Verification of Statecharts using Temporal Logic

Daniel Bruns

Seminar Formale Software-Entwicklung
Institut für theoretische Informatik, Universität Karlsruhe
Sommersemester 2008

1 Introduction

Statecharts play an important role in describing dynamic views in specification. Since they are based on a formal semantic model, they are often considered subject to formal verification. Approaches for verification via model checking have been made as early as statecharts were introduced. Thums et. al.[TSOR04] pursue another path: Statechart semantics are described using *Interval Temporal Logic* (ITL)[CMZ08]. The idea is to symbolically execute the statechart and prove temporal properties such as "$state_x$ is eventually reached" or "the value of variable z is always at least 0". For that purpose a sequent calculus has been established and successfully implemented into the *KIV verification tool*[BRS+08].

In the following we will first give a short introduction to both statecharts and ITL and then concentrate on translation issues, though we tend to avoid going into all-too-technical detail.

2 Statecharts

Statecharts were first introduced by Harel et. al. in 1984 by the well-known STATEMATE standard[HLN+90]. UML state machines are largely based on

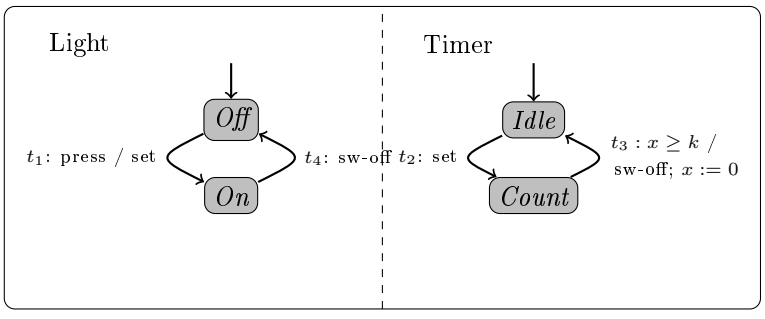

Figure 1: Example of a statechart with and-states and static reaction

this definition. Statecharts possess a hierarchical structure of states and substates. They can be *basic states*, i.e. with no substates, *or-states*, i.e. exactly one substate is active at once or *and-states*, i.e. all substates are active at the same time. Transitions between states are labelled with *guard conditions* and *actions*. In the example given in Fig. 1, the transition from "off" to "on" is named t_1 and has guard "press" and action "set". In addition, states can be labelled with *static reactions* within them and actions on entering or leaving a state can be defined. However these features do not extend the expressiveness of statecharts and can easily be simulated.

3 The logical framework

We will be using Interval Temporal Logic (ITL), first introduced by Moszkowski in 1985. An *interval* formally is a finite or infinite sequence of first order structures. The intuitive meaning of a sequence is the time interval on which assumptions are made by temporal formulae. The well-known temporal operators such as $\Box$ (always), $\Diamond$ (eventually) and $\circ$ (next) can be derived. In contrast to logics like Linear Time Logic (LTL), to deal with finite intervals, a *weak next* operator $\bullet$ is also featured, which does not necessarily require a following point in time to exist.

Definition 1 (Semantics of ITL)

- $(\tau_0, \ldots) \vDash \psi$ *iff* $\tau_0 \vDash \psi$ *for a pure predicate logic formula* ψ
- $(\tau_0, \tau_1, \ldots) \vDash \Box\phi$ *iff* $\tau_0 \vDash \phi$ *and* $(\tau_1, \ldots) \vDash \Box\phi$
- $(\tau_0) \vDash \Box\phi$ *(interval of length 1) iff* $\tau_0 \vDash \phi$
- $(\tau_0, \tau_1, \ldots) \vDash \circ\phi$ *iff* $(\tau_1, \ldots) \vDash \phi$
- $(\tau_0) \vDash \circ\phi$ never *holds*

The other operators are derived from their duals: $\Diamond\phi := \neg\Box\neg\phi$ and $\bullet\phi := \neg\circ\neg\phi$. ITL is far more expressive than LTL. Not only the finiteness of the time structure can be asserted, but properties like "ϕ holds on every second point in time".

Based on the above definitions we are able to establish a sequent calculus. We will give only two rules as an example:

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \bullet\Box\phi, \Delta}{\Gamma \vdash \Box\phi, \Delta}\ box - right \tag{1}$$

$$\frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \circ\Diamond\phi \vdash \Delta}{\Gamma, \Diamond\phi \vdash \Delta}\ diamond - left \tag{2}$$

4 Translation into ITL

The basic idea behind translation of statecharts into ITL is to treat traces of statechart configurations as an interval. A *configuration* is determined by the valuation of all statechart variables. A progression from one configuration to another will be called a *step*. Every step computes the variable valuation for

the following configuration. Therefore we assign to every variable x a primed counterpart x' which valuates to the value of x in the next configuration, i.e. $\beta_{\tau_i}(x') = \beta_{\tau_{i+1}}(x)$. We associate every state with a predicate of the same name indicating whether it is active. The statechart itself is considered a set of axioms from which information on transitions is derived.

