Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science 113 (2005) 3–21

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Semantics and Runtime Monitoring of TLCharts: Statechart Automata with Temporal Logic Conditioned Transitions

Doron Drusinsky[1]

Computer Science Department, Naval Postgraduate School, Monterey, CA, USA
Time Rover Inc., Cupertino, CA, USA

Abstract

This paper describes the semi-formal semantics and a run-time monitoring technique for TLCharts, a visual specification language that combines the visual and intuitive appeal of non-deterministic Harel Statecharts with formal specifications written in Linear-time (Metric) Temporal Logic (LTL and MTL). We describe an automata-theoretic semantics for non-deterministic statecharts with negation and state overlapping and extend it to cater for temporally annotated transitions, thereby providing a simple automata theoretic semantics for TLCharts. We also describe a run-time monitoring technique for TLCharts.

Keywords: Harel Statecharts, temporal logic, finite automata, semantics, formal specification

1 Introduction

Temporal Logic is a special branch of modal logic that investigates the notion of time and order. Linear-time Temporal Logic (LTL) is an extension of propositional logic where, in addition to the well-known propositional logic operators, there are four future-time operators ($\Diamond$-*Eventually*, $\Box$-*Always*, *U*-*Until*, O-*Next*) and four dual-past time operators. Pnueli [19] suggested using LTL for reasoning about concurrent programs. Since then, several researchers have used LTL to state and measure correctness of concurrent programs, protocols, and hardware (e.g., [18,19]). Metric Temporal Logic (MTL) was suggested by

[1] Email: ddrusins@nps.edu

doi:10.1016/j.entcs.2004.01.025

Chang, Pnueli, and Manna as a vehicle for the verification of real time systems [4]. MTL extends LTL by supporting the specification of relative-time and real-time constraints. With MTL, all four LTL future-time operators can be characterized by relative-time and real-time constraints, specifying the duration of the temporal operator. Temporal Logic with Time Series constraints (TLS) was suggested by Drusinsky [6] as an extension of MTL which enables temporal specifications that assert about time-series properties such as stability, monotonicity, and min-max values.

Ever since first published [12], and later incorporated into the OMT methodology and UML standard, Harel statecharts have been described in numerous papers and books (e.g., [3,**?**]). Statecharts extend finite state diagrams with hierarchy (state nesting), concurrence, and history states. Harel Statecharts are typically used for design analysis and implementation; for example, Brugge suggests using statecharts in the design analysis phase of an object oriented UML based design methodology [3]. A formal semantics of Harel statecharts has been suggested in [13]. This paper uses new a automata theoretic semantics for statecharts first suggested in [5]. The semantics lends itself to the inherently non-deterministic semantics required by the TLCharts formalism. It also supports statecharts and TLCharts with overlapping states.

Timed automata extend non-deterministic automata with timed transitions, similar to the *timeout* event mechanism of Harel statecharts [1]. With TLCharts MTL conditions can be used in addition to timeout events. This is an important feature because the LTL *until* operator can neither be simulated in a trivial way by timed automata, nor can nesting of temporal operators.

Sowmya and Ramesh suggested in [21] to use temporal logic assertions with statechart qualities by applying temporal logic in a hierarchical manner; the resulting language is a new hierarchical form of the textual temporal logic formalism. In comparison, our hybrid language is a true automata-theoretic hybrid with a unified syntax and semantics; the resulting language is highly visual and familiar, with special LTL annotation of some transitions.

Enciso et-al. [15] suggest LNint-e, a logic that enables statecharts to temporal logic transforms using a new temporal logic which combines points and intervals to specify the dynamic behavior of programs. In contrast, our suggested language maintains, for the most part, the syntax and semantics of both languages and retains the visual appeal of statecharts.

The Mathworks' Stateflow statechart tool has a so-called temporal logic extension. Stateflow events and conditions can use the four operators *after*, *before*, *at*, and *every*. These four operators are essentially extended versions of the LTL eventuality operator. Most notably, the Stateflow formalism lacks non-determinism, negation, and an operator equivalent to LTL's *Until* operator.

Non-deterministic Finite Automate (NFA) are sometimes used as a specification language [16]. The TLChart formalism suggested in this paper extends the NFA formalism in two ways: it suggests using non-deterministic statecharts instead of flat and sequential NFA formalism, and supports the annotation of transitions with LTL, MTL and TLS conditions.

Existing commercial tools such as the Temporal Rover [9] provide support for LTL and MTL assertion checking within statechart states. Such a tool synergy is less effective than a unified formalism suggested in this paper in that it does not provide semantics for the hybrid language and does not enable unrestricted hybrid representations, such as the statecharts transition function using MTL guards.

In this paper we describe TLCharts, a formalism that visually and intuitively resembles Harel statecharts while enabling temporal-logic conditioned transitions; similar semantics supports a hybrid of statecharts and extended regular expressions. Such hybrids are useful for the specifying abstract non-deterministic temporal properties inside a statechart specification.

Our primary motivation for developing the TLCharts specification language is the concern for effective, early phase specifications, namely system specification that is performed before system design and implementation. This is different than the most prevalent application of formal specifications, which is mostly concerned with the specification of correctness properties for a given, existing, system.

2 An Entry System Control Example

The entry example consists of the following four conditions: (system) *begin*, (system) *end*, *keyPressed, doorOpen* (where *doorClosed* = ¬*doorOpen*) and *alarm*. A particular requirement of interest is:

R: a session is the interval between a *begin* and an *end-condition*. For every such session a *keyPressed* must be repeatedly sensed within two-minute intervals or else an *alarm* must sound within 10 seconds until *keyPressed* is sensed. Also according to this specification, once the alarm sounds then the assertion has succeeded and no more alarms are permitted. The *end-condition* is defined to be true whenever there is an interval which starts with the door closed followed by an *end* being repeatedly sensed until a later time when *begin* is sensed.

