Crossing the Gap between Semantics and Practice: Crafting a Compiler for Statecharts

Extended Abstract

Andrzej Wąsowski*

IT University of Copenhagen
wasowski@it-c.dk

1 Introduction

Statecharts and various dialects based on state machines are popular both as main-stream modeling languages and as a specification formalism used in verification and model checking. The community of users and researchers agrees that one of the main advantages of the language is its good support for complete methodology – support for modeling, verification and automatic program synthesis.

The wide acceptance of complete methodology contradicts with reality. Program synthesis from statechart models is discussed much more seldom than other phases of the development process. Inefficient generation methods discourage end-users from high-level description languages. Moreover very limited dissemination of results builds a definitely incorrect impression that hardly anything is available on software synthesis from statecharts (see sample opinion in [4]).

2 Statecharts compilation survey

In the first part of the talk I will briefly present a survey of statechart compilation methods trying to determine the purpose and the specifics of each method. I will use following four sources of information:

1. Hardware logic synthesis from statecharts had been widely discussed in eighties (see for instance [5]). These approaches are seldom interesting for software generation because of the inherent concurrency in hardware implementations.
2. Numerous semantics definitions for statecharts describe the behaviors of modeling constructs using various theoretical frameworks. For most of these frameworks a corresponding implementation scheme can be derived.
3. Papers on verification of statecharts propose some model optimizations useful for making the model-checking process easier. Some of this model transformations/representations are interesting from program synthesis perspective as well.
4. Finally there are some direct contributions in program synthesis for statecharts.

After having reviewed (some part of) this material I have identified a few distinctive strategies. In this talk I will pick up typical examples and try to characterize them briefly. Interesting approaches are: logics-based equational methods [8, 2], hierarchy preserving methods[3], flattening methods [6] and global automata methods [7]. I should also briefly mention some program verification oriented methods (for example [9]) and comment on approaches of popular object-oriented CASE tools ([11] contains a small survey).

3 SCOPE compiler overview

In the second part of the talk, I will describe some experience with implementation of SCOPE[1], a simple high-level code generator for statecharts. The semantic structure should only be sketched shortly to explain meaning of various components in generated code. The details are described elsewhere [10]. I shall mainly focus on the structure of the compiler itself.

Our compiler takes a model as an input and translates it to runtime representation. This representation consolidated with a static driver is interpreted at runtime (see fig. 1 for overview of the environment).

The task performed by hierarchy preserving translator seems to be relatively simple. However apparent difficulties emerge when ones tries to implement such translator in a clean and efficient way. It

* I would like to thank Peter Sestoft of IT University in Copenhagen for guidance throughout the project.

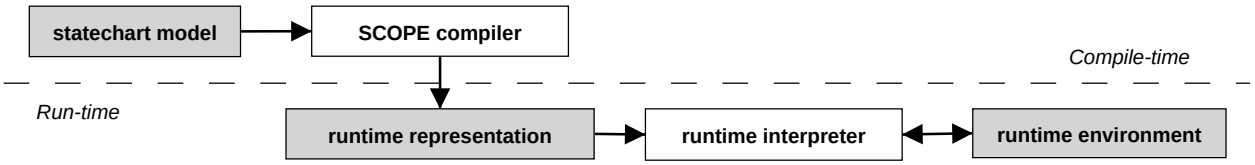

Fig. 1. Flow of model execution in visualSTATE environment.

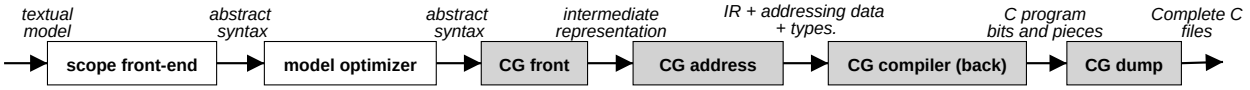

Fig. 2. Phases of statechart compiler as implemented in SCOPE. Shaded rectangles represent components of automatic code synthesizer.

is no longer obvious how the tool should be sliced into phases, and how those phases should be ordered. Traditional knowledge on the compiler structure does not help too much. The textbooks are oriented on generating machine language programs, while we are producing ... arrays of integers. SCOPE compiler uses a tiny bytecode language suitable for representing arrays of integer references as an intermediate representation.

Runtime representation consists of several integer arrays reflecting state hierarchy, transitions, history vectors, current state vector, signal queue and dispatch tables for actions and guards. The arrays are not compressed excessively at the moment. The only optimization made is inference of minimal integer types to be used as cell types.

To achieve that I have split the compiler into four separate phases (see fig.2). Each phase of the compiler is small and simple. Each welcomes another kind of optimizations to be added in future. I will comment on possibilities for optimizations in various components. The choices include global model transformation, sequential and parallel compaction of transitions, broadcasts unrolling, table compressions, etc.

Despite the lack of model optimizations SCOPE produces executables which are about 15% smaller than counterparts created with industrial implementation.

References

[1] SCOPE: A statechart compiler. http://www.mini.pw.edu.pl/~wasowski/scope.

[2] Beauvais, J.-R., Houdebine, R., Guernic, P. L., Rutten, E., and Gautier, T. A translation of Statecharts and Activitycharts into Signal equations. Tech. Rep. No 1182, IRISA: Institut de Recherche en Informatique et Systemes Aléatoires, Rennes Cedex, France, May 1998.

[3] Behrmann, G., Kristoffersen, K., and G.Larsen, K. Code generation for hierarchical systems. In *NWPT'99 – The 11th Nordic Workshop on Programming Theory* (Uppsala, Sweden, Sept. 1999).

[4] Björklund, D., Lilius, J., and Porres, I. Towards efficient code synthesis from statecharts. In *Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists. Workshop of the pUML-Group held together with the UML'2001* (Toronto,Canada, October 1st, 2001), A. Evans, R. France, and A. M. B. Rumpe, Eds., Lecture Notes in Informatics P-7, GI.

[5] Drusinsky, D. *On Synchronized Statecharts*. PhD thesis, Department of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1988.

[6] IAR Inc. IAR visualSTATE®. http://www.iar.com/Products/VS/.

[7] Maraninchi, F. The Argos language: Graphical representation of automata and description of reactive systems. In *Proceedings of the IEEE Workshop on Visual Languages* (Kobe, Japan, October 1991).

[8] Maraninchi, F., and Halbwachs, N. Compiling ARGOS into boolean equations. In *Proc. 4th Int. School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT)* (Uppsala, Sweden, Sept. 1996), vol. 1135 of *Lecture Notes in Computer Science*, Springer-Verlag.

[9] Sekerinski, E., and Zurob, R. iState: A statechart translator. In *UML 2001 - The Unified Modeling Language, Toronto, Canada, October 2001* (2001), M. Gogolla and C. Kobryn, Eds., Lecture Notes in Computer Science 2185, Springer-Verlag, pp. 376 – 390.

[10] Wąsowski, A., and Sestoft, P. On the formal semantics of visualSTATE statecharts. Tech. Rep. TR-2002-19, IT University of Copenhagen, Sept. 2002.

[11] Zündorf, A. Rigorous object oriented software development with Fujaba. Unpublished Draft, 2000.