Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 211 (2008) 191–200
www.elsevier.com/locate/entcs

Towards Verifying Model Transformations

Anantha Narayanan[1,2] and Gabor Karsai[3]

Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37235, USA

Abstract

In model-based software development, a complete design and analysis process involves designing the system using the design language, converting it into the analysis language, and performing the verification and analysis on the analysis model. Graph transformation is increasingly being used to automate this conversion. In such a scenario, it is very important that the conversion preserves the semantics of the design model. This paper discusses an approach to verify this semantic equivalence for each transformation. We will show how to check whether a particular transformation resulted in an output model that preserves the semantics of the input model with respect to a particular property.

Keywords: Graph Transformation, Verification, Bisimulation.

1 Introduction

Domain specific modeling languages (DSMLs) greatly simplify the task of the system designer, presenting a higher level of abstraction that is easy to work with. DSMLs also facilitate analysis by providing an appropriate abstraction. However, it is not always the case that the same language is suitable for both design and analysis. For instance, Statecharts are very powerful for designing concurrent systems, but their analysis is usually not simple. Extended Hybrid Automata (EHA) were introduced in [3] as an intermediate, simpler language with a more restricted syntax. Subsequent work [4] has shown that this intermediate format can be used to generate verification models that may be verified using model checking tools such as SPIN [5].

Graph transformation has been suggested as a powerful and convenient method for transforming design models into analysis models. The transformation must

[1] The research described in this paper has been supported by a grant from NSF/CSR-EHS, titled "Software Composition for Embedded Systems using Graph Transformations", award number CNS-0509098.

[2] Email: ananth@isis.vanderbilt.edu

[3] Email: gabor.karsai@vanderbilt.edu

doi:10.1016/j.entcs.2008.04.041

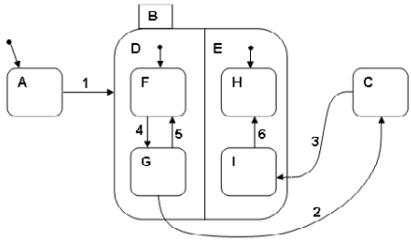


Fig. 1. A sample Statechart model

ensure that the analysis model preserves the semantics of the design model, and truthfully represents the design. As a first step towards this goal, it would be useful to establish that the transformed model is semantically equivalent to the source model, with respect to the property we wish to verify. In this paper, we study this notion of equivalence between the two graphs, and a way to check if there exists a bisimulation relation between the graphs. If it is possible to prove that the analysis model behaves in exactly the same way as the design model with respect to a certain property, then we can conclude that checking for the property in the analysis model is equivalent to checking for the same property in the original design model. In the following sections, we will go through the basics of graph transformation principles and tools, and demonstrate our approach to checking the equivalence using Statechart models and EHA models.

2 Background

2.1 Model Integrated Computing

Model Integrated Computing (MIC) [1] is an approach to system development using domain specific models to represent the architecture and behavior of the system and its environment. The development process involves the creation of a meta-model that defines the abstract syntax of the domain, from which a Domain Specific Design Environment (DSDE) is generated. The DSDE can be used to create domain specific models. These models are usually transformed to other formats, such as executable code, or to perform analysis. The MIC tool suite containing GME [6] and GReAT [7] were used in developing the examples for this paper.

2.2 GReAT

The transformations in this paper will be written in GReAT [7], a language for specifying graph transformation rules. GReAT belongs to the class of practical graph transformation systems such as AGG [8], PROGRES [9] and FUJABA [10]. It uses UML and OCL to specify the domains of the transformation.

GReAT allows users to compose source and target meta-models by defining temporary vertex and edge types that can span across multiple domains and will be used temporarily during the transformation. This enables us to tie the different domains together to make a larger, heterogeneous domain that encompasses all the

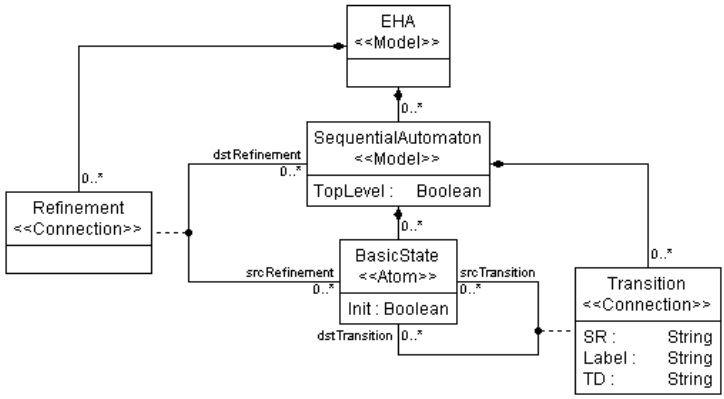


Fig. 2. EHA meta-model in UML

domains and cross-links. This feature plays an important role in our approach to verifying transformations.

