Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 139–145
www.elsevier.com/locate/entcs

Proof Transformation via Interpretation Functions: Results, Problems and Applications

Piotr Kosiuczenko

Department of Computer Science
University of Leicester
Leicester, UK

Abstract

Change is a constant factor in Software Engineering process. Redesign of a class structure requires transformation of the corresponding OCL constraints. In a previous paper we have shown how to use, what we call, interpretation functions for transformation of constraints. In this paper we discuss recently obtained results concerning proof transformations via such functions. In particular we detail the fact that they preserve proofs in equational logic, as well as proofs in other logical systems like propositional logic with modus ponens or proofs using resolution rule. Those results have direct applications to redesign of UML State Machines and Sequence Diagrams. If states in a State Machine are interpreted by State Invariants, then the topological relations between its states can be interpreted as logical relations between the corresponding formulas. Preservation of the consequence relation can bee seen as preservation of the topology of State Machines. We indicate also an unsolved problem and discuss the mining of its positive solution.

Keywords: UML, Redesign, Proof Transformation, Constraint Transformation, State Machines, Sequence Diagrams, Formal Methods.

1 Introduction

Unified Modelling Language provides textual and diagrammatic means for system specification (cf. [13]). Systems and their real-world environments are modelled using abstractions such as Classes Diagrams, State Machines or Sequence Diagrams. There are different software engineering processes which

[1] This research was partially funded by the EU project Leg2Net and EU project AGILE.

doi:10.1016/j.entcs.2004.08.040

can be applied. In the old fashioned Waterfall Model, one has to begin with a correct requirement specification, make refinements to obtain the design and then implement the design specification. These steps can be adequately described using the notion of refinement (cf. e.g. [8]). This works correctly, if the requirements do not change and the software developers have a clear idea regarding how to proceed. In practice however, a specification constantly changes due to a number of factors including changed or new client requirements, new technology enablers and so on. In such a case extensive manual reengineering of system specification and design is needed. Today's software engineering processes embrace change as being a constant factor. In this case, requirements tracing is much harder to achieve. The notion of refinement with its monotonicity assumption can barely model such changes. For example, if an interface or class signature changes, a formula or a constraint which described a property concerning classes implementing this interface may no longer make sense. No tool in the market allows one for an automatic transformation of constraints. Changes have to be made to the specification manually, which is very time consuming and error prone.

A number of approaches to redesign of UML class models exist already. The best known is the refactoring approach [1]. It provides simple patterns for code and class structure modification to extend and modify a system without altering its behavior. The interpretation function, used in abstract algebra [11], transforms a single operation into a complex term. Graph rewriting systems may be used to transform specifications (cf. e.g. [3]).

In a previous paper [8] (see also [5]) we have studied the redesign of UML State Invariants with OCL constraints, as well as the transformation and tracing of constraints. We have introduced a new notion of interpretation function for redesign of State Invariants which extends, in a natural way, the notion introduced in [11]; an interpretation function is a compositional function generated by a mapping satisfying conditions analogous to orthogonality in term rewriting systems. Our concept of redesign is more general than the concept of refinement since we do not assume that properties are only added or refined, but we allow for changing them in an arbitrary way. For example a number of design level classes might be restructured or a specification level class might be split into several design level classes. Properties, which have to be preserved, may concern dependencies between classes, associations, attributes or generalization relationships. They are expressed by OCL formulas and transformed. Our approach is motivated by the notion of abstraction as it is used in UML [13]. Interestingly, our approach allows us for not only an automatic constraint transformation but also for an automatic proof transformation. In the technical report [6], we have shown that several kinds of entailment re-

lations are preserved by interpretation functions; in particular such functions preserve equational proofs, proofs using propositional tautologies, resolution rule and induction. This allows one to save the clerical work of redoing proofs after transformation of a Class Diagram.

In this paper we present briefly the results contained in the technical report [5] and discuss their applications to redesign of State Machines and Sequence Diagrams. In section 2, we present briefly the idea of interpretation functions and the acompanying results. In section 3, we show how this idea and results can be applied to redesign of State Machines and Sequence Diagrams. In section 4, we conclude this paper and list some open problems.

