Formal Verification of Timed Systems: A Survey and Perspective

FARN WANG

Invited Paper

An overview of the current state of the art of formal verification of real-time systems is presented. We discuss commonly accepted models, specification languages, verification frameworks, state-space representation schemes, state-space construction procedures, reduction techniques, pioneering tools, and finally some new related issues. We also make a few comments according to our experience with verification tool design and implementation.

Keywords*—Embedded systems, formal methods, formal verification, models, real-time systems, specification, temporal logics, theory, tools.*

I. Introduction

Real-time systems differ from untimed systems in that their behavioral correctness relies not only on the results of their computations, but also on the clock times when the results are produced. Formal verification means to rigorously explore the correctness of system designs expressed as mathematical models, most likely with the assistance of modern computers. From our viewpoint, there have been the following three motivations for the heated research on the formal verification of real-time systems in the last two decades.

- With the success of formal verification in the very large scale integration (VLSI) industry [53], it is natural to expect that similar success can be repeated in the formal verification of real-time systems. In particular, the achievement of binary decision diagram (BDD) technology [52] has raised the hopes and confidence of the industry for the verification of real-time systems.
- With the readiness of concrete theoretical frameworks for the verification of real-time systems [5], [11], [113], [127]–[129], [133], [197], both programmers and theoreticians are eager to see how the theory adapts to real-world projects.
- With the increasing scope and complexity of embedded systems and resulting state-space explosion, it is becoming less and less likely that we can run a sufficient number of simulation traces to gain both enough coverage of the state spaces and enough confidence in the systems with a project schedule. Further, even if it were feasible to have extensive coverage of the system, the potential of a single untested sequence of events that no one thought of to cause system failure is also of concern.

In the last two decades, many achievements in the formal verification of real-time systems have been reported, from various solid theory foundations to complex implementation techniques and formal verification of many real-world projects [37], [189], [203], [206]. Still the intrinsic complexity of various framework for real-time system verification is forbiddingly high. The verification problems of timed systems are usually exponentially more complex than their untimed counterparts. For example, the model-checking problem of computation tree logic (CTL) is in PTIME[1] [60], [61], while that of timed CTL (TCTL) is in PSPACE[2] [5]. Thus, in the foreseeable future, it will be difficult to use formal techniques alone for decisive answers to complex verification tasks.

But this does not mean that we are pessimistic about the future of formal verification. On the contrary, with most major projects currently spending over 50% of their development costs in verification and integration, there are tremendous opportunities for using formal verification to sizably reduce the

Manuscript received November 15, 2003; revised April 19, 2004. This work was supported in part by the National Science Council (NSC) of Taiwan, R.O.C., under Grants NSC 92-2213-E-002-103 and NSC 92-2213-E-002-104, and in part by the System Verification Technology Project of the Industrial Technology Research Institute, Taiwan, R.O.C.

The author is with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: farn@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/JPROC.2004.831197

[1]PTIME problems can be solved with time complexity polynomials to the input sizes in bit counts.

[2]PSPACE problems may incur memory consumption polynomials to the input sizes in bit counts.

explosive growth of verification and integration costs and to enhance the quality of system designs in industry. On the one hand, for complex real-time systems, formal verification will likely be used to enhance the intelligence and performance of simulation and testing. For example, coverage metrics can be more precisely mapped to the functions to be verified. On the other hand, for targets with clean modularity and interface, formal verification can be used to rigorously check the components and the interfaces and gradually could be accepted as standard methods in the automation of industrial quality control. Actually it is claimed that this latter approach has already had a dramatic effect on the SLAM project of Microsoft, which plans to incorporate model-checking capability in its Windows driver development kit (DDK) [34].

In this paper, we give a review of these many achievements so that readers can use the paper as an index to the literature. We organize the paper according to the various research topics in formal verification, including models in Section II, description and specification languages in Section III, verification frameworks in Section IV, representation of state space in Section V, constructions of state-space representations in Section VI, reduction techniques for representations in Section VII, some tools in Section VIII, and some other issues in Section IX, including symbolic simulation, parametric analysis, controller synthesis, probabilistic analysis, and worst case execution time (WCET) analysis.

There are many frameworks to choose from to complete a verification task. Each framework has its unique advantage and may incur an intrinsic challenge. We feel it is better to let the readers be informed of the challenges in the verification frameworks of his/her choice. Thus we have cited many complexity results of various verification problems in the paper. Complexity of a verification problem means the order of growth of required resources to solve the problem with respect to the input sizes in bit counts. The resources can be CPU times, memory space, message counts, power consumption, etc. But in this paper, we are mainly concerned with CPU times and memory space. Some jargon of the complexity classes includes *PTIME*, *NP-complete*,[3] *PSPACE*, *EXPTIME*,[4] *EXPSPACE*,[5] *nonelementary complexities*,[6] and *undecidability*.[7]

Heitmeyer and Mandrioli have also written a handbook on formal verification techniques for real-time computing [106]. Regarding techniques for untimed systems, a classic book is one by Clarke *et al.* [65]. A previous survey paper in this regard is by Ostroff [179].

[3]NP problems means that we can guess a solution in time complexity polynomials to the input sizes. NP-complete problems are the hardest ones in NP and are in general considered untamable problems in computer sciences.

[4]EXPTIME problems consume CPU times exponential to the sizes of inputs in bit counts.

[5]EXPSPACE is the set of problems that at most consume memory capacity exponential to the input sizes in bit counts. EXPSPACE-complete problems are harder than EXPTIME problems.

[6]Nonelementary complexities are like $2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$ with the heights of the exponent stacks at least proportional to the input sizes in bit counts.

[7]Undecidable problems do not guarantee termination. In general, it is not possible to design algorithms (procedures that guarantee termination) for undecidable problems.

Before you go on, we remind you that the paper may show the author's intentional or unconscious bias toward each approach in formal verification. After all, the amount of space needed to explain the details of each subfield is a subjective decision.

II. Models

Formal verification grows from formal or mathematical logics [41], in which we discuss the grammar (syntax) and meaning (semantics) of logic formulas. It is possible to associate the same grammar with different styles of meaning. The meaning of a logic formula is defined as a set of *models*. A *model* in mathematical logic is a domain of values and some functions on the domain. Without such formal definitions, rigorous and mechanical verification of real-time systems will be impossible.

Intuitively, in the forum of specification and verification, a model is a behavior of a system description (or specification). According to the various frameworks we use, a model for a real-time system can be a state set, a state sequence, an event sequence, a state tree, or an infinite domain with relations. Some other possibilities can also be found in [80].

A. Linear Time Versus Branching Time

We can view a computation either as a linear sequence with only one future or as a tree with many possible futures. The former is called *linear-time semantics* [185], while the latter is called *branching-time semantics* [60], [61].

Linear-Time Temporal Logics: The research on automatic verification of computer programs was initiated when Pnueli proposed using *linear-time propositional temporal logic (LPTL)* [185] to specify and compute the behaviors of computer systems. LPTL is a subclass of modal logic [41] with *possible-world semantics* and modal operators: $\Box$ (for all possible worlds) and $\Diamond$ (there exists a possible world). In LPTL, $\Box$ is interpreted as "from now on, at all states" (or henceforth, always), while $\Diamond$ is interpreted as "from now on, there exists a state" (or eventually). For example, we may have the model of a railroad crossing system. `approaching` and `down` are two atomic propositions, and we want to specify that whenever an approaching train is detected, from that state on, eventually the gate is down. In LPTL, this can be expressed as

$$\Box(\texttt{approaching} \rightarrow \Diamond \texttt{down}).$$

Two other commonly used modal operators are $\bigcirc$ (next) and $\mathcal{U}$ (until). $\bigcirc p$ means that p is true in the next state. $p\mathcal{U}q$ means that p is true until q is true.

In defining logics, people usually try to use minimal syntax structures. Usually $\Diamond p$ and $\Box p$ can be defined as the shorthands for true$\mathcal{U}p$ and $\neg\Diamond\neg p$, respectively. On the other hand, Kamp showed that $p\mathcal{U}q$ cannot be modeled with modal operators $\Box$, $\Diamond$, and $\bigcirc$ [131].

In the late 1980s, people added the concept of "clock time" to LPTL. That is, a global clock is assumed in the model such that the global clock does not have to increment its reading

at every state. Initially, Ostroff [178] discussed issues in expressiveness and complexity with quantification and linear constraints of clock readings in LPTL. For example, we may write

$$\forall x \exists y \Box \Big((T = x \wedge \texttt{approaching}) \\ \rightarrow \Diamond (T = y \wedge y - x \leq 300 \wedge \texttt{down}) \Big). \quad (1)$$

Here T is the special variable for the reading of the global clock at the current state.

Alur and Henzinger proposed *timed propositional temporal logic (TPTL)* [15]. TPTL has a clock-reading freezing modal operator and uses binary difference constraints between frozen clock readings. The intuition is that quantifications on clock readings following $\Box$ are universal, while the ones following $\Diamond$ are existential. For example, we may write

$$\Box x. (\texttt{approaching} \rightarrow \Diamond y. (y - x \leq 300 \wedge \texttt{down})) \quad (2)$$

which means that the gate will be down in 300 s from any state in mode approaching. Note that this formula specifies something different from (1). In (1), time y is independent of the states quantified by the modal operator $\Box$, while in (2), it is not.

Alur and Henzinger also defined *metric temporal logic (MTL)*, which allows the specification of timing distances between states quantified by adjacent modal operators [16]. For example, we may write

$$\Box(\texttt{approaching} \rightarrow \Diamond_{\leq 300} \texttt{down})$$

to specify the same property as (2). A nice exploration of various discrete-time extensions of LPTL is [16].

In 1992, Wang *et al.* extended TPTL to *Asynchronous PTL (APTL)* for distributed systems with clock jitters [228]. Specifically, they redefine the semantics of clock differences in distributed systems with a timing precedence relation. The idea is that instead of comparing the values of clock readings, we now compare the temporal precedence of clock readings. For example, we may write $\Box[x, y]x + 2 \lhd y + 1$, which means that we have two distributed clocks such that at every state, if the reading of the two clocks are x and y, respectively, then every second tick of the first clock must precede the next tick of the second clock. Putting it another way, for every tick of clock 2, clock 1 will tick at least twice.

Branching-Time Temporal Logics: The intuition behind linear-time logics is that there is only one future. With branching-time logics, the possibility of many futures is assumed, and modal operators $\exists$, $\forall$ are provided to specify the relation among different futures. Path quantifier $\exists$ means "there exists a run from now on" and $\forall$ means that "for all runs from now on." For example, in CTL [60], [61]

$$\forall \Box (\texttt{approaching} \rightarrow \forall \Diamond \texttt{down})$$

means that whenever the monitor is in mode approaching, along all runs henceforth, the gate will eventually be closed. Note that in CTL and its extensions, linear-time modal operators ($\Box$, $\Diamond$, $\mathcal{U}$, $\bigcirc$) must immediately follow a path quantifier

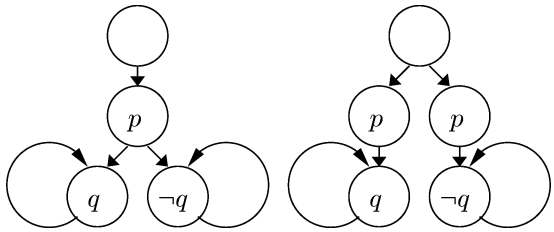

Fig. 1. Two behaviors that CTL can differentiate.

($\exists$, $\forall$). In the literature of CTL, modal operators $\exists$, $\forall$, $\Box$, $\Diamond$, $\mathcal{U}$, and $\bigcirc$ are written as E, A, G, F, U, and X, respectively.

In semantics, an LPTL formula defines a set of linear state sequences, while a CTL formula defines a set of computation trees. CTL and LPTL are not comparable in expressiveness. For example, the LPTL formula $\Box\Diamond p$ says p states happen infinitely many times. Also, the *nonZeno* requirement [113] in TPTL is $\Box x.\Diamond x > 1$. These properties are known to be inexpressible in the timed extensions of CTL. On the other hand, CTL allows for the reasoning of properties between possible future behaviors. For example, the CTL formula

$$\forall \Box (p \rightarrow ((\exists \bigcirc q) \wedge (\exists \bigcirc \neg q)))$$

can differentiate the two behaviors in Fig. 1, while no LPTL formula can.

Emerson *et al.* proposed *real-time CTL (RTCTL)* [83], which uses formulas like $\exists \neg p \mathcal{U}^{\leq k} p$ to specify the existence of a state sequence along which p becomes true within the next k states. This type of property is suitable for cycle-based systems like VLSI, in which the discrete global clock ticks at every discrete state change [83].

Harel *et al.* [104] discussed the expressiveness and complexity of the CTL extension with universally quantified clock variables and arbitrary linear clock constraints.

The most used branching-time temporal logic for real-time systems is *TCTL* [5], which supports modal operators like $\exists\Box_{\sim c}$, $\exists\Diamond_{\sim c}$, $\exists\mathcal{U}_{\sim c}$, $\forall\Box_{\sim c}$, $\forall\Diamond_{\sim c}$, and $\forall\mathcal{U}_{\sim c}$. Here c is a natural number and $\sim$ is one of $\leq$, $<$, $=$, $>$, or $\geq$. For example

$$\forall \Box (\texttt{approaching} \rightarrow \forall \Diamond_{\leq 300} \texttt{down})$$

means that whenever the monitor is in mode approaching, along all runs henceforth, the gate will be closed in 300 s.

Beside its modality, TCTL also differs from TPTL and the like in that TCTL is defined with dense-time clock models. Thus, there is no modal operator involving $\bigcirc$.

Integration of Linear Time and Branching Time: Emerson and Halpern defined a unified specification language, called CTL^*, for both LPTL and CTL [81]. Remember that in CTL, we require that linear-time modal operators must immediately follow path quantifiers. If this restriction is lifted in CTL, then we get CTL*. Regarding LPTL, each LPTL formula specifies the set of linear sequences satisfying it. Thus, intuitively, LPTL is a subclass of CTL* because every LPTL formula implicitly carries a universal quantifier on all computation sequences. That is, an LPTL formula ϕ characterizes the same set of state sequences as the CTL* formula $\forall\phi$.

It is also natural to extend CTL* to TCTL*. Möller discussed how to model check a subclass of TCTL* [167]. Wang carried out experiments using model-checking algorithms on the subclass of TCTL* which contains fairness assumptions [220].

B. Discrete Time Versus Dense Time

In the definition of real-time system models, we can require that all time readings are integers and all clocks increment their readings at the same time. This is discrete-time semantics [15], [127]. The other choice is dense-time semantics [11], [78], which means that time readings can be rationals or reals and all clocks increment their readings at a uniform rate.

Discrete-time models are suitable for synchronous systems where all concurrent processes share the same global clock. Example languages are TPTL and MTL [16]. Dense-time models are better for distributed systems with multiple clocks and timers, which can be tested, set, and reset independently. Examples include TCTL [5].

Note that in our explanation of discrete-time models, we require that there is a single global clock. It may seem that such a requirement is inappropriate for distributed systems with digital clocks. In fact, after the sampling or detection of an event in an embedded system, the occurrence time of the event is still stored in a digital format. Thus, philosophically, discrete-time semantics with distributed clocks seems a plausible choice. But in general, if the clocks are distributed and do not increment their readings at the same time, then we still have to implicitly record the ticking order of the clocks so that the clocks still increment their readings at the same rate. The information pieces of ticking orders are of factorial complexity, which is the same as that of region graph construction for dense-time models [5], and do not save any computation resources. Henzinger *et al.* discussed the relation between systems with distributed digital clocks and those with distributed dense clocks [112].

Clock-reading models can also affect the verification complexity. For example, the satisfiability problem of TPTL is in EXPSPACE, while its dense-time version is undecidable. Intuitively, discrete-time models may lead to lower complexity in verification and analysis, since there are many fewer states. But in practice, the impact on complexity is tricky. Specifically, with the symbolic techniques for timed and hybrid systems [18], [113], state spaces are represented as Boolean combinations of linear constraints of state variables. An important operation is checking the emptiness of state-space representations. However, the emptiness problem of linear constraints is in PTIME for reals (in dense-time models) and NP-complete for integers (in discrete-time models).