We refer the reader to the infusion pump example of [7] which consists of a requirement similar to R. This reference also contains an LTL/MTL specification for the infusion pump requirement and analyzes subtle specification errors that result from the use of linear time temporal logic. A deterministic Harel statechart specification of requirement R is illustrated in Fig. 1. The discussion in [7] compares the accuracy of the statechart of Fig. 1 with that of the TLChart of Fig. 2.

Fig. 1 and Fig. 2 are both legal TLCharts, i.e. Harel statecharts are a special case of TLCharts, and so are LTL and MTL assertions (using a diagram with a single state and transitions annotated with LTL).

TLChart extend Harel statecharts in the following ways:

(i) Some transitions are annotated with LTL, MTL or TLS conditions, such as the transition labeled *alarm U keyPressed* in Fig. 2.

(ii) TLChart's support non-deterministic behavior.

(iii) TLCharts flavor of non-determinism incorporates the specification of both good and bad computations with ambiguities resolved via a priority-based resolution scheme. From an automata theoretic perspective this amounts to existential non-determinism with negation.

(iv) TLCharts can have overlapping states.

A TLChart input string represents a sequence of combinations of stimuli and corresponding system responses; for example, a sequence may contain *keyPressed* - generated by the environment, as well as alarm - a system generated response. The TLChart of Fig. 2 describes legal (accepted) and illegal (rejected) sequences. From a verification standpoint, a rejected string means that the systems behavior does not comply with the specification, typically due to an incorrect system reaction to the input stimuli. This application of diagrams to specification rather than programming and design explains the existence of a sink state (the Error state), which does not typically exist in a design phase statechart.

3 TLCharts: Syntax and Semantics

3.1 Syntax

In this paper we consider Harel statecharts as first described in [Ha], including state hierarchy, concurrence, and history states. Hence, no state overlapping is permitted; this assumption will be changed in the next section. For simplicity, we assume that statechart transitions are annotated with conditions and no events, although we expect TLChart to be used and applied with events and conditions, much like UML statecharts. We use the automata-theory-oriented notation where transitions are annotated with symbols from a finite input alphabet Σ. A practical generalization of the automata model is to use

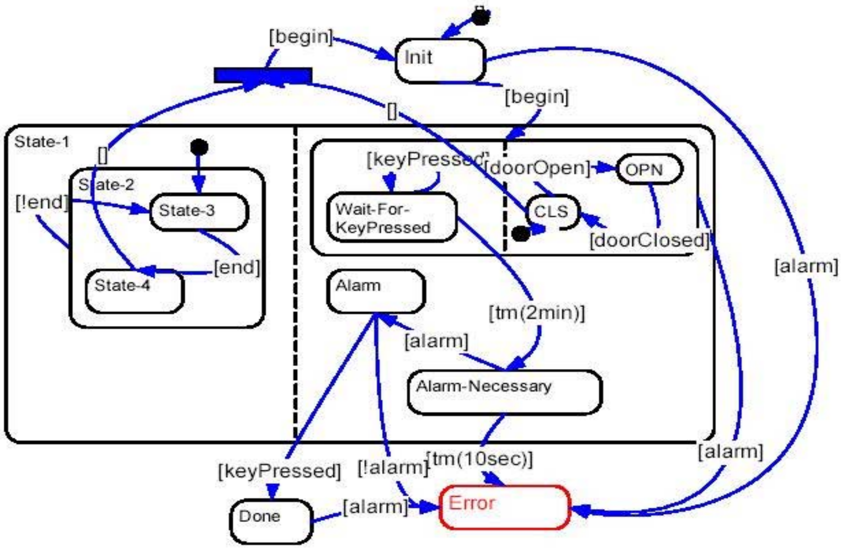

Fig. 1. Deterministic Harel statechart specification for requirement R. A Harel statechart is by definition also a TLChart.

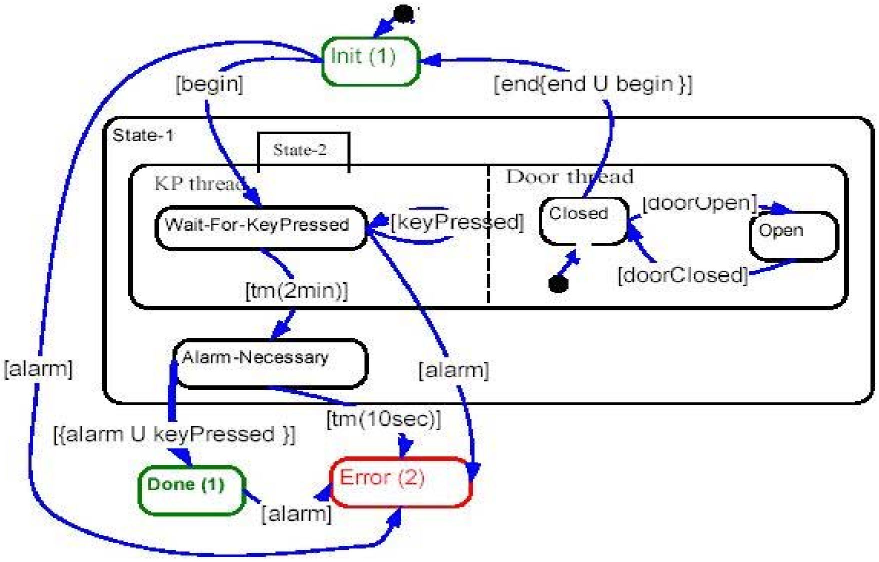

Fig. 2. TLChart specification for requirement R1 (all states other than Error are good states; all states with no specified priority have default, i.e., lowest, priority).

the power set of Σ as the actual input alphabet; this generalization enables multiple simultaneous input channels to the statechart device. Consequently, transition labels are subsets of Σ, where a transition labeled *label* means informally that all symbols s∈*label* must be concurrently present on the input tape for the transition to be traversed.

TLChart transitions are annotated with one or both of the following types of conditions: *propositional* and *temporal*. Propositional conditions are subsets of Σ. Temporal conditions include all legal LTL and MTL formulae. In Fig. 2 temporal conditions are represented using curly braces. Hence [*end* {*end U begin*}] represents the propositional condition *end* and the temporal condition *end U begin*. When using (extended) regular expressions instead of LTL the equivalent condition is [*end* {*end* begin*}].

