Integrated Graphic Environment to Develop Applications based on Attribute Grammars

Tibor Gyimóthy*, Zoltán Alexin* and Róbert Szücs**

*Research Group on the Theory of Automata,
Hungarian Academy of Sciences,
H-6720 Szeged Aradi vértanúk tere 1.
e-mail: h42gyi@ella.hu and h157ale@ella.hu

**Cogito, Software Research and Development Ltd.
H-6725 Szeged Szentháromság utca 75/B.

Abstract. This paper presents an overview of an integrated graphic environment called **P-GEN**$^{(1)}$ to develop applications based on attribute grammars. The system has a modular structure to enable integration of different modules for all phases of the processing of an attribute grammar specification. This environment contains routines for transforming attribute grammar specifications written in different formalisms to an internal representation. The system assists the interpretation of the most important evaluation strategies. The graphic representation of the parse trees and attributed dependency graphs is also supported.

1 Introduction

Attribute grammars [Knu68] are probably the most adequate of the tools available for the specification and automated implementation of compilers, interpreters and translators. However only a few commercial applications are based on attribute grammars the main reason for which probably being the lack of a true user-friendly integrated environment. In the following we present some requirements for such a system.

Perhaps the most difficult problem is the absence of a standard formalism for the attribute grammar specification. Hence such an environment has to contain modules to convert specifications written in different metalanguages. In addition it is convenient for the user if he can make the attribute grammar specification interactively as well.

* This work has been supported by the Cogito Ltd, Hungary and the presentation was made possible by OTKA under grant No. 99/10441.

The system needs a well defined interface that enables the incorporation of different kinds of parser modules. In many applications it is useful if that the user can direct the tree-construction corresponding to the (already) known set of production rules (e.g. Syntax Directed Editors). Ambiguous attribute grammars are applied in numerous fields of artificial intelligence (e.g. syntactic pattern recognition, natural language processing). For these grammars it is useful if the computed attribute values can be used during the parsing. Incremental methods for parser generation are also included thus providing the possibility of interactive grammar specification. Of course the system must contain parser generator modules for the most important parsing methods (LL(1), LALR(1)). The graphical representation of the parsing process helps the user in the development of a grammar.

The system has to contain interfaces to build different attribute evaluators, but the real help for a user is if an interpreter for the attribute evaluation is involved in the system. In such a case the evaluation process takes place before the user's eyes. The evaluation process can be executed step by step, may be interrupted at any time during which the computed values of the attributes can be requested from the system. After this the processing can continue to the next step. The graphic display of the attributed dependency graph is very useful (e.g. the syntax-tree and the attributes of this tree can be viewed).

A system developed by taking into consideration the previous points has the advantage that beginners can define usable attribute grammars in a short time while experts can use it for debugging and design. Such a system could be a link between different attribute grammar systems.

The system called **P-GEN** that has these properties is currently under development and we now give an overview of this project. The system has a modular structure like **ELI** [GHL92] to integrate different modules for all the phases of processing an attribute grammar specification. Similar to the **TALE** [JKP91] system a graphic presentation is given to assist the user in the development. A unique feature of our system is that it gives a common platform from which different attribute grammar formalisms are converted to a standard one.

This paper is organized as follows. Section 2 is a short overview of attribute grammar specification possibilities in the system. In Section 3 we discuss the parsing problem. In Section 4 the concept of the attribute evaluation is presented. In Section 5 we demonstrate the user interface and utilities of the system. Finally in Section 6 we present our conclusions. We assume that the reader is familiar with attribute grammars.

2 Attribute Grammar Specification

There are not many developers who use attribute grammars to produce applications, in spite of the fact that numerous attribute grammar systems have been constructed during the past few years [DJL88]. In addition there is no indication that one of the attribute grammar metalanguage will become the standard. This is - in our opinion - the main obstacle to the spreading of attribute grammar systems i.e. there are not enough human and financial resource to develop a suitable environment. There are no standard libraries for particular tasks and because of the different formalisms sometimes even the understanding of the specifications is hard.