C. State Based Versus Event Driven

A state is a snapshot of a system at a moment in time. In the sense of control, it is everything that we need to know about the system at that moment in order to determine the future for all future input sequences. Mathematically, it consists of the recording of the values of all variables and control locations. For instance, at a railroad crossing, the variable `gate` can be used for the gate position of the present state. State-based real-time temporal logics are usually derived from their untimed counterparts [60], [61], [185]. Examples include TCTL [5], RTCTL [83], TPTL, MTL [16], and APTL [228].

An event represents an instantaneous change of states and may trigger a series of responses in a system. For example, the detection of an approaching train at a railroad crossing is an event that will hopefully cause the gate to close. Event-driven system descriptions are very natural in the world of embedded system engineering. One pioneering work that incorporates timed events into the linear-time model is *real-time logic (RTL)* by Jahanian and Mok [127]. RTL is a linear-time event-driven logic with event occurrence-time functions. For example, we may write

$$\forall i\,(\texttt{approaching}(i) + 300 \geq \texttt{down}(i))$$

which means that after the ith train `approaching` event has occurred, the ith gate `down` event must happen in 300 s. RTL is a subclass of first-order integer arithmetic with monadic functions. Yang *et al.* [233] have also designed a temporal logic, synchronous real-time event logic (SREL), based on event countings along state sequences in a discrete-time domain.

One thing to note is that in synchronous cycle-based systems, e.g., VLSI circuits, people usually treat state and event as the same thing, since variables change values only at the beginning of each clock cycle and then remain steady in the cycle. But for distributed real-time systems, this treatment may lead to imprecise modeling, since there is no common clock among the many distributed sites.

It also has long been argued that neither pure state-based nor pure event-based languages quite support the natural expressiveness desirable for the specification of real-world systems [57], [122], [134], [137], [174]. Recently, Wang has proposed an extension to TCTL for the specification and model checking of behaviors involving both states and events with fairness assumptions [220]. For example, in the railroad crossing system, we may want to say that when the monitor signals the controller to `lower` the gate, then the gate must be `down` in 300 s. This can be specified as $\forall\Box^{\texttt{lower}\geq 1}\forall\Diamond_{\leq 300}\texttt{down}$. Here $\Box^{\epsilon\geq c}p$ means that whenever c events of type ϵ happen, p must immediately be true.

III. Description and Specification Languages

A. Timed Automata

The model of *timed automata* was first proposed by Alur and Dill [11]. A timed automaton is a finite-state automaton equipped with a finite set of clocks which can hold nonnegative real values. It is structured as a directed graph whose nodes are *modes (control locations)* and whose arcs are *transitions*. The example of a monitor process for trains approaching a railroad crossing is in Fig. 2. The monitor is in mode *far* when all trains are far from the crossing, or in mode *approaching* when a train will arrive at the crossing within 300 s, or in mode *crossing* when a train is at the crossing, or in mode *passed* when trains have just left the crossing within

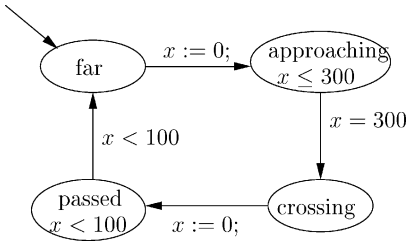

Fig. 2. The model of gate–monitor.

the last 100 s. The ovals represent modes, while the arcs represent transitions. The modes are labeled inside the ovals with *invariance conditions* which are Boolean conjunctions. All states in a mode must satisfy the corresponding invariance condition. At any moment, the timed automaton can reside in only one mode. The transitions are labeled with *triggering conditions* and a set of clocks to be reset during the transitions. A timed automaton can make a transition only if it satisfies the corresponding triggering condition. The invariance conditions and triggering conditions are Boolean combinations of clock (difference) inequalities. In a timed automaton's operation, one of the transitions can be triggered when the corresponding triggering condition is satisfied. Upon being triggered, the automaton instantaneously transits from one mode to another and resets some clocks to zero. In between transitions, all clocks increase their readings at a uniform rate.

A state of a timed automaton is a recording of its present mode and the readings of all clocks. With discrete-time models, all clocks will have integer readings and increment their readings at the same instant. With dense-time models, clock readings are reals (or rationals). A computation of a timed automaton can be defined as an infinite sequence of time-state pairs such that the time components form a nondecreasing and divergent real number sequence.

A timed automaton is a nondeterministic machine and does not have to make transitions as long as the invariance condition is satisfied. One usual way to force transitions is by using an invariance condition. For example, in Fig. 2, in the "approaching" mode, we require that $x \leq 300$ to force the transition out of the mode in 300 time units.

In practice, a real-time system is usually described as a set of process timed automata, each representing the behavior of an autonomous process. Such a decomposition is natural in the description and construction of concurrent and distributed systems. By decomposing a timed automaton to a set of process automata, engineers can greatly simplify their tasks in the modeling of complex, concurrent behaviors. The timed automaton can be constructed as the product automaton of the process timed automata. The mode set of the product automaton is now the Cartesian product of the mode sets of the process timed automata. The invariance condition is defined by the conjunction of the invariance conditions of the modes of the process timed automata. The behavior of the product timed automaton is an interleaving of transitions of the process timed automata.

What makes timed automata interesting is that their model-checking problem is in PSPACE [5]. Moreover,

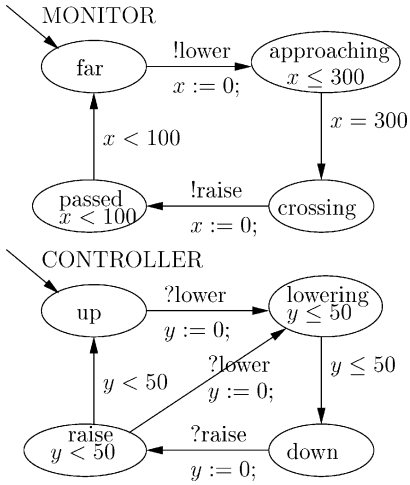

Fig. 3. The model of gate–monitor–controller.

under certain conditions, symbolic manipulation techniques for timed automata have performed well enough to verify many industrial systems [113].

B. Communicating Timed Automata

Complex state transitions may require cooperation among processes to construct. For example, in the railroad crossing example, at the time when an approaching train is detected, the monitor sends a "signal" to the gate controller to move the gate down. Conceptually we can decompose the system into two processes: the train monitor and the gate controller. The sending and receiving of the signal can best naturally be modeled by the interaction between the monitor and the controller. There are also two process transitions involved in the behavior, i.e., the detection of the approaching train and the starting of the gate's moving down.

The first language device designed to "glue" process transitions into a (global) transition is the *channel* concept for binary synchronization in Hoare's *Communicating Sequential Processes* (CSP) [115]. Such a device can greatly help to improve the modularity of model descriptions. For example, in the embedded control system for a railroad crossing, we may have a channel called `lower` for the communication of the command to move the gate down. The language device `!lower` represents the sending (or output) event through the channel, while `?lower` represents the receiving (or input) event through the same channel. Two process transitions labeled, respectively, with the sending event and receiving event through the same channel must happen at the same instant to make a legitimate global transition. The interactive synchronization between a train monitor and a gate controller can be modeled with the *communicating timed automaton* in Fig. 3. When the monitor detects that a train is approaching the railroad crossing, it sends out a signal through channel `lower` to the controller to move the gate down. A process transition with an input event that cannot find a peer process transition with matching output event simply cannot be executed.

CSP-style synchronization channels were incorporated in communicating real-time state machines (CRSM) [198] in

1992. Such devices are now supported in real-time system model-checkers like I/O automata [90], [133], [157], UPPAAL [39], [183], SGM [120], [222], [223], and the region-encoding diagram (RED) [213], [216], [217], [219].

A similar synchronization device is used in HyTech [18] and Kronos [75], [237] with no distinction between senders and receivers. Each process is declared with the set of synchronizers to respond to. Then when a synchronizer happens, all processes declared to respond to it will have to make a process transition. Such a device is handy in modeling broadcasting, multicasting, and other multiparty synchronizations. For example, to model the bus collision behavior in carrier sense multiple access with collision detection (CSMA/CD) protocol [175], the bus process may execute a transition labeled with the event `Collision_Detected`, which will force every sender process to respond to a transition with the same event. It is also possible to combine basic CSP-style binary synchronizations to construct such complex multiparty synchronizations. For example, in RED [219], to model the `bus collision` event in CSMA with n senders, the bus needs to execute a transition with n `!Collision_Detected` labels, and, thus, forces each of the n senders to respectively respond with a transition with one `?Collision_Detected` label.

C. Hybrid Automata

Hybrid automata [18], [108] are used to model embedded systems with continuous variables, whose values may increment/decrement at various rates. Arbitrary linear constraints are allowed for invariance conditions and triggering conditions. Timed automata [11] are a special subclass of hybrid automata in which all continuous variables increment their values at a uniform rate and only upper-bound and lower-bound inequalities of clocks are allowed. A hybrid automaton shares the same structure as timed automata, except that in each mode, the increment rate of each continuous variable is independently specified.

In general, hybrid automata are not subject to algorithmic verification, but there are subclasses which are [109].

D. Logics

Logic formulas can also be used to describe system behavior. In such frameworks, the system descriptions (including the models for systems and environments) and the specifications are all put down in the same language. In its operation, a logic theory is defined by a set of axioms and inference rules. Every formula that can be constructed through repetitive applications of the inference rules is called a *theorem*. Two key issues are *soundness* and *completeness*. Soundness means that the theorems in the theory are consistent, i.e., do not contradict one another. Completeness means that every theorem can be proved in a finite number of steps of inference rule applications. When we use logics for verification, we usually put down the system description as axioms and inference rules and the specification formulas as a theorem. A proof plan may also involve several lemmas and corollaries as intermediate steps to the establishment of the goal theorem.

To know when to stop inferencing in case the specification formula is actually not a theorem, we need to prove a *small model theorem* to establish the maximum number of inference steps for the proof of the specification formula. The complexity of the verification procedure depends on the complexity of the small models.

Propositional temporal logics, linear time and branching time, have been discussed in Section II-A. The advantage of such frameworks is that they usually come with small model theorems and algorithms to check whether a formula is a theorem or not. The disadvantage is that they are usually not expressive enough to model complex behaviors with, for instance, queues, stacks, counters, range-unbounded variables, polynomial constraints, etc.

First-order and higher order logics have also been used [127], [197] for specification and verification. The advantage is that such logics are very expressive for modeling complex behaviors. The disadvantage is that in general, it is not possible to design algorithms to check whether a formula is a theorem or not.[8] First-order logics seem a good compromise between expressiveness and computability, since they are complete in general. That is, we still have *semidecision* procedures that are guaranteed to construct proofs if the specification formulas are indeed theorems.

On the other hand, higher order logics [96], [181] are in general incomplete. Interested readers are refered to [176] for a tutorial on Isabelle/HOL. Mattolini and Nesi presented *temporal-interval logic with compositional operators (TILCO)* [161] with formal proof support from Isabelle/HOL.

E. Timed Process Algebras

Timed CSP was first designed by Reed and Roscoe [188] and later modified by Davies and Schneider [74]. It supports the following three grammar rules with timing constraints:

$$\text{Wait } t | P_1 \overset{t}{\triangleright} P_2 | P_1 \overset{\downarrow}{t} P_2.$$

"Wait t" means to wait for t time units.

$P_1 \overset{t}{\triangleright} P_2$ models the timeout in communication. It behaves as P_1 until, at time t, when no synchronization has happened yet, the control is transfered to P_2.

$P_1 \overset{\downarrow}{t} P_2$ models the interrupt in embedded systems. It behaves as P_1 until at time t, regardless of synchronizations, when control is transfered to P_2.

The time value domain in timed CSP is dense. A variety of semantics has been defined. The simplest one associates a program with a set of timed traces. Schneider wrote a book in this regard [194]. In the book, manual proofs are illustrated and a refinement relation from untimed CSP programs to timed CSP programs is presented. With the introduction of the event *tock*, which advances time by one unit, Schneider also showed how to translate timed CSP programs to untimed CSP programs with tock events.

Baeten *et al.* have done substantial work in extending the process algebra known as the *algebra of communicating*

[8] In the jargon of computer science, the *decision problems* of such logics are said to be *undecidable* or *incomputable*.

processes (ACP) to the real-time domain [30]–[32]. The time models can be dense or discrete. Both absolute time and relative times can be specified. Time-stamped actions can be used to combine actions with the passage of time. The integration operator $\int_{v \in V} P$ allows composition over a continuum of alternatives. The initial abstraction operator $\surd v.F(v)$ defines a mapping from a parametric initial setting of real variable v to a set of processes, whose behaviors depend on the value of v.

Work on the *calculus of communicating systems (CCS)* with timing can be found in [59], [170], and [234].

Another timed extension of process algebra is PACSR [152], [184], which supports resource-awareness and probability reasoning in embedded system design.

LOTOS is an ISO standard specification language based on process algebra [43], [123]. Its real-time extension is called *Enhanced Timed-LOTOS (ET-LOTOS)* [124], [150], [151].

F. Timed Petri Nets

Petri nets [87], [130] are convenient for modeling concurrent systems. In a Petri net, we have places, which may hold tokens, and transitions, which may consume some tokens from the places and produce some other tokens in other places. A state of a Petri net is called a *marking*, which assigns a number of tokens to each place. Since each place can hold an unbounded number of tokens, Petri nets are infinite-state systems. One special thing about Petri nets is that they lack the capability to test the nonexistence of tokens. Specifically, a transition may happen if all its input places have tokens. But no action can be taken when there is no token in some input place. Theoretically, Petri nets are equivalent to *counter machines without zero-test capability* [97] and *vector addition systems* (VAS) [118], [132].

Several classes of timed extensions of Petri nets have been proposed [3], [42], [47], [93], [163], [187]. For example, Merlin and Faber defined *time Petri nets*, in which each transition is associated with a clock that records the time lapse since it was last enabled [163]. Clock readings can be natural numbers in discrete-time models or dense numbers in continuous-time models. State may change due to transitions and time passage. Each transition also has two attributes: earliest firing time (Eft) and latest firing time (Lft). An enabled transition can only fire when its clock reading is no less than its Eft, and if continuously enabled, it must be fired before its Lft. Thus, time may not increment beyond the minimum deadlines set by the Lft of all enabled transitions. This semantics that "some enabled transition must fire" is different from the nondeterministic semantics of timed automata, in which a continuously enabled transition does not have to be fired before the Lft.

Ghezzi *et al.* proposed another Petri net extension, called *Time Environment/Relation (ER) nets*, for timed systems [93]. Each token, instead of each transition, is associated with a time stamp. Transitions can be triggered only when the earliest time and latest time requirements are met with respect to the token time stamps. They also presented three axioms which must be satisfied in each action to maintain the natural concepts of time stamps.

Adbulla and Nylen defined *timed Petri nets* [3], in which each token is associated with a clock which can be reset at the time of transition. This resembles the clock reset operations of timed automata. There is also no obligation to fire an enabled transition before the Lft expires.

Serugendo *et al.* extended Merlin and Faber's time Petri net to *real-time synchronized Petri nets* with synchronizations between a set of objects, modeled with real-time Petri nets [195].

Girault and Valk editted a handbook on formal methods based on Petri nets [94]. Cerone and Maggiolo-Schettini wrote a survey paper on various approaches to extending Petri nets to specification and verification of timed systems [56].

G. Graphical Languages

Statecharts [103] were introduced to help users describe behavioral hierarchies of untimed concurrent systems in a graphical style. System operations can be hierarchically decomposed to parallel and serial modes. Such behavioral hierarchy has inspired various compositional frameworks for verification and analysis, e.g., assume–guarantee reasoning [1], [17], [99], [162], [205], and state refinement verification [9], [232].