A TLChart without state overlapping induces an *and/or* state tree as illustrated in Fig. 3. A TLChart with overlapping states, such as the TLChart of Fig. 4, induces an *and/or* state Directed Acyclic Graph (DAG), as illustrated in Fig. 5.

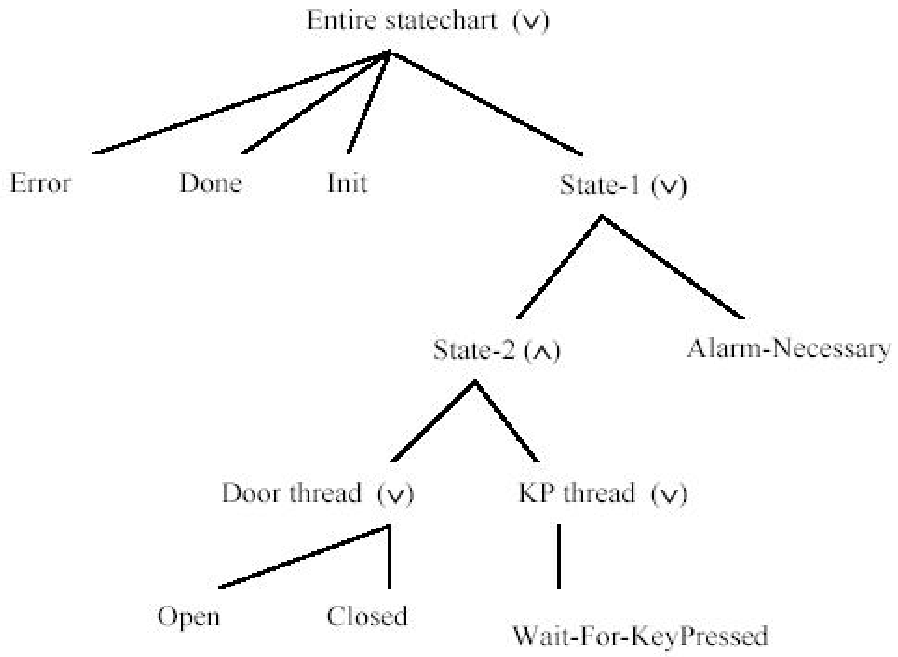

Fig. 3. State *and/or* tree for Fig. 2.

3.2 Semantics without Temporal Conditions

The TLChart formalism specifies requirements using formal languages. The semantics of a TLChart is interlingua-based, using an *Equivalent Non-Deterministic Automaton* (ENFA) [5,?]. Once defined in terms of its ENFA, a TLChart defines correctness properties in a manner that resembles formal logic specification, such as temporal logic specification. It observes a given input tape and decides whether this tape is acceptable or not. In real life terms the input tape corresponds to a combined sequence of inputs to-, and manifested outputs from-, a given system.

Definition 3.1 A *configuration* C is a subset of the TLChart state set in which states of C are pairwise orthogonal, i.e., if for every pair $r, s \quad \in \quad C$, the least common ancestor of $r$and$s$in the state tree is an *and* state.

A configuration C is *maximal* if it is not a subset of any other configuration with a larger cardinality.

For example, in Fig. 2, the configuration {*Wait-For-KeyPressed*} is not maximal while {*Wait-For-KeyPressed*, *Closed*} and {*Wait-For-KeyPressed*, *Open*} are maximal. Note that state configurations do not, in general, contain information about corresponding superstates, such as *Wait-For-KeyPressed* and *Closed* residing under *State-2*, which in turn resides under *State-1*. This information is not necessary because, absent state overlapping, state hierarchy is unique. However, we will change this notation when we describe TLCharts with overlapping states.

The ENFA's state set consists of all possible maximal configurations. Intuitively, this set represents a Cartesian product of state sets of concurrent TLChart threads. We denote a TLChart state within a configuration C as a *constituent* of C.

As a preliminary step, before we describe the ENFA's transition relation, note that we can replace statechart and TLChart hierarchical transitions, such as *State*-1→*Init* in Fig. 2, with concurrence, using a new concurrent thread with one inner state, e.g. *State*-1*a*. The hierarchical transition is then replaced with the transition *State*-1*a* →*Init*. Also, in the following discussion we represent TLChart transitions as annotated binary relations over configurations. For example, in Fig. 1, the hyper edged transition labeled *begin* is the annotated relationship $(\{\text{CLS}, \textit{State-4}\}, \{\textit{Init}\})_{begin}$ and, in Fig. 2, the transition *Closed* → *Init* is represented by the annotated relationship $(\{\textit{Closed}\}, \{\textit{Init}\})_{end,endUbegin}$.

For simplicity, we will first define consider TLCharts with no temporal conditions

Definition 3.2 A *global transition* is an annotated binary relationship between configurations.

Given a TLChart transition $(b', c')_{a'}$ we say it is a *member TLChart-transition* of a global transition $(b,c)_a$ if $b' \subseteq b$, $c' \subseteq c$, and $a' \subseteq a$.

Given a global transition $t=(b, c)_a$ we denote its set of member TLChart transitions as $m(t)$. For example, in Fig. 1, t= ({*OPN, State-3*}, {*CLS, State-4*}$_{\{doorClosed,end\}}$is a global transition, where OPN$\rightarrow_{doorClosed}$ *CLS* and *State-3*$\rightarrow_{end}$ *State-4* are two members of t.

Definition 3.3 Let $(b', c')_{a'}$ be a member transition of some transition $(b, c)_a$, then every state in b' (c') is said to be a *source* (*target*) state in b (c), and we denote the set of all source (target) states for $(b, c)_a$ as the *source* (*target*) *set* of $(b, c)_a$.

Note that member TLChart transitions of a global transition t=$(b, c)_a$ are always non-conflicting otherwise c would not be a valid configuration.

Definition 3.4 A global transition t_1=$(b, c)_a$ is *maximal* if (i) b and c are maximal configurations, and (ii) no other global transition t_2=$(b, e)_a$ exists such that $m(t_1) \quad \subset \quad m(t_2)$.