2 Interpretation Functions

Interpretation functions proved to be very useful as a vehicle for an automatic transformation of OCL constraints when changes to Class Diagrams are performed [6] (see also [4]). For example, if an attribute a of type Integer is replaced by a path $b.x$, where b is an association pointing to another class and x is an attribute of that class, then every OCL constraint or State Invariant containing a has to be modified.

This kind of modifications can be performed using interpretation functions, i.e. partial functions generated by mappings satisfying conditions analogous to orthogonal term rewriting systems (cf. e.g. [12]). A domain of such a mapping satisfies conditions valid for all domains of orthogonal term rewriting systems; i.e. all terms in the domain are linear and non-overlapping.

Interpretation functions have several useful properties. They allow us to transform OCL specifications. The idea is that the designer or implementer who changes a class structure maps modified Model Elements on the target Model Elements, the transformation of the corresponding constraints being accomplished automatically. Such functions preserve equational proofs, proofs using propositional tautologies, resolution rule and proofs by induction [5]. This allows one to save the clerical work of redoing proofs of this kind after transformation of Class Diagrams.

3 Applications

Interestingly, results concerning preservation of consequence relation have also implications for redesign of other UML diagrams. In the following three subsections we consider application of our concepts to State Machines and Sequence Diagrams.

3.1 State Machines

One of the most useful kind of UML diagrams are the so called State Machines [13]. A State Machine is composed of a number of states connected by edges corresponding to transitions. States can be structured; one state can contain several other states called substates. In UML, "a state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event" [13]. Consequently States in a State Machine correspond to formulas; in the first case the formulas are called State Invariants. Such formulas may describe values of object attributes and inter-relationship between different objects; they can be expressed for example in OCL. States of a State Machine can be represented by trees, in particular by propositional formulas (cf. [9]). We argue that topological relation between states in a State Machine can be interpreted as logical relations between the corresponding formulas. In this case, preservation of the entailment relation can be seen as preservation of the topology of State Machines. It is natural to assume that the invariant corresponding to a substate implies the invariant corresponding to its superstate, since the invariant corresponding to the substate should be more restrictive. This condition (we call it state monotonicity) can be formally expressed as follows:
For every two states s_1 and s_2, s_1 is a substate of s_2 if and only if the formula corresponding to s_1 implies the formula corresponding to s_2.

On the other hand, one can be interested, if the states of a State Machine cover all possibilities. In the case of a State Machines describing the behavior of an object, this means that for every combination of the objects attributes, there is a state covering this combination. When states are interpreted by invariants, this covering property can be equivalently expressed by the requirement that for every combination of attributes, there is a State Invariant describing this combination. Formally, let F_i for $i = 1, ..., n$, be formulas corresponding to all states of a State Machine M; the states of M cover all possibilities if and only if the formula $F_1 \vee ... \vee F_n$ is a tautology. Equivalently, $F_1 \vee ... \vee F_n$ can be proved without using domain specific formulas.

A stronger property (we call it exhaustiveness) says that for every, so called, or-state all its substates cover all possibilities. Formally, let s be an or-state, let F be the corresponding State Invariant, let s_1,..., s_n be all substates of s and let $F_1, ..., F_n$ be the corresponding invariants. F is exhaustive if and only if F is logically equivalent to the disjunction $F_1 \vee ... \vee F_n$. We say that states in a State Machine are non-overlapping, if for every two different substates s_1 and s_2 of an or-state s, the conjunction of the corresponding invariants $F_1 \wedge F_2$ is logically false.

There are several other useful properties of this kind which can be expressed by logical relations between formulas. For example that all reachable states are defined by non-contradictory formulas or that disjunctions of formulas corresponding to orthogonal states are equivalent to the superstate invariant (a condition analogous to exhaustiveness).

As explained above, transformation of a Class Diagram requires the transformation of the corresponding constrains (for example OCL constraints). Consequently, if the states of a state Machine are described by formulas, those formulas may need to be changed. If implication in first order logic is preserved by interpretation functions, then all properties described above are preserved by such functions. The results presented in [5] show, that interpretation functions preserve proofs using equational reasoning, resolution rule, propositional tautologies and induction. Consequently at the moment we can say that if the above mentioned properties are proved using those kinds of reasoning, then they are preserved by interpretation functions. If for example, there is a proof of the state-monotonicity property using above mentioned ways of reasoning, then after transformation via an interpretation function this property remains valid after transformation.