In the following, we will firstly calculate possible steps and then construct corresponding ITL-formulae inserted into our sequent calculus.

4.1 Calculation of possible steps

We recursively calculate the set Σ of statechart steps which can possibly be taken. Each step is given as a tuple (γ, T), where γ is a predicate logic formula defining preconditions (active states and guard configuration) and T is the set of transitions which may be taken.

- For a *basic state* s it is always empty since there are no transitions within s at all.
- For an *or-state*, T is required to be non-conflicting, i.e. no two transitions within the same state must be taken.
- *And-states* take care that T may be more than a singleton since all substates are active at the same time. So, their set of possible steps calculates via Cartesian product.

In the above example there are theoretically 32 possible steps, since there are 4 possible state configurations and for any of them there are 4 possible guard configurations for the transitions, summing up to 16. Additionally, the static reaction could take place in state configuration. (Given a reasonable starting configuration however there are fewer steps, e.g. the static reaction can only occur within "Count".) Calculated steps would be $(Count \wedge x \geq k \wedge On \wedge \neg swoff, \{t_3\})$ or $(Off \wedge \neg press \wedge Idle \wedge \neg set, \emptyset)$.

So far, we have considered any step allowed by the statechart. In a more general definition, Σ is restricted by an ITL sequent $\Gamma \vdash \Delta$, where Γ contains information about currently active (resp. inactive) states and Δ is expected to contain properties to be proven. E.g. if $\neg On \in \Gamma$, then $(On \wedge \ldots, \{t4\})$ is not a possible step.

Reconsider the example in Figure 1. If the light is switched on, we will enter the state "Count". But if $x < k$ neither transition nor static reaction can be executed. The corresponding step element therefore has an empty transition set. This does not mean there is no step and nothing can be done. The resulting step is called a *macro step*. In contrast to *micro steps* it does not modify the configuration of the statechart but reads events from the environment (in our example there are none) and generates a special event called *tick*. Tick symbolizes an external time event and – in our example – enables the static reaction as the upfollowing micro step. This is not to be confused with the time structure of ITL which is associated with *every* step.

4.2 Establishing sequent calculus rules

To a given statechart $\mathcal{S}$ and sequent $\Gamma \vdash \Delta$ we examine every possible step simultaneously. Each step $\sigma = (\gamma, T) \in \Sigma(\mathcal{S}, \Gamma, \Delta)$ gives rise to three formulae:

- $cond(\sigma)$ – the condition which is just γ,
- $exec(\sigma)$ which executes the actions symbolically and
- $next(\sigma)$ which contains information about active states in the next configuration.

Since events are associated with predicates, every event occurring in T is set to true for the next configuration by *exec*. For a macro step, both events from the environment and tick are set to true, too. $next(\sigma)$ consists of the conjunction of all active states and the negation of inactive states for the next configuration (depending on the targets of T). This results in the following sequent rule:

$$\frac{\Gamma, \bigvee_{\sigma \in \Sigma} (cond(\sigma) \wedge exec(\sigma) \wedge \circ next(\sigma)) \vdash \Delta}{\mathcal{S}, \Gamma \vdash \Delta}\ sc - unfold \tag{3}$$

4.3 Proving the assumptions

So far we have shown how to derive a formal description of possible steps. Now it is time to use our sequent calculus to prove some properties. Recall the above example once again. A desired property might be that once the light is on, it will eventually be off (without any environmental activities). The corresponding sequent would be $On \vdash \Diamond Off$. With the above construction, this can be proven in a finite number (more precisely $\mathcal{O}(k)$) of sequent calculus steps as shown (heavily simplified) in Fig. 2. Over the first few steps, we eventually reach a

$$\cfrac{\cfrac{\cfrac{\cfrac{(*)}{\mathcal{S}, Idle, Off \vdash Off, \circ\Diamond Off}\ diamond - right}{\mathcal{S}, Idle, Off \vdash \Diamond Off}}{\vdots}}{\cfrac{\cfrac{\mathcal{S}, x \doteq k, On, Count \vdash \Diamond Off}{On, Count, x' \doteq k, \circ(\mathcal{S} \wedge On \wedge Count) \vdash Off, \bullet\Diamond Off}\ sc - step}{On, Count, x' \doteq k, \circ(\mathcal{S} \wedge On \wedge Count) \vdash \Diamond Off}\ diamond - right'}$$

$$\vdots$$

$$\mathcal{S}, On, Count \vdash \Diamond Off$$

$$\vdots$$

$$\mathcal{S}, On \vdash \Diamond Off$$

Figure 2: A proof tree for $On \vdash \Diamond Off$

configuration where "Count" is active and the value of x' equals k[1]. The newly introduced rule $sc - step$ finally takes our statechart to the next configuration. Therefore leading next operators are removed, primed variables are replaced and unprimed variables are discarded:

$$\frac{\Gamma, \zeta_0, \ldots, \zeta_n \vdash \eta_0, \ldots, \eta_m, \Delta}{\Gamma', \circ\zeta_0, \ldots, \circ\zeta_n \vdash \bullet\eta_0, \ldots, \bullet\eta_m, \Delta'}\ sc - step \tag{4}$$