2.3 Statecharts

State machines, based on Harel's statecharts [11] are used in UML to represent the reactive behavior of systems. State machines are constructed from *states* and *transitions*. States may be simple, composite or concurrent. States may be connected by directed edges called transitions. Transitions connecting states contained in different levels of hierarchy are called *inter-level* transitions. Figure 1 shows an example of a Statechart. Transitions 2 and 3 in the figure are inter-level. A *state configuration* is a maximal set of states that the system can be active in simultaneously. State configurations are closed upwards, meaning that if a system is in a state A, then it must also be in A's parent state. Some valid state configurations in Figure 1 are {A}, {B, F, H} and {B, G, I}.

2.4 EHA

Extended Hierarchical Automata (EHA) were introduced as an alternate representation to provide formal operational semantics for Statechart diagrams. EHA offer an alternative simplified hierarchical representation for Statecharts that helps in correctness proofs [2]. The meta-model for EHA in UML is shown in Figure 2.

Each Statechart model can be represented by one EHA model. Every compound state in the Statechart model is represented by a Sequential Automaton in the EHA. There is one top level Sequential Automaton for the EHA, which represents the initial automaton. Each state in the Statechart has a corresponding Basic State in the EHA. If a state is compound in the Statechart, then it is further "refined" into a Sequential Automaton in the EHA, which will contain Basic States corresponding to all the states within the compound state in the Statechart. Similarly, these states may be refined further.

Transitions in EHA are always within a single Sequential Automaton, i.e. there

are no inter-level transitions in an EHA. Inter-level transitions in Statecharts are elevated based on the scope of the transition, to the Sequential Automaton representing the lowest common ancestor of the start and end states of the transition in the Statechart. EHA transitions have special attributes called "source restriction" and "target determinator", which keep track of the actual source and target of the transition in the Statechart. The conversion of Statechart models into EHA models will be discussed in detail in the next section.

3 Verifying graph transformations

Graph transformation systems such as GReAT allow users to transform models of one meta-model to models of another meta-model using a collection of pattern matching rules. However, it is not certain whether the output of the transformation preserves the semantics of the source model that we intend to analyze. Important semantic information may easily be lost or misinterpreted in a complex transformation, due to errors in the graph rewriting rules or in the processing of the transformation. We need a method to verify that the semantics that we are interested in analyzing are indeed preserved across the transformation.

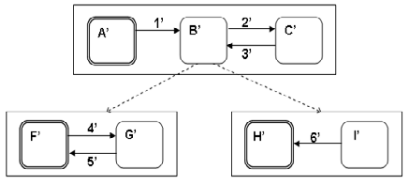
We propose an approach to check whether the semantics of the input model were preserved in the output model of a transformation. We are *not* trying to prove the correctness of the graph transformation rules in general, but check if a *particular* generated model is a valid representation of a *particular* source model, in order to verify a particular property about the source model. We accomplish this by defining an equivalence relation between objects of the input and the output model, and use this to check if the two models are similar in behavior.

3.1 Bisimilarity

Two systems can be said to be *bisimilar* if they behave in the same way, i.e. one system simulates the other and vice-versa. A bisimulation relation can be defined formally as follows.

Definition 3.1 Given a labeled state transition system (S, Λ, $\rightarrow$), a *bisimulation relation* is defined as an equivalence relation R over S, such that for all p, $q \in$ S, if (p, q) is in R, and for all $p' \in$ S and $\alpha \in \Lambda$, $p \rightarrow^{\alpha} p'$ implies that there exists a $q' \in$ S such that $q \rightarrow^{\alpha} q'$ and (p', q') is in R, and conversely, for all $q' \in$ S, $q \rightarrow^{\alpha} q'$ implies $p \rightarrow^{\alpha} p'$ and (p', q') is in R.

