The Development Graph Manager MAYA

Serge Autiez ﬀ and Dieter Hutter 2 and Till Mossakowski 3 and Axel Schäfer 2

1 FR 6.2 Informatik, Saarland University, P.O. Box 15 11 50, D 66041 Saarbrücken,
Autiez@gis.uni-sb.de
2 DFKI GmbH, Stuhlsatzenhausweg 3, D 66123 Saarbrücken, hutter@dfki.de
3 FB 3, University of Bremen, P.O. Box 330 440, D 28334 Bremen, till@tzi.de

1 Overview

The use of formal methods is propagated to increase the security and safety of developed software. However, the logical formalization of software systems is error-prone. As even the verification of small-sized industrial developments require several person months, specification errors revealed in late verification phases pose an incalculable risk for the overall project costs. Thus, an evolutionary formal development approach is absolutely indispensable in all applications so far, as it was hardly ever the case that the development steps were correctly designed in the first attempt. The search for formally correct software and the corresponding proofs is more like a formal reflection of partial developments rather than just a way to assure and prove more or less evident facts.

The MAYA-system supports an evolutionary formal development as it allows the user to specify and verify his/her development and maintain and reuse the verification work already done when changing the specification. MAYA supports the use of various (structured) specification languages like CASL [CASL98] and VSE-SL [VSE96] to formalize the software development. Moreover, MAYA provides a generic interface to plug additional parsers for the support of other specification languages.

The textual specifications are translated into a structured logical representation called a development graph [AHMS00,Hut00], which is based on the notions of consequence relations and morphisms and makes arising proof obligations explicit. The user can tackle these proof obligations with the help of interconnected theorem provers like Isabelle [Pau94] or INKA [AHMS99].

The failure of proving one of these obligations usually gives rise to modifications of the underlying specification (see Figure 1). MAYA supports
this evolutionary process as it calculates minimal changes to the logical representation to readjust it to a modified specification while preserving as much verification work as possible. If necessary it also adjusts the database of the interconnected theorem prover. Furthermore, MAYA hands on explicit information how the axiomatisation has changed and provides also available (due to the changes invalidated) proofs of the same problem to allow for a re-use of proofs inside the theorem provers. Information about a proof provided by the theorem provers is used to optimize the maintenance of the proof during the evolutionary development process.

2 From Textual to Logical Representation

The specification of a formal development in MAYA is always done in a textual way using specification languages like CASL or VSE-SL. MAYA comprises parsers to translate such specification into the MAYA-internal specification language DGRL (“Development Graph Representation Language”). DGRL provides a simply-typed lambda calculus to specify the local axiomatisation of a theory in a higher order logic. While unstructured specifications are solely represented as signature together with a set of logical formulas, the structuring operations of the specification languages (like for instance then, and, or with in CASL) are translated into the structure of a development graph. Each node of this graph corresponds to a theory. The axiomatisation of this theory is split into a local part which is attached to the node as a set of higher-order formulas and into global parts, denoted by ingoing definition links, which import the axiomatisation of other nodes via some consequence morphisms. While a so-called local link imports only the local part of the axiomatization of the source node of a link, global links are used to import the entire axiomatization of a source node (including all the imported axiomatizations of other nodes). In the same way local and global theorem links are used to postulate corresponding relations between nodes (see [AHMS00] for details).

Changes of specifications are done inside the textual representation. Parsing a modified specification results in a modified DGRL-specification. In order to support a management of change, MAYA computes the differences of both DGRL-specifications and compiles them into a sequence of basic operations in order to transform the development graph corresponding to the original DGRL-

Fig. 2. Systemarchitecture of MAYA
specification to a new one corresponding to the modified DGRL-specification. Examples of such basic operations are the insertion or deletion of a node or a link, the change of the annotated morphism of a link, or the change of the local axiomatisation of a node. As there is no optimal solution to the problem of computing differences between two specifications, MAYA uses heuristics based on names and types of individual objects to guide the process of mapping corresponding parts of old and new specification.

Since the differences of two specifications are computed on the basis of the internal DGRL-representation, new specification languages can be easily incorporated into MAYA by providing a parser from this language to DGRL.

3 Maintaining the Development Graph

The development graph is the central datastructure to store and maintain the formal (structured) specification, the arising proof obligations and the status of the corresponding verification effort (proofs) during a formal development. The development graph is always synthesized or manipulated with the help of the previously mentioned basic operations (insertion / deletion / change of nodes / links / axiomatisation) and MAYA incorporates sophisticated techniques how these operations will affect proof obligations or proofs stored within the development graph. They incorporate a notion of monotonicity of theories and morphisms, they take into account the sequence in which objects are inserted into the development graph and they make use of the knowledge of the proofs provided by the interconnected theorem provers.

MAYA distinguishes between proof obligations postulating properties between different theories, like the notion of \texttt{view} in \texttt{CASL} or \texttt{satisfies} in \texttt{VSE-SL}, and lemmata speculated within a single theory, e.g. with the \texttt{implies} annotation in \texttt{CASL}. As theories correspond to subgraphs within the development graph, a relation between different theories, represented by a global theorem link, corresponds to a relation between different two subgraphs. Each change of these subgraphs can affect this relation and would invalidate previous proofs of this relation. Therefore, MAYA decomposes relations between different theories to individual relations between the local axiomatization of a node and a theory (denoted by a local theorem link). Each of these relations decomposes again into a set of proof obligations postulating that each local axiom of the node is a theorem in the target theory wrt. the morphism attached to the link. These proof obligations are tackled by the interconnected theorem provers. Keeping track of these different decompositions, MAYA is able to adjust them once the development graph has been altered.

While definition links establish relations between theories, theorem links denote postulated lemmata about such relations. Thus, the reachability between two nodes establishes a formal relation between the connected nodes (i.e. the theory of the source node is part of the theory of the target node wrt. the morphisms attached to the connecting links). MAYA uses this property to prove relations between theories also by searching for paths between the correspond-
ing nodes (instead of decomposing the corresponding proof obligation in the first place).

Proof obligations which have to be tackled by interconnected theorem provers occur when verifying a local theorem link or proving a speculated lemma. In both cases the proofs are done within a specific theory. Thus, conceptually each theory may include its own theorem prover. In practice MAYA makes use of generic theorem provers which are fed with the axiomatization of the current theory. Switching between different proof obligations may cause a change of the underlying current theory and thus a change of the underlying axiomatization. MAYA provides a generic interface to plug-in theorem provers (based on a XML-RPC protocol) which allows for an incremental update of the database of the prover.

4 Conclusion

The MAYA-system is implemented mostly in Common Lisp while parts of the GUI, which are common to the OMEGA-system [], are written in Mozart. The CASL-parser is provided by the CoFI-group in Bremen. The MAYA-system is freely available from the MAYA-webpage at www.dfki.de/inka/maya.html.

Future extensions of the system will include a treatment of hiding [MAH01], a uniform treatment of different logics based on the notion of heterogenous development graphs [Mos01], and the maintenance of theory-specific control information for theorem provers (like tactics or proof plans).

References

[Mos01] T. Mossakowski. Heterogeneous development graphs and heterogeneous borrowing. Submitted