Science of Computer Programming 96 (2014) 191–210

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

The formalization and implementation of Adaptable Parsing Expression Grammars

Leonardo V.S. Reis[a,*], Roberto S. Bigonha[b], Vladimir O. Di Iorio[c], Luis Eduardo S. Amorim[c]

[a] Departamento de Computação e Sistemas, Universidade Federal de Ouro Preto, Brazil
[b] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil
[c] Departamento de Informática, Universidade Federal de Viçosa, Brazil

HIGHLIGHTS

- We propose an adaptable model based on Parsing Expression Grammars.
- We added attributes to PEG so extensibility is achieved by means of grammar attributes.
- The model is formally defined.
- The implementation of the adaptable model is detailed.
- The adaptable model preserves PEG legibility.

ARTICLE INFO

Article history:
Received 30 March 2013
Received in revised form 8 February 2014
Accepted 21 February 2014
Available online 27 February 2014

Keywords:
Extensible languages
Adaptable grammars
PEG

ABSTRACT

The term “extensible language” is especially used when a language allows the extension of its own concrete syntax and the definition of the semantics of new constructs. Most popular tools designed for automatic generation of syntactic analysers do not offer any adequate resources for the specification of extensible languages. When used in the implementation of features like syntax macro definitions, these tools usually impose severe restrictions. For example, it may be required that macro definitions and their use reside in different files; or it may be impossible to perform the syntax analysis in one single pass. We claim that one of the main reasons for these limitations is the lack of appropriate formal models for the definition of the syntax of extensible languages.

This paper presents the design and formal definition of *Adaptable Parsing Expression Grammars*, an extension to the *Parsing Expression Grammar* (PEG) model that allows the manipulation of its own production rules during the analysis of an input string. The proposed model compares favourably with similar approaches for the definition of the syntax of extensible languages. An implementation of the model is also presented, simulating the behaviour of packrat parsers. Among the challenges for this implementation is the use of attributes and on the fly modifications on the production rules at parse time, features not present in standard PEG. This approach has been used on the definition of a real extensible language, and initial performance tests suggest that the model may work well in practice.

* Corresponding author.
E-mail addresses: leo@decsi.ufop.br (L.V.S. Reis), bigonha@dcc.ufmg.br (R.S. Bigonha), vladimir@dpi.ufv.br (V.O. Di Iorio), luis.amorim@ufv.br (L.E.S. Amorim).

http://dx.doi.org/10.1016/j.scico.2014.02.020
0167-6423/© 2014 Elsevier B.V. All rights reserved.

```
 1 grammar ForLoop extends{Expression,Identifier}
 2   Expr |:=
 3     for {i:Id ← e:Expr,? Space}* do block:Expr end
 4       ⇒ // ... define a transformation to pure Fortress code
 5 end
 6
 7 ...
 8
 9 // Using the new construct
10 g1 = <1,2,3,4,5>
11 g2 = <6,7,8,9,10>
12 for i ← g1, j ← g2 do
13   println ``('' i ``,'' j ``)''
14 end
```

Fig. 1. A Fortress program with a syntax macro.

1. Introduction

In recent years, we have witnessed important advances in parsing theory. For example, Ford created *Parsing Expression Grammars (PEG)* [1], an alternative to Context Free Grammars for describing syntax, and packrat parsers [2], top-down parsers with backtracking that allow unlimited lookahead and a linear parse time. Parr has devised a new parsing strategy called LL(*) for the ANTLR tool, which allows arbitrary lookahead and recognizes some context-sensitive languages [3]. Visser proposed SGLR [4,5], combining scannerless parsing with generalized LR parsing, and addressing important issues related to extensibility and compositionality of parsers. New engines have been created, such as YAKKER [6], which allows the control of the parsing process by arbitrary constraints, using variables bound to intermediate parsing results. All the advances listed above, however, do not include important features for the definition of extensible languages, especially when extensibility may be considered at parse time.

As a simple example of desirable features for the implementation of extensible languages, Fig. 1 shows an excerpt from a program written in the Fortress language [7]. Initially, a new syntax for loops is defined, and then this syntax is used in the same program. In details, line 1 introduces the definition of *ForLoops* as an extension of *Expression* and *Identifier*. Line 2 says that the grammatical rule that defines *Expr* is to be extended with the new right hand side defined on line 3. The new alternative, defined in line 3, begins with a terminal symbol *for*, followed by a symbol group, another terminal *do*, a nonterminal *Expr* and a terminal *end*. The names *i*, *e* and *block* are only aliases to the nonterminal, which may be used in the transformation part, not showed in Fig. 1. The expression between the symbols { and } is a sequence of zero or more patterns of the form *Id* ← *Expr* with an optional comma and a required white space (nonterminal Space) at the end. The lines 12 to 14 show a use of this new syntax.

Standard tools for the definition of the syntax of programming languages are not well suited for this type of extension, because the syntax of the language is modified while the program is processed. The Fortress interpreter, written with the tool Rats! [8], uses the following method: it collects only the macro (extension) definitions in a first pass, processes the necessary modifications to the grammar, and then parses the rest of the program in a second pass [9]. A tool that is able to parse the program in Fig. 1 in one pass must be based on a model that allows syntax extensions.

Our concern is the lack of tools for helping the definition and implementation of extensible languages, such as Fortress. Our goal is to provide a solution for this problem, which may be used in practice. We propose *Adaptable Parsing Expression Grammars (APEG)*, a model that combines the ideas of *Extended Attribute Grammars*, *Adaptable Grammars* and *Parsing Expression Grammars*. The main goals that the model has to achieve are:

- to offer facilities for adapting the grammar during the parsing process, without adding too much complexity to the base model (in this case, PEG);
- to allow an implementation with reasonable efficiency.

Satisfying the first requirement, the model may be considered a viable formal approach to describe the syntax of extensible languages. An efficient implementation allows automatic generation of parsers that may be used in practice.

1.1. From Context-Free to Adaptable Grammars

Context Free Grammars (CFGs) are a formalism widely used for the description of the syntax of programming languages, but not powerful enough to describe context dependent aspects of any interesting programming language, let alone languages with extensible syntax. In order to deal with context dependency, several augmentations to the CFG model have been proposed, and the most commonly used are variations on *Attribute Grammars* (AGs) [10]. First introduced by Knuth to assign semantics to context-free languages, AGs have subsequently been used for several other purposes, such as the specification of static semantics of programming languages [11,12].

Authors like Christiansen [13] and Shutt [14] argue that, in AG and other extensions for CFGs, the clarity of the original base CFG model is undermined by the power of the extending facilities. Christiansen gives as an example an attribute