Run-time Monitoring and Recovery of Harel Statecharts using Prioritized Non-deterministic Statechart Specifications

Doron Drusinsky
Time Rover, Inc.,
11425 Charsan Lane, Cupertino, CA, 95014
www.time-rover.com

Abstract

This paper describes the StateRover, a new graphical editor, code generator, run-time monitor, and run-time recovery armor-platter for Harel statecharts augmented with specifications written using prioritized non-deterministic statecharts and metric temporal logic. The StateRover integrates prioritized non-deterministic statechart specifications with deterministic UML-statecharts thereby enabling run-time recovery of deterministic statecharts upon violation of formal requirement specifications. We also compare a Kasas State bounded existance specification pattern written using non-deterministic statecharts with its LTL counterpart.

Harel Statecharts and Statechart Specifications

Harel statecharts have been described in numerous papers and books since first published by Harel [Ha] and later incorporated into the OMT methodology and eventually into the UML standard, (e.g. [Br, RB]). Statecharts extend finite state diagrams with hierarchy (state nesting), concurrence, and history states. Harel Statecharts are typically used for design analysis and modeling; for example, Brugge suggests using statecharts in the design analysis phase of an object oriented UML based design methodology [Br]. A formal semantics of Harel statecharts has been suggested in [HN]. The tools described in this paper rely on an automata theoretic semantics for statecharts described in [D3].

In [D3] Drusinsky described TLCharts, a hybrid of non-deterministic Harel statecharts and temporal logic conditions on statechart transitions and as statechart actions. The StateRover tool described in this paper combines non-deterministic Harel statecharts and temporal logic assertions as statechart actions.

Run-time Monitoring and Run-time Execution Recovery

Run-time Execution Monitoring (REM) is class of methods of tracking the temporal behavior of an underlying application. REM methods range from simple print-statement logging methods to run-time tracking of complex formal requirements (e.g., written in temporal logic) for verification purposes. Recently, NASA used REM for the verification of flight code for the Deep Impact project [DG]. The U.S. Missile Defense Agency (MDA) is currently applying REM to the verification of a new Ballistic Missile Defense System [Ca]. Published REM methods typically use temporal logic, Metric Temporal Logic (MTL), and regular expressions as a specification language [D1]. The StateRover tool described in this paper uses non-deterministic statecharts as a primary specification language but also enables the annotation of states with MTL assertions.

Run-time Execution Recovery (RER) from violations of formal requirement is a technique that integrates REM and the Monitored System (MS) in a closed loop so that the system, once notified of a formal specification violation, throws an exception and manages this exception in a predetermined manner. In [D5] Drusinsky describes a RER technique based on the heterogeneous coupling of REM and source code Java exception handling. This method uses a two layered approach where the REM tool manages specification and monitoring while recovery is performed using Java exception handling. In contrast, our technique uses statecharts as a single medium for specification, monitoring and recovery.

Non-deterministic Statechart Assertions

[D3, D4] described the formal and semi-formal semantics of non-deterministic statecharts. Consider the non-deterministic statechart of Fig. 2. It describes an assertion named *Assertion-1*, which states that no more than 3 new cars can be sensed within a 30 second interval. Non-determinism is implemented using multiple active objects, one per possible Statechart computation. The statechart contains two types of actions, *local* actions such as nCnt++, and *winner* actions such as the print action in the *Error* state. Local actions execute locally within every non-deterministic computation. In contrast, winner actions are resolved using a prioritization scheme and only the winner computation performs its winner actions. For example, in Fig. 2, if one computation is visiting state *Count* (which has priority 1 by default) and another is visiting state *Error* (which has a designated priority 2), then the higher priority state *wins* and its winner action is executed, i.e., the *Error* state print action is executed. In general, all

0-7803-9197-7/05/$20.00 © 2005 IEEE.

Authorized licensed use limited to: Naval Postgraduate School. Downloaded on October 14, 2008 at 14:05 from IEEE Xplore. Restrictions apply.

winner actions from all highest priority active states are executed. This priority scheme enables customized assertion actions within non-deterministic statecharts. Custom winner actions implement the distinction between good and bad behavior commonly used by specification languages, such as a temporal logic assertion succeeding or failing. This is unlike the TLChart notation of [D3], which designates specific good and bad state types.