3.2 Sequence Diagrams

UML 2.0 introduces conditions to Sequence Diagrams (SD) [14]. They do not play such a central role as state invariants in State Machines. On the other hand, the structuring mechanisms in SD are of different kind; these are operations on sets of traces to facilitate behavior specification. The previous section contains a list of State Machines properties which can be defined in logical terms. In a similar way one can define properties of Sequence Diagrams. In this case, those constraints concern composition of trace specifications. For example, one can require that conditions in a Sequence Diagram, which is obtained using parallel composition, are non-contradictory or that Sequence Diagrams combined by alternative composition have exclusive pre-conditions. As in the case of State Machines, the preservation of logical consequences implies that such properties are preserved by interpretation functions.

4 Concluding Remarks and Future Work

In this paper we show that results presented in [6] and in [5] have direct applications to transformation of UML diagrams, in particular to State Machines and Sequence Diagrams. We show that logical relations between State Invariants in a State Machine are preserved, if they can be proved using certain kinds of proofs.

There are still several open questions. In particular, it is not known, if interpretation functions preserve entailment relation of first order logic. This question may seem purely theoretical, but a positive answer would have very interesting implications for State Machines and Sequence diagrams; it would mean, for example, that interpretation functions preserve topology of State Machines. Even if the answer is negative, the results obtained so far prove to be useful.

In the future, we are going to further study the properties of interpretation functions in logical terms. We are going to investigate, if the entailment relation of first order logic is preserved by interpretation functions. In first order logic there is an equivalence between semantic and syntactic consequence. Our results obtained up to now focus mainly on the syntax. Institutions approach the problem of transformation from the model theoretic point of view (cf. e.g. [10]). We are going to study the relation of interpretation functions to institutions. We plan also to implement a tool supporting class redesign, transformation and tracing of model elements. Such a tool will be very helpful since a purely manual transformation of complex OCL constraints is very laborious and error prone.

References

[1] Fowler, M., *Refactoring: Improving the Design of Existing Code*, Reading, Mass., Addison-Wesley, 2000.

[2] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, *Design Patterns*, Addison- Wesley, Reading, 1995.

[3] Grosse-Rhode, M., and Presicce, F., and M. Simeoni, *Formal software specification with refinements and modules of typed graph transformation systems*, Journal of Computer and System Sciences, **64(2)**, 2002.

[4] Kosiuczenko, P., *Formal Redesign of UML Class Diagrams*, in (Evans, A., France, R., Moreira, A., Rumpe, B. Eds): Proc. of pUML Workshop on Practical UML Based Rigorous Development Methods, Toronto. GI-Edition, Lecture Notes in Informatics, 2001.

[5] Kosiuczenko, P., *Proof Transformation via Interpretation Functions*, Technical Report nr. 2004/27, University of Leicester, 2004, 16 pages: http://www.cs.le.ac.uk/~pk82/Theory1.1.pdf.

[6] Kosiuczenko, P., *Redesign of UML Class Diagrams: a formal Approach*, submitted for publication, 2004.

[7] Kosiuczenko, P., *Proof Transformation via Interpretation Functions: Results, Problems and Applications*, to appear in ENTCS, 2004.

[8] Lano, K., *Formal object oriented development*, Springer, Berlin, 1995.

[9] Lütgen, G., and M. Mendler, *Statecharts: From Visual Syntax to Model-Theoretic Semantics*, in proc. of Wirtschaft und Wissenschaft in der Network Economy, Tagungsband der GI/OCG-Jahrestagung, K. Bauknecht, W. Brauer, Th. Mück (eds.), Viena, 25.09.2001.

[10] Tarlecki, A., *Institution: An Abstract Framework for Formal Specifications*, In (Astesiano, E., Kreowski, H. -J., Krieg-Brückner, B. Eds.): Algebraic Foundations of System Specification, Springer, 1999.

[11] Taylor, W., *Characterizing Malcev conditions*, Algebra Universalis, **3**, Springer, Berlin, 1973, 351–397.

[12] Terese et. al., *Term rewriting systems*, Cambridge University Press, 2003.

[13] OMG, *Unified Modeling Language Specification*, Version 1.5, 2003.

[14] OMG, *Unified Modeling Language Specification*, Version 2.0 (pending), 2004.