This paper describes the effort of building a general purpose system based on attribute grammar that have the properties listed in the previous section. First we developed an internal representation of the I/O data structures of the lexical, syntactic and attribute evaluation modules. These data structures can be enriched when necessary.

The input to the system is a simple text file which can be edited by any conventional text editor. Naturally the user has the choice of using the attribute evaluator interpreter (see Section 4), he may if he wishes start the whole generating process from a batch.

In the ***import*** menu-item of the system different kinds of attribute grammar specifications can be read and converted to the internal data structures. In the ***export*** menuitem the internal representation of an attribute grammar can be transformed to different attribute grammar formalisms. At present the system contains routines to import from and export to **PROF-LP** [GHK89].

An additional feature of our system is that the attribute grammar specification can be written interactively. In this case the system checks the consistency of the specification during editing. The function of the interactive part of our system is similar to that used by the interactive **PROF-LP** as presented in [GHK89].

3 Building The Syntax-tree

The best founded part of compiler theory is the syntactic parsing (e.g. algorithms to generate effective LL(1) and LALR(1) parsers are presented in [ASU86]).

Our system offers the opportunity to use a separate parser. The user can use his "own" parser, but this parser is expected to output a sequence of productions as the syntax-tree used to derive the actual input. Our system provides built-in LL(1) and LALR(1) parser generators.

In certain fields of attribute grammar applications the underlying CF grammar itself is ambiguous (e.g. natural language interfaces, syntactic pattern recognition). To deal with these grammars a backtrack parser generator has been developed in our system [GYT86]. The generated parsers use the LL(1) tables so a lot of redundant backtracks can be eliminated. A further characteristic of the generated parsers is that the parsing can be influenced by the evaluated attributes. Calling the start symbol of an ambiguous grammar repeatedly allows all possible derivations of the grammar to be constructed for a given input.

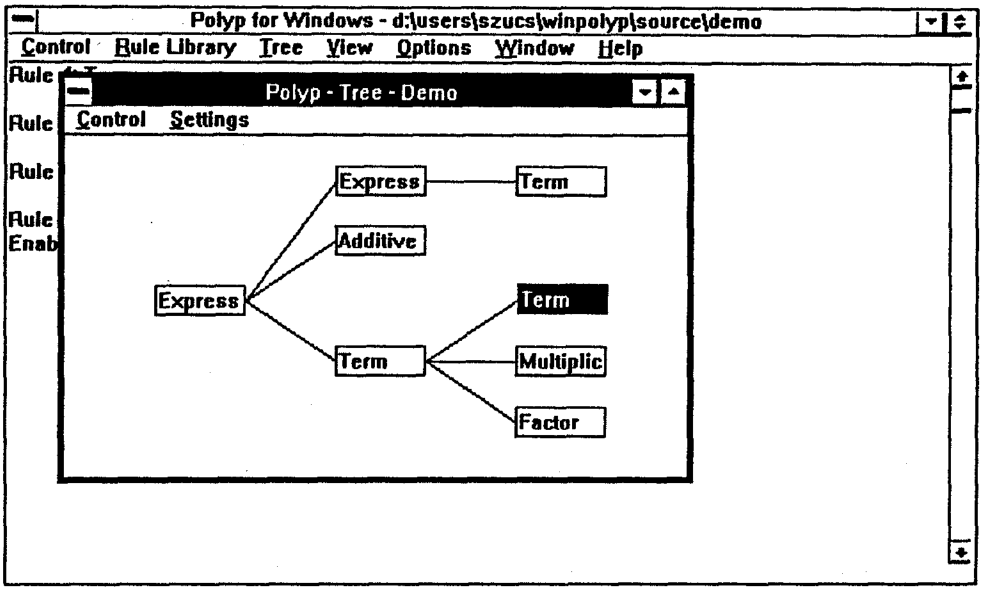

Polyp for Windows - d:\users\szucs\winpolyp\source\demo
Control Rule Library Tree View Options Window Help
Rule
Rule
Rule
Rule
Enab
Polyp - Tree - Demo
Control Settings
Express
Term
Additive
Express
Term
Term
Multiplic
Factor

Figure 1.

