Refinement and Verification of Responsive Control Systems

Karla Morris[1(✉)], Colin Snook[2], Thai Son Hoang[2], Geoffrey Hulette[1], Robert Armstrong[1], and Michael Butler[2]

[1] Sandia National Laboratories, Livermore, CA, USA
knmorri@sandia.gov
[2] ECS, University of Southampton, Southampton, UK

Abstract. Statechart notations with 'run to completion' semantics, are popular with engineers for designing controllers that respond to events in the environment with a sequence of state transitions. However, they lack formal refinement and rigorous verification methods. Event-B, on the other hand, is based on refinement from an initial abstraction and is designed to make formal verification by automatic theorem provers feasible. We introduce a notion of refinement into a 'run to completion' statechart modelling notation, and leverage Event-B's tool support for theorem proving. We describe the difficulties in translating 'run to completion' semantics into Event-B refinements and suggest a solution. We outline how safety and liveness properties could be verified.

Keywords: Run-to-completion · Statecharts · Refinement

1 Introduction

Reactive Statecharts are open systems capable of receiving potentially non-deterministic input. This work, which builds on our previous work [7,8], exposes a srhallow embedding of open Statecharts semantics in Event-B. Statecharts provide a graphical language, generalized from state machines, that is popular with engineers, variants of which appear in Matlab Simulink/Stateflow [6] and the Ansys tools. Particularly attractive is providing accessibility to abstraction/refinement via Rodin/Event-B which has an intuitive metaphor in the Statechart semantics [7,8]. The commercial tools have similar ideas expressed as encapsulation and composition but not entailing any formal guarantees. The hope is that engineers can better understand the origin of proof obligations in refinements and achieve formal guarantees earlier in their designs where it is most tractable.

Related work has developed a number of different semantics all with different purposes and outcomes [2,3,5]. Because our contribution is focused on a mapping to Event-B, safety property preserving refinement is key. Event-B provides not only a definition of refinement but a rubric for finding valid refinements and this

A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 272–277, 2020.
https://doi.org/10.1007/978-3-030-48077-6_23

is carried over into the Statecharts work presented here. In our version of Statechart semantics, refinement means a subsetting of traces from an abstraction. This has the beneficial effect of preserving safety properties from abstraction to refinement and permits proofs to be discharged at the highest tractable level of abstraction. It is at the highest level of abstraction that proofs are presumably the easiest to discharge.

2 Background

SCXML is a modelling language based on Harel statecharts [12]. State-Chart XML (SCXML) follows a 'run to completion' semantics, where trigger events may be needed to enable transitions. Trigger events are queued when they are raised, and then one is de-queued and consumed by firing all the transitions that it enables, followed by any (un-triggered) transitions that then become enabled due to the change of state caused by the initial transition firing. This is repeated until no transitions are enabled, and then the next trigger is de-queued and consumed. There are two kinds of triggers: internal triggers are raised by transitions and external triggers are raised by the environment (non-deterministically for the purpose of our analysis). An external trigger may only be consumed when the internal trigger queue has been emptied.

Event-B is a formal method for system design [1,4]. It uses *refinement* to introduce system details gradually into the formal model. An Event-B model contains two parts: *contexts* and *machines*. Contexts contain *carrier sets*, *constants*, and *axioms* constraining the carrier sets and constants. Machines contain *variables* v, *invariants* $\mathsf{I}(\mathsf{v})$ constraining the variables, and *events*. An event consists of a guard denoting its enabled-condition and an action defining the value of variables after the event is executed. In general, an event e has the form: **any** t **where** $\mathsf{G}(\mathsf{t},\mathsf{v})$ **then** $\mathsf{S}(\mathsf{t},\mathsf{v})$ **end** where t are the event parameters, $\mathsf{G}(\mathsf{t},\mathsf{v})$ is the guard of the event, and $\mathsf{S}(\mathsf{t},\mathsf{v})$ is the action of the event.

Machines can be refined by adding more details. Refinement can be done by extending the machine to include additional variables (*superposition refinement*) representing new features of the system, or by replacing some (abstract) variables by new (concrete) variables (*data refinement*).

UML-B State-machines provides a diagrammatic modelling notation for Event-B in the form of state-machines and class diagrams [9–11]. The diagrammatic models relate to an Event-B machine and generate or contribute to parts of it.

Each state is encoded as a boolean variable and the current state is indicated by one of the boolean variables being set to TRUE. An invariant ensures that only one state is set to TRUE at a time. Events change the values of state variables to move the TRUE value according to the transitions in the state-machine.

