Model Checking at the Requirements Level

David N. Jansen
Thesis advisor: Roel Wieringa
Universiteit Twente, Faculteit der Informatica, Enschede, The Netherlands
{dnjansen,roelw}@cs.utwente.nl

Research abstract

1 Research hypothesis and proposed solution

Hypothesis We want to look at uses of model checking (or, in general, formal verification) in early stages of software development. Although formality and precision are not identical, formality can help developers expressing their thoughts more precisely. Model checking serves as an extra tool that helps in finding errors and omissions in a requirements specification, not only in an implementation. So, it augments the quality of the requirements specification.

However, complete formalisation is not useful, at least in the beginning of a development project. We have to deal with a combination of informal and formal parts of the specification to be written, and we can only check the formal part.

A single software development method never applies to all software problems, and many problems are solved by a combination of methods and techniques. Therefore, it is not sensible to devise just one method for the application of model checking at requirements level.

Solution We want to give directions and hints how to plug model checking into several methods and how to combine the informal and formal elements. We provide an example software tool for model checking. Developers write the formal parts of the requirement specification and of the desired properties using a tool with an interface to a model checker.

TRADE is a possible combination of informal methods. The article [23] combines TRADE with model checking. An example of a hint may be: it is more useful to consider both the system under development and its environment and to check a combined model of both. Section 3 gives more details on our contribution to the solution.
2 Prior research

The model checking algorithm has been described already in [10]: to check whether a finite-state system verifies a specification stated as CTL formula. Much more research on model checking has been done, e.g. for TCTL [3] and for ACTL [12].

Model checking is most often applied in later phases of software development, where the models to be checked are already quite low-level and almost fixed. See, e.g., the case study [16].

Examples of (partly) formal methods are: KAOS [11] defines a tree of goals to be achieved by a (planned) system. SCR [17] is a formal method based on tables for the specification of required behaviour of software systems. Cheng is working on the transition from informal to formal, too [7]. There are plenty of informal software development methods. For a survey of many structured and object-oriented methods, see [22].

3 Expected contributions

I have already done the first part of the work:

A simple logic language We have defined ATCTL, a combination of several extensions of CTL: actions (ACTL) and real-time (TCTL) [19]. This gives users the choice to express desired properties in the way they like.

We have defined a translation from ATCTL to TCTL, which allows to check ATCTL models against ATCTL formulae with a TCTL model checker, e.g. Kronos [5, 24].

TCM extension We have extended the drawing tool TCM [13] to serve as a front end to the model checker Kronos. The statecharts drawn with TCM are translated according to the semantics defined by Eshuis [15], which is a formalisation of the UML statechart semantics.

The following parts are still to be done:

Property patterns are schemas where the user may fill in a gap in a property which is otherwise stated in sensible natural language [14]. They can help the user in formulating the desired properties.

Jackson’s problem frames [18] give schemas for simple problems. For each problem frame, Jackson provides a method to handle it. (As most problems are not so simple, we will have to divide them into simple parts and apply several methods to the parts.)

Is there a relation between property patterns and problem frames? We think that there are typical properties one would like to check in a specific problem frame.

Methodic hints for integration of informal and formal specifications Especially in these early stages, many descriptions are still informal. We investigate which parts of the descriptions can be formalized and checked.

2
We want to provide hints on how to interface informal and formal parts in a requirements engineering method. It is not intended to give another method, but to show how to plug in model checking into existing methods.

**Case studies and examples** We are looking at some standard examples, e.g. the Production Cell controller [20] and the Steam Boiler controller [1]. The case studies have two goals: first, we may learn how to embed model checking into different methods. Second, they provide evidence that it is possible to combine existing methods with model checking.

## 4 Obstacles and further research

We wanted to rely on Kronos as a model checker because it is one of the few model checkers which are able to handle real-time properties, and the only one we know to understand TCTL. However, Kronos does not suffice: it only accepts models with up to 65536 locations. We want to handle also larger examples to provide evidence for our research results.

There are several possibilities for further research:

- There are other real-time model checkers: UPPAAL [2] and RED [21]. These model checkers only allow for reachability analysis and don’t include TCTL nor a similar logic. It is possible to check desired properties using an observer automaton.
- We also consider properties without real time. Many property patterns can be stated as pure temporal properties. Other model checkers (e.g., SMV [8, 9]) can handle these properties. Other interesting variants of CTL include probability [4] and deontic logic [6]. We can use these variants to express more specific patterns.

## References


