TEMPORAL LOGIC

Heiko Krumm

University of Dortmund, Department of Computer Science

Symbolic logic generally supports the reasoning with propositions, i.e., with statements to be evaluated to true or false. Temporal logic is a special branch of symbolic logic focussing on propositions whose truth values depend on time. That contrasts with the classical logic point of view where the truth value of a repeatedly uttered proposition must always be the same and must neither depend on the modalities of the repetition nor on additional information. Temporal propositions typically contain some (explicit or implicit) reference to time conditions, while classical logic deals with timeless propositions. For instance consider the following examples:

A: "The moon is a satellite of the earth"
B: "The moon is rising"
C: "The moon is setting"

Proposition *A* can be viewed as timeless, since it is true in past, present, and future. In contrast, the propositions *B* and *C* have a temporalized aspect and refer to the implicit time condition "now". Consequently temporal logic applies to time-related universes of discourse where behaviors and courses of events are of interest. As classical logic formulas can characterize static states and properties, temporal logic formulas can describe sequences of state changes and properties of behaviors.

Classical logic comprises different logics; several variants of propositional logic, first order predicate logic, etc., exist with different sets of logical operators and inference rules. Likewise some temporal logics were proposed which differ with respect to their formula syntax, semantics, and expressiveness. A temporal logic, however, basically results from an extension of a classical propositional or predicate logic by temporal quantifiers introducing temporalized modalities. Usually, there are at least the two quantifiers ■ (denoting "always") and ◆ (denoting "eventually") and typical formulas are similar to following examples:

D: ◆ *B*	"The moon will be rising eventually"
E: ■ ◆ *B*	"The moon will be rising again and again"
F: ■ (*B* ⇒ ◆ *C*)	"Moon rise leads to moon setting"

The example formula *D* is true, if the moon is rising now or will be rising in some future point of time. Formula *E* exemplifies that combinations of temporal quantifiers can denote more complex time conditions, e.g., "always eventually" can correspond to the natural language term "again and again". Finally, formula *F* is an example of a "leads-to" pattern describing that always a precondition *B* will eventually result in a postcondition *C*.

Due to its temporal quantifiers temporal logic is a convenient and appropriate means to reason with time-related propositions. Indeed, classical logic can also handle temporal properties, but the formulas tend to be complicated since points of time have to be explicitly represented in the underlying universe. The formula *E* may serve as example and underpin the usefulness of temporal logics. The easy-to-read temporal logic formula *E* corresponds to following predicate logic formula: "For all subjects *x* a subject *y* exists such, that – if *x* is a point of time – *y* is a point of time equal or later to *x* and the moon is rising at *y*".

History

Temporal logic is rooted in the field of exact philosophy and is a variant of modal logic. Modal logic deals with propositions which are interpreted with respect to a set of possible worlds. The truth value of propositions depends on the respective world and basically two operators "necessarily" and "possibly" exist which denote that a proposition is true in all possible worlds res. in some possible worlds. Even the ancient Greek philosophy schools of the Megarians, Stoics, and Peripatetics as well as Aristotle used some temporalized form of these modal operators. During the Middle Ages Arabian and European logicians resumed and refined the ancient approaches in order to discern different types of necessity and possibility. In modern times, the interest in symbolic logic grew during the first half of the 20th century, and – with some delay – new modal and temporal logic approaches occurred. First publications date back to the 1940's.

In particular, the logicians Prior, Rescher, Kripke, and Scott contributed to the development of modern temporal logic. Kripke presented a formal possible world semantics for modal logics. Prior proposed a temporal interpretation. An ordered set of possible worlds can correspond to a temporal sequence of states. In result, the two basic modal operators "necessarily" and "possibly" become the temporal quantifiers "always" and "eventually". Based on the linearity of time additional operators like "next" and "until" as well as past operators were introduced. Rescher and Urquhart outlined the history and introduced several major systems of temporal logic in [5]. In 1974, Burstall proposed the application of temporal logic in computer science for the first time. Pnueli improved this approach in [4], which is regarded as the classic source of temporal logic based program specification and verification.

Computer Science Application

In several fields of computer science there is a needs for the formal description of event-discrete processes and the corresponding reasoning. In the main, we have to mention the formal specification and verification of so-called reactive systems, the formalization of real-life processes as well as the semantics of natural language commands to be modeled in artificial intelligence, and finally the handling of dynamic consistency conditions in data base systems.

We focus on reactive systems. In particular, Manna and Pnueli recognized in [3] that reactive systems are of growing interest and that temporal logic is well-suited for their formal specification and verification. In contrast to those programs which transform starting states into final results and which may be specified by pre- and postconditions, reactive systems interact with their environment during runtime and the course of interactions and system events is essential. The range of reactive systems is wide and growing. It comprises embedded systems, process control systems, and all types of interactive, concurrent or distributed hard- and software systems. Due to their inherent concurrency, their elaborated fault-tolerance, coordination, and interaction mechanisms distributed systems are rather complex reactive systems and usually need particular design and development tools which support the formal handling of dynamic aspects. Here, temporal logic is profitably applied with respect to following topics:

1. Formal specification: Temporal logic formulas serve as precise, concise and binding descriptions of systems and components (e.g., as proposed by Lamport, Manna, and Pnueli in [2] res. [3]).
2. Formal verification: The rules of a temporal logic proof calculus are applied to show the correctness of a temporal logic specification with respect to more abstract system specifications (e.g., in [2] and [3]).
3. Requirements description: During the early system design the results of the requirements constraining the functional system behavior are represented by a set of temporal logic formulas.
4. Specification checks: Even if the design specifications use other means than temporal logic (e.g., other formal description techniques, *see* SDL, Estelle, and LOTOS, *see also* UNITY), temporal logic may be applied additionally in order to describe requirements and plausibility conditions. Meanwhile several approaches exist which support the tool-based checking of formal system specifications with respect to temporal logic conditions (*see* Model Checking).