Intuitively, t_1 pairs maximal configurations b and c by firing a *maximal* set of non-conflicting TLChart transitions that are enabled in configuration b using the inputs symbols in a. For example, in Fig. 1, ({*OPN, Wait-For-KeyPressed, State-3*}, {*CLS, Wait-For-KeyPressed, State-4*}$_{\{doorClosed,end\}}$ is a maximal transition but ({*OPN, Wait-For-KeyPressed, State-3*}, {*CLS, Wait-For-KeyPressed, State-3*}$_{\{doorClosed,end\}}$is not maximal because it did not fire the enabled TLChart transition *State-3*$\rightarrow_{end}$ *State-4*.

ENFA transitions extend maximal transitions in that they might affect states in an indirect way, i.e., without those states being source or target states of any member transition. To this end, we need a few more definitions.

Recall that a statechart (are therefore a TLChart) default state is a designated initial state within a particular level of nesting. Hence, we define the following.

Definition 3.5 A state d *is default under state* s if d is a descendant of $s$in the state tree, d is a default state, and all ancestors of d under s are default states. A state d is a *default state under a configuration* C if d is a default state under the least common ancestor (*lca*) of the elements of C in the state tree.

In Fig. 2 for example, state *Closed* is default under state *Door Thread*, but not default under the TLChart root. Finally, we define ENFA transitions.

Definition 3.6 Let t=$(b, c)_a$ be a maximal transition and let b', c' be the source and target sets of t, respectively. t is a *legal ENFA transition* if for every state $s \in c - c'$ (i.e., s is in c but is not an explicit target state of some member transition), s is a default state under c or else $s \in b - b'$ (i.e., s was not affected by the transition).

For example in Fig. 1, the maximal transition ($\{$*Init*$\}$, $\{$*CLS, For-KeyPressed, State-3*$\}_{\{begin\}}$ is an ENFA transition where the least common ancestor of configuration $\{$*CLS, Wait-For-KeyPressed, State-3*$\}$ is state *State*-1, and *CLS, For-KeyPressed,* and *State-3* are all defaults under *State-1.*

Note that conflicting simultaneously enabled ENFA transitions induce nondeterminism. This is the case in the TLChart of Fig. 2 when *keyPressed*, *doorOpen*, and *alarm* are all true while in configuration $\{$*Closed, Wait-For-KeyPressed*$\}$ enabling two ENFA transitions, one resulting in configuration $\{$*Open, Wait-For-KeyPressed*$\}$ while the other results in configuration $\{$*Error*$\}$.

3.3 Semantics with Temporal Conditions

While standard semantics for LTL are defined using infinite models, in this paper our primarily interest is with finite linear model semantics. Hence, for example, *Eventually* ρ is satisfied if there exists state s in the finite linear model that satisfies ρ.

An LTL model relates to an automatons input tape in the following straightforward way. An LTL model consists of a finite sequence of states with Boolean propositions and corresponding truth assignments assigned to each state. For example, consider a model with two states (i.e., two cycles), where $\{$*begin,* $\neg$*end, KeyPressed,* $\neg$*alarm, doorOpen*$\}$ is the truth assignment for state 0 (interpreted as cycle 0), and $\{\neg$*begin,* $\neg$*end, KeyPressed,* $\neg$*alarm, doorClosed*$\}$ is the truth assignment for state 1. This model is therefore obviously exchangeable with an automaton input tape with the symbol $<$*begin,* $\neg$*end, KeyPressed,* $\neg$*alarm, doorOpen*$>$ in position 0 and $<\neg$*begin,* $\neg$*end, KeyPressed,* $\neg$*alarm, doorClosed*$>$ in position 1. In other words, each Boolean proposition p_i and its negation $\neg \mathrm{p}_i$ form an alphabet Σ_i. The input alphabet Σ for the ENFA is then the Cartesian product of all Σ_i alphabets.

We now incorporate temporal conditions into ENFA behavior. First, note that every ENFA transition has a pair of propositional and temporal conditions, which are the respective conjunctions of all propositional and temporal conditions annotating its member TLChart transitions. Hence we represent an ENFA transition as an annotated binary relation $(b, c)_{a,\rho}$ where a is a propositional condition and ρ is a temporal condition. Temporal conditions affect ENFA behavior via the definition of a computation. Given an input tape, a conventional one-way non-deterministic Finite Automaton (NFA) computation is essentially a sequence of consecutive transitions and corresponding tape head moves to the right; details are available in [16]. ENFA's extend this well known definition by requiring that for every transition t_i in the computation the input tape is observed from position i into the future and back to the past, but without moving the tape head. The transition t_i is then enabled only if the temporal condition is satisfied by the tape, while considering position number i as cycle 0.

Definition 3.7 Let $\sigma = \sigma_1.\sigma_2...\sigma_n$ be an input tape and let $C = c_0.\mathrm{c}_1 \ldots \mathrm{c}_n$be an ENFA computation. C is a *computation on* σ if c_0 is an initial configuration and $\forall i$, $0< i \leq \ n$,the ENFA contains a transition $(c_{i-1},c_i)_{a,\rho}$such that $a \subseteq \sigma_i$,and $\sigma,i \models \rho$ using standard LTL and MTL semantics (e.g., [4]).

