Abstract. We describe formal steps in the design of iState, a tool for translating statecharts into programming languages. Currently iState generates code in either Pascal, Java, or the Abstract Machine Notation of the B method. The translation proceeds in several phases. The focus of this paper is the formal description of the intermediate representations, for which we use class diagrams together with their textual counterparts. We describe how the class diagrams are further refined. The notions of representable, normalized, and legal statecharts are introduced, where normalized statecharts appear as an intermediate representation and code is generated only for legal statecharts.

1 Introduction

Statecharts, conceived as a visual formalism for the design of reactive systems [3], extend finite state diagrams by hierarchy, concurrency, and communication. These three extensions allow the specification of complex reactive systems that would be impractical without them. Because of the appeal of the graphical notation, statecharts have gained popularity and are now part of object-oriented modeling techniques [4,8,9].

The concepts of hierarchy, concurrency and communication, while intuitive on their own, interact in intricate ways. As a result, various formal semantics of statecharts and statechart tools interpret their interaction in different ways or impose different constraints, e.g. [2,5]. Our approach is to use a semantics of statecharts defined by a direct translation into a (nondeterministic) programming language, with the goal of the generated code being comprehensible: Having readable code allows us to get confidence in the translator and in the original statechart. Furthermore, we can use the translation scheme for explaining statecharts and illustrate this by the translator, rather than having to use a “third domain” for the definition. Finally, in order to allow the generated code to be further analyzed, it must be understood in the first place.

Such a translation scheme from statecharts into the Abstract Machine Notation of the B method [1], an extension of Dijkstra’s guarded commands, was presented in [10]. The use of AMN is motivated by three aspects: first, AMN supports nondeterminism, allowing nondeterministic statecharts to be translated to nondeterministic AMN machines (that can be further refined into executable machines). This way, the nondeterminism is being preserved rather than eliminated by the translation, as other tools typically do. Secondly, AMN supports the parallel or independent composition of statements, which allows a simple translation of concurrent states. Thirdly, the B method allows invariants to be
stated and checked. Invariants can express safety conditions and checking the invariant will then ensure that every event is going to preserve all safety conditions.

Compared to translation schemes used by other tools, ours can be characterized as event-centric rather than state-centric, as the main structure of the code is that of events. The scheme is suitable for those kind of reactive systems where events are processed quickly enough so that no queuing of events is necessary and where blocking of events is undesirable. To our experience so far, the resulting code is not only comprehensible, but compact and efficient as well.

The tool we built for implementing this translation scheme, iState, operates in four phases as shown in Figure 1. In this paper we focus on formally describing the intermediate representations, namely the representable, normalized, and legal statecharts. We refer to [10] for an illustration of the translation scheme and to [11] for details on the translation algorithms and for larger examples.

The input to iState is a textual representation of statecharts that is defined by a grammar. This representation can also be visualized through a LATEX package. However, the Strategy pattern has been used for allowing other importers to be added, for example a graphical front end. Currently iState generates code in AMN, Pascal, and Java. The code generator uses an intermediate representation (which is similar to AMN) so that other languages can be added as needed. All complete statecharts in this paper have been processed and drawn by iState and its accompanying LATEX package.

Section 2 presents briefly all statechart elements and the event-centric translation scheme. This prepares for the formalization of statecharts by class diagrams, together with their textual counterpart, in Section 3. This section illustrates a general way of translating class diagrams into a textual form. Normalized and flawed statecharts are characterized in Section 4 and legal statecharts are characterized in Section 5. The statechart model is refined by eliminating associations and basic algorithms on statecharts are discussed in Section 6. We give an example in Section 7 and conclude with a discussion in Section 8.

2 An Event-Centric Translation Scheme

We give the translation scheme of statecharts to AMN first for plain state diagrams, then for hierarchy, concurrency, and communication. Programs in AMN, called machines, consists of a section declaring sets (types), a section declaring variables, an invariant section, an initialization section, and a section with operations. The initialization has to establish the invariant and all operations have to preserve it.