Though this definition is given in terms of a single set S, we can think of equivalence of two transition systems in terms of a global set containing both the system's states. In our approach to verifying whether the semantics are preserved across a transformation, we will check whether there is a bisimulation relation between the source model and the target model.



Transition Label	SR	TD
2'	G'	
3'		I', F'

Fig. 3. Sample EHA model

3.2 Transforming Statecharts into EHA

The EHA notation for Statecharts can be obtained by a graph transformation process [2]. The basic steps of the transformation are listed below:

(i) Every Statechart model can be transformed into an EHA model, with one top level Sequential Automaton in the EHA model.

(ii) For every (primitive or compound) state in the Statechart (except for regions of concurrent states), a corresponding basic state is created in the EHA.

(iii) For every composite state in the Statechart model, a Sequential Automaton is created in the EHA model, and a "refinement" link connects the Basic State in the EHA corresponding to the state in the Statechart, to the Sequential Automaton in the EHA that it is refined to.

(iv) All the contained states in the composite state are further transformed by repeating steps (ii) and (iii). The top level states in the Statechart will go into the top level Sequential Automaton in the EHA.

(v) For every non-interlevel transition in the Statechart model a transition is created in the EHA between the Basic States corresponding to the start and end states of the transition in the Statechart model.

(vi) For every inter-level transition in the Statechart model, we trace the scope of the transition to find the lowest parent state s_P that contains both the source and the target of the transition. A transition is created in the EHA, in the Sequential Automaton corresponding to s_P. The source of the transition in the EHA is the Basic State corresponding to the highest parent of the source in the Statechart that is within s_P, and the target in the EHA is the Basic State corresponding to the highest parent of the target in the Statechart that is within s_P. The transition in the EHA is further annotated, with the "source restriction" attribute set to the basic state corresponding to the actual source in the Statechart, and the "target determinator" set to the basic state corresponding to the actual target in the Statechart.

Figure 3 shows the EHA model obtained by transforming the Statechart model shown in Figure 1. The table on the top right of the figure shows the values for the *source restriction* and *target determinator* annotations for two of the transitions.

3.3 Behavioral equivalence of the Statechart model and the EHA model with respect to reachability

A "state configuration" in a Statechart is a valid set of states that the system can be active in. If a state is part of an active configuration, then all its parents are also part of the active configuration. A transition in the Statechart can take the system from one state configuration to another state configuration, where the source and target states of the transition are subsets of the initial and final state configurations. A state configuration S_f in the Statechart is said to be "reachable" from a state configuration S_i if there exists a series of valid transitions that can take the system from S_i to S_f.

Similarly, a state configuration in an EHA model is a set of Basic States. If a Basic State is part of an active configuration, and is part of a non-toplevel Sequential Automaton, then the Basic State that is refined into this Sequential Automaton is also a part of the active configuration. For instance, B', F', I' is a valid active configuration in Figure 3. A transition in the EHA can take the system from one state configuration to another state configuration, where the union of the source of the transition and its source restriction are a subset of the initial state configuration, and the union of the target of the transition and its target determinator are a subset of the final state. A state configuration S_f in the EHA is said to be "reachable" from a state configuration S_i if there exists a series of valid transitions that can take the system from S_i to S_f.

An EHA model truly represents the reachability behavior of a Statechart model, if every reachable state configuration in the Statechart has an equivalent reachable state configuration in the EHA and vice versa.

For every state s in the Statechart, we have a unique Basic State s' in the EHA. We can specify an equivalence relation R, such that $(s, s') \in R$ and say that s' is equivalent to s. A state configuration S in the Statechart is equivalent to a state configuration S' in the EHA if for all $s \in S$ there is an equivalent $s' \in S'$, and for all $s' \in S'$, there is an equivalent $s \in S$. Furthermore, for every transition t in the Statechart, we have a unique transition t' in the EHA. We can specify an equivalence relation R_t, such that $(t, t') \in R_t$ and say that t' is equivalent to t.

Given the relations R and R_t, we can check if there is a bisimulation relation between the two models using the following definition.

Definition 3.2 Given a state configuration S_A in the Statechart model, and its equivalent state configuration S_B in the EHA model, the equivalence is a bisimulation if for each transition t from S_A to a state configuration S_A' in the Statechart, there exists an equivalent transition t' in the EHA from S_B to a state configuration S_B', and S_B' is equivalent to S_A' (and vice versa)

If this relation is a bisimulation, then verifying the EHA model for reachability will be equivalent to verifying the Statechart model for reachability. If the check fails, it means that there was an error in the transformation.