For example, using the entry system TLChart of Fig. 2, consider the input tape (using straightforward abbreviations of the entry system conditions): $\sigma = \sigma_1.\sigma_2.\sigma_3.\sigma_4.\sigma_5.\sigma_6=$

$\{B,\neg E,KP,\neg A,DC\}.\{\neg B,\neg E,\neg KP,\neg A,DC\}.$
$\{\neg B,\neg E,KP,\neg A,DO\}.\{\neg B,\neg E,KP,A,DC\}.$
$\{\neg B,E,\neg KP,\neg A,DC\}.\{B,\neg E,\neg KP,\neg A,DO\}.$

The following C_1computation is enabled by σ; each line is considered as a cycle, starting at cycle 0:

$\{Init\}\rightarrow_B$
$\{Wait\text{-}For\text{-}KeyPressed,\ Closed\}\rightarrow_{(none)}$
$\{Wait\text{-}For\text{-}KeyPressed,\ Closed\}\rightarrow_{KP,DO}$
$\{Wait\text{-}For\text{-}KeyPressed,\ Open\}\rightarrow_A$
{Error} (a sink state)

Similarly, the following C_2computation is also enabled by σ:

$\{Init\}\rightarrow_B$
$\{Wait\text{-}For\text{-}KeyPressed,\ Closed\}\rightarrow_{(none)}$
$\{Wait\text{-}For\text{-}KeyPressed,\ Closed\}\rightarrow_{KP,DO}$
$\{Wait\text{-}For\text{-}KeyPressed,\ Open\}\rightarrow_{KP,DC}$
$\{Wait\text{-}For\text{-}KeyPressed,\ Closed\}\rightarrow_\rho$\{Done\}

where ρ is the temporal condition $E \quad U \quad B$. ρ is enabled on cycle 4 because the input tape then points to σ_5 and the tape suffix is $\sigma_5.\sigma_6=$

$\{\neg B,\ E,\neg KP,\neg A,DC\}.\{B,\neg E,\neg KP,\neg A,DO\}$ which satisfies ρ.

Like their logical counterpart, ENFAs represent assertions about a system. They do so using a formal language mechanism, namely by accepting or rejecting strings (tapes). A classical NFA accepts a string using an existential criterion, namely, if a computation ending in a final state exists. A dual universal automaton ($\forall$-FA) accepts a string if all computations end in a final state. Combining both acceptance criteria results in an alternating automaton. Alternatively, an existential NFA with negation can be used instead of a combination of both acceptance criteria. ENFA supports negation using (i) negation inside temporal conditions, (ii) a combination of *good* (accepting) and *error* (rejecting) states. For a given input string $s$an ENFA has one or more possible computations on s, some of which end in a good state while others end in an error state. Conflicts are resolved using a priority scheme where the winning computation is the computation whose last visited state configuration contains a TLChart state *St* whose priority is higher than all other TLChart states in all competing configurations. If *St* is a good state then the TLChart accepts the input string otherwise the TLChart rejects it. For example, in Fig. 2 consider two computations on the input string σ described earlier, $C_1$and C_2. C_1 ends in the configuration $\{Error\}$ where the *error* state *Error* has priority 2. C_2ends in the configuration $\{Done\}$ where *good* state *Done* has priority 1. We use the engineering convention where lower integer values represent higher priorities; hence, σ is accepted because *Done* has a higher priority than *Error*.

Whenever the priority scheme cannot resolve conflicts we arbitrarily select the error computation as overriding. Likewise, whenever a single computation ends in a configuration that contains both good and error states, then we arbitrarily select the error state as overriding.

TLCharts support methods for real-time constraint specification. The first method uses Harel statechart *timeout* (*tm*) events, not unlike the mechanism used by timed automata. The second method uses MTL; consider a variant of Fig. 2 where the pair of transitions *Wait-For-KeyPressed*$\rightarrow_{tm(2min)}$*Alarm-Necessary*, and *Wait-For-KeyPressed*$\rightarrow_{keyPressed}$ *Wait-For-KeyPressed* are replaced with a single transition *Wait-For-KeyPressed*$\rightarrow_{\rho}$*Alarm-Necessary*, where $\rho = \neg\Diamond_{\leq 2min}$ *keyPressed.* Though similar, the two approaches differ with respect to the timing in which state *Alarm-Necessary* is reached. With the first representation *Alarm-Necessary* is reached after two minutes, while under second representation the transition is traversed after one cycle. In this context, we suggest a special visual delay construct, represented with thick edges, which can only be used with the following unnested temporal conditions: $\Box_{\leq d}\rho$ ($\Box\rho$ with an MTL upper bound d), $\Diamond\rho$, and $\rho U\psi$. It means that the transition is traversed only when the temporal condition becomes true, i.e., when the MTL upper bound d in for $\Box_{\leq d}\rho$ is reached, or when ψ is true in $\Diamond\psi$ or $\rho U\psi$. Hence, in Fig. 2, the transition *Done*$\rightarrow_{alarm}$*Error* is enabled only after the final *keyPressed* input that satisfies the preceding transitions' temporal condition (*alarm U keyPressed*) is detected. A thick transition $A \rightarrow_{\rho U\psi} B$is formally but a shorthand representation of a pair of *thin* transitions $A \rightarrow_{\rho U \bigcirc\psi} A'andA' \rightarrow_{\psi} B$,where A' is a new sibling of A in the state tree. A similar approach is used for the other types of thick transitions.

From a semantic perspective, real-time measurements, used by statechart timeout events and MTL constraints, are represented in our ENFA model using a standard monotonically increasing positive integer function that maps each tape cell with a real-time value.

Note that though visually similar to Harel Statecharts, TLCharts are actually used and applied more like a temporal logic specification in the following sense. TLCharts do not describe the token by token reaction of a reactive system to environment stimuli. Rather, TLCharts consider a complete input string s, which combines both environment inputs s_in and system outputs s_out; a TLChart asserts about the legality of an s_out system response to the s_in stimuli.

4 TLCharts with Overlapping States.

The proposed automata theoretic statechart semantics described in Section 3 caters for statecharts with overlapping states [17]. Consider the TLChart of Fig. 4, a variant of the TLChart of Fig. 3 but with overlapping states. In Fig. 4, state *State-OVLP* is an *and* state that shares its substates with the concurrent threads of state *State*-2. Fig. 4 induces the DAG state graph of Fig. 5. The intuitive meaning of the state overlap in Fig. 4 is that it is illegal for a key to be pressed while the valve is open.