The StateRover Tool

The StateRover tool consists of a graphical editor, a code generator, an animation engine, and is integrated with the TemporalRover Metric Temporal Logic (MTL) code generator and monitor [D1, D2] for assertion checking in states. The StateRover contains two primary code generators: a code generator for deterministic UML-statechart models combined with flowcharts and MTL assertions, and a code generator for non-deterministic statechart assertions combined with flowcharts. Although the StateRover consists of two separate code generators, the specification assertions are integrated within the UML-statechart model, as illustrated by the Traffic-Light Controller (TLC) statechart example of Fig. 1.

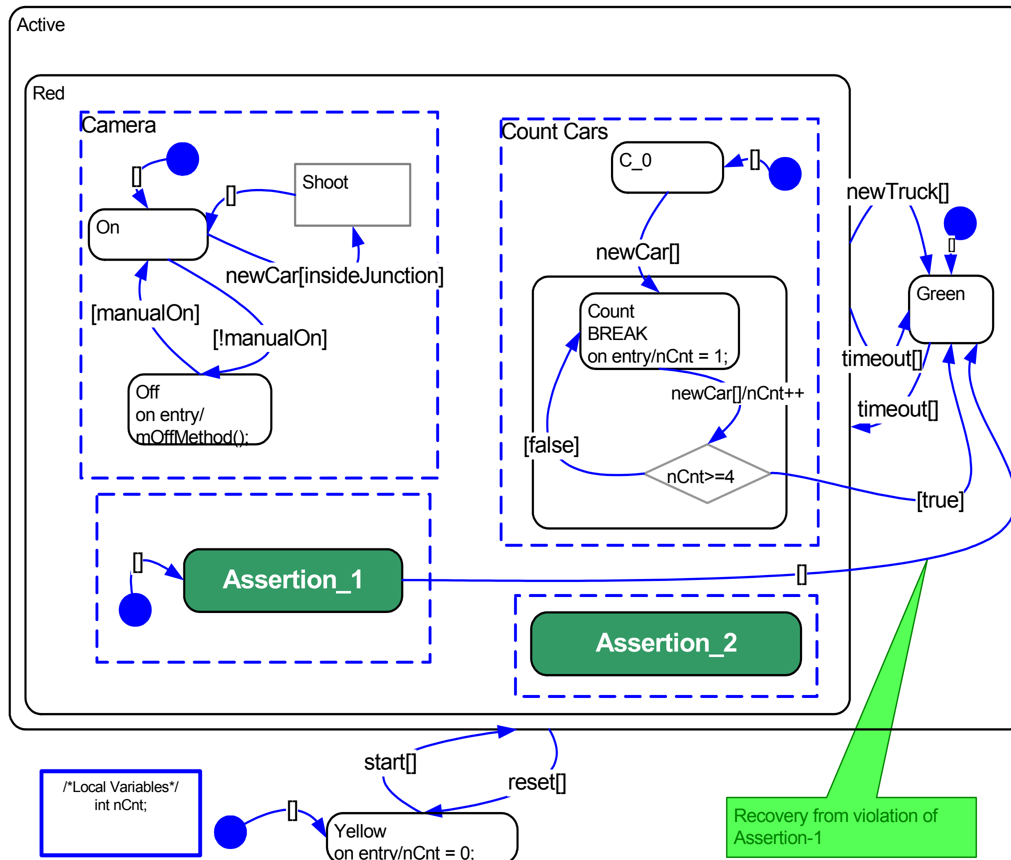


Figure 1. The primary (deterministic) TLC statechart armor plated with Assertion_1