3.4 Checking for bisimilarity by using cross-links to trace equivalence

GReAT allows us to link input model elements to target model elements using special associations that belong to a composite meta-model, and we call them *cross-links*. These cross-links are maintained throughout the transformation, and used to trace the equivalence relations R and R_t.

When a transformation creates the Basic States and the transitions in the target EHA model, it is known to which states and transitions they correspond to in the Statechart model. What is not certain is whether all states in the Statechart are transformed correctly, all composite states are refined correctly, all transitions are transformed correctly, and all transitions connect the correct sets of states. When a rule matches a state or a transition in the Statechart and creates the equivalent Basic State or transition in the EHA, a cross-link association called "equivalentTo" is created between the Statechart object and its corresponding EHA object. When the transformation completes, the relations R and R_t can be traced using these associations.

Rather than checking for all possible state configurations in the Statechart, it would be more efficient to consider every transition in the Statechart and its minimal required source configuration. Any superset of this state configuration will be a valid starting configuration, and will not have to be investigated further. For every transition t in the Statechart model, and its equivalent transition t' in the EHA model, if their start state configurations S_A and S_B are equivalent, and also their end state configurations S_A' and S_B' are equivalent, then there exists a bisimulation for this particular instance, according to our definition.

The implementation follows straightforwardly from the discussion. At the end of the transformation, we have access to the source model graph, the output model graph, and also the cross-links between the two. We collect the set of all the transitions from the source graph. For each transition in this set, we find the equivalent transition in the EHA by following the "equivalentTo" cross-link. Now we can compute the minimal source state configuration S_A for the transition in the Statechart model, and the source state configuration S_B for the EHA model. We check the equivalence of S_A and S_B by taking every state s in S_A, finding its equivalent state s' form the EHA, and checking if s' is in S_B, and vice versa. The target states are also checked similarly. If this check succeeds for all transitions in the Statechart, and there are no more transitions in the EHA, then the two systems can be said to be bisimilar with respect to checking reachability. In other words, we can conclude reachability in the Statechart model by verifying it in the EHA model. If this check fails, then there may be errors in the transformation, and the generated EHA model does not truly represent the input Statechart model.

The final step is checking the reachability in the EHA model. [4] provide ways to generate a Promela model from an EHA model, which can be checked using the SPIN model checker. To check the reachability of a certain state configuration, a claim can be attached to the SPIN model that verifies whether that configuration is reachable in the model. Alternately, a claim can be made in SPIN that says that the state is not reachable. If it is indeed reachable, the SPIN verifier refutes this

claim and presents a counter-example, as a trace that leads to this state configuration. This represents a valid series of transitions in the EHA that leads to the specified state configuration. As a corollary, we may use the cross-links created during the transformation, to reproduce this trace in the Statechart model. In this way, reachability in the Statechart model can be verified by verifying it in the EHA model.

It should be noted that the technique described above is not an attempt to prove the correctness of the graph transformation rules in general. This is a method to verify if a particular instance of a transformation is valid, and must be executed for each transformation individually. We also do not try to prove the general semantic equivalence of models. We identify the equivalence relations with respect to a specific property and test if there is a bisimulation. The complexity of the transformation is not increased significantly by this method. As the cross-links are created every time the objects of the output model are created, and as we directly trace these cross-links during checking, the complexity of the check is proportional to the size of the model, and not the state space of the model. In other words, we can perform this check without actually having to execute the models.

4 Related work

We now discuss some related work in the area of automatic verification using model checking, graph transformations and other types of proofs.

4.1 Verifying properties by converting models into an intermediate format

[2] [3] convert Statechart models into EHA models. [4] create Promela models from the EHA models, which can be verified using the SPIN model checker. Our approach will be useful in these instances, to provide a certificate that the intermediate formats truly preserve the property we wish to verify using them. An interesting research problem is whether our approach can be used to check whether the generated Promela model (which is code in plain text) truly represents the EHA model it was generated from.