Modecharts [128] were the first timed extension of statecharts. Their semantics was defined with RTL [127] in discrete time-domain. Timing intervals and discrete events can be used as triggering conditions for transitions.

Statecharts support many powerful primitives, like exceptions, group transitions, and history. As a result, their semantics is complex. Hierarchic reactive module (HRM) is a model description language for timed systems with restrictions on transitions to simplify the semantics [14]. Syntactically, transitions in HRM can only connect to entry/exit points of structural modules. Transitions are forbidden from jumping directly to inner modules.

Timed unified modeling language (UML) is the real-time extension of UML [72], which is in turn a variation of statecharts. Translation schemes from timed UML to various verification tool languages have been studied and implemented [73], [88], [136], [146], [172].

IV. Verification Framework

There are the following four major approaches to computationally verifying timed systems.

A. Satisfiability Checking

In this framework, we write both the system behavior description B and specification S as logic formulas and try to prove that $B \rightarrow S$ is a theorem (i.e., *tautology*) of the underlying axioms. In reality, we usually check if $B \wedge \neg S$ is a *contradiction*, or equivalently $B \wedge \neg S$ is *unsatisfiable*. In the implementation, we can use the *tableau method* to construct an untimed Kripke structure and check if $B \wedge \neg S$ is satisfied at the initial nodes in the structure. A tableau is a small

model that can be used to check the existence of a model for $B \wedge \neg S$. Conceptually it is a directed graph $\langle V, E \rangle$ such that V is the set of possible worlds (or states) and E is the set of transitions from world to world.

In the following, we illustrate the tableau construction for TPTL satisfiability checking [15]. Assume that we are given a TPTL formula ϕ such that negations only appear in front of atomic constraints like p and $x + c \sim y + d$. This is possible because of deMorgan's law and $\neg\Box\phi_1 \equiv \Diamond\neg\phi_1$, $\neg\Diamond\phi_1 \equiv \Box\neg\phi_1$, and $\neg(\phi_1 \mathcal{U} \phi_2) \equiv (\Box\neg\phi_2) \vee ((\neg\phi_2)\mathcal{U}\neg(\phi_1 \wedge \phi_2))$. Given a TPTL formula ϕ, $\texttt{closure}(\phi)$ is a set of TPTL formulas constructed with the following induction rules.

- $\phi \in \texttt{closure}(\phi)$ and true $\in \texttt{closure}(\phi)$.
- If $\phi_1 \vee \phi_2 \in \texttt{closure}(\phi)$ or $\phi_1 \wedge \phi_2 \in \texttt{closure}(\phi)$, then $\phi_1 \in \texttt{closure}(\phi)$ and $\phi_2 \in \texttt{closure}(\phi)$.
- If $\phi_1 \mathcal{U} \phi_2 \in \texttt{closure}(\phi)$, then ϕ_1, ϕ_2, $\bigcirc\phi_1 \mathcal{U} \phi_2 \in \texttt{closure}(\phi)$.
- If $\Box\phi_1 \in \texttt{closure}(\phi)$, then ϕ_1, $\bigcirc\Box\phi_1 \in \texttt{closure}(\phi)$.
- If $\Diamond\phi_1 \in \texttt{closure}(\phi)$, then ϕ_1, $\bigcirc\Diamond\phi_1 \in \texttt{closure}(\phi)$.
- If $\bigcirc\phi_1 \in \texttt{closure}(\phi)$, then $\phi_1 \in \texttt{closure}(\phi)$.
- If $x.\phi_1 \in \texttt{closure}(\phi)$, then $\phi_1[x := T] \in \texttt{closure}(\phi)$ where T is the symbol for current time and $\phi_1[x := \epsilon]$ is identical to ϕ_1 except that every occurrence x is replaced by ϵ.
- If $x + c \sim T + d \in \texttt{closure}(\phi)$ where $c, d \in N$ and $\sim \in \{<, \leq, =, \geq, >\}$, then $x + c \sim T + d - 1 \in \texttt{closure}(\phi)$. The case for $T + d \sim x + c$ is symmetric.

Here, for convenience, we assume only inequalities like $x + c \sim y + d$, where x and y are clock variables and c and $d \in N$ are used.

A node v in V for ϕ is a subset of $\texttt{closure}(\phi)$ that satisfies the following *node consistency conditions*.

- true $\in v$.
- If $T + c \sim T + d \in v$, the truth value of $c \sim d$ is in v.
- If $\neg\phi_1 \in v$, then $\phi_1 \notin v$.
- If $\phi_1 \vee \phi_2 \in v$, then $\phi_1 \in v$ or $\phi_2 \in v$.
- If $\phi_1 \wedge \phi_2 \in v$, then $\phi_1 \in v$ and $\phi_2 \in v$.
- If $\phi_1 \mathcal{U} \phi_2 \in v$, then either $\phi_2 \in v$ or both $\phi_1 \in v$ and $\bigcirc\phi_1 \mathcal{U} \phi_2 \in v$.
- If $\Box\phi_1 \in v$, then both $\phi_1 \in v$ and $\bigcirc\Box\phi_1 \in v$.
- If $\Diamond\phi_1 \in v$, then either $\phi_1 \in v$ or $\bigcirc\Diamond\phi_1 \in v$.
- If $x.\phi_1 \in v$, then $\phi_1[x := T] \in v$.

If $(v, v') \in E$, then the following *transition consistency conditions* must also be maintained in E, i.e., exactly one of the following two is true.

1) For all $\bigcirc\phi_1 \in v$, then $\phi_1 \in v'$.
2) For all $\bigcirc\phi_1 \in v$, then $\phi_1^T \in v'$, where ϕ_1^T is identical to ϕ_1 except that every constraint like $x + c \sim T + d$ with $c, d \in N$ is respectively replaced with $x + c \sim T + d - 1$. (The case for $T + d \sim x + c$ is symmetric.)

Case 1 models the passage of zero time units, while case 2 models the passage of one time unit. Also note that when a constraint like $x + c \sim T - 1$ is generated, we will no longer ask for the decrements from $T - 1$. Thus, $T - 1$ serves as a flag, since no matter when we freeze T to x, the truth value of $x + c \sim T - 1$ is already determined at the moment when $T - 1$ is generated.

The small model theorem of TPTL says that if there is a model for ϕ, then there is such a model:

- whose nodes are in the powerset of $closure(\phi)$ and satisfy the node consistency condition; and
- whose transitions satisfy the transition consistency condition; and
- that contains a cycle such that
  - the cycle is reachable from an initial state node; and
  - if $\Diamond\phi_1$ is labeled in some node in the cycle, there is another node labeled with ϕ_1 in the cycle; and
  - if $\phi_1 \mathcal{U} \phi_2$ is labeled in some node in the cycle, there is another node labeled with ϕ_2 in the cycle.

The satisfiability problem of linear-time temporal logics TPTL [15], TETL, MTL [16], and APTL [228] are EXPSPACE-complete.

It is also possible to use temporal logics with dense-time semantics to do satisfiability checking. However, the satisfiability problems of TPTL with dense-time semantics and TCTL are all undecidable [5], [16]

B. Model Checking

The framework of *model checking* [60], [61] means that the system descriptions are given as automata, the specification formulas are given as temporal logic formulas, and we want to check if all models of a given system description satisfy a given specification formula. One popular framework in this category is the TCTL model-checking problem [5] in which the system descriptions are timed automata, while the specifications are TCTL formulas. The TCTL model-checking problem is PSPACE-complete. Note that the framework in [5] only permits atomic constraints like $x \sim c$ with $\sim \in \{\leq, <, =, >, \geq\}$. For about one decade, people have straightforwardly extended the framework with constraints like $x - y \sim c$ [113]. Recently, Bouyer [46] showed that the model-checking algorithms in [5], [113] is not correct for such an extension.

An important subclass in the model-checking framework is *safety checking*, i.e., the model checking of formulas like $\forall\Box\eta$ where η is a propositional formula for the safety property of a system. In implementation, the safety analysis problem is usually translated to the negation of the *reachability problem*, i.e., whether $\exists\Diamond\neg\eta$ is true or not. Most of the implementations in model checking have focused on the efficiency enhancement of safety checking. The major reason for this may be that fully model checking complex timed systems is indeed too difficult.

It is also possible to model check linear-time temporal logics like TPTL [15] or MTL [16]. In this framework, we want to verify that every computation of the timed automata is also a state sequence of the linear-time property.

Model checking has also been applied to frameworks other than automata. FDR is a commercial model-checker for CSP but does not quite support real-time system modeling [89].

The idea is to check whether, by hiding some internal variables, a behavior description in CSP can be identified as a refinement of another specification in CSP.

There is also model-checking research for Petri nets. One common approach is to put in restrictions to reduce Petri nets to finite-state systems, which are then subject to algorithmic model checking. In general, the reachability (of a specification marking) problem of Petri nets is decidable but without known elementary complexity. Another commonly accepted framework for verification with Petri nets is called the *coverability* problem, in which we are given a specification marking v and want to see if there is a reachable marking v' such that v' is no less than v place by place.

C. Simulation

It is also possible to use state-transition systems (automata) for both system behavior descriptions and specifications. This framework can be useful when a specification is too complex to put down in temporal logics. After all, a picture is worth a thousand words. Given a behavior model description A and a specification B, intuitively, we want to check whether A simulates B, i.e., every behavior of A is also a behavior of B with respect to the input and output events (i.e., observable events) and the times at which those events occur. However, we need to define behavior more clearly.

In linear time, a behavior is an (infinite or finite) sequence of events and the occurrence times of those events. Such a sequence together with the event occurrence times is called a *timed trace*. The framework of *trace inclusion* verifies whether A *implements* B, i.e., the timed traces of A are also timed traces of B. This framework may not have the power to discern certain behaviors. For example, in Fig. 1, although the two systems have the same set of traces, the choices at p states are not the same. In addition, Alur and Dill showed that the inclusion problem of timed traces is undecidable [12].

An alternative framework is *simulation* [158], [159], [209], which, in formal verification literature, does not mean that we build a mathematical model in the programming language C, execute the model with an inference engine, and then observe the trace [66], [224]. Instead, *simulation* is a relation between states of two model descriptions at an abstraction level (regarding the same set of observable inputs and outputs). Given a behavior model description A and a specification B, intuitively, we want to check if there is a simulation relation from A to B (i.e., A simulates B). For convenience, we write $\nu \xrightarrow{\delta,e}_A \nu'$ iff in A, we can transit from ν to ν' by first letting time progress by δ (a nonnegative real) and then executing a transition labeled with event e. A relation γ between the state sets of A and B is a simulation iff for every $(\nu_1, \nu_2) \in \gamma$:

- ν_1 and ν_2 are the same, predicate by predicate, at the abstraction level;
- if $\nu_1 \xrightarrow{\delta,e}_A \nu_1'$ for some ν_1', then $\nu_2 \xrightarrow{\delta,e}_B \nu_2'$ for some ν_2' of B such that $(\nu_1', \nu_2') \in \gamma$; and
- for every initial state ν_1 of A, there is an initial state ν_2 of B such that $(\nu_1, \nu_2) \in \gamma$.

Timed simulation has been used extensively for systems modeled with various extensions of I/O automata [44], [105], [141], [154]–[156], [160]. Since I/O automata [133] have first-in-first-out (FIFO) queues and their verification problem is in general undecidable, a lot of the work was carried out using theorem provers (see Section IV-D). Taşıran *et al.* showed that the simulation checking problem for timed automata is EXPTIME [209].

The third alternative is *timed bisimulation*, which is also a relation between states of two (timed) automata at an abstraction level. For convenience, given a relation γ, we let $\check{\gamma} = \{(b,a) | (a,b) \in \gamma\}$. Given two model descriptions A and B, a relation γ is a timed bisimulation between A and B iff γ is a timed simulation from A to B and $\check{\gamma}$ is a timed simulation from B to A. Čerāns showed that timed bisimulation checking is decidable [55]. Lasota showed that timed bisimulation checking is also decidable for timed basic parallel processes (BPPs)[9] [145]. Bisimulation-based verification of timed automata can be found in [71], [153]. Work on discrete-time models can be found in [147]–[149].

D. Theorem Proving

This approach stems from very early research in artificial intelligence [58]. In this framework, verification engineers manually design a verification plan (proof sketch) and then use theorem provers to mechanically check the correctness of the reasoning steps in the plan. Since many of the theorem provers accept undecidable classes of logic formulas, the approach usually does not guarantee the termination of individual mechanical verification tasks. If a user feels that a prover cannot finish a task, he/she may have to either intervene with expertise (formalized as axioms or proof strategies) or change the verification plan. Fulfillment of a verification plan depends heavily on the users' profound knowledge of the underlying logics and proficiency in using the tools.

Various algorithms were developed to check subclasses of first-order logics. Shostak [202] and Nelson and Oppen [173] presented algorithms to decide unquantified combinations of some fragments of first-order logics. Shostak also presented algorithms (congruence closure) to decide equality with uninterpreted functions [200] and methods, like loop residue [201] and SUP-INF [199], to decide linear arithmetic. Oppen presented algorithms for checking *Presburger arithmetic* formulas[10] [177]. Techniques from propositional calculus [52], [98] can also be employed to check propositional fragments of first-order or high-order logics.

There is also an extensive library of term rewriting techniques for first-order and higher order logics [77], [135]. Research has shown that controlled heuristics on term-rewriting rules are important.

[9]BPPs are systems constructed with rules $P ::= p | P_1 \| P_2 | 1 \triangleright P_1$, where p is an atomic process and $1 \triangleright P_1$ is the process of the passage of one time unit followed by process P_1.

[10]Presburger arithmetic is the theory of linear constraints like $c_1 x_1 + \ldots + c_n x_n \sim d$, Boolean operators, and quantification on variables $x_1, \ldots, x_n$.

Shankar designed an operator $\lfloor P \rfloor$ (read "since" P) in a state-based model in the well-documented theorem prover Prototype Verification System (PVS) [197]. The operator measures the time that has elapsed since P last held. It is implemented with three axioms in PVS. The first specifies the initial value of each $\lfloor P \rfloor$. The second and third respectively define when $\lfloor P \rfloor$ should remain the same and when it should increment in an action. Users can also define their own axioms in PVS for convenience. Work along this line can be found in [24], [25], [117], and [193].

V. Representations of State Spaces

Many verification frameworks rely on the analysis of reachable state spaces. The efficient manipulation of representations of reachable state spaces is fundamental to efficient verification of real-time systems. One important work in this regard is [5], in which Alur *et al.* presented a finite representation, called the *region graph*, for the dense-time state space of timed automata, and then proved the PSPACE-completeness of the TCTL model-checking problem. A region graph is a directed graph whose nodes are called *regions* and whose arcs represent either time progress or discrete transitions between regions. A region is a state subspace with three characteristics. The first is the control location of the states, the second is the integer parts of clock readings in the states up to the biggest timing constants used in the automata and the TCTL formula, and the third is the ordering among the fractional parts of clock readings in the states.

Region graphs are important in establishing complexity. For practical verification, symbolic data structures can usually yield more compact representations. In the following sections, we discuss some work in this regard.

A. Difference-Bounded Matrices

Since Dill proposed to use the difference-bounded matrix (DBM) to record the time space of real-time systems [78], the DBM has been adopted by two major model checkers: Kronos [75], [237] and UPPAAL [39] and has become the most popular data structure for such a purpose.

A DBM is a two-dimensional array. Each entry in a DBM records the difference between two clocks' readings of a state in the space characterized by the DBM. Zero is also treated as a special clock. Conceptually, given a set X of clocks, a DBM D is a mapping from $(\{0\} \cup X)^2$ to a set of elements like $(\sim, c)$ such that:

- $\sim$ is either $\leq$ or $<$;
- $c \in \{-C, \ldots, -1, 0, 1, \ldots, C, C^+\}$ where C is the biggest timing constant used in the real-time systems or in the specification and C^+ means any constant greater than C; and
- $c = C^+ \Rightarrow \sim = <$.