While the UML-B translation deals with the basic data formalisation of state-machines it differs significantly from the semantics discussed in this manuscript. UML-B adopts Event-B's simple guarded action semantics and does not have a concept of triggers and run-to-completion. Here we make use of UML-B's state-machine translation but provide a completely different semantic by generating

a behaviour into the underlying Event-B events that are linked to the generated UML-B transitions.

3 Run to Completion

The run to completion semantics is specified via an abstract basis that is extended by the model. The specification of this basis consists of an Event-B *context* and *machine* that are the same for all input models and are refined by the specific output of the translation. This allows us to introduce an abstract behaviour of transitions queueing and using triggers which is gradually refined to introduce the actual triggering and transitions of the specific example being modelled. It would not otherwise be possible for newly introduced transitions to modify the abstract queues. The basis context introduces a set of all possible triggers, which is partitioned into internal and external triggers (e.g. FutureInternalTrigger and FutureExternalTrigger respectively), some of which will be introduced in future refinements. Each refinement partitions these trigger sets further to introduce concrete triggers, leaving a new abstract set to represent the remaining triggers yet to be introduced.

The basis machine declares variables that correspond to the internal and external queues, the dequeued trigger and a flag that signals when a run to completion macro-step has been completed (no un-triggered transitions are enabled). The abstract event futureTriggeredTransitions represents a combination of transitions that are triggered by the trigger presently ready to dequeue, dt. The actions of these transitions may also raise triggers of their own.

In the process of refining a model, a designer takes advantage of the non-determinism in the abstraction to introduce new triggers and statechart behaviour that refines abstract events. By default a run may non-deterministically complete at any stage until no un-triggered transitions are enabled (when completion is the only choice left). This allows for future refinements that may strengthen the guards of transitions (e.g. by introducing new nested states as the source of a transition) Such guard strengthening refinements correspond to earlier (i.e. weaker) completion, hence the need to allow for this behaviour in the abstraction. When a refinement level is reached for which the designer wants to verify a property that relies on a particular control response within the current run, early completion must be disallowed. This is done by specifying (as an annotation in the SCXML model) that the transitions involved in the run are *finalized*. The SCXML translation tool will then automatically strengthens the guards of the completion events to ensure that the run to completion sequence is not interrupted early by non-deterministic behaviour.

The translation of a specific SCXML model extends that described in [7,8] with the following additions:

Trigger Queues in Basis: The encoding of trigger queues in the abstract basis machine has been improved so that triggers are properly dequeued before potential use, which allows triggers to be discarded if the controller cannot respond to them. This more accurately reflects the SCXML semantics.

Finalisation: Transitions can be flagged as finalised which means their guards can not be strengthened in subsequent refinements. This allows them to be 'enforced' when they are enabled (i.e. completion cannot occur until they have fired) which is needed for verification.

Restricted Raising of Internal Triggers: Once a trigger is introduced it must immediately be raised at that refinement level by any transitions that wish to do so. It cannot be raised in later refinements except by newly introduced transitions. This restriction was necessary to make simulation more useful by removing non-deterministic raising of triggers in anticipation of refinements.

Context Instantiation: The axioms of the basis context, that allow future triggers to be added, have been improved so that ProB[1] can automatically create an instantiation.

A tool to automatically translate SCXML models into UML-B has been produced.

4 Statechart Refinement

Our system includes three refinement rules.

1. Guard conditions on a transition can be strengthened; this can done by adding textual guards to the transition, or changing the source of the transition to a nested state.
2. Transitions can have additional actions, provided they do not modify variables appearing in the abstraction; this can be accomplished by adding textual action to the transition or by changing the target to nested state.
3. A statechart can be embedded within a state of another statechart – sometimes called hierarchical composition or hierarchical refinement.

Via the translation explained in Sect. 3, these rules rely on the usual Event-B proof obligations to ensure that they do indeed yield refinements in the Event-B semantics.

5 Verification of Safety Properties

In a state-chart model we naturally wish to verify properties P, about other parallel statechart regions and auxiliary data, that are expected to hold true in a particular state S. Hence, all of the safety properties that we consider are of the form: S=TRUE ⇒ P, where the antecedent is implicit from the containment of P within S.

SCXML models represent components that respond to received triggers and are not perfectly synchronised with changes in the monitored properties. Hence, P may be temporarily violated until the system responds by leaving the state

[1] ProB is an animator, constraint solver and model checker for the B-Method. https://www3.hhu.de/stups/prob.