Another option available when building a derivation-tree is that the user can control the tree creation himself. The derivation-tree appears on the screen and the tree building can be directed by cursor movements (clicking, dragging) - see Figure 1. - thus the tree evolves templatewise. (A template is a labelled and attributed inner node of the derivation-tree.) We plan to integrate a complete syntax directed editor generator into the system.

The interactive specification of attribute grammars requires the incremental generation of parsers especially large grammars. There are methods for generating LL(1) and LALR(1) parsers incrementally [HKR89][GHK89]. The system will eventually be augmented with such parser generators.

4 The Attribute Evaluation

Our main goal was the interpretation of different attribute evaluators. The evaluation can be performed step by step or continuously, the values of the attribute instances can be printed out in response to a user request.

The result of the parser module is a derivation-tree which is read by the interpreter which in turn builds its own internal attributed tree. Each node contains space to store attribute values occurring in the regarded production rule. The central part of this module is a routine that can compute an arbitrary attribute instance (presuming that all other needed attributes have already been computed). The routine looks for the semantic function that sets the value of the corresponding attribute, then pushes the parameters, i.e. the known attributes, onto the stack then finally interprets the function call and stores the computed attribute value. Hence all of the semantic equations are expected to be a function. In order to make perfect calls, functions have to be declared in the attribute grammar specification. Declaration means the declaration of function names, the parameter types and order, and the type of the return value.

By using this basic routine it is not too difficult to implement different attribute evaluation strategies (e.g. in the case of the visit-sequence oriented evaluators [Kas80] this routine is called by the actual attribute instances).

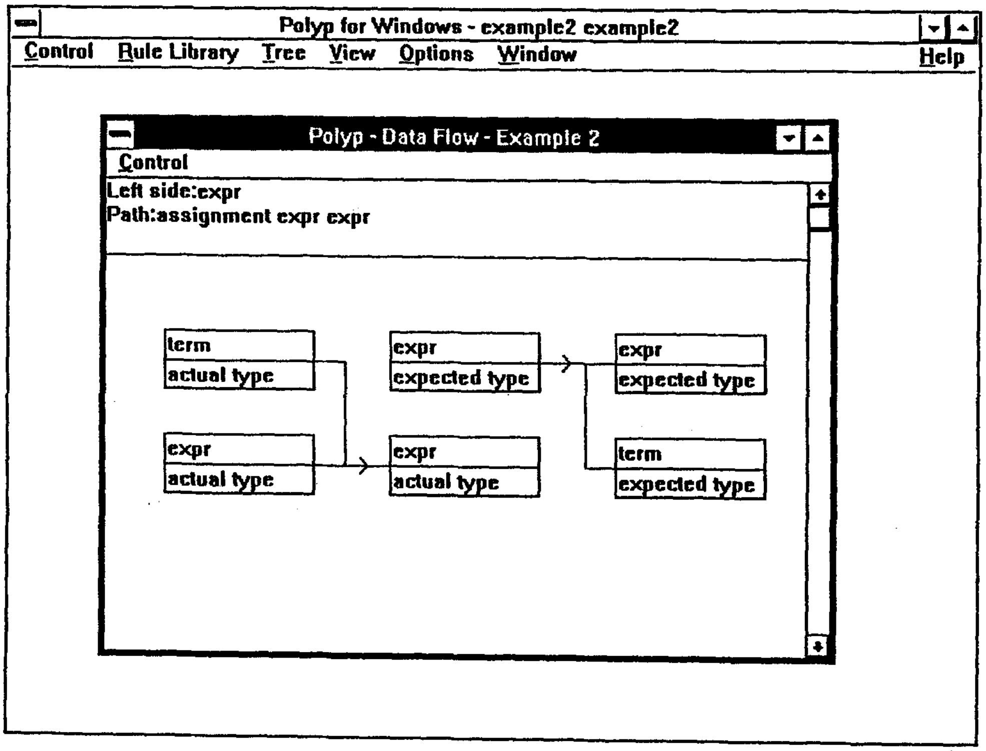


Figure 2.

