Static Specification Completeness Checking of UML State Machines

Zsigmond Pap

Safe programs never process such operations, that can cause accident, human casualty or environment harm. The design of safe programs is always verified according to the specification, but if the specification is incomplete or inconsistent, verification and validation can't find the possible implementation errors, which may result in accidents. Accordingly, the checking of the completeness and consistency of the specification is cru- cial.

There are numerous formal specification methods proposed in the literature. Generally they use a descrip- tion language that can be easily processed and checked by a computer. The specification check must be fast and efficient. This means, for example, that the checker should not build the full reachability tree correspond- ing to the specification, because this is a very slow process and requires a lot of resources. Our goal is to work out a static checking method and tool, which can check some aspects of the specification without gener- ating the reachability tree.

Our tool aims at checking specifications developed using UML, the Unified Modeling Language. Especially the formalism used to describe the dynamic behaviour of the system, i.e. the Statechart specification should be checked. Statecharts, as extension of state diagrams, consist of states and labelled transitions, but allow state hierarchy and concurrency. A state can be normal or complex. Complex states can be concurrent super- states, which result in a hierarchical tree of states. There are special (pseudo) states, like the initial state or the history state. The transitions are labelled by a trigger event, a guard condition and actions. When an event occurs, all transitions triggered by this event can fire if the state machine is in the specified source state, and the guarding condition is true. When a transition fires, the state machine goes to the destination state, and all actions associated with the transition start. There are special transitions. The destination state of a conditional transition depends on some conditions. The fork transition allows starting concurrent sub-automata, the join transition help to step into a single state from concurrent states. The completion transition has no trigger event, fires immediately when the guarding condition becomes true.

The completeness checking of the Statechart specification is based on the following rules:

- All possible trigger events should be considered when the transitions leaving a given state are examined. There must not be ambiguous, non-deterministic situations, when the state machine can reach different states and there is no priority between the corresponding transitions (triggered by the same event). How- ever, two transitions triggered with the same event can be unambiguous, if the guarding conditions can't be true at the same time.
- In a hierarchical model the sub-states inherit the transition from their parent states. To verify the com- pleteness, it is sufficient to check the leaf-states only, i.e. all non pseudo-states without sub-states. If at least two transitions have true guard conditions and can fire at the same time and on the same hierarchy level, the specification is ambiguous.
- The join and fork transitions can be transformed to normal transitions with special guard conditions. The conditional transitions need the following checking: the branch cases must form a tautology, and one of them must be true in any case.
- If there is a state without incoming transitions (except the initial pseudostate), then this state is not reach- able, which means a specification error.
- If in a given hierarchy level the guard conditions form a tautology, a transition on a higher level might be unnecessary.
- Each state of a program must be limited in time. Accordingly, from every stable state must start a transi- tion triggered by a time-out event. This transition fires when the maximum time in that state is exceeded.

Our checker was integrated with the commercial UML CASE tool "MID Innovator". From the repository of this tool a script can export the design to an Oracle database via ODBC. The checker program works on this database, which makes the navigation of the model easier. The specification completeness checking was implemented in Prolog. The rules mentioned above are expressed as Prolog questions. Our implementation has an interface toward SQL, the language used to handle the database. Accordingly, every Prolog question is converted into SQL commands. The interface sends these to the Oracle server, and converts the answers into Prolog predicates.

Up to now, the checker was successfully used to examine the completeness and consistency of several (small scale) UML designs.

****