From a semantics perspective, ENFA state configurations for TLCharts with overlapping states contain all state nesting information. Hence, the situation where *State-KP* and *Open* are simultaneously visited has two distinct possible representations as ENFA state configurations: {*State*-1, *State*-2, *State-KP, Open*}, and {*State*-1, *State-OVLP*, *State-KP, Open*}. Therefore, the following two computations are distinct, though when considering only leaf states the cycle #1 configurations look alike:

{*Init*}$\rightarrow_B$
{*State*-1, *State*-2, *Wait-For-KeyPressed*, *Closed*}$\rightarrow_{KP,DO}$
{*State*-1, *State*-2, *State-KP*, *Open*}$\rightarrow_{DC}$
{*State*-1, ***State-2***, *State-KP*, *Closed* }
and
{*Init*}$\rightarrow_B$
{*State*-1, *State*-2, *Wait-For-KeyPressed*, *Closed*}$\rightarrow_{KP,DO}$
{*State*-1, ***State-OVLP***, *State-KP*, *Open*}$\rightarrow_{(any)}$
{*Error*}

Given that the second computation ends in *Error*, a state with higher priority than any of the states in {*State*-1, *State*-2, *State-KP*, *Closed*}, the TLChart rejects the input, effectively stating that *State-KP* and *Open* cannot be visited simultaneously.

Fig. 4 contains another instance of state overlapping, where two *or* states overlap, namely *NoAlarm* and *State-1.* Hence, when state *Wait-For-KeyPressed*

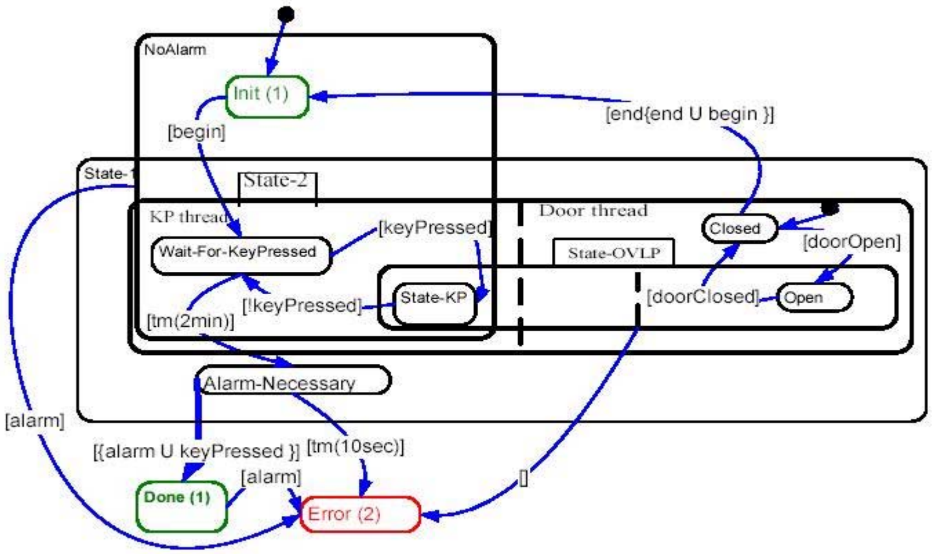

Fig. 4. A TLChart with overlapping states.

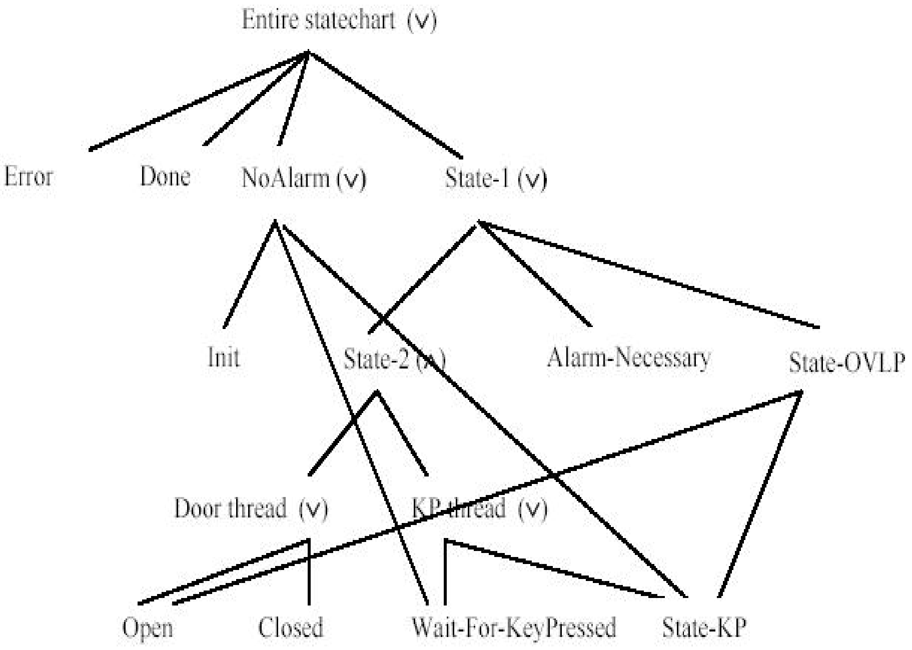

Fig. 5. State *and/or* DAG for the TLChart of Fig. 4.

is visited, it can be considered as residing under state *NoAlarm* or under state *State-1,* each case resulting in a different configuration. Therefore, the following two computations are possible:

$\{Init\} \rightarrow_B$
$\{State\text{-}1,\ State\text{-}2,\ Wait\text{-}For\text{-}KeyPressed,\ Closed\} \rightarrow_{alarm}$
$\{State\text{-}1,\ State\text{-}2,\ Wait\text{-}For\text{-}KeyPressed,\ Closed\}$
and
$\{Init\} \rightarrow_B$
$\{NoAlarm,\ Wait\text{-}For\text{-}KeyPressed\} \rightarrow_{alarm}$
$\{Error\}$

The formal semantics described earlier is extended to support TLCharts with overlapping states in the following way. Given a TLChart T with overlapping states, we say that *T's state DAG contains the tree Tr*, if *Tr* is a spanning tree *in T and is a legal state-tree.* Using the existing semantics of Section 3, every state-tree ST_i contained in the state DAG induces an ENFA, which in turn induces a set S_i of possible, competing, computations; the final outcome of these competing computations is then determined using a priority scheme. The new extended semantics defines the *DAG computation set* S_{DAG} as the union of allS_i sets, and the final outcome is then determined using the priority scheme applied to all computations in $S_{DAG.}$

5 Armor Plating Specifications

Run time assertion checking is a common method for armor-plating programs against unexpected errors. Recently, in [8], armor plating method using runtime monitoring of LTL and MTL assertions combined with exception handling was suggested.