For each two $x, x' \in \{0\} \cup X$, $D(x, x') = (\sim, c)$ means that $x - x' \sim c$. A time space characterizable by a DBM is called a *zone*.

A DBM can represent a convex state space in the time space. Efficient operations like intersection and normalization to all-pair shortest-path form, can be performed. But a DBM cannot represent a concave state space.

Annichini *et al.* [21] have extended DBM with parameters for the semialgorithmic analysis of counter and clock systems.

B. BDD-Like Data Structures

BDD [52] is a minimum canonical form for propositional logic and has become an indispensible technology in hardware verification. Topologically, a BDD is an acyclic directed graph with a single source and two sinks (for FALSE and TRUE, respectively). It can represent both disjunctions and conjunctions. Each node is labeled with a decision atom, and the outgoing arcs are labeled with the values of the corresponding decision atom. It is minimum because BDD has the least representation size for any state space with respect to a given variable ordering. It is canonical because there is exactly one BDD for any given state space. This canonicality feature also implies that equality checking between state spaces and emptiness checking of a state space can be done efficiently.

The first paper to discuss how to use BDD to encode zones (actually for asynchronous systems with clock jitters) was by Wang *et al.* in 1993 [227]. They discussed how to use BDD with decision atoms like $x_i + c \leq x_j + d$ to model check timed automata. Here, c and d are timing constants with magnitude no greater than the biggest constant used in the behavior model and specification. Each decision atom can assume a Boolean truth value. The approach may suffer from bad performance, since the size of the decision atom domain is already proportional to the timing constants and, thus, exponential to the input size. However, they did not report implementation or experiments. In 1996, Balarin implemented the same scheme and reported experiments with approximation techniques [33]. In 1999, Møller *et al.* used the same idea to devise a data structure called a *difference decision diagram (DDD)* and discussed many manipulation techniques [168], [169].

The *numerical decision diagram (NDD)* [26] uses binary encoding for clock readings, and its performance is very sensitive to timing-constant magnitude.

The *clock-difference diagram (CDD)* [36] uses decision atoms like $x-y$ and labels arcs with disjoint intervals. For example, a node label $x-y$ together with an arc label (3,5] constitute the constraint $3 < x - y \leq 5$. CDDs were only used in UPPAAL [36] as recording devices of zones constructed with DBMs. No model-checking and reachability algorithms were implemented with CDDs in [36].

RED [213], [214] encodes the ordering of fractional parts of clock readings in the variable ordering and has achieved very high space efficiency for symmetric systems with large numbers of clocks and small timing constants. RED is a canonical representation of timed automata state subspaces. But for large timing constants, REDs performance degrades rapidly.

Then in 2001, Wang proposed the *clock-restriction diagram (CRD)* [216]–[218], which has a structure similar

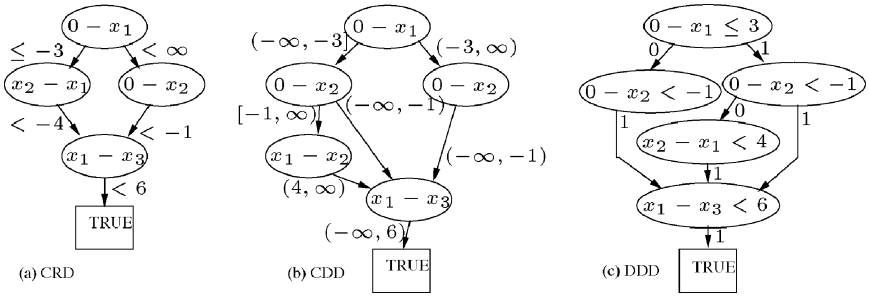

Fig. 4. Comparison among CRD, CDD, and DDD.

to CDD. The major difference between CRDs and CDDs is that the arcs from a node in CDDs are labeled with "DISJOINT" intervals, while those from a node in CRDs are labeled with upperbounds, which are structurally overlapping. Due to this minor difference, CRDs may avoid the representation fragmentation problem which are observed in CDDs with respect to some benchmarks. A complete set of symbolic manipulation algorithms with CRDs, including weakest precondition constructions, strongest postcondition constructions, and normalizations, has been developed and reported in [219]. According to the experience reported in [219], CRDs lead to clearer algorithm structures in the manipulation of dense-time state-space representations than CDDs. It was also reported that with careful programming, CRD technology outperforms DBM [216], [217], [219]. Similar experiments can be found in [196].

The size of a BDD can be reduced by only recording those paths to sink TRUE. The CRD, CDD, and DDD for the state space $(0-x_1 \leq -3 \wedge x_2-x_1 < -4 \wedge x_1-x_3 < 6) \vee (0-x_2 < -1 \wedge x_1 - x_3 < 6)$ is in Fig. 4.

C. Sets of Constraints

For linear hybrid systems, whose timing constraints may not be pairwise differences of clock readings, the state spaces can then be represented as unions of convex polyhedra [8], [102], each of which is bounded by a set of linear constraints. Early work in this regard is the linear relation analysis [69], [101] in the forum of *abstract interpretation* [67], [68].

Wang proposed a BDD-like data structure, called the *hybrid-restriction diagram (HRD)*, for the representation and manipulation of hybrid automata state spaces and reported experiments [221].

VI. Construction of Time-Space Representations

Since verification problems are highly complex for real-time systems, only with tight integration between data structures and algorithms can we effectly verify real-world system designs. The computation of real-time systems can be viewed as the interleaving of time progresses and discrete transitions [111]. Technically, given a state-space representation, we need to implement two procedures: $\texttt{time}(\eta)$ and $\texttt{xtion}(\eta, e)$ that construct the state-space representations of time progress and state transition e, respectively, from states in η. Then we apply these two procedures to iteratively construct the state-space representations reachable from the initial states. Specifically, given the set of transition E and the representations for the initial condition I and the invariance constraint Φ of a system, we execute the following loop, until $\eta = \eta'$:

```
for (η = I ∧ Φ, η' = false; η ≠ η';) {
  η' = η; η = η ∨ Φ ∧ time(Φ ∧ ⋁_{e∈E} xtion(η, e))
}
```

After the loop, η is the representation for the state space reachable from the initial condition.

The transitions in timed systems are usually presented as rules (or guarded commands) in the form "$\tau \longrightarrow x := \epsilon$;" which means that when the *triggering condition* τ is satisfied, the *assignment* of ϵ (an expression) to x can be executed. Computationally

$$\texttt{xtion}(\eta, \text{“}\tau \longrightarrow x := \epsilon;\text{”}) \equiv x = \epsilon \wedge \exists x(\eta \wedge \tau).$$

The existentially quantified x can be removed with a Fourier–Motzkin elimination.

On the other hand, the basic symbolic manipulation to derive constraints after time progress from a zone, i.e., procedure time() in Section IV-B, is in [113]. For simplicity, we assume that we are given a zone with only operator $\leq$ as follows:

$$\bigwedge_i x_i \leq c_i \wedge \bigwedge_j -y_j \leq d_j \wedge \bigwedge_k x_k - y_k \leq e_k.$$

Here, x_i, y_j, x_k, and y_k are clock variables, and c_i, d_j, and e_k are integer constants in $\{-C, -C+1, \ldots, 0, \ldots, C\}$ with C as the biggest timing constant used in the behavior description and specification. In practice, both $\leq$ and $<$ can be used. We want to determine the *postcondition* of time progress, i.e., what the zone becomes after the progress of time $\delta \geq 0$, with $\delta \in \mathcal{R}$. If we use the symbol x to represent the reading of clock x after time progress of δ, then the reading of clock x before the time progress can be symbolically represented as $x - \delta$. Prior to the time progress, the above zone constraint must be satisfied; we have

$$\exists\delta \begin{pmatrix} \delta \geq 0 \wedge \bigwedge_i x_i - \delta \leq c_i \wedge \bigwedge_j -(y_j - \delta) \leq d_j \\ \wedge \bigwedge_k (x_k - \delta) - (y_k - \delta) \leq e_k \end{pmatrix}$$
$$= \exists\delta \begin{pmatrix} -\delta \leq 0 \wedge \bigwedge_i x_i - \delta \leq c_i \wedge \bigwedge_j -y_j + \delta \leq d_j \\ \wedge \bigwedge_k x_k - y_k \leq e_k. \end{pmatrix}$$

The new existentially quantified variable δ can be eliminated by deriving inequalities like

$$x_i - \delta - y_j + \delta \le c_i + d_j \equiv x_i - y_j \le c_i + d_j$$

for each pair of $x_i - \delta \le c_i$ and $-y_j + \delta \le d_j$ (including $-\delta \le 0$). Thus, we have

$$\exists\delta \begin{pmatrix} -\delta \le 0 \wedge \bigwedge_i x_i - \delta \le c_i \wedge \bigwedge_j -y_j + \delta \le d_j \\ \wedge \bigwedge_k x_k - y_k \le e_k \\ \wedge \bigwedge_{i,j} x_i - y_j \le c_i + d_j \wedge \bigwedge_j -y_j \le d_j \end{pmatrix} = \begin{pmatrix} \bigwedge_k x_k - y_k \le e_k \\ \wedge \bigwedge_{i,j} x_i - y_j \le c_i + d_j \wedge \bigwedge_j -y_j \le d_j \end{pmatrix}$$

again with a Fourier–Motzkin elimination of δ. Since new constraints like $x_i - y_j \le c_i + d_j$ can be generated iteratively, it seems that we may not be able to converge to a fixpoint in the process of symbolic state-space representation construction. Fortunately, according to [78], when $c_i + d_j$ is bigger than the biggest timing constant (i.e., C) used in the model description and the specification formula, then we can discard the corresponding inequality. And when $c_i + d_j$ is smaller than $-C$, we can replace the inequality with $x_i - y_j < -C$. The replacements do not change the results of verification, since they are region equivalent.

A more efficient formulation for calculating the time progress postcondition is used in [196] and [219] and does not need to introduce the variable δ. This is done in two steps by first deducing transitivity constraints $x_i - y_j \le c + d$ through each pair of constraints like $x_i \le c$ and $-y_j \le d$, and then removing all upperbound constraints like $x_i \le c$.

With the capability to derive zones after discrete transitions and time progresses, we can (explicitly or implicitly) construct a zone graph whose nodes are zones and whose arcs represent time progresses and discrete transitions. However, such a framework can still be enhanced in efficiency. In the following, we survey various techniques used in the construction of state-space representations for real-time systems.

A. Backward Versus Forward Analysis

The time progress operation in the previous section reasons into the future. *Forward reachability analysis* uses such a basic step (together with operations for forward discrete transitions) to iteratively construct the representation for the state spaces forwardly reachable from the initial state.

We can also define a basic manipulation step for backward time progress. This can be achieved by substituting each x for $x + \delta$ instead of $x - \delta$. *Backward reachability analysis* uses such a basic step (together with operations for backward discrete transitions) to iteratively construct the representation for the state spaces backwardly reachable from the goal state.

For the widely used framework of safety checking, backward and forward analyses can both perform well. But backward analysis is almost mandatory for model checking because the modalities of strong "until" ($\mathcal{U}$, $\exists\mathcal{U}$, U, or EU) and strong "eventually" ($\Diamond$, $\exists\Diamond$, F, or EF) can only be efficiently handled with backward fixpoint computation.

B. Compositional Approach

Engineers usually describe their systems as a set of concurrent modules and design each module to work correctly with respect to uncertainty in the environment. Thus, it is natural to expect that such defensive designs can significantly reduce verification complexity. Various compositional verification strategies have been proposed to implement such expectations. Laroussinie *et al.* [142], [144] proposed a procedure to iteratively refine the specification formulas in a timed modal logic which accepts formulas like $\langle e\rangle\eta$ (eventually, after event e, η is satisfied), $[\delta]\eta$ (for every time passage by $\delta \ge 0$, η is always true), and $\langle\delta\rangle\eta$ (eventually, there is a time passage by $\delta \ge 0$ such that η is true). Given concurrent timed automata $A_1, \ldots, A_m$ and such a formula ϕ, the model-checking task is to prove $A_1|\ldots|A_n \models \phi$, where $A_1|\ldots|A_n$ is the composition of the concurrent modules. Then we can refine the specification as ϕ/A_n and inductively prove $A_1|\ldots|A_{n-1} \models \phi/A_n$. Here ϕ/A_n is a refinement of ϕ with respect to A_n. Intuitively, ϕ/A_n is obtained by restricting ϕ with transition relations of A_n encoded in the timed modal logic. This process is then repeated until we can prove or disprove true $\models (\ldots(\phi/A_n)\ldots)/A_1$.

In [223], Wang and Hsiung proposed a compositional framework, called *state-graph manipulators (SGM)*, to construct the global state graphs. SGM takes advantage of compositional verification to support a user-friendly interface. They use state graphs of concurrent modules as high-level data objects and package various verification techniques into SGMs to help users manipulate state graphs without the prerequisite deep knowledge of verification technology. They have also shown that by applying various reduction SGMs to the intermediate state graphs produced after each binary composition, significant enhancement can be obtained in verification performance.

In [114], compositionality is utilized in the framework of refinement of reactive modules [17]. Intuitively, module P is a refinement of Q, in symbols $P \preceq Q$, if each observable behavior of P is also a behavior of Q. We want to prove $P_1||P_2 \preceq Q$ by proving $P_1 \preceq Q$ and $P_2 \preceq Q$. However, this is usually impossible. The techique in [114] relies on the derivation of assumptions A_1 and A_2 from P_1 and P_2, respectively. Then in the framework of assumption–guarantee, we can prove the guarantee by instead proving $P_1||A_2 \preceq A_1$, $A_1||P_2 \preceq A_2$, and $A_1||A_2 \preceq Q$.

C. On-the-Fly Approach

In some early verification tools [53], the global state-space representations are constructed without regard to whether the states are reachable or not. In reality, if we have better knowledge which allows us to not represent unreachable states, then significant representation and manipulation complexity can be reduced. This is the basic idea of on-the-fly construction of state-space representations [45], [110]. Global state-space representations are constructed in a minimal way. Intuitively, we do not generate anything that is not used to answer the verification problem.

D. Normalizations

One challenge in real-time system verification arises in the fact that there is not an efficient and canonical (unique) representation of state spaces. Regions [5] and RED [213] are canonical but are also of high complexity. Zones, convex polyhedra [69], [101], DBM [78], CDD [36], DDD [168], [169], and CRD [216], [217] are not canonical and incur complexity both in the need to check equality/containment between representations and in the possibility of representing/manipulating the same state spaces many times.

The purpose of normalization is to reduce the number of possible representations for the same state space. A normal form has to be defined first. The most popular one is *closure form (or tight form)* of zones, which means all constraints are tight [78]. Closure form zones can be obtained with an all-pair shortest-path algorithm in the style of Kleene's closure. In [143], *reduced form* is defined to have the minimum number of constraints for each zone and it also exhibits space efficiency. In [216], [217], [219], the performance of various normal form possibilities of CRDs are investigated.

Recently, people have also researched using a satisfiability-checking procedure (SAT procedure) to efficiently decide the emptiness of real-time state space [207]. In this way, normalization is traded for the efficiency of many off-the-shelf SAT solvers [171], [238], [239].

VII. Reduction Techniques

Reduction techniques are used to simplify state-space representations and promote verification efficiency. The theoretical foundation is bisimulation equivalence [165], which characterizes states that can be distinguished by TCTL models. If we find that two state-space representations ϕ_1 and ϕ_2 describe bisimilarly equivalent states and ϕ_2 has lower space complexity than ϕ_1, then we can replace ϕ_1 with ϕ_2.

Approximation is another reduction approach, which may not preserve bisimulation equivalence.

In the following, we briefly discuss several reduction techniques for timed systems.

A. Inactive Variable Elimination

A variable is *inactive* in a state if along all computations from the state, the variable will not be read again before being written to. Two states are bisimilarly equivalent iff all variables other than those inactive ones have the same contents in the two states. In [76], Daws and Yovine presented a technique to detect inactive clocks in timed concurrent systems with synchronizations. They also presented techniques to detect clocks with equal readings and only represent one of them in the state-space representation.

