Abstract

What is the practical applicability of attribute grammars? As we show in this paper, attribute grammars are at least good enough for the prototyping of fully functional interactive tools. Going from a definition of a language and the functionality of its tools to an attribute grammar is a discipline in need of a systematic approach, for which we give some initial material. As is inevitable when a system is extensively used (in our case the Cornell Synthesizer Generator), this paper also proposes extensions to the attribute grammar formalism and its supporting systems.

1 Introduction

This paper represents, in some way, a view from the trenches. How we prototyped tools contributing to a specification environment for LOTOS is the main topic here. Attribute grammars were chosen because they promised to be a good prototyping approach to language based software development, and the close relation between attribute grammars and the description of tool functions helps ensure the correctness of the tool prototypes. Needless to say, doing some large developments based on attribute grammars has given us some insights in what can be done with them and what their limitations are. Knowing some techniques for structuring attribute grammars is very important. Yet, the methodology of programming with attribute grammars is not fully developed. One of the aims of this paper is to contribute to that methodology. Usage of attribute grammars in our tools is also discussed in another paper [vE89b], which has more emphasis on the functionality of the tools produced.
The attribute system selected for our work is the Cornell Synthesizer Generator (SG) [TR89a, TR89b]. Its important features are that it is a fully functional generator of structure editors employing incremental attribute evaluation. Moreover, the fact that these generated editors work on a window system greatly contributes to their appeal.

The structure of this paper is as follows. We assume some familiarity with attribute grammars and the concept of structure editors. In section 2 we discuss the kind of tool functionality that we want to prototype for LOTOS tools, and show that the concept of structure editing has wide applicability for that. Section 3 is devoted to attribute grammar programming techniques and, by necessity, also comments on limitations and possible extensions. Creating interactive user interfaces is the subject of section 4.

2 Functionality of LOTOS Tools

Most interactive tools can be seen as editors. In an editor the user issues commands to navigate through, manipulate or view the edit-object. For example, the edit-object of a conventional text editor is a sequence of lines of characters. In a database system the edit-object is an entire database and can be accessed through a database query language. Tools that support a specification language operate on expressions in this language and on objects that denote the semantics of these expressions (as in a programming language: the program, its execution state and assertions). The SG generates language specific editors from a high level language specification. Editor specifications are written in a formalism (SSL) that is based on attribute grammars.

LOTOS (Language Of Temporal Ordering Specification) is a language developed for the formal specification of communication protocols and services. It was developed by ISO for the work on Open Systems Interconnection and is now international standard IS8807. LOTOS is based on Milner's CCS (Calculus of Communicating Systems) [Mil80]. Its data type structure is based on the abstract data type language ACT ONE [EM85]. An overview of LOTOS, including specifications and theory, can be found in [vEVD89]. A tutorial on LOTOS can also be found in [BB87].

In process algebraic approaches, such as LOTOS and CCS, the emphasis is on describing systems as behaviours. A behaviour is a sequence of events and event-offers. An event is the result of a synchronous interaction of behaviours. A behaviour is described by a behaviour expression. A behaviour expression can be composed by combining atomic event-offers and by combining other behaviour expressions with operators such as choice, parallel, enable, disable etc. Behaviour expressions can also be named and parameterised.

Semantically, a behaviour expression denotes a transition system of states and transitions (corresponding to events) between states. At each state of the behaviour a number of events are possible. The set of these events is called the menu of that state. If no events are possible, the state is usually called a deadlock state. All possible sequences of events form the tree of behaviour, also called the communication tree. Each node in the tree is a state, and each edge corresponds to an event, which is possibly parameterised with values. The communication tree is an important concept that is the basis of a number of