TLCharts offer an opportunity for armor-plating specifications using *over-specification*, namely by adding temporal conditions to an otherwise fully specified TLChart or statechart. Consider for example requirement R and the corresponding TLChart of Fig. 2. A correctness property φ of interest, expressed in MTL, is that in state *Wait-For-KeyPressed*: $(\neg\Diamond_{\le 120}\ keyPressed)$ => $\Diamond_{[120,130]}\ (alarm\ U\ (keyPressed \wedge \Box\neg alarm)$. Therefore, Fig. 2 and 4 can be armor-plated with a transition *Wait-For-KeyPressed*$\rightarrow_{\neg\varphi}$*Error*. This transition overlaps at least two other transitions: (i) it can, with the appropriate conditions, fire together with *Wait-For-KeyPressed*→ *Alarm-Necessary* → *Error*, or (ii) it can, with the appropriate conditions, fire together with *Wait-For-KeyPressed*→ *Alarm-Necessary* → *Done*→ *Error.* The purpose of over-specification is therefore to provide additional assurance that a specific requirement is satisfied.

A different flavor of armor plating involves the addition of temporal conditions as guards to propositional conditions. Such armor plating is useful when designing an implementation statechart, where is can be used for additional safeguards assuring that the specification is conformed to. For example, the developer of the statechart of Fig. 1 can armor-plate the transition $Alarm \rightarrow_{keyPressed} Done$ with the temporal guard $\neg\Diamond alarm$ resulting in the transition $Alarm \rightarrow_{keyPressed \wedge \neg\Diamond alarm} Done.$ By doing so the developer has carried a specification requirement into the design level representation all while using a single coherent formalism.

6 Runtime Monitoring of TLChart Specifications

Our runtime monitoring method for TLCharts (with no state overlapping) combines a particular statechart code generation technique with an existing MTL monitoring tool described in [6,9]. A corresponding product named the StateRover is currently under development by Time Rover, Inc.

The particular statechart Code Generation (CG) algorithm we use is a derivative of a hardware synthesis technique for Harel statecharts described in [10]. This synthesis technique generates a hardware Programmable Logic Array (PLA) representation of the control logic for a hardware controller implementation of a statechart. A PLA is a two-level sum-of-products Boolean logic device. The PLA implements the logic responsible for configuration changes. Configurations are stored in a state-register. In programming terms each PLA product term corresponds to an *if* statement in a conventional programming language, while the hardware state register corresponds data store for the Present Configuration (PC). Hence, the entire statechart is represented as a collection of *if* statements, one per transition in the original statechart, each changing certain states within the PC. We denote each such *if*-statement as an *if-block*. An if-block for the Fig. 1 transition $\{State\text{-}4,\ CLS\} \rightarrow \{Init\}$ is:

```
if ((PC[1]==STATE4) && (PC[3]=CLS) && begin)) { // guard
PC[0]=INIT; PC[1]=0; // assignment of next PC
PC[2]=0; PC[3]=0;
}
```

An if-block consists of a *guard* condition and an *assignment* of elements of the next PC.

TLChart monitoring is performed using CG. The TLChart CG method is a variation of the statechart CG method described earlier with the following modifications:

(i) Instead of a single PC data store, the generated code contains a plurality of potential PC data stores; this plurality of PC stores implements non-deterministic behavior, where each PC store stores the present state within a single computation.

(ii) If-blocks might contain temporal conditions within their guards.

(iii) Each PC store is marked with a tag using one of four values representing the Cartesian product of a logical value (*good/error*) and a Boolean finality value (*steady/transient*). A PC store also contains a simple *counter*. The logical value of the PC is error if any state of the PC is an error state, or else it is marked as good.

The PC finality marking represents the status of a computation with respect to temporal conditions under evaluation. The PC finality marking provides information about the possibility that the PC logical value might change in the future. For example, consider a computation $\ldots \rightarrow \{\mathit{Alarm\text{-}Necessary}\} \rightarrow_{alarmUkeyPressed} \{\mathit{Done}\} \rightarrow_{alarm} \{\text{Error}\}$ in Fig. 2. If a PC is $\{\mathit{Error}\}$ then its logical value is error, but if the finality marking is transient it means that the computation has a potential of not reaching the *Error* state, where the actual outcome depends on the on-going evaluation of a temporal condition such as $alarmU$ *keyPressed*.

Monitoring begins with a set of assigned PC stores, one per initial configuration, all marked as good and steady and having their counters reset to zero. For each configuration PC, a *PC change cycle* consists of firing all if-blocks, resulting in a change of PC. Hence, a new PC set is induced by the current PC set and the current input symbols.

Within a single PC change cycle, the PC holds its value, tag, and counter as generated by the previous change cycle for that PC. Then, each if-block contributes to a change in the PC in the following way. For an if-block that contains only propositional conditions: PC's counter and tag are not affected. Such an if-block affects the PC contents in an obvious way only if it guard succeeds. For example, for the TLChart of Fig. 2, let PC=$\{$*Wait-For-KeyPressed*, *Closed*$\}$, then after receiving the input stimuli *doorOpen* the PC changes to PC=$\{$*Wait-For-KeyPressed*, *Open*$\}$.