Wang and Hsiung presented techniques to detect the inactiveness of variables in local state graphs considering the read–write behaviors in concurrent systems [223].

B. Partial-Order Reduction

A major cause for state-space explosion in verification is that we have to enumerate all the sequences (i.e., total ordering) of events in concurrent systems, like the ordering among the clock readings, clock resets, etc. But in reality, engineers rarely rely on the exact ordering of such sequences to assure the correctness of their system designs. Partial-order reduction has been realized in various strategies for untimed systems [92], [95], [116], [138], [182] and has been proven to be a valuable technique for untimed system verification.

For timed systems, Pagani presented a notion of independence between transitions based on global-time semantics of timed automata [180]. Yoneda *et al.* proposed a partial-order technique for model checking timed linear-time temporal logic on time Petri nets [38], [235], [236]. Bengtsson *et al.* used a definition of independence similar to [235], [236] and let clocks run without synchronization until communication time in a partial-order semantics. Minea [166] extended further the work of Bengtsson *et al.* [38].

C. Internal Transition Bypassing

A transition is *internal* if it is not observable from the outside. By eliminating state information related to internal transitions, significant reduction in the complexity of state graphs can usually be achieved. Similar concepts date back to the *internal actions* of process algebrae [115], [165]. A recent similar concept is the *invisible transition* by Miller and Katz [164]. Wang and Hsiung extended the technique to timed systems with concurrent read/write operations [223]. In [148] and [149], related work by Lawford *et al.* on discrete-time systems can be found.

D. Symmetry Reduction

In a concurrent system with symmetric processes running different copies of the same program, we can permute the roles of processes to transform state-space representations to normalized representations [84]. Symmetry reduction can be viewed as a special case of static partial-order reduction [138]. Wang presented a BDD-like data structure and a symmetry reduction technique [213]. The technique is quite efficient for fully symmetric systems with small timing constants. Wang also presented a symmetry reduction method for general timed systems based on an analysis of zone constraints and reported his experiments [218].

E. Approximation

The most general theory of *abstract interpretation* for the analysis of computer systems was invented by Cousot *et al.* [67], [68]. There are two strategies for approximation: underapproximations and overapproximations. With underapproximations (or overapproximations) of state-space representations, we approximate a reachable state space with a representation for its subset (or superset respectively) in the hopes of reducing representation and manipulation complexity. In [231], Wong-Toi presented a powerful overapproximation technique called *convex-hull approximation*. The idea is to represent a union of convex hulls with a minimal convex hull encompassing all convex hulls in the union. The technique has been proven very useful in checking

safety properties and has been used in many tools [39], [48], [75], [183], [237].

However, for BDD-like data structures, convex-hull approximation is difficult to implement if we consider variable-ordering interleaving between discrete variables and clock constraint variables. Special approximation techniques in this regard can be found in [33], [224].

In [210], Tripakis and Yovine developed various methods to approximate timed systems with untimed automata. The expectation was to exploit the existing rich infrastructure in algorithms and tools for the verification of untimed systems.

An interesting technique was developed in the last several years to automatically generate abstract predicates for the discretization of timed and hybrid systems [63]. The idea was to use the counterexample (a computation that invalidates the specification) capability available in most model checkers. Initially, we start verification with a very abstract behavior model. If the model checker at hand says the specification is satisfied, then we stop. Otherwise, a counterexample is generated and an abstract predicate is constructed to break the computation of the counterexample. The process continues until we either find no more counterexamples or find a counterexample that cannot be broken. Applications of this technique to hybrid systems can be found in [10], [62].

VIII. Tools

There are many verification tools for timed systems. In the following, we discuss some of them based on their popularity and technical achievements.

A. Modechart Toolset

The modechart toolset (MT) [66], [190] is a joint effort between the University of Texas, Austin, and the Naval Research Laboratory, Washington, DC. It accepts modecharts [126], [128], a timed extension of statecharts, and allows natural description of behavior hierarchy. It includes a graphical user interface for creating, modifying, and browsing modecharts. It also supports three types of analysis. First, modecharts can be converted to a first-order logic, RTL [127], and from there, we can use a theorem prover to analyze properties. Second, MT supports simulation with flexible interfaces so that users can have vivid visual effects through various computer animations and virtual reality displays [51]. Finally, MT also has a model checker for RTL formulas [129], [208].

B. PVS

PVS [191], [197] is one of the well-known general-purpose specification and verification environments for higher-order logics. It is based on Shostak's satisfiability-checking methods for first-order logics [70]. Intelligent case analysis is employed to take advantage of efficient techniques for deciding propositional formulas [98]. Term rewriting techniques are used with heuristics [77], [135]. Also, PVS uses higher order logics to support type checking, as in traditional programming languages [192]. The tool can be downloaded at http://pvs.csl.sri.com/.

C. Hytech

HyTech [108] is a model checker for linear hybrid systems [18]. Its specification is written in *integrator CTL (ICTL)*. Users can describe synchronizations between processes, specify the urgency of transitions, and ask for diagnostic error-trace generation. The feature of parametric analysis, although it is not guaranteed to halt, may provide extensive power to provide informative feedback to the users. The tool can be downloaded at http://www-cad.eecs.berkeley.edu/~tah/HyTec/. A graphical user interface is provided by the UPPAAL team.

D. Kronos

Kronos is a TCTL model checker for timed automata [48], [75], [237]. Its timed automata model does not allow variables and can be cumbersome in the description of data operations. It supports rendezvous among processes. It starts verification with on-the-fly construction of the product automata. Backward and forward analysis are both supported, but model checking must be executed with backward analysis. The data structure for clock constraints is DBM. Counterexamples can also be generated. The tool can be downloaded at http://www-verimag.imag.fr/TEMPORISE/kronos/.

E. UPPAAL

UPPAAL [39], [183] has now grown into an integrated tool environment for modeling, validation, and verification of real-time systems modeled by a network of automata with high-level data objects like range-bounded integers and arrays. The project is now a collaboration between Uppsala University, Uppsala, Sweden, and Aalborg University, Aalborg, Denmark. It is basically a forward reachability analyzer of timed systems. The data structure for clock constraints is DBM. Users can describe systems with automata templates, with mode urgency, and with synchronizations. The uniform graphical user interface allows editing, symbolic simulation, and verification. Moreover, the simulator allows the users to configure the level of details of the simulated systems to be displayed.

Various options are provided for bit-state hashing, inactive clock reduction, compact memory management, convex-hull approximation, and counterexample generation. The tool is available at http://www.docs.uu.se/docs/rtmv/uppaal/. Recently, Möller did research on extending UPPAAL with restricted inevitability analysis of the specification properties like $\forall\Diamond_{\leq d}\eta$, where d is the deadline for inevitability and η is a propositional formula [167]. Hendriks and Larsen investigated taking advantage of time granularities at different control locations for verification efficiency [107]. Behrmann *et al.* proposed a static analysis technique on triggering condition [35].

F. PARAGON

PARAGON stands for *Process-algebraic Analysis of Real-time Applications with Graphics-Oriented Notations*.

It was developed at the University of Pennsylvania, Philadelphia. Its basic system description language is algebra of communicating shared resources (ACSR) [152], [184], which is designed to model time delays created in a priority scheduling environment. A graphical version of ACSR is graphical communicating shared resources (GCSR). In addition to the visual simulator of GCSR, ACSR also has a verification tool called the VERSA toolkit, which is capable of rewriting rules to ACSR and exploring of state spaces of the constructed state machines. It can be obtained at http://www.cis.upenn.edu/~lee/paragon.html. ACSR has subsequently been extended with probability for process failure. A model checker against μ-calculus, called LCSR, is also available.

G. SGM

SGM is a compositional model checker for communicating timed automata [223]. It is designed to provide a user-friendly verification environment to users who are nonexperts in verification technology. The framework of SGM allows users to view process state graphs as high-level data-objects and package various verification techniques (e.g., on-the-fly binary composition in forward analysis, inactive clock elimination, symmetry reduction, etc.) so that users can manage verification complexity at an abstract level. Through applying reduction SGMs in between application of binary composition SGMs, significant reduction in representation complexity can be obtained. The tool was developed as a joint project between Academia Sinica and National Chung-Chen University, Taiwan, R.O.C. It is now available at http://www.cs.ccu.edu.tw/~pahsiung/sgm/. A graphical user interface is also supported.

H. MOCHA

MOCHA [4], [19] is a joint project among the University of California, Berkeley, the University of Pennsylvania, and the State University of New York, Stony Brook. The model of MOCHA consists of discrete-time reactive modules [17]. The specification formulas are given in alternating temporal logic (ATL), designed to specify collaborative and adversary interactions in concurrency. The popular CTL is a subclass of ATL. ATL model checking is achieved with the BDD engine VIS [50] developed at the University of California, Berkeley. Other than that, MOCHA also supports automated refinement checking and counterexample generation. The tool can be downloaded at http://www-cad.eecs.berkeley.edu/~mocha/.

I. IOA Toolset

The IOA toolset [90], [91] is a platform for the development of reliable complex distributed systems modeled with I/O automata [133]. The toolset accepts timed and hybrid system descriptions with FIFO message channels. In the toolset, the following functions are supported.

- A simulator that lets the users observe sample executions of IOA programs.
- Interface programs that extract axioms and proof obligations from IOA specifications so that users can conveniently use theorem provers like Larch Prover [100] and Isabelle/HOL [176] to verify their system designs.
- The Daikon invariants detector that examines the output of the simulator and proposes properties that are likely to be invariants of the IOA program.
- A code generator that synthesizes programs in languages like C++ and Java from IOA programs.

The toolset is available at http://theory.lcs.mit.edu/tds/ioa

J. TReX

TReX [23] is a tool for automatic analysis of infinite-state systems. Its input language is timed automata extended with parameters, counters, and lossy channels. It uses *simple regular expressions (SRE)* [2], [22], constrained parametric DBMs [21], and first-order arithmetic formulas to represent state spaces. Its core is a forward/backward exploration algorithm for the construction of state space, although termination is not guaranteed. It uses efficient extrapolation techniques to approximate the state-space representations.

Specification is given as an observer automaton. Counterexample traces for diagnosis can be generated and used to synthesize constraints to remedy the system. Constraints for liveness properties can also be analyzed and synthesized. The tool can be downloaded at http://www-verimag.imag.fr/~annichini/trex/.

K. CMC

CMC [142] is a compositional model checker [144] for networks of timed automata. An extension to hybrid systems is also available [54]. Like UPPAAL, CMC only supports the verification of safety properties and time-bounded liveness properties. The tool is available at http://www.lsv.ens-cachan.fr/~fl/cmcweb.html.

L. RED

RED is a project to design a TCTL model-checker with BDD-like data structures for representation and manipulation of dense-time state spaces. In the beginning, it used data structure *RED* (the same name as the tool) for symmetric systems [213]. The project went on to the development of a new data structure, CRD, for asymmetric systems [216]–[218]. Now RED supports symbolic simulation with a graphical user interface and code and region coverage estimations [226]. It is also equiped with the capability for full TCTL model checking, counter example generation, forward/backward analysis, a C-like model-description language, comment-line assertion checking, and symmetry reduction. RED was developed at both Academia Sinica and National Taiwan University, Taiwan, ROC and can be downloaded for free at http://cc.ee.ntu.edu.tw/~val. Recent extensions include a BDD-like data structure for the parametric safety analysis of linear hybrid systems [221], a model-checking algorithm with weak and strong fairness assumptions, and a specification language for both states and events for distributed real-time semantics [220].

M. PRISM

PRISM [139] stands for *probabilistic symbolic model checker*. Users can reason with the probability that a specification can be satisfied in a probabilistic system [140]. The project has been carried out in the University of Birmingham, Birmingham, U.K. The tool set supports both discrete-time and dense-time system analysis and is available at http://www.cs.bham.ac.uk/~dxp/prism/. Please check Section IX-D for more references.

IX. Other Issues

A. Symbolic Simulation

Formal verification is highly complex in both time and space. Many verification tasks are still beyond the reach of state-of-the-art automatic verification. On the other hand, simulation has served industry as the main tool for verification for decades. Compared with formal verification, simulation is much more efficient and can give users a realistic visualization of system behaviors. But now with the realization of systems on a chip containing millions of transistors, there are not enough resources to run enough traces to achieve sufficient functional coverage. In the foreseeable future, it is likely that simulation and formal verification will complement each other to assure the quality of industrial designs.

Symbolic simulation is a balance between simulation and formal verification. Instead of using specific recording of concrete states, we use symbolic representations of state spaces. Thus, intuitively, a symbolic trace may cover a huge set of concrete traces in traditional simulation.

Symbolic simulation can be very useful in the early stage of debugging. It is now supported in many formal verification tool packages for timed systems [108], [183], [219]. In simulation and testing, the concept of coverage has been useful in estimating how much of a target function has been verified and identifying coverage holes. Traditional coverage metrics for VLSI include line coverage, finite-state automata arc coverage, and state coverage [40]. Wang *et al.* presented symbolic techniques for estimating region coverage in dense-time state spaces [226]. Such techniques can be used to evaluate the progress of verification projects.

B. Parametric Analysis

Traditionally, verification problems have been formulated to only ask for "YES" or "NO" concerning the correctness of the system designs. In reality, engineers need more informative analysis techniques which help to mathematically characterize their systems' behaviors. Such systems are usually specified with symbolic constants, called *parameters*, whose values may engender different behaviors of the models. Setting and calibrating these parameters is a crucial task for the engineers developing hybrid systems. Parametric analysis is a research area for deriving constraints for a model to satisfy a specification. Hybrid system model checkers [2], [18], [21]–[23], [108] naturally come with the capability of parametric analysis but do not guarantee termination. In [20], Alur *et al.* showed that when three clocks are compared with parameters, the emptiness problem of parametric timed automata is undecidable. Moreover, a parameter α is called an *upper bound (or lower bound)* if it is used in a constraint like $x < \alpha$ or $x \leq \alpha$ (respectively $x > \alpha$ or $x \geq \alpha$), assuming that all negations have been pushed into the inequality operators of clock constraint atoms. A parametric timed automaton is *bipartite* iff none of its parameters are both upper bound and lower bound. In [121], it is shown that the emptiness problem of bipartite parametric timed automata is decidable. A parametric timed automaton is an upper-bound (or lower-bound) timed automaton iff all its parameters are upper bounds (lower bounds). In [230], it is shown that the shape of the parameter valuation spaces of upper-bound parametric timed automata are computable in double exponential time, while the ones of lower-bound parametric timed automata are in PSPACE.

In the parametric TCTL model-checking framework with parameters in TCTL formulas and no parameters in the timed automata, Wang *et al.* presented algorithms to compute the characterizations for the parameter solutions [211], [215], [229].

Emerson and Trefler worked on parametric analysis algorithms in the framework of discrete-time model checking with branching-time logic PRTCTL, which allows parametric quantification of event counts and timing constraints [85].

Alur *et al.* pursued the parametric analysis problem in the framework of discrete-time systems with parameterized linear-time temporal logic [13]. Especially they researched the emptiness, universality, and finiteness problems of the satisfying parameter valuation set. They also discussed the optimization problem according to certain criteria.

Wang reported a speed-up technique for parametric safety analysis of hybrid systems [221]. The observation is that while exploring the state space of hybrid systems, once a parameter constraint (i.e., one which involves only parameters) is derived along a computation, it must be satisfied from then on. Thus, while we are constructing the state-space representation, we can also keep a recording of the parameter space representation (which is usually much simpler). If a newly constructed state subspace does not increase the parameter space, then we can prune the exploration. It is reported that for some benchmarks, the technique can significantly enhance the performance of parametric analysis. Moreover, in some cases, the parameter space converges, while the state space does not.

C. Controller Synthesis

Following the seminal work of Ramadge and Wonham [186], the use of automata and formal languages to reason about *controllability* of *discrete event dynamic systems* has received much attention in the control community in the past decade. The *controller synthesis* problem, simply speaking, is to find out whether, for a given system, there is a controller through which the interaction between the system and the controller results in only computations of "good" behavior. (If such a controller exists, it is also desirable to construct it

effectively.) Brandin and Wonham also extended the work to discrete-time models [49].