[1] This is achieved due to the very same construction, but we decided to concentrate on the last step.

Here, ζ_i and η_j may be the only temporal logic formulae. They must all be prefixed with a next operator, which can be done using temporal logic rules as given in Section 3. In contrast, Γ' and Δ' require to be pure predicate logic. Γ resp. Δ is derived by removing all prime symbols ($'$) and "discarding" (by applying skolemization) all unprimed variables (i.e. values which held during the last configuration).

Obviously, not every (provable) assuption can be verified in a finite number of steps. E.g. we want to prove that $\Box(x \leq 6)$ is derivable from the statechart. Since it lacks final states, we have to prove that the property holds for an infinite number of configurations. However, the initial configuration reappears within a finite number of steps. With some generalization of the sequent we now can easily close the proof by induction (or invariant method with decreasing variant).

5 Tool support

The construction method presented above has been successfully implemented into the KIV (Karlsruhe Interactive Verifier) system. KIV can be used for both interactive and automated deduction. It does not only feature sequent calculus for temporal logics such as ITL, but uses statecharts natively as data structures to operate on. KIV features a declarative language with hierarchical definition

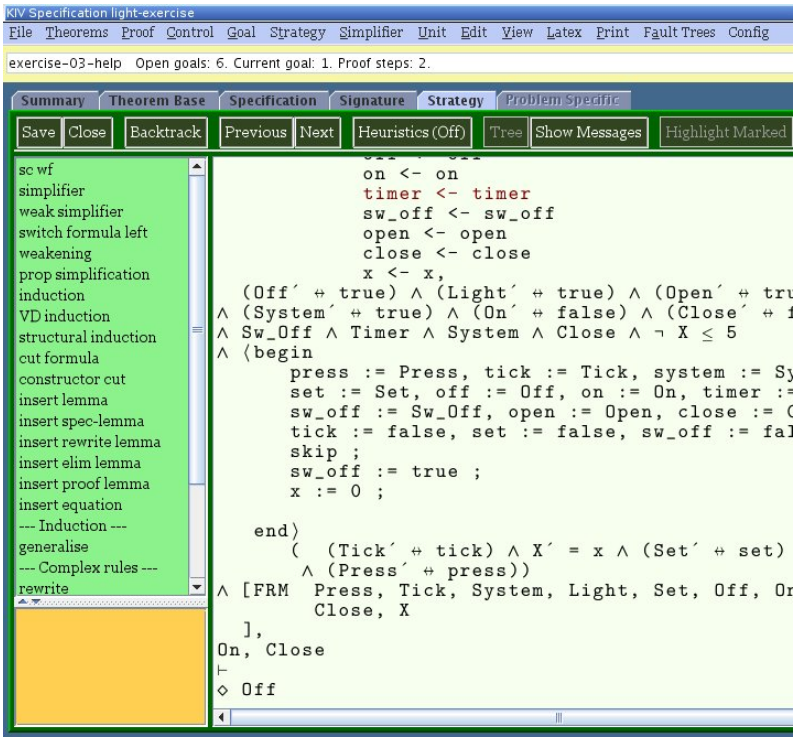

Figure 3: KIV screenshot (proving $On \vdash \Diamond Off$)

of states. Current development aims at graphical display and editing.

References

[BRS+08] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, Kurt Stenzel, Andreas Thums, Holger Grandy, Jonathan Schmitt, Dominik Haneberg, and Frank Ortmeier. A practical course on KIV. http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/, June 2008.

[CMZ08] Antonio Cau, Ben Moszkowski, and Hussein Zedan. Interval temporal logic. http://www.cse.dmu.ac.uk/STRL/ITL//itlhomepage.html, January 2008.

[HLN+90] D. Harel, H. Lachover, A. Naamad, Amir Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, SE-16(4):403–414, April 1990.

[TSOR04] Andreas Thums, Gerhard Schellhorn, Frank Ortmeier, and Wolfgang Reif. Interactive verification of statecharts. In H. Ehrig, editor, *Integration of Software Specification Techniques for Applications in Engineering*, pages 355 – 373. Springer LNCS 3147, 2004.