Now consider an if-block for a TLChart transition with a temporal condition, such as the transition $\{\mathit{Alarm\text{-}Necessary}\} \rightarrow \{\mathit{Done}\}$ of Fig. 2; the guard for this if-block contains the temporal condition $\rho = alarm\ U\ keyPressed$. When the guard executes, it increments the PC's counter and invokes a code snippet for ρ that is generated by a temporal logic code generator such as the Temporal Rover [9]. The temporal logic related code executes every cycle and results in one of four values, which combine a Boolean *logic* (*success/fail*) result with a Boolean *finality* (*done/not-done*) result describing whether the current logical result can change in the future [9]. Hence, results from the code snippet for ρ affect the PC in question in one of four possible ways:

(i) If the code snippet for ρ results in a *fail-done* pair then the PC in question is not valid anymore and is removed from the set of possible PC's.

(ii) If the code snippet for ρ results in a *success-done* pair then the counter for the PC in question is decremented, and the code snippet dies (it will not execute in a future cycle).

(iii) If the code snippet for ρ results in a *fail-not-done* or *success-not-done* pair then the PC's finality marking is marked as transient.

The PC counter for a particular PC store S counts the number of temporal logic conditions within the computation for S that are still under evaluation. Whenever S is zero it means that no temporal logic code condition within the computation for S is alive; the PC's finality marking is then marked as steady.

7 Conclusion

Harel statecharts and LTL are well-researched and advocated specification languages for reactive systems. Harel statecharts are widely popular through their UML counterpart. LTL on the other hand, is advocated primarily by the academia. While Harel statecharts are visual and deterministic, LTL is textual, logical and non-deterministic. TLCharts combine both formalisms thereby enabling specifications that are visual but also logical and non deterministic when needed. TLCharts have a straightforward formal automata based semantics that supports a meaningful interpretation of statecharts with state overlapping. With TLCharts, temporal conditions are *anchored* in states, such as *alarm U keyPressed* being anchored in the state *Alarm-Necessary* in Fig. 2. This eliminates the need to use deeply nested LTL, when using the pure LTL alternative, or to provide a fully deterministic statechart, when using the Harel statechart alternative. We call this property *just in time TL*. In addition, TLCharts enable specification armor plating.

Although this paper describes TLCharts as a hybrid of statecharts and temporal logic, the notation can be extended to support regular expressions. In fact, a tool currently under development by Time Rover, Inc., will support both temporal logic and regular expression conditions within TLChart diagrams.

Clearly, TLCharts can be abused; a single state TLChart with highly nested LTL and MTL conditions is a legal TLChart and so is a fully deterministic, implementation level detailed, Harel statechart. Further research is needed to establish when each constituent capability of this new formalism actually contributes a significant added value to the specification effort.

Separate research, done under the auspices of the Naval Postgraduate School in Monterey, is investigating the application of TLCharts to the on-going Battle Manager of the U.S. Missile Defense system.

References

[1] R. Alur, and D. Dill, A Theory of Timed Automata, Theoretical Computer Science, 126:183-235, 1994.

[2] G.S. Avrunin, J. C. Corbett, and M. B. Dwyer, Property Specification Patterns for Finite-State Verification, 2nd Workshop on Formal Methods in Software Practice, March 1998.

[3] B. Bruegge, Object-Oriented Software Engineering: Conquering Complex and Changing Systems, Prentice Hall, ISBN 0-13-489725-0.

[4] E. Chang, A. Pnueli and Z. Manna, Compositional Verification of Real-Time Systems, Proc. 9'th IEEE Symp. On Logic In Computer Science, 1994, pp. 458-465.

[5] D. Drusinsky, On Synchronized Statecharts, Ph.D. Dissertation, Weizmann Institute of Science, 1988.

[6] D. Drusinsky, Monitoring Temporal Rules Combined with Time Series, Proc. 2003 Computer Aided Verification Conference (CAV), pp. 114-117.

[7] D. Drusinsky, Visual Formal Specification using (N)TLCharts: Statechart Automata with Temporal Logic and Natural Language Conditioned Transitions. International Workshop on Parallel and Distributed Systems: Testing and Debugging (PADTAD), 2004.

[8] D. Drusinsky, Specs Can Handle Exceptions. Embedded Developers Journal, November 2001, pp. 10-14.

[9] D. Drusinsky, The Temporal Rover and the ATG Rover. Proc. SPIN 2000 Workshop. Springer Verlag Lecture Notes in Computer Science, 1885, p. 323-329.

[10] D.Drusinsky, A State Assignment for Single-block Implementation of Statecharts. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10(12): 1569-1575, December 1991.

[11] D. Drusinsky and David Harel, On the Power of Bounded Concurrency I: Finite Automata. Journal of the ACM, 41(3): 517-539, May 1994.

[12] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8, pp. 231-274, 1987

[13] D. Harel and A. Naamad, The Statemate Semantics of Statecharts. ACM Tran. of Software Engineering and Methodology, 5(4) Oct 1996.

[14] C. Eisner, D. Fishman, J. Havlicek, Y. Lustig, A. McIsaac, D. Van Campenhout, Reasoning with Temporal Logic on Truncated Paths, Proc. 2003 Computer Aided Verification Conference (CAV), pp. 27-39.

[15] M.Enciso, I. P. de Guzm'an, C.Rossi, Using Temporal Logic to Represent Dynamic Behaviour of UML Statecharts, in: ECOOP 2002 Workshop on Integration and Transformation of UML Models (2002).

[16] J. Hopcroft. and J. Ullman,Theory of Formal Languages and Automata, Addison Wesley, 2'nd edition, 2001, ISBN 0-201-44124-

[17] Kahanna, C.A., Statecharts with Overlapping States, M.S. Thesis, Dept. of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel, 1986 (in Hebrew).

[18] Z. Manna and A. Pnueli, Verification of Concurrent Programs: Temporal Proof Principles, Proc. of the Workshop on Logics of Programs, Springer LNCS, 1981 pp. 200-252.

[19] A. Pnueli, The Temporal Logic of Programs, Proc. 18'th IEEE Symp. on Foundations of Computer Science, 1977, 46-57.

[20] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, W. Lorensen, Object Oriented Modeling and Design, Prentice Hall, ISBN 0-13-629841-9.

[21] S. R. Sowmya and S. Ramesh, Extending Statecharts with Temporal Logic, IEEE Transactions on Software Engineering, Vol. 24, No. 3, March 1998 – 1998.