The TLC of Fig. 1 controls lights on a junction between two roads, *Main* (M) and *Secondary* (S). The states of the TLC statechart reflect that traffic-light color assignment to the lights on the M road. While in Red, the statechart monitors and counts cars waiting on the M road and turns lights Green when four or more cars are waiting or when a truck is waiting. Note that StateRover diagrams use variables, actions and conditions from an underlying programming language (e.g., C, C++ or Java). While in the TLC is in the Red state, there is an on-going camera activity orthogonal to the counting activity. Also, while in Red, the non-deterministic statechart *Assertion_1* of Fig. 2 checks that no more than 3 new cars can be sensed within a 30-second interval. Assertion_1 is invoked using the StateRover's sub-statechart mechanism designed to incorporate external statecharts within a given statechart without physically copying the external diagram into the user statechart. *Assertion_1* is code generated using the non-deterministic StateRover code generator. When the TLC enters the Red state it constructs and resets the *Assertion_1* object. Thereafter, as long as the TLC is in the Red state, every event sensed by the TLC statechart object is passed

Authorized licensed use limited to: Naval Postgraduate School. Downloaded on October 14, 2008 at 14:05 from IEEE Xplore. Restrictions apply.

on to the *Assertion_1 object,* which performs its own (non-deterministic) state changes per that event. Assertion_2, which is active within the *Active* state, will be discussed in the sequel.

We denote the UML-statechart model as the *primary model*. In general a primary model is armor-plated with a plurality of non-deterministic statechart assertions represented as sub-statecharts.

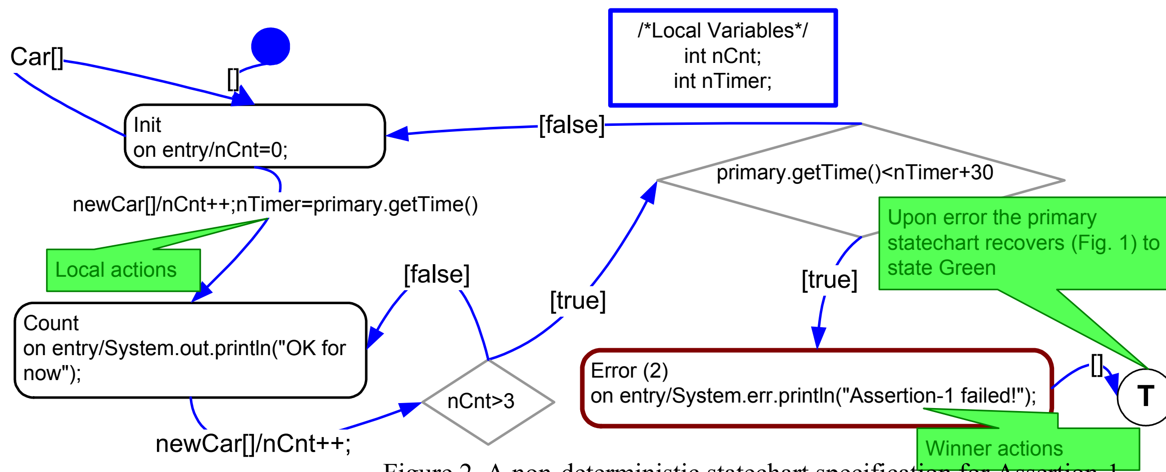


Figure 2. A non-deterministic statechart specification for Assertion-1

Prioritized non-deterministic statechart specifications under the StateRover environment enjoy all of the features available to describe deterministic statecharts within the StateRover, such as Harel statechart hierarchy, concurrence and history states, integrated flowcharts within statecharts, sub-statecharts, and critical regions - a visual specification of synchronization over shared resources. For example, Fig. 3a illustrates the StateRover specification of a bounded-existence specification pattern taken from the Kansas State specification patterns library [KSU] specifying that *transitions to P-states occur at most 2 times between states Q and R*. The LTL specification as suggested by [KSU] is: []((Q & <>R) -> ((!P & !R) U (R | ((P & !R) U (R | ((!P & !R) U (R | ((P & !R) U (R | (!P U R)))))))))). Compositionality of prioritized non-deterministic statecharts is achieved using the sub-statechart mechanism, as illustrated by the example of Q in Fig. 3b. We argue that that non-deterministic statecharts are visual, simpler and more readable than the corresponding temporal logic code; in addition, the non-deterministic statechart specification enjoys a more flexible counting mechanism: it can count many more than two occurrences of P without any significant change in the specification. In fact, the counting value can be defined dynamically, in run-time.