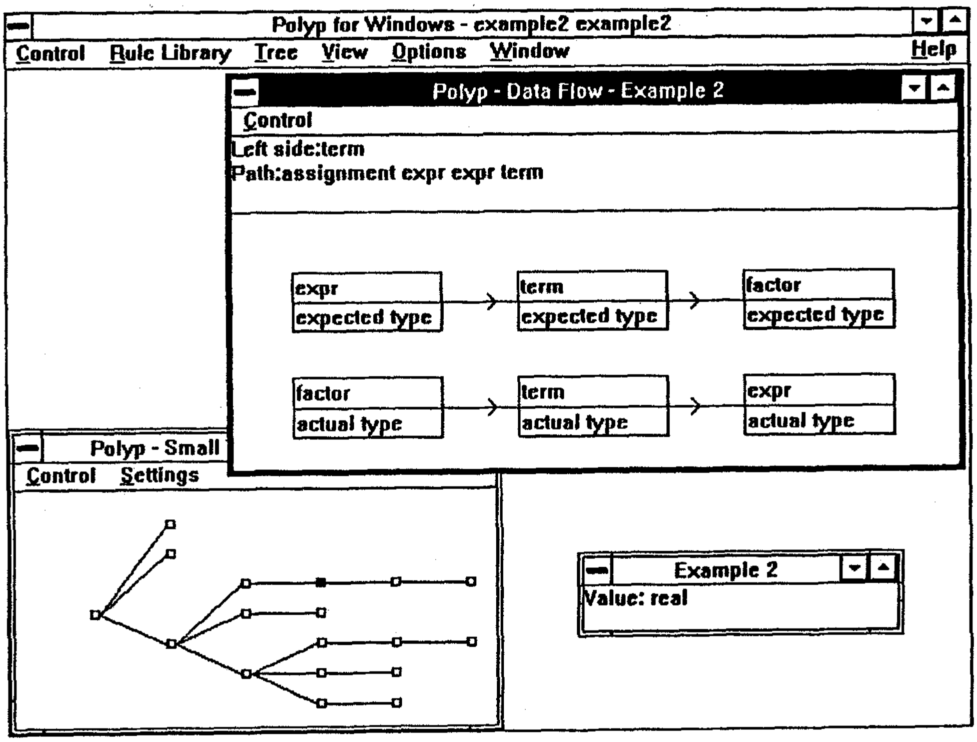
Since the evaluated trees can be modified directly the integration of the incremental attribute evaluator modules will be useful [Alb90].

To give as much freedom as possible we have developed a dynamic evaluator. The system computes the set of those attributes which can be computed in the next step, then the user chooses the next attribute to be computed and instructs the interpreter to do a step and so on and so on ... .

The system can display the attributed dependency graph of a node (see Figure 2). The global flow of the evaluation process can be followed by viewing the previouslymentioned syntax-tree display and the local flow can be followed by viewing the dependency graph.

5 User Interface and Utilities

P-GEN is written in C++[(2)] and requires the MS Windows 3.xx[(3)] graphical environment. Our system is devoted to becoming a standard specification tool for attribute grammars and to be an integrated tool for running, testing and debugging the generated parsers and evaluators.



Polyp for Windows - example2 example2
Control Rule Library Tree View Options Window Help
Polyp - Data Flow - Example 2
Control
Polyp - Small
Control Settings
Example 2

Left side:term
Path:assignment expr expr term

expr expected type → term expected type → factor expected type

factor actual type → term actual type → expr actual type

Value: real

Figure 3.