Linear and Branching Time

Usually, a temporal logic can be classified as so-called linear-time logic which considers behaviors modeled as linear sequences of states. Within one behavior, each state has exactly one future. Additionally, so-called branching-time logics are known. Here, the formulas refer to tree-structured behaviors where a state can have several futures. The behavior-trees can directly correspond to tree models of non-deterministic systems (e.g., synchronization and communication trees, *see* Calculus of Communicating Systems). A corresponding prominent branching-time logic is CTL (computation tree logic, proposed by Clarke, Emerson, and Sistla in [1]). Its temporal quantifiers directly support the navigation in behavior trees.

Non-deterministic systems, however, have not necessarily to be modeled by behavior trees. Likewise, a set of linear state sequences can form a model of a non-deterministic system where each state sequence corresponds to one possible evolution of the system. In comparison with this linear-time approach, branching-time logics additionally provide for notions of potential behaviors since branching-time formulas can describe properties of branches which correspond to subsets of the possible execution sequences while linear-time formulas generally state properties of all possible sequences.

Variants

Besides of the mentioned distinctions between temporal propositional and predicate logics and between linear-time and branching-time logics, there exist further variants. Some introduce additional temporal quantifiers like “always in the past”, “sometimes in the past”, “next”, “precedes”, “until”, and “leads-to”. Others extend the time model, e.g., in order to describe time-intervals or real-time quantifications. Furthermore, partial-order temporal logics were proposed which directly refer to partial-order representations of concurrency (*see* Concurrency Model).

Example

To exemplify the application of temporal logic for the specification and verification of systems we outline some formula and proof patterns proposed by Lamport in [2] with respect to the Temporal Logic of Actions TLA which is a compact linear-time logic for the reasoning on state-transition systems. He considers the two commonly known classes of properties, invariance and eventuality. Moreover, the correctness of design refinements can be proven with respect to the preservation of properties.

An invariance property P is expressed by a formula "$\blacksquare\, P$" where P is a predicate logic formula describing a set of execution states. Inter alia P may specify following typical correctness conditions of a system:

1. Partial correctness: P is an implication of the form "system terminated $\Rightarrow$ correct results computed".
2. Deadlock freedom: P applies to a set of states, the system is not deadlocked.
3. Mutual exclusion: P asserts that at most one process is in a critical section.

By means of auxiliary history variables all interesting safety properties of a system can be expressed as invariance properties (*see* Safety Property).

The formal proof of invariance properties is supported by a proof rule applying induction on the course of system execution steps. At first, one proves that each initial state implies P. Furthermore, each transition class of the system has to be considered. Each transition has to transform states where P is true into successor states where P is true again.

Eventuality properties assert that some events will eventually happen during each execution of a system. The following typical properties can be easily expressed in temporal logic:

1. Termination: A formula of the form "$\blacklozenge$ terminated" can assert that each execution leads to a state where the system is terminated.
2. Live service: Each state representing that a service request is pending will be followed by a state the request is served: "$\blacksquare$ (requested $\Rightarrow \blacklozenge$ served)".
3. Fair message transfer: If a message is sent often enough over a loose channel, then it is eventually delivered: "($\blacksquare\, \blacklozenge$ sent) $\Rightarrow$ ($\blacklozenge$ delivered)".

Eventuality properties can express the typical liveness requirements of systems (*see* Liveness Property).

The proof of eventuality properties can be reduced to the proof of a series of transitive leads-to properties of the form "$\blacksquare\, (P \Rightarrow \blacklozenge\, Q)$". The proof of a single leads-to property is supported by the so-called lattice rule which is based on the existence of a well-founded order. The order asserts that a finite number of execution steps is sufficient to reach a state where Q is true.

Systems can be described by formulas on abstract levels as well as on more implementation-near ones. Thus, specifications, refinement steps of a design, and implementations can be represented. That is of great interest, since valid implications correspond to system refinements which are correct in the usual understanding of system developers. Let the formula *Spec* describe a system S on a more abstract level. A formula *Impl* describes a correct refinement of S, if the implication formula "*Impl* $\Rightarrow$ *Spec*" is provable.

References

[1] E.M. Clarke, E.A. Emerson, and A.P. Sistla, *Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications*, ACM Transactions on Programming Languages and Systems, 8(2): 244-263, 1986

[2] L. Lamport, *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, 16(3):872-923, 1994

[3] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992

[4] A. Pnueli, *The Temporal Logic of Programs*, Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46-57, 1977

[5] N. Rescher and A. Urquhart, *Temporal Logic*, Springer-Verlag, 1971

Cross Reference:

CTL *see* Temporal Logic

Formal Specification *see* Temporal Logic

Formal Verification *see* Temporal Logic

TLA *see* Temporal Logic

Dictionary Terms:

Concurrency Model

Model representing the global dynamics of a system which consists of concurrently acting components. Mainly, there are two types of concurrency models. Interleaving models induce a total temporal ordering of all component actions. Thus, the system is assumed to perform a global sequence of actions and the model reduces concurrency to non-determinism. In contrast, partial-order models represent the temporal independence of concurrent events directly. Concurrent events are not comparable with respect to the order of events.

Liveness Property

Property of a system concerning its dynamics and expressing that the system will eventually show a particular behavior within a finite period of time. Together with safety properties (*see* Safety Property) liveness properties can be used to characterize the principal functionality of distributed systems.

Safety Property

Property of a system concerning its dynamics and expressing that the system behavior never injures particular conditions, e.g., never enters forbidden states. Together with liveness properties (*see* Liveness Property) safety properties can be used to characterize the principal functionality of distributed systems.