Assertion_2 of Fig. 4 is an example of a *past-time* assertion. It specified, for the primary TLC controller of Fig. 1, that *at least three cars should be detected within the 30 second interval that precedes a time in which the primary controller enters the Green state*.

Run-time Recovery of Primary Statechart Models using Non-deterministic Statechart Assertions

The TLC example of Figures 1 and 2 illustrates the StateRover recovery mechanism. The non-deterministic statechart assertion of Fig. 2 detects a violation of the requirement it changes states to the Error state, which has a higher priority than all other states in the assertion and is therefore the *winner* state. Using the terminal-state notion (**T**) the Assertion_1 sub-statechart terminates its execution, and the parent TLC moves from the Assertion_1 state (a substate under the Red state, orthogonal to the Counter and Camera threads) to the Green state. In other words, the primary TLC statechart recovers by moving to the Green state when the assertion fails.

Using the sub-statechart notation the same assertion specification can be reused in multiple locations in one or more statecharts. Each use induces a new assertion object.

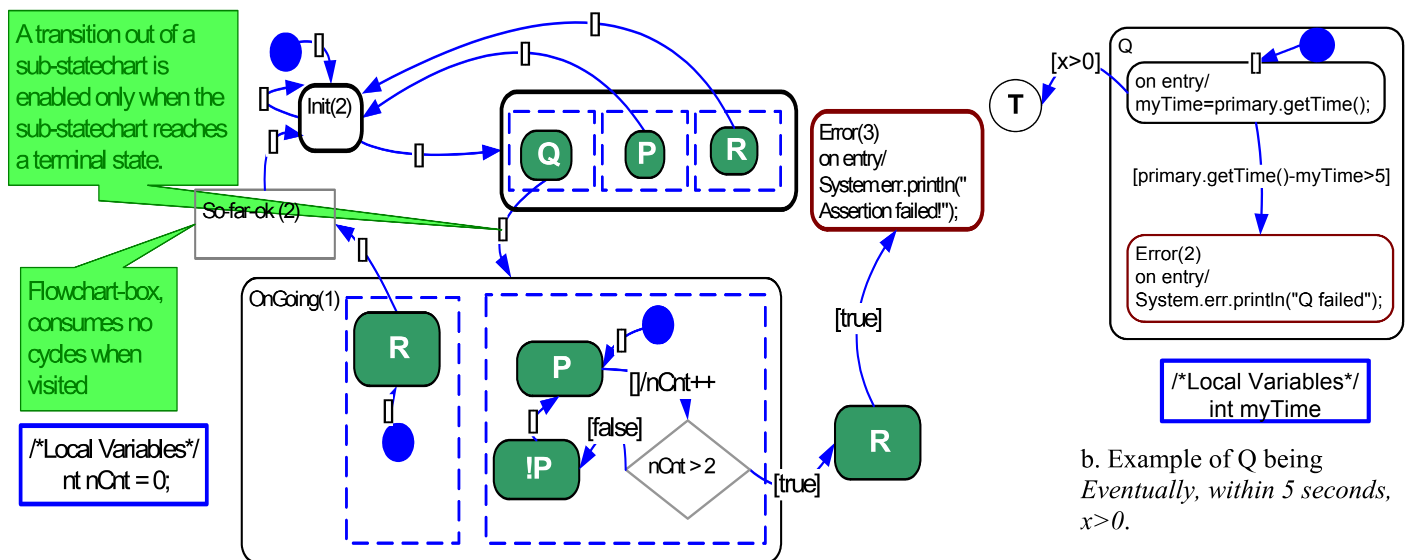
References

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

[DG] D. Drusinsky, G. Watney, Applying Run-Time Monitoring to the Deep-Impact Fault Protection Engine, 28'th IEEE/NASA Software Engineering Workshop, 2003.