Interested readers are also referred to [28] for a *symbolic* approach for controller synthesis in the framework of timed automata. As opposed to providing only yes/no answers in the conventional framework of controller synthesis, recent papers [27], [29] dealt with *quantitative* properties of behaviors for controllable timed automata.

In [229], Wang and Yen presented a unified framework for controller synthesis and parametric analysis. They developed an algorithm to compute the parametric characterization for the synthesis of controllers.

D. Probabilistic Analysis

The idea of probabilistic model checking is to attach probability distribution to transitions so that we know how probable it is that a specification can be satisfied [6], [140], [204]. Kwiatkowska *et al.* proposed probabilistic TCTL to put down statements like [140]

$$x.[\text{ready } \forall \mathcal{U}(x \leq 3 \wedge \text{ run})]_{>0.6}$$

which means that the probability of transiting from state ready to run within 3 s is greater than 0.6. Note here that the clock constraint is used with a reading freezing modal operator [15], [113].

Sproston discussed the model-checking problem for probabilistic hybrid systems [204].

E. WCET Analysis

Verification frameworks like TCTL model checking assume the timing constants used in transitions and specifications are given by users. But in practice, to come up with precise estimation of such timing constants can be a difficult task. The research on WCET analysis [86] focuses on the execution time analysis of program implementations. The cycle times of machine instructions and hardware features like cache lookahead and interrupt handling all have to be taken into consideration in the analysis.

In [212], Wang has proposed an algebraic framework to compute the characterization of execution times of dynamic, recursive, and concurrent systems.

X. Summary and Perspectives

This is a survey of the rapid development of formal verification technology for real-time systems. Theoretically speaking, formal verification has the advantage of functional completeness. That is, if a formal verification tool tells you that a system description is correct, then the design is indeed free of bugs in the abstraction level of the description. In contrast, it is usually difficult to know when a verification task is complete with simulation and testing. But at this moment, with its intrinsic high complexity, formal verification still presents one of the most formidable challenges to computer science, engineering, and human intelligence. As a consequence, people usually have to accept much more abstract system models in formal verification than in simulation and testing. Such high-level abstractions usually lead to insufficient description power for real-world systems and false negations as verification results.

Yet, through the effort of many researchers around the world in the last two decades, we feel that in the future, formal verification will become an indispensible technology in guaranteeing the quality of real-time system designs. Moreover, even though fully automatic formal verification may still be beyond the reach of the state of the art for the foreseeable future, the technology we have accrued in the research of formal verification of real-time systems will still be a valuable asset in automating the verification of large-scale systems. Instead of promoting formal verification as an ultimate solution, which can only answer YES/NO if the computers are fortunate enough to complete the formal verification tasks after several months, formal methods for fast and incremental feedback to engineers and managers should be better appreciated. For fast feedback, we feel that abstraction techniques will be important. If verification tools can quickly respond albeit with imprecise results, engineers can use the response as a guideline to revise their industrial designs. As for incremental feedback, at this moment, there is a lack of such devices for verification management by numbers. We propose the use of *symbolic coverage estimation* techniques [226] for this purpose, since coverage techniques have already been proven valuable in the past few decades in industry. For example, it would be difficult to argue for the value of your work when after three months, the computer is still running the PSPACE verification problem without termination. But if the formal verification tool can tell you that after three months' work, 60% functional coverage has been achieved, then you can perhaps guide the tool to work on coverage holes for better verification efficiency. Moreover, management will have a better appreciation for your effort and a rough estimation of the resources needed to obtain sufficient confidence in the project. For example, we may very well integrate coverage techniques with reachability analysis of timed automata as follows [226].

Compute the numerical estimations v and f,
  respectively, of the initially covered portions
  and the whole target function.
`for(`$\eta = I \wedge \Phi; v/f <$ `threshold;)`{
  $\eta = \eta \vee \Phi \wedge$ `time(`$\Phi \wedge \bigvee_{e \in E}$`xtion(`$\eta, e$`));`
  Compute the new numerical estimation v
    of the covered portions of the target function.
}

Typical coverage metrics include visited-state coverage, line coverage, region coverage [226], etc. In this framework, the managers and engineers have better numerical decision support in deciding when to stop the verification session by setting a goal threshold of functional coverage.

Acknowledgment

The author would like to thank the anonymous reviewers of the Proceedings of the IEEE. Their comments and sug-

gestions have lead to the addition of new material of which the author was unaware. The author also would like to thank Prof. H.-C. Yen for his comments on the manuscript.

REFERENCES

[1] M. Abadi and L. Lamport, "Conjoining specifications," *ACM Trans. Program. Lang. Syst.*, vol. 17, pp. 507–534, 1995.

[2] P. A. Abdulla, A. Bouajjani, and B. Jonsson, "On-the-fly analysis of systems with unbounded, lossy, FIFO channels," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1427, pp. 305–318.

[3] P. A. Abdulla and A. Nylen, "Timed Petri nets and BQOs," in *Lecture Notes in Computer Science, Applications and Theory of Petri Nets 2001*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2075, pp. 53–70.

[4] L. D. Alfaro, R. Alur, R. Grosu, T. A. Henzinger, M. Kang, R. Majumdar, F. Mang, C. Meyer-Kirsch, and B. Y. Wang, "Mocha: A model-checking tool that exploits design structure," in *Proc. 23rd Int. Conf. Software Engineering (ICSE'01)*, pp. 835–836.

[5] R. Alur, C. Courcoubetis, and D. L. Dill, "Model checking for real-time systems," in *Proc. 5th Annu. IEEE Symp. Logic in Computer Science*, 1990, pp. 414–425.

[6] ——, "Model checking for probabilistic real-time systems," in *Lecture Notes in Computer Science, Automata, Languages and Programming*. Heidelberg, Germany: Springer-Verlag, 1991, vol. 510, pp. 115–136.

[7] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Lecture Notes in Computer Science, Hybrid Systems*. Heidelberg, Germany: Springer-Verlag, 1993, vol. 736, pp. 209–229.

[8] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, pp. 3–34, 1995.

[9] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky, "Hierarchical modeling and analysis of embedded systems," *Proc. IEEE*, vol. 90, pp. 11–28, Jan. 2003.

[10] R. Alur, T. Dang, and F. Ivancic, "Counter-example guided predicate abstraction of hybrid systems," in *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2619, pp. 208–223.

[11] R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *Lecture Notes in Computer Science, Automata, Languages and Programming*. Heidelberg, Germany: Springer-Verlag, 1990, vol. 443, pp. 322–335.

[12] ——, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, pp. 183–235, 1994.

[13] R. Alur, K. Etessami, S. L. Torre, and D. Peled, "Parametric temporal logic for 'model measuring'," in *Lecture Notes in Computer Science, Automata, Languages and Programming*. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1644, pp. 159–168.

[14] R. Alur and R. Grosu, "Modular refinement of hierarchic reactive machines," in *Proc. Annu. Symp. Principles of Programming Languages 2000*, pp. 390–402.

[15] R. Alur and T. A. Henzinger, "A really temporal logic," in *Proc. 30th Annu. IEEE Symp. Foundations of Computer Science*, 1989, pp. 164–169.

[16] ——, "Real-time logics: complexity and expressiveness," *Information and Computation*, vol. 104, pp. 35–77, 1993.

[17] ——, "Reactive modules," *Formal Methods Syst. Design*, vol. 15, pp. 7–48, 1999. Preliminary version: *Proc. 11th IEEE Logic in Computer Science 1996*, pp. 207–218.

[18] R. Alur, T. A. Henzinger, and P. H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. Software Eng.*, vol. 22, pp. 181–201, Mar. 1996.

[19] R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Taşıran, "MOCHA: Modularity in model checking," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1427, pp. 521–525.

[20] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Parametric real-time reasoning," in *Proc. 25th ACM Symp. Theory of Computing*, 1993, pp. 592–601.

[21] A. Annichini, E. Asarin, and A. Bouajjani, "Symbolic techniques for parametric reasoning about counter and clock systems," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 1855.

[22] ——, "Symbolic verification of lossy channel systems: applications to the bounded retransmission protocol," in *Lecture Notes in Computer Science, Tools and Analysis for the Construction of Analysis of Systems*. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1579.

[23] A. Annichini, A. Bouajjani, and M. Sighireanu, "TReX: a tool for reachability analysis of complex systems," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2102.

[24] M. Archer, "TAME: using PVS strategies for special-purpose theorem proving," *Ann. Math. Artif. Intell.*, vol. 29, no. 1/4, pp. 201–232, Feb. 2001.

[25] M. Archer and C. Heitmeyer, "TAME: A specialized specification and verification system for timed automata," presented at the Work-in-Progress (WIP) 17th IEEE Real-Time Systems Symp. (RTSS'96), Washington, DC, 1996.

[26] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse, "Data-structures for the verification of timed automata," in *Lecture Notes in Computer Science, Hybrid and Real-Time Systems*, 1997, vol. 1201, pp. 346–360.

[27] E. Asarin and O. Maler, "As soon as possible: time optimal control for timed automata," in *Lecture Notes in Computer Science, Hybrid Systems: Computation and Control*, F. Vaandrage and J. van Schuppen, Eds. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1569, pp. 19–30.

[28] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Lecture Notes in Computer Science, Hybrid Systems II*, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. Heidelberg, Germany: Springer-Verlag, 1995, vol. 999, pp. 1–20.

[29] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," in *Proc. IFAC Symp. System Structure and Control*, 1998, pp. 469–474.

[30] J. C. M. Baeten and J. A. Bergstra, "Real time process algebra with infinitesimals," in *Algebra of Communicating Processes 1994*, A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, Eds., 1995, pp. 148–187.

[31] ——, "Discrete time process algebra: absolute time, relabive time, and parametric time," *Fundam. Inform.*, vol. 29, no. 1/2, pp. 51–76, 1997.

[32] J. C. M. Baeten and C. A. Middelburg, "Process algebra with timing: real time and discrete time," in *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds. New York: Elsevier, 2001, ch. 10.

[33] F. Balarin, "Approximate reachability analysis of timed automata," in *Proc. IEEE Real-Time Systems Symp.*, 1996, pp. 52–61.

[34] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "SLAM and static driver verifier: technology transfer of formal methods inside Microsoft," presented at the Integrated Formal Methods 2004 Conf. , Canterbury, U.K..

[35] G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen, "Static guard analysis in timed automata verification," in *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2619, pp. 254–277.

[36] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi, "Efficient timed reachability analysis using clock difference diagrams," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1633, pp. 341–353.

[37] J. Bengtsson, W. O. D. Griffoen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and Y. Wang, "Verification of an audio protocol with bus collision using UPPAAL," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1102, pp. 244–256.

[38] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, "Partial order reductions for timed systems," in *Lecture Notes in Computer Science, CONCUR'98*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1466, pp. 485–500.

[39] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL—a tool suite for automatic verification of real-time systems," in *Lecture Notes in Computer Science, Hybrid Control Systems*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1066, pp. 232–243.

[40] L. Bening and H. Foster, *Principles of Verifiable RTL Design, a Functional Coding Style Supporting Verification Processes in Verilog*, 2nd ed. Norwood, MA: Kluwer, 2001.

[41] *Handbook of Logic and Language*, J. v. Benthem and A. T. Meulen, Eds., North-Holland, 1997.

[42] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. Software Eng.*, vol. 17, pp. 259–273, Mar. 1991.

[43] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Comput. Netw. ISDN Syst.*, vol. 14, no. 1, pp. 25–29, 1987.

[44] D. Bosscher, I. Polak, and F. Vaandrager, "Verification of an audio control protocol," in *Lecture Notes in Computer Science, Formal Techniques in Real-Time and Fault-Tolerant Systems*. Heidelberg, Germany: Springer-Verlag, 1994, vol. 863, pp. 170–192.

[45] A. Bouajjani, S. Tripakis, and S. Yovine, "On-the-fly symbolic model-checking for real-time systems," in *Proc. IEEE Real-Time Systems Symp.*, 1997, pp. 25–34.

[46] P. Bouyer, "Untameable timed automata!," in *Lecture Notes in Computer Science, STACS 2003*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2607, pp. 620–631.

[47] F. D. J. Bowden, "Modeling time in Petri nets," presented at the 2nd Australian–Japan Workshop Stochastic Models, 1996.

[48] M. Bozga, C. Daws, and O. Maler, "Kronos: a model-checking tool for real-time systems," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1427, pp. 546–550.

[49] B. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Trans. Automat. Contr.*, vol. 39, pp. 329–342, Feb. 1994.

[50] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, "VIS: a system for verification and synthesis," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, vol. 1102, pp. 332–334.

[51] M. Brockmeyer, C. Heitmeyer, F. Jahanian, and B. Labaw, "A flexible, extensible simulation environment for testing real-time specifications," in *Proc. IEEE Real-Time Technology and Applications Symp. 1997*, pp. 125–135.

[52] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.

[53] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *Proc. IEEE Symp. Logic in Computer Science*, 1990, pp. 428–439.

[54] F. Cassez and F. Laroussinie, "Model-checking for hybrid systems by quotienting and constrains solving," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 1855, pp. 373–388.

[55] K. Čerāns, "Decidability of bisimulation equivalence for parallel timer processes," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1992, vol. 663, pp. 302–315.

[56] A. Cerone and A. Maggiolo-Schettini, "Time-based expressivity of time Petri nets for sytem specification," *Theor. Comput. Sci.*, vol. 216, no. 1–2, pp. 1–53, 1999.

[57] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha, "State/event-based software model checking," in *Lecture Notes in Computer Science, Integrated Formal Methods*. Heidelberg, Germany: Springer-Verlag, 2004, vol. 2999, pp. 128–147.

[58] C.-L. Chang, R. C. Lee, and R. C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*. New York: Academic, 1997.

[59] L. Chen, "An interleaving model for real-time systems," in *Lecture Notes in Computer Science, Logical Foundations of Computer Science*. Heidelberg, Germany: Springer-Verlag, 1992, vol. 620, pp. 81–92.

[60] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Lecture Notes in Computer Science, Logic of Programs*. Heidelberg, Germany: Springer-Verlag, 1981, vol. 131, pp. 52–71.

[61] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal-logic specifications," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 2, pp. 244–263, 1986.

[62] E. M. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald, "Verification of hybrid systems based on counterexample-guided abstraction refinement," in *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2619, pp. 192–207.

[63] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model-checking," *J. Assoc. Comput. Mach.*, vol. 50, no. 5, pp. 752–794, Sept. 2003.

[64] E. M. Clarke, O. Grumberg, M. Minea, and D. Peled, "State-space reduction using partial-ordering techniques," *Int. J. Softw. Tools Technol. Transf.*, vol. 2, no. 3, pp. 279–287, 1999.

[65] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: MIT Press, 2001.

[66] P. C. Clements, C. L. Heitmeyer, B. G. Labaw, and A. T. Rose, "MT: a toolset for specifying and analyzing real-time systems," in *Proc. IEEE Real-Time Systems Symp. 1993*, pp. 12–22.

[67] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs of by construction or approximation of fixpoints," in *Conf. Rec. 4th Annu. ACM Symp. Principles of Programming Languages*, 1977, pp. 238–252.

[68] ——, "Abstract interpretation and application to logic programs," *J. Logic Program.*, vol. 13, no. 2–3, pp. 103–179, 1992.

[69] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," presented at the 5th ACM Symp. Principles of Programming Languages, Tucson, AZ, 1978.

[70] D. Cyrluk, P. Lincoln, and N. Shankar, "On Shostak's decision procedure for combinations fo theories," in *Lecture Notes in Computer Science, Automated Deduction (CADE-13)*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1104, pp. 463–477.

[71] P. R. D'Argenio, "Algebras and Automata for Timed and Stochastic Systems," Ph.D. dissertation, Dept. Comput. Sci., University of Twente, Enschede, The Netherlands, 1999.

