Formal Models for Encapsulation, Structure and Hierarchy in Distributed Systems

m.c.w. geilen

e-mail: geilen@ics.ele.tue.nl

Abstract

Automated verification tools allow a designer to express wanted or unwanted behaviour of the system and search for satisfaction or violation of these behaviours. It is the state-space explosion problem that makes that verification tools are typically applied to small models and that careful modelling is required to keep the number of reachable states small. For this reason, current languages and logics often operate on a low level of abstraction, offering little structuring and abstraction concepts.

It is possible to use verification methods in other stages of the design process as well. In such cases, one would like to be able to use these logical properties in the same design framework, using such concepts as structuring and hierarchy, encapsulation, inheritance and so forth. A calculus supporting structural hierarchy and encapsulation is introduced with suitable equivalences. Furthermore a temporal logic is defined that fits to this calculus and it is shown how such logics can be used to specify properties of hierarchical systems.

The CCCS calculus

We define a calculus called CCCS (component based CCS), that describes a static structure of communicating and identifiable components.

The syntax of the calculus allows process expressions to be labelled with identifiers.

The semantics of the calculus describes a labelled transition system from which every component’s individual contribution can be deduced.

Temporal logic for components

We can define temporal logics that match the component based calculus and that are able to explicitly refer to behaviour of individual components. For example the formula

\[ X.p \equiv Y.q \Rightarrow Z.r \]

defines a property named p of component X in terms of the properties q and r of components Y and Z.

Thus this kind of logics provides a good framework for encapsulating a component’s inner structure. In the example properties of components Y and Z are referred to by a name (q and r respectively). Allowing these properties to be defined together with the components themselves.

Equivalences for CCCS

The transitions of our calculus carry information about the components that were involved in the transition.

If we define a standard bisimulation equivalence, this implies that an external observer is able to see everything happening inside the system. This might be undesired.

Observability can be limited by using an abstraction function that (partly) abstracts from component information, giving rise to an equivalence relation \( \approx \). If we abstract away all information about component \( f \), if-ACCS we arrive at the normal CCS equivalence. If we do not abstract the equivalence is approximately the location equivalence of [1]. A more abstract observability leads to a strictly weaker equivalence relation:

\[ f \leq g \Rightarrow \exists f \mathbin{\approx} g \]


Example

One can specify a component based version of Milner’s famous jobshop example. The entire system is a component called jobshop. Two jobbers, a hammer and a mallet can be identified as the components jobber1, jobber2, Hammer and Mallet.

Jobshop \( \equiv \) Jobshop,\( \Omega \) ( Hammer, isUsed \( \Rightarrow \) Q Hammer, isAvailable )

The above property of the Jobshop expresses that for any behaviour of the Jobshop component, it is always the case that whenever the Hammer is being used, it will eventually become available again. This can be expressed without knowing how to express the availability of the hammer.

Conclusions & future work

We have introduced a calculus CCCS and temporal logics for hierarchical systems. We have introduced syntax, semantics and equivalences for the calculus, possible temporal logics for components and we have shown how they can be used to specify properties of hierarchical systems. It has been shown that the logics offer a useful tool to deal with encapsulation of a component’s internals. The reduction interpretation leads to a compositional next operator in the case of linear time temporal logic.

Future work includes the extension of the syntax and semantics of POOSL with object identifiers. Furthermore the definition of a property language for POOSL is required, giving not only syntax for the temporal logic itself, but also for specifying the atomic propositions, tools need to be developed or extended for automatically analysing properties. The temporal logics should also be extended with real-time features in order to be useful for POOSL.