4.2 Operational semantics using graph transformations

[12] [13] [14] are some works on using graph transformation rules to specify the dynamic behavior of systems. [14] presents a meta-level analysis technique where the semantics of a modeling language are defined using graph transformation rules. A transition system is generated for each instance model, which can be verified using a model checker. [15] verifies if a transformation preserves certain dynamic consistency properties by model checking the source and target models for properties p and q, where property p in the source language is transformed into property q in the target language. This transformation requires validation by a human expert. Our method does not check whether the models themselves satisfy a property, but automatically does check whether the models are equivalent with respect to that

property.

4.3 Certifiable program generation

[16] considers the problem of verification of generated code by focusing on each individual generated program, instead of verifying the program generator itself. The generator is extended such that it produces all logical annotations that are required for formal safety proofs in a Hoare-style framework. These proofs certify that the program does not violate certain conditions during its execution. While the proofs in this case are not related to semantic correctness, the idea of providing an instance level certificate of correctness instead of proving the correctness of the generator has been a great motivation for our ideas.

5 Summary

We have described a method for checking if a certain execution of a transformation produced an output model that preserved the semantics of the input model. This check is important when the output model is used for verification and analysis, as errors in the transformation may result in an output model that does not truly represent the input model. We are studying how such an equivalence can be established when the target model abstracts away a lot of detail in the source model. Our method does not attempt to prove the correctness of the transformation itself, but checks whether a particular execution produced a correct result. This check does not adversely affect the complexity of the transformation.

References

[1] Sztipanovits J., Karsai G., "Model-Integrated Computing", *IEEE Computer*, pp. 110-112, April, 1997.

[2] Varró D. "A Formal Semantics of UML Statecharts by Model Transition Systems", Proc. ICGT 2002: 1st International Conference on Graph Transformation, *LNCS*, vol. 2505, pp. 378-392.

[3] Mikk E., Lakhnech Y., and Siegel M., "Hierarchical automata as model for statecharts", In R. Shyamasundar and K. Euda, editors, ASIAN97 Third Asian Computing Conference. Advances in Computer Science, volume 1345 of *LNCS*, pages 181196. Springer-Verlag, 1997.

[4] Latella D., Majzik I., and Massink M., "Automatic verification of a behavioral subset of UML statechart diagrams using the SPIN model-checker", *Formal Aspects of Computing*, 11(6), pp. 637 664, 1999.

[5] Holzmann G., "The model checker SPIN", *IEEE Transactions on Software Engineering*, 23(5), pp. 279-295, 1997.

[6] Ldeczi A. et. al., "Composing Domain-Specific Design Environments", *IEEE Computer*, November 2001, pp. 44-51.

[7] Agrawal A., Karsai G., Ledeczi A., "An End-to-End Domain-Driven Software Development Framework", 18th Annual ACM SIGPLAN Conference on Object- Oriented Programming, Systems, Languages, and Applications (OOPSLA), Anaheim, California, October 26, 2003.

[8] Gottler H., "Attributed graph grammars for graphics", H. Ehrig, M. Nagl, and G. Rosenberg, editors, Graph Grammars and their Application to Computer Science, LNCS 153, pages 130-142, Springer-Verlag, 1982.

[9] Schürr A., Winter A. J., and Zündorf A., In [20], chap. The PROGRES Approach: Language and Environment, pp. 487 550. World Scientific, 1999.

[10] Nickel U., Niere J., and Zündorf A.. Tool demonstration: The FUJABA environment. In The 22nd International Conference on Software Engineering (ICSE). ACM Press, Limerick, Ireland, 2000.

[11] Harel D., "Statecharts: A visual formalism for complex systems", *Science of Computer Programming*, 8(3), pp. 231274, 1987.

[12] Corradini A., Heckel R., Montanari U. "Graphical operational semantics", In Proc. ICALP2000

[13] Schmidt A., Varró D., "CheckVML: A Tool for Model Checking Visual Modeling Languages", In Proc. UML 2003: 6th International Conference on the Unified Modeling Language, *LNCS*, vol. 2863, pp. 92-95.

[14] Varró D., "Automated Formal Verification of Visual Modeling Languages by Model Checking", *Journal of Software and Systems Modeling*, col. 3(2), pp. 85-113.

[15] Varró D. and Pataricza A., "Automated Formal Verification of Model Transformations", In Critical Systems Development in UML 2003, pp. 63-78.

[16] Denney E., Fischer B., "Certifiable Program Generation", GPCE 2005, *LNCS*, vol. 3676, pp. 17-28.