[72] W. Damm, B. Josko, and H. Hungar, "A compositional real-time Semantics of STATEMATE designs," in *Lecture Notes in Computer Science, Compositionality: The Significant Difference*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1536, pp. 186–238.

[73] A. David and M. O. Möller, "From HUPPAAL to UPPAAL—A translation from heirarchical timed automata to flat timed automata," Dept. Comput. Sci., Aarhus Universitet, Aarhus, Denmark, Tech. Rep. BRICS RS-01-11, 2001.

[74] J. Davies and S. Schneider, "A brief history of timed CSP," *Theor. Comput. Sci.*, vol. 138, no. 2, pp. 243–271, 1995.

[75] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The tool KRONOS," in *Lecture Notes in Computer Science, Hybrid Systems*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1066, pp. 208–219.

[76] C. Daws and S. Yovine, "Reducing the number of clock variables of timed automata," in *Proc. IEEE Real-Time Systems Symp.*, 1996, pp. 73–81.

[77] N. Dershowitz and J.-P. Jouannaud, *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, J. van Leeuwen, Ed. Amsterdam, The Netherlands: North-Holland, 1990, ch. 6, pp. 243–320.

[78] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1989, vol. 407, pp. 197–212.

[79] B. P. Douglass, *Real-Time UML*. Reading, MA: Addison-Wesley, 1998.

[80] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Amsterdam, The Netherlands: Elsevier, vol. B, pp. 995–1072.

[81] E. A. Emerson and J. Y. Halpern, ""Sometimes" and "not never" revisited: on branching versus linear time temporal logic," *J. Assoc. Comput. Mach.*, vol. 33, no. 1, pp. 151–178, 1986.

[82] E. A. Emerson and C.-L. Lei, "Modalities for model checking: branching time logic strikes back," in *Science of Computer Programming 8*. Amsterdam, The Netherlands: Elsevier Science Publishers B. V. (North-Holland), 1987, pp. 275–306.

[83] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan, "Quantitative temporal reasoning," *J. Real Time Syst.*, vol. 4, pp. 331–352, 1992.

[84] E. A. Emerson and A. P. Sistla, "Utilizing symmetry when model-checking under fairness assumptions: an automata-theoretic approach," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 4, pp. 617–638, July 1997.

[85] E. A. Emerson and R. J. Trefler, "Parametric quantitative temporal reasoning," in *Proc. IEEE Symp. Logic in Computer Science*, 1999, pp. 336–343.
[86] J. Engblom, A. Ermedahl, M. Sjoedin, J. Gustafsson, and H. Hansson, "Worst-case execution-time analysis for embedded real-time systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 4, pp. 437–455, 2001.
[87] J. Esparza, "Petri nets, commutative context-free grammers and basic parallel processes," in *Lecture Notes in Computer Science, Fundamentals of Computation Theory*. Heidelberg, Germany: Springer-Verlag, 1995, vol. 965, pp. 221–232.
[88] T. Firley, M. Huhn, K. Diethers, T. Gehrke, and U. Goltz, "Timed sequence diagrams and tool-based analysis—a case study," in *Lecture Notes in Computer Science, UML'99*. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1723, pp. 645–660.
[89] Formal Systems (Europe) Ltd.. Failure-Divergence Refinement—FDR2 User Manual. [Online]. Available: http://www.fsel.com
[90] S. J. Garland and N. A. Lynch, "The IOA language and toolset: Support for designing, analyzing, and building distributed systems," Massachusetts Inst. Technol., Cambridge, Tech. Rep. MIT/LCS/TR.
[91] ——, "Using I/O automata for developing distributed systems," in *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitaraman, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2000, ch. 13, pp. 285–312.
[92] R. Gerth, R. Kuiper, D. Peled, and W. Penczek, "A partial order approach to branching time temporal logic model checking," in *Proc. 3rd Israel Symp. Theory of Computing and Systems*, 1995, pp. 130–139.
[93] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze, "A unified high-level Petri net formalism for time-critical systems," *IEEE Trans. Software Eng.*, vol. 17, pp. 160–172, Feb. 1991.
[94] C. Girault and R. Valk, *Petri Nets for Systems Engineering—A Guide to Modeling, Verification, and Applications*. Berlin, Germany: Springer-Verlag, 2003.
[95] P. Godefroid, *Lecture Notes in Computer Science, Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Space Explosion Problem*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1032.
[96] M. J. C. Gordon, "HOL: a proof generating system for higher-order logic," in *VLSI Specification, Verification, and Synthesis*, G. Birtwistle and P. A. Subrahmanyam, Eds. Norwood, MA: Kluwer, 1988, pp. 73–128.
[97] S. Greibach, "Remarks on blind and partially blind one-way multicounter machines," *Theor. Comput. Sci.*, vol. 7, pp. 311–324, 1978.
[98] J. F. Groote and J. P. Warners, "The propositional formula checker Heer-Hugo," *J. Automat. Reason.*, vol. 24, no. 1–2, pp. 101–125, Feb. 2000.
[99] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 843–871, 1994.
[100] J. V. Guttag, J. J. Horning, S. J. Garland, K. D. Jones, A. Modet, and J. M. Wing, *Larch: Languages and Tools for Formal Specification*. New York: Springer-Verlag, 1993.
[101] N. Halbwachs, "Détermination automatique de relations linéaire vérifiées par les variables d'un programme," Thèse de 3e cycle, University of Grenoble, Grenoble, France, 1979.
[102] ——, "Delay analysis in synchronous programs," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1993, vol. 697, pp. 409–423.
[103] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comp. Program.*, vol. 8, no. 3, pp. 231–274, 1987.
[104] D. Harel, H. Lachover, A. Naamad, and A. Pnueli, "Explicit clock temporal logic," in *Proc. IEEE 5th Symp. Logic in Computer Science*, 1990, pp. 402–413.
[105] C. Heitmeyer and N. A. Lynch, "The generalized railroad crossing—a case study in formal verification of real-time systems," in *Proc. 15th IEEE Real-Time Systems Symp.*, 1994, pp. 120–131.
[106] C. Heitmeyer and D. Mandrioli, *Formal Methods for Real-Time Computing*. New York: Wiley, 1996.
[107] M. Hendriks and K. G. Larsen. (2002) Exact acceleration of real-time model checking. *Electron. Notes Theor. Comput. Sci.* [Online]. Available: http://www.elsevier.nl/locate/entcs/volume65.html
[108] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, "HyTech: a model-checker for hybrid systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 1, pp. 110–122, 1997.
[109] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?," *J. Comput. Syst. Sci.*, vol. 57, pp. 94–124, 1998. Preliminary version: *Proc. 27th Symp. Theory of Computing 1995*, pp. 373–382.
[110] T. A. Henzinger, O. Kupferman, and M. Y. Vardi, "A space-efficient on-th-fly algorithm for real-time model-checking," in *Lecture Notes in Computer Science, CONCUR'96*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1119, pp. 514–529.
[111] T. A. Henzinger, Z. Manna, and A. Pnueli, "An interleaving model for real-time," in *Proc. 5th Int. Conf. Information Technology*, 1990, pp. 717–730.
[112] ——, "What good are digital clocks?," in *Lecture Notes in Computer Science, Automata, Languages, and Programming*. Heidelberg, Germany: Springer-Verlag, 1992, vol. 623, pp. 545–558.
[113] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Inform. Comput.*, vol. 111, no. 2, pp. 193–244, 1994.
[114] T. A. Henzinger, S. Qadeer, and S. Rajamani, "You assume, we guarantee: methodology and case studies," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1427, pp. 440–451.
[115] C. A. R. Hoare, *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[116] G. J. Holzmann and D. Peled, "An improvement in formal verification," in *Proc. 7th IFIP WG6.1 Int. Conf. Formal Description Techniques,*, 1994, pp. 177–194.
[117] J. Hooman, "Compositional verification of real-time applications," in *Lecture Notes in Computer Science, Compositionality—The Significance Difference*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1536, pp. 276–300.
[118] P. Hopcroft and J. Pansiot, "On the reachability problem for 5-dimensional vector addition systems," *Theoret. Comp. Sci.*, vol. 8, pp. 135–159, 1979.
[119] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.
[120] P.-A. Hsiung and F. Wang, "User-friendly verification," presented at the 1999 IFIP TC6/WG6.1 Joint Int. Conf. Formal Description Techniques and Protocol Specification, Testing, and Verification, Beijing, China.
[121] T. Hune, J. Romijn, M. Stoekinga, and F. Vaandrager, "Linear parametric model checking of timed automata," in *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems* . Heidelberg, Germany: Springer-Verlag, 2001, vol. 2031, pp. 189–203.
[122] M. Huth, R. Jagadeesan, and D. Schmidt, "Modal transition systems: a foundation for three-valued program analysis," in *Lecture Notes in Computer Science, Proc. European Symp. Programming 2001*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2028, pp. 155–169.
[123] *ISO/IEC-JTC1/SC21/WG1/FDT/C:LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*, IS 8807, Feb. 1989.
[124] *ISO/IEC JTC1/SC33: ISO/IEC FCD 15437—Enhancements to LOTOS (E-LOTOS)*, ISO/IEC JTC1/SC33 N0188, May 1998.
[125] F. Jananian, "Verifying properties of systems with variable timing constraints," in *Proc. IEEE Real-Time Systems Symp. 1989*, pp. 319–328.
[126] F. Jahanian, R. S. Lee, and A. K. Mok, "Semantics of modechart in real-time logic," in *Proc. 21st Hawaii Int. Conf. Systems Science 1988*, pp. 479–489.
[127] F. Jahanian and A. K. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 890–904, Sept. 1986.
[128] ——, "Modechart: a specification language for real-time systems," *IEEE Trans. Software Eng.*, vol. 20, pp. 933–947, Dec. 1994.
[129] F. Jahanian and D. A. Stuart, "A method for verifying properties of modechart specifications," in *Proc. IEEE Real-Time Systems Symp. 1988*, pp. 12–21.
[130] P. Jancar and F. Moller, "Checking regular properties of Petri nets," in *Lecture Notes in Computer Science, CONCUR'95*. Heidelberg, Germany: Springer-Verlag, 1995, vol. 962, pp. 348–362.
[131] J. A. W. Kamp, "Tense logic and the theory of linear order," Ph.D. dissertation, Univ. California, Los Angeles, 1968.
[132] R. M. Karp and R. E. Miller, "Parallel program schemata," *J. Comput. Syst. Sci.*, vol. 3, pp. 147–195, 1969.

[133] D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager, "Timed I/O automata: a mathematical framework for modeling and analyzing real-time systems," presented at the 24th Int. Real-Time Systems Symp., Cancun, Mexico, 2003. Full version: D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager, "The theory of timed I/O automata, Massachusetts Inst. Technol. Lab. Comput. Sci., Cambridge, MA, Tech. Rep. MIT-LCS-TR-917.

[134] E. Kindler and T. Vesper, "ESTL: a temporal logic for events and states," in *Lecture Notes in Computer Science, Application and Theory of Petri Nets*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1420, pp. 365–384.

[135] J. W. Klop, *Handbook of Logic in Computer Science*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds. Oxford, U.K.: Oxford Univ. Press, 1992, vol. 2, ch. 1, pp. 1–117.

[136] A. Knapp, S. Merz, and C. Rauh, "Model checking timed UML state machines and collaborations," in *Lecture Notes in Computer Science, Formal Techniques in Real-Time and Fault-Tolerant Systems*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2469, pp. 395–414.

[137] D. Kozen, "Results on the propositional mu-calculus," *Theor. Comput. Sci.*, vol. 27, pp. 333–354, 1983.

[138] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigun, "Static partial order reduction," in *Lecture Notes in Computer Science, Research and Development in Knowledge Discovery and Data Mining*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1394, pp. 345–357.

[139] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: probabilistic symbolic model-checker," in *Lecture Notes in Computer Science, Computer Performance Evaluation*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2324, pp. 200–204.

[140] M. Kwiatkowska, G. Norman, R. Segala, and J. Sprotson, "Automatasic verification of real-time systems with discrete probability distributions," *Theor. Comput. Sci.*, vol. 286, no. 1–2, 2002.

[141] B. W. Lampson, N. A. Lynch, and J. F. Sogaard-Andersen, "Correctness of at-most-once message delivery protocols," in *Proc. 6th IFIP Int. Conf. Formal Techniques for Networked and Distributed Systems*, 1993, pp. 385–400.

[142] F. Laroussinie and K. G. Larsen, "CMC: a tool for compositional model-checking of real-time systems," in *Proc. IFIP TC6/WG6.1 Joint Int. Conf. Formal Description Techniques and Protocol Specification, Testing, and Verification'98*, pp. 439–456.

[143] K. G. Larsen, F. Larsson, P. Pettersson, and Y. Wang, "Efficient verification of real-time systems: compact data-structure and state-space reduction," in *Proc. IEEE Real-Time Systems Symp. 1998*, pp. 14–24.

[144] K. G. Larsen, P. Petterson, and W. Yi, "Compositional and symbolic model-checking of real-time systems," in *Proc. IEEE Real-Time Systems Symp. 1995*, pp. 76–87.

[145] S. Lasota, "Decidability of strong bisimilarity for timed BPP," in *Lecture Notes in Computer Science, CONCUR'02*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2421, pp. 562–578.

[146] L. Lavazza, G. Quaroni, and M. Venturelli, "Compiling UML and formal notations for modeling real-time systems," presented at the 8th Eur. Conf. Software Engineering, Vienna, Austria, 2001.

[147] M. Lawford, "Model reduction of discrete real-time systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1997.

[148] M. Lawford, J. S. Ostroff, and W. M. Wonham, "Model reduction of modules for state-event temporal logics," in *Proc. IFIP TC6/WG6.1 Joint Int. Conf. Formal Description Techniques and Protocol Specification, Testing, and Verification'96*, pp. 263–278.

[149] M. Lawford and W. M. Wonham, "Equivalence preserving transformations of timed transition models," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 1167–1179, July 1995.

[150] G. Leduc and L. Leonard, "An introduction to ET-LOTOS for the description of time-sensitive systems," *Comput. Netw. ISDN Syst.*, vol. 29, no. 3, pp. 271–292, 1997.

[151] L. Leonard and G. Leduc, "A formal definition of time in LOTOS," *Formal Aspects Comput.*, vol. 10, pp. 248–266, 1998.

[152] I. Lee, P. Bremond-Gregoire, and R. Gerber, "A process algebraic approach to the specification and analysis of resource-bound real-time systems," *Proc. IEEE*, vol. 82, pp. 158–171, Jan. 1994.

[153] H. Lin and W. Yi, "Axiomating timed automata," *Acta Inform.*, vol. 38, no. 4, pp. 277–305, 2002.

[154] V. Luchangco, E. Söylemez, S. Garland, and N. A. Lynch, "Verifying timing properties of concurrent algorithms," in *Proc. 7th IFIP Conf. Formal Techniques*, 1994, pp. 259–273.

[155] N. A. Lynch, "Simulation techniques for proving properties of real-time systems," in *Lecture Notes in Computer Science, A Decade of Concurrency: Reflections and Perspectives (REX School/Symposium, Noordwijkerhout, The Netherlands, June 1993)*. Heidelberg, Germany: Springer-Verlag, 1994, vol. 803, pp. 375–424.

[156] ——, *Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann, 1995.

[157] N. Lynch and M. R. Tuttle, "An introduction to input/output automata," *CWI-Q.*, vol. 2, no. 3, pp. 219–246, Sept. 1989.

[158] N. A. Lynch and F. W. Vaandrager, "Forward and backward simulations for timing-based systems," in *Lecture Notes in Computer Science, Real Time: Theory in Practice 1991*. Heidelberg, Germany: Springer-Verlag, 1992, vol. 600, pp. 397–446.

[159] ——, "Forward and backward simulations part II: Timing-based systems," CWI, Amsterdam, The Netherlands, Rep. CS-R9314, 1993.

[160] N. A. Lynch and H. B. Weinberg, "Proving correctness of a vehicle maneuver: deceleration," in *Proc. 2nd Eur. Workshop Real-Time and Hybrid Systems*, 1995, pp. 196–203.