Authorized licensed use limited to: Naval Postgraduate School. Downloaded on October 14, 2008 at 14:05 from IEEE Xplore. Restrictions apply.

[D1] D. Drusinsky - The Temporal Rover and ATG Rover. Proc. Spin 2000 Workshop, Springer Lecture Notes in Computer Science, 1885, pp. 323-329.
[D2] D. Drusinsky - Monitoring Temporal Rules Combined with Time Series, Proc. 2003 Computer Aided Verification Conference (CAV), pp. 114-117.
[D3] D. Drusinsky - Semantics and Runtime Monitoring of TLCharts: Statechart Automata with Temporal Logic Conditioned Transitions, Fourth Workshop on RunTime Verification, ETAPS'04 Conference. Invited paper.
[D4] 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. Invited paper.
[D5] D. Drusinsky - *Specs Can Handle Exceptions.* Embedded Developers Journal, November 2001, pp. 10-14. (http://eet.com/embedsub/archive.html).
[Ha] D. Harel, A Visual Formalism for Complex Systems, Science of Computer Programming, 8, pp. 231-274, 1987.
[KSU] Specification Patterns, http://patterns.projects.cis.ksu.edu/



b. Example of Q being *Eventually, within 5 seconds, x>0.*

a. Non-deterministic statechart specification of bounded existence. P, Q, R and !R refer to sub-statecharts for the respective specifications.

Figure 3. Non-deterministic statechart and LTL specifications of a KSU bounded existence property.

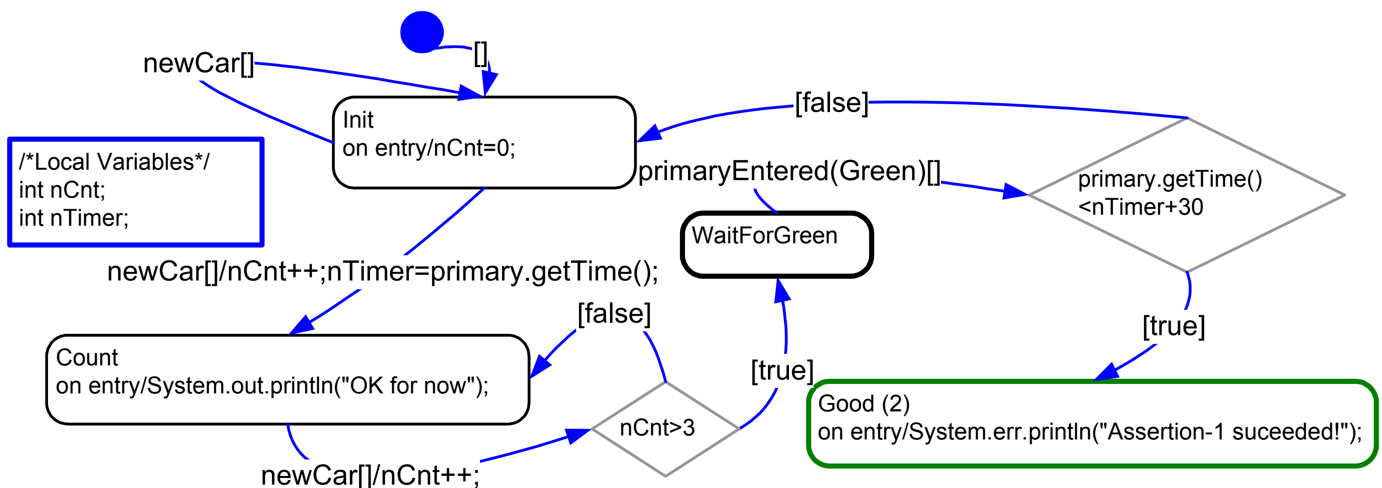


Figure 4. The non-deterministic statechart specification for past-time assertion Assertion-2. Note that Priorities are relative, i.e., priorities in sub-statecharts assertions are considered lower priorities in primary statechart.

Authorized licensed use limited to: Naval Postgraduate School. Downloaded on October 14, 2008 at 14:05 from IEEE Xplore. Restrictions apply.