S in which the property is expected to hold. To cater for this we express P in a modified form P' that allows time for the response to take place. There are two forms of reaction that can be used to exit S; a) an untriggered transition, or b) a transition that is triggered by an internally raised trigger. For a), the modified property P' becomes P $\vee$ *untriggered transitions are not complete*, and for b) P' becomes P $\vee$ *trigger* t *is in the internal queue or dequeued* (where t is the internal trigger raised when the violation of P is detected).

6 Verification of Control Responses

It is sometimes possible to construct a model that satisfies some invariant (e.g. safety) properties, but does not behave in a useful way. Therefore, as well as verifying invariant properties, we would like to verify the system's responsiveness. More specifically in this case, we want to ensure that the controller responds to external triggers to make appropriate modifications to the system variables. These kind of live responses can not be verified by proof of invariants since they are temporal properties. Instead, we can express the property in Linear Temporal Logic (LTL) and use the ProB model checker to verify it.

In general, our liveness properties will have the following form:

$$\mathsf{G}([\mathsf{external\_trigger\_event}] \Rightarrow \mathsf{F}\{\mathsf{predicate}\}),$$

where the predicate concerns variables v that the system maintains, and may refer to old values old(v) that existed when the external trigger occurred. To specify a liveness property to be verified, a special LTL element is added to the SCXML model with attributes, property (a string of the above form) and refinement (an integer indicating the refinement level at which the property should be verified). The translator generates a separate 'branch' refinement for each LTL property to be verified. In this special refinement, history variables are added to record the value at the state when the external trigger occurs, of any variables that are referenced as 'old' values. A text file is automatically generated containing the LTL property to be checked.

In this generated version, an assumption of strong fairness is added for all other events in the model. (This assumption is stronger than necessary since some events will not affect the outcome, but is easier to generate and is sufficient for our verification aim).

$$\mathsf{SF}[\mathsf{e1}] \wedge \mathsf{SF}[\mathsf{e2}] \ldots \Rightarrow \mathsf{G}([\mathsf{external\_trigger\ \_event}] \Rightarrow \mathsf{F}[\mathsf{predicate}])$$

This property is then verified using the LTL facility of the ProB model checker.

7 Conclusion

Statecharts are useful and widely used by engineers for modelling the design of control systems that respond to sensed changes in the environment. Event-B provides an effective language for formally verifying properties via incremental

refinements. However, it is not straightforward to apply the latter to the former. We have developed a technique for introducing refinement of Statecharts that can be translated to Event-B for verification. Invariant properties about the expected coordination of states can be added and are interpreted with additional allowance for the reactions to take place in the control system. Such invariants prove automatically with the existing Rodin theorem provers. We use an LTL model checker as a complementary process for verifying expected reactions to environmental triggers.

In future work we intend to formalise the semantics of our extended SCXML notation in order to define its notion of refinement and correspondence to Event-B.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Syriani, L.L.E., Sousa, V.: Structure and behavior preserving statecharts refinements. Sci. Comput. Program. **170**(15), 45–79 (2019)
3. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
4. Hoang, T.S.: An introduction to the Event-B modelling method. In: Romanovsky, A., Thomas, M. (eds.) Industrial Deployment of System Engineering Methods, pp. 211–236. Springer, Heidelberg (2013)
5. Maraninchi, F.: The Argos language: graphical representation of automata and description of reactive systems. In: IEEE Workshop on Visual Languages (1991)
6. MATLAB. 9.7.0.1190202 (R2019b). The MathWorks Inc., Natick, Massachusetts (2019)
7. Morris, K., Snook, C.: Reconciling SCXML statechart representations and Event-B lower level semantics. In: HCCV - Workshop on High-Consequence Control Verification (2016)
8. Morris, K., Snook, C., Hoang, T.S., Armstrong, R., Butler, M.: Refinement of statecharts with run-to-completion semantics. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2018. CCIS, vol. 1008, pp. 121–138. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12988-0_8
9. Said, M.Y., Butler, M., Snook, C.: A method of refinement in UML-B. Softw. Syst. Model. **14**(4), 1557–1580 (2015)
10. Snook, C.: iUML-B statemachines. In: Proceedings of the Rodin Workshop 2014, Toulouse, France, pp. 29–30 (2014). http://eprints.soton.ac.uk/365301/
11. Snook, C., Butler, M.: UML-B: formal modeling and design aided by UML. ACM Trans. Softw. Eng. Methodol. **15**(1), 92–122 (2006)
12. W3C. State chart XML SCXML: State machine notation for control abstraction, September 2015. http://www.w3.org/TR/scxml/