[161] R. Mattolini and P. Nesi, "An interval logic for real-time system specification," *IEEE Trans. Software Eng.*, vol. 27, pp. 208–227, Mar. 2001.

[162] K. L. McMillan, "A compositional rule for hardware design refinement," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 1997, vol. 1254, pp. 24–35.

[163] P. Merlin and D. J. Faber, "Recoverability of communication protocols—Implications of a theoretical study," *IEEE Trans. Comput.*, vol. COM-24, pp. 1036–1043, Sept. 1976.

[164] H. Miller and S. Katz, "Saving space by fully exploiting invisible transitions," *Formal Methods Syst. Design*, vol. 14, pp. 311–332, 1999.

[165] R. Milner, *Communication and Concurrency*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[166] M. Minea, "Partial order reduction for model-checking of timed automata," in *Lecture Notes in Computer Science, CONCUR'99*. Heidelberg, Germany: Springer-Verlag, 1999, vol. 1664, pp. 431–446.

[167] M. O. Möller. (2002) Parking can get you there faster—Model augmentation to speed up real-time model checking. *Electron. Notes Theor. Comput. Sci.* [Online]. Available: http://www1.elsevier.com/gej-ng/31/29/23/117/51/show/Products/notes/index.htt

[168] J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard, "Difference decision diagrams," presented at the Annu. Conf. Eur. Assoc. Computer Science Logic (CSL), Madrid, Spain, 1999.

[169] ——, "Fully symbolic model-checking of timed systems using difference decision diagrams," presented at the Workshop on Symbolic Model-Checking (SMC), Trento, Italy, 1999.

[170] F. Moller and C. Tofts, "A temporal calculus of communicating systems," in *Lecture Notes in Computer Science, CONCUR'90*. Heidelberg, Germany: Springer-Verlag, 1990, vol. 458, pp. 401–415.

[171] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," presented at the 39th Design Automation Conf., Las Vegas, NV, 2001.

[172] D. Muthiayen, "Real-time reactive system development—A formal approach based on UML and PVS," Ph.D. dissertation, Concordia University, Montreal, QB, Canada, 2000.

[173] G. Nelson and D. C. Oppen, "Simplification by cooperating decision procedures," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 2, pp. 245–257, 1979.

[174] R. De Nicola and F. Vaandrager, "Three logics for branching bisimulation," *J. Assoc. Comput. Mach.*, vol. 42, no. 2, pp. 458–487, 1995.

[175] X. Nicolin, J. Sifakis, and S. Yovine, "Compiling real-time specifications into extended automata," *IEEE Trans. Software Eng. (Special Issue on Real-Time Systems)*, vol. 18, pp. 794–804, Sept. 1992.

[176] T. Nipkow, L. C. Paulson, and M. Wenzel, *Lecture Notes in Computer Science, Isabelle/HOL*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2283.

[177] D. C. Oppen, "A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic," *J. Comp. Syst. Sci.*, vol. 16, pp. 323–332, 1978.

[178] J. S. Ostroff, *Temporal Logic of Real-Time Systems*, ser. Advanced Software Development Series. New York: Wiley, 1990.

[179] ——, "Formal methods for the specification and design of real-time safety critical systems," *J. Softw. Syst.*, vol. 18, no. 1, pp. 33–60, April 1992.

[180] F. Pagani, "Partial orders and verification of real-time systems," in *Lecture Notes in Computer Science, Formal Techniques in Real-Time and Fault-Tolerant Systems*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1135, pp. 327–346.
[181] L. C. Paulson, "Isabelle: the next 700 theorem provers," in *Logic and Computer Science*, P. Odifreddi, Ed. New York: Academic, 1990, pp. 361–386.
[182] D. Peled, "Combining partial order reductions with on-the-fly model checking," *Formal Methods Syst. Design*, vol. 8, pp. 39–64, 1996.
[183] P. Pettersson and K. G. Larsen, "UPPAAL2k," *Bull. Eur. Assoc. Theor. Comput. Sci.*, vol. 70, pp. 40–44, 2000.
[184] A. Philippou, O. Sokolsky, R. Cleaveland, I. Lee, and S. Smolka, "Probabilistic resource failure in real-time process algebra," in *Lecture Notes in Computer Science, CONCUR 98*, 1998, vol. 1466, pp. 389–404.
[185] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. IEEE Symp. Foundations of Computer Science*, 1977, pp. 45–57.
[186] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, pp. 206–230, 1987.
[187] R. R. Razouk and C. V. Phelps, "Performance analysis using timed Petri nets," in *Proc. 6th Protocol Specification, Testing, and Verification Conf.*, 1984, pp. 561–576.
[188] G. M. Reed and A. W. Roscoe, "A timed model for communicating sequential processes," in *Lecture Notes in Computer Science, Automata, Languages, and Programming*. Heidelberg, Germany: Springer-Verlag, 1986, vol. 226, pp. 314–323.
[189] J. Romijn, "A timed verification of the IEEE 1394 leader election protocol," in *Proc. 4th Int. ERCIM Workshop Formal Methods for Industrial Critical Systems (FMICS'99)*, pp. 3–29.
[190] A. Rose, M. Perez, and P. C. Clements, "Modechart toolset user's guide," Center Comput. High Assurance Syst., Naval Res. Lab., Washington, DC, Tech. Rep. NRL/MRL/5540-94-7427, 1994.
[191] J. Rushby, "Theorem proving for verification," in *Lecture Notes in Computer Science, Modeling and Verification of Parallel Processes*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 2067, pp. 39–57.
[192] J. Rushby, S. Owre, and N. Shankar, "Subtypes for specifications: Predicate subtyping in PVS," *IEEE Trans. Software Eng.*, vol. 24, pp. 709–720, Sept. 1998.
[193] M. Schenke and E.-R. Olderog, "Transformational design of real-time systems—part I: From requirements to program specification," *Acta Inform.*, vol. 36, no. 1, pp. 1–65, 1999.
[194] S. Schneider, *Concurrent and Real-Time Systems: The CSP Approach*. New York: Wiley, 1999.
[195] G. D. M. Serugendo, D. Mandrioli, D. Buchs, and N. Guelfi, "Real-time synchronized Petri nets," in *Lecture Notes in Computer Science, Application and Theory of Petri Nets*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2360, pp. 142–162.
[196] S. A. Seshia and R. E. Bryant, "Unbounded, fully symbolic model checking of timed automata using Boolean methods," in *Lecture Notes in Computer Science, Computer-Aided Verification*, 2003, vol. 2725, pp. 154–166.
[197] N. Shankar, "Verificatoin of real-time using PVS," in *Lecture Notes in Computer Science, Computer Aided Verification* . Heidelberg, Germany: Springer-Verlag, 1993, vol. 697, pp. 280–291.
[198] A. Shaw, "Communicating real-time state machines," *IEEE Trans. Software Eng.*, vol. 18, pp. 805–816, Sept. 1992.
[199] R. E. Shostak, "On the SUP-INF method for proving Presburger formulas," *J. Assoc. Comput. Mach.*, vol. 24, no. 4, pp. 529–543, Oct. 1977.
[200] ——, "An algorithm for reasoning about equality," *Commun. ACM*, vol. 21, no. 7, pp. 583–585, July 1978.
[201] ——, "Deciding linear inequalities by computing loop residues," *J. Assoc. Comput. Mach.*, vol. 28, no. 4, pp. 769–779, Oct. 1981.
[202] ——, "Deciding combinations of theories," *J. Assoc. Comput. Mach.*, vol. 31, no. 1, pp. 1–12, Jan. 1984.
[203] D. P. L. Simons and M. I. A. Stoelinga, "Mechanical verification of the tree 1394a root contention protocol using uppaal2k," Comput. Sci. Inst. Nijmegen, Nijmegen, The Netherlands, Tech. Rep. CSI-R0009, 2000.
[204] J. Sproston, "Decidable model-checking of probabilistic hybrid automata," in *Lecture Notes in Computer Science, Formal Techniques in Real-Time and Fault-Tolerant Systems*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 1926, pp. 31–45.
[205] E. W. Stark, "A proof technique for rely-guarantee properties," in *Lecture Notes in Computer Science, Foundations of Software Technology and Theoretical Computer Science*, 1985, vol. 206, pp. 369–391.
[206] T. Stauner, O. Müller, and M. Fuchs, "Using HyTech to verify an automotive control system," in *Lecture Notes in Computer Science, Hybrid and Real-Time Systems*. Heidelberg, Germany: Springer-Verlag, 1997, vol. 1201, pp. 139–153.
[207] O. Strichman, S. A. Seshia, and R. E. Bryant, "Deciding separation formulas with SAT," in *Lecture Notes in Computer Science, Computer Aided Verification*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2404, pp. 209–222.
[208] D. A. Stuart, "Implementing a verifier for real-time systems," in *Proc. IEEE Real-Time Systems Symp. 1990*, pp. 62–71.
[209] S. Taşıran, R. Alur, R. P. Kurshan, and R. K. Brayton, "Verifying abstractions of timed systems," in *Lecture Notes in Computer Science, CONCUR'96*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1119, pp. 546–562.
[210] S. Tripakis and S. Yovine, "Analysis of timed systems using time-abstracting bisimulations," in *Formal Methods Syst. Design*, 2001, vol. 18, pp. 25–68.
[211] F. Wang, "Parametric timing analysis for real-time systems," *Inform. Comput.*, vol. 130, no. 2, pp. 131–150, Nov. 1996. Preliminary version: *Proc. 10th Symp. Logic in Computer Science 1996*, pp. 112–122.
[212] ——, "High-level execution time analysis," in *Lecture Notes in Computer Science, Formal Methods for Real-Time and Probabilistic Systems*. Heidelberg, Germany: Springer-Verlag, 1997, vol. 1231, pp. 325–339.
[213] ——, "Efficient data-structure for fully symbolic verification of real-time software systems," in *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 1785, pp. 157–171.
[214] ——, "Region encoding diagram for fully symbolic verification of real-time systems," in *Proc. 24th Annu. Int. Computer Software and Applications Conf.*, Taipei, Taiwan, R.O.C., 2000, pp. 509–515.
[215] ——, "Parametric analysis of computer systems," *Formal Methods Syst. Design*, vol. 17, pp. 39–60, 2000.
[216] ——, "RED: model-checker for timed automata with clock-restriction diagram," presented at the Workshop on Real-Time Tools, Aalborg, Denmark, 2001.
[217] ——, "Symbolic verification of complex real-time systems with clock-restriction diagram," presented at the IFIP Int. Conf. Formal Techniques for Networked and Distributed Systems , Cheju Island, Korea, 2001.
[218] ——, "Symmetric model-checking of concurrent timed automata with clock-restriction diagram," presented at the Int. Conf. Real-Time and Embedded Computing Systems and Applications 2002, Tokyo, Japan.
[219] ——, "Efficient verification of timed automata with BDD-like data-structures," *Int. J. Softw. Tools Technol. Transf. (Special Issue for VMCAI'2003)*. Preliminary version: *Lecture Notes in Computer Science, Verification, Model Checking, and Abstract Interpretation.* Heidelberg, Germany: Springer-Verlag, 2003, vol. 2575.
[220] ——, "Model-checking distributed real-time systems with states, events, and multiple fairness assumptions," presented at the 10th Int. Conf. Algebraic Methodology and Software Technology, Stirling, U.K., 2004. Preliminary version: *Lecture Notes in Computer Science, Algebraic Methodology and Software Technology.* Heidelberg, Germany: Springer-Verlag, 2004, vol. 3116.
[221] ——, "Symbolic parametric analysis of linear hybrid systems with BDD-like data-structures," presented at the 16th Computer Aided Verification Conf., Boston, MA, 2004. Preliminary version: *Lecture Notes in Computer Science, Computer Aided Verification.* Heidelberg, Germany: Springer-Verlag, 2004, vol. 3114.
[222] F. Wang and P.-A. Hsiung, "Automatic verification on the large," in *Proc. 3rd IEEE High-Assurance Systems Engineering Symp.*, 1998, pp. 134–141.
[223] ——, "Efficient and user-friendly verification," *IEEE Trans. Comput.*, vol. 51, pp. 61–83, Jan. 2002.
[224] F. Wang, G.-D. Hwang, and F. Yu, "Symbolic simulation of industrial real-time and embedded systems—experiments with the bluetooth baseband communication protocol," *J. Embedded Comput.*, vol. 1, no. 1, 2004, to be published.

[225] ——, "TCTL inevitability analysis of dense-time systems," in *Lecture Notes in Computer Science, Implementation and Application of Automata*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2759, pp. 176–187.
[226] ——, "Numerical coverage estimation for the symbolic simulation of real-time systems," in *Lecture Notes in Computer Science, Formal Techniques for Networked and Distributed Systems—FORTE 2003*. Heidelberg, Germany: Springer-Verlag, 2003, vol. 2767, pp. 160–176.
[227] F. Wang, A. Mok, and E. A. Emerson, "Symbolic model-checking for distributed real-time systems," in *Lecture Notes in Computer Science, FME '93: Industrial-Strength Formal Methods*. Heidelberg, Germany: Springer-Verlag, 1993, vol. 670, pp. 632–651.
[228] F. Wang, A. K. Mok, and E. A. Emerson, "Real-time distributed system specification and verification in APTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 4, pp. 346–378, Oct. 1993.
[229] F. Wang and H.-C. Yen, "Parametric optimization of open real-time systems," in *Lecture Notes in Computer Science, Static Analysis*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2126, pp. 299–318.
[230] ——, "Reachability solution characterization of parametric real-time systems," *Theor. Comput. Sci. (Special Issue for the 8th Conf. Implementation and Application of Automata)*, to be published.
[231] H. Wong-Toi, "Symbolic Approximations for Verifying Real-Time Systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1995.
[232] S. Yamane, "Formal specification and verification method for hard real-time s7ystems," *IPSJ J.*, vol. 42, no. 6, 2001.
[233] J. Yang, A. K. Mok, and F. Wang, "Symbolic model-checking for event-driven real-time systems," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 386–412, Mar. 1997.
[234] W. Yi, "Real-time behavior of asynchronous agents," in *Lecture Notes in Computer Science, CONCUR'90*. Heidelberg, Germany: Springer-Verlag, 1990, vol. 458, pp. 502–520.
[235] T. Yoneda and H. Schlingloff, "Efficient verification of parallel real-time systems," *Formal Methods Syst. Design*, vol. 11, no. 2, pp. 187–215, 1997.
[236] T. Yoneda, A. Shibayama, B.-H. Schlingloff, and E. M. Clarke, "Efficient verification of parallel real-time systems," in *Lecture Notes in Computer Science, Computer Aided Verification*, 1993, vol. 697, pp. 321–332.
[237] S. Yovine, "Kronos: A verification tool for real-time systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1/2, pp. 123–133, Oct 1997.
[238] L. Zhang and S. Malik, "The quest for efficient Boolean satisfiability solvers," presented at the Int. Conf. Computer Aided Deduction (CADE 2002), Copenhagen, Denmark.
[239] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," presented at the Int. Conf. Computer-Aided Design 2001, San Jose, CA.

Farn Wang received the B.S. degree in electrical engineering from National Taiwan University in 1982, the M.S. degree in computer engineering from National Chiao-Tung University in 1984, and the Ph.D. degree in computer sciences from the University of Texas, Austin, in 1993.

From 1986 to 1987, he was a Research Assistant in Telecommunication Laboratories, Ministry of Communications, Taiwan, R.O.C. From 1993 to 1997, he was an Assistant Research Fellow at the Institute of Information Science (IIS), Academia Sinica, Taiwan, R.O.C. From 1997 to 2002, he was an Associate Research Fellow at IIS. In 2002, he became an Associate Professor in the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. His research interests are in automating human verification experiences to develop verification tools with precision and efficiency. He architected and implemented several tools for the verification of timed and hybrid systems. These tools include RED, a model-checker for timed and hybrid systems, and SGM, an efficient and user-friendly verification tool for timed systems.