Reading outer attribute grammar specification into the system is supported by conversion routines.

The attribute grammar specification can be typed interactively by means of structured dialog boxes after which the system checks the typed production rule to determine wether it is consistent with the previously typed rules. The attributed trees constructed by an attribute grammar can be stored in the system. If an attribute grammar is modified the system checks the consistency of the constructed trees. The system can generate the valid trees for a grammar. The nodes of the trees can be selected by different ways (e.g. by name, by rule-name, by attribute-name).

There are several possibilities for the interpretation of the attribute evaluations. The user can request evaluation concerning to a standalone attribute instance, to a group of attributes and to a total subtree. The process is shown on the graphic representation of the syntax-tree or on the dependency graphs. The computed attribute values can be displayed on the screen (see Figure 3). The initial values can be set manually or the computed values can be changed.

6 Summary

P-GEN is a unified graphic environment to develop attribute grammar specifications. The different specification methods can be linked together. So far the **PROF-LP** import/export routines are implemented. Thus the system can be tested by the attribute grammar applications written in **PROF-LP** dialect.

Our system can be used for different applications - it can be considered as a user-friendly integrated development system; The most convenient specification methods can be used as well as parser generator and evaluator modules; The debugging facilities make the process of development faster by an unestimatable amount; The system has applications in teaching where it may be used to demonstrate the working of parsers and attribute evaluators; And last but not least we refer to a project of our research group that deals with the learning of attribute grammars in a pattern recognition environment where an interactive system provides great help in evaluating the different learning strategies.

7 References

[Alb90] H. Alblas: Concurrent Incremental Attribute Evaluation. In: P. Deransart, M. Jourdan (eds.): Attribute Grammars and their Applications Lecture Notes in Computer Science 461. Berlin: Springer Verlag 1990. pp. 343-358

[ASU86] A.V. Aho, R. Sethi, J.D. Ullman: Compilers - Principles, Techniques and Tools. Addison-Wesley 1986.

[DJL88] P. Deransart, M. Jourdan, B. Lorho: Attribute Grammars. In: Lecture Notes in Computer Science 323. Berlin: Springer Verlag 1988.

[GHK89] T. Gyimóthy, T. Horváth, F. Kocsis, J. Toczki: Incremental Algorithms in PROF-LP. In: D. Hammer (eds.) 2nd Workshop on Compiler-Compilers and High-Speed Compilation 1988. Lecture Notes in Computer Science 371. Berlin: Springer Verlag 1989. pp 93-102

[GHP92] R.W. Gray, V.P. Heuring, S.P. Levi, A.M. Sloane, W.M. Waite: ELI: A Complete, Flexible Compiler Construction System In: Communications of the ACM. February, 1992. Vol. 35. No. 2. pp 121-130

[GYT86] T. Gyimóthy, J. Toczki: Syntactic Pattern Recognition in the HLP/PAS System. Acta Cybernetica 8, Vol 1. Szeged: 1987. pp 79-88

[HKR89] J. Heering, P. Klint, J.G. Rekers: Incremental Generation of Parsers In: Proc. of ACM Sigplan '89 Conference on Programming Language Design and Implementation, Portland, Oregon, 1989. ACM Sigplan Notices 24, 7 1989. pp 179-191

[JKP91] E. Järnwall, K. Koskimies, J. Paakki: The Design of the Tampere Language Editor. Department of Computer Science University of Tampere Technical Report A-1991-10.

[Kas80] U. Kastens: Ordered Attributed Grammars. Acta Informatica 13. 1980. pp 229-256

[Knu68] D.E. Knuth: Semantics of Context-Free Languages Mathematical Systems Theory 2 (June 1986.) pp 127-146

(1) P-GEN is a product of Cogito Co. Ltd.

(2) C++ is a product of Borland Inc.

(3) Windows 3.xx is a product of Microsoft Inc.