Computer Languages, Systems & Structures 36 (2010) 203–222

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

A platform for the automatic generation of attribute evaluation hardware systems

Alexandros C. Dimopoulos, Christos Pavlatos *, George Papakonstantinou

National Technical University of Athens, School of Electrical and Computer Engineering, Heroon Polytechniou 9, 15780 Zografou, Athens, Greece

ARTICLE INFO

Article history:
Received 6 February 2009
Received in revised form
27 August 2009
Accepted 30 September 2009

Keywords:
Attribute grammars
FPGA
Attribute evaluation
Hardware
Semantic evaluation

ABSTRACT

Attribute grammars (AG) allow the addition of context-sensitive properties into context free grammars, augmenting their expressional capabilities by using syntactic and semantic notations, making them in this way a really useful tool for a considerable number of applications. AGs have extensively been utilized in applications such as artificial intelligence, structural pattern recognition, compiler construction and even text editing. Obviously, the performance of an attribute evaluation system resides in the efficiency of the syntactic and semantic subsystems. In this paper, a hardware architecture for an attribute evaluation system is presented, which is based on an efficient combinatorial implementation of Earley's parallel parsing algorithm for the syntax part of the attribute grammar. The semantic part is managed by a special purpose module that traverses the parse tree and evaluates the attributes based on a proposed stack-based approach. The entire system is described in Verilog HDL (hardware design language), in a template form that given the specification of an arbitrary attribute grammar, the HDL synthesizable source code of the system is produced on the fly by a proposed automated tool. The generated code has been simulated for validation, synthesized and tested on an Xilinx FPGA (field programmable gate arrays) board for various AGs. Our method increases the performance up to three orders of magnitude compared to previous approaches, depending on the implementation, the size of the grammar and the input string length. This makes it particularly appealing for applications where attribute evaluation is a crucial aspect, like in real-time and embedded systems. Specifically, a natural language interface is presented, based on a question-answering application from the area of airline flights.

1. Introduction

Some 40 years ago Knuth [1] extended context free grammars (CFG) by allowing the addition of context-sensitive properties and thus introducing the appealing formalism of attribute grammars (AG). Specifically, semantic rules and attributes were added to CFGs augmenting their expressional capabilities and therefore offering many advantages in the domain of language specification, analysis and translation. AGs allow high-level context-sensitive properties of individual constructs in a language to be described in a declarative way and to be automatically computed for any program in the language. The primary field of AG usage is in computer languages [2] but they are also convenient in fields such as Artificial

* Corresponding author. Tel.: +30 6945779553.
E-mail addresses: alexdem@cslab.ece.ntua.gr (A.C. Dimopoulos), pavlatos@cslab.ece.ntua.gr (C. Pavlatos), papakon@cslab.ece.ntua.gr (G. Papakonstantinou).

1477-8424/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2009.09.003

Intelligence [3,4], Pattern Recognition [5] or even Biomedicine [6]. Regardless the field of application, in an AG, knowledge is represented using syntactic and semantic (attribute evaluation rules) notation. For the first (syntax) part, a parser is responsible for the recognition of the syntax and the construction of the parse tree as well. For the second part, that of the tree traversal and semantic evaluation, referred in the following as tree decoration, an evaluator is needed. The evaluator traverses the parse tree, which is covered with applicable rules in each node and executes the corresponding actions, associating attribute values in each node. Typically, the values of an attribute can be defined in terms of values of other attributes.

Attribute grammar evaluation is an operation that is usually divided into two subparts, the syntactic and the semantic. Concerning the syntactic analysis, one significant factor that can positively influence the performance of the CFG parser is undoubtedly the selected parsing algorithm. Two well-known parsing algorithms for general CFGs are the Earley's algorithm [7] and the Cocke–Younger–Kassami (CYK) algorithm [8]. Both of them are basically a dynamic programming procedure and have a time complexity $O(n^3|G|)$, where n is the length of the input string and $|G|$ is the size of the grammar. A closer look on the previously mentioned parsing algorithms shows that there are some strong connections [9] between the two. After the introduction of Earley's and CYK algorithms, several modifications [9] and improvements via parallelization [10–13] have been proposed for these algorithms. Chiang and Fu [10] and Cheng and Fu [11] have presented designs using VLSI arrays for the hardware implementation of the aforementioned parsing algorithms, although they do not propose an efficient implementation for the operator they use, while Ibbara [12] and Ra [13] presented software implementations running on parallel machines. All these approaches are not implemented in reconfigurable hardware and the scale of the hardware is input string length dependent. The hardware oriented approach was reinvigorated by presenting implementations in reconfigurable FPGA (field programmable gate arrays) boards of the CYK algorithm [14,15] and Earley's algorithm [16]. The early architectures, proposed in [14,15], either fail to fully exploit the available parallelization of the parsing algorithms or demand excessive storage. Whereas the software approaches by Ibbara [12] and Ra [13] execute parts of the parsing algorithms sequentially and, thus, do not achieve the maximum possible speed-up. On the other hand, existing hardware methodologies must overcome the complexity imposed by the operations of the parsing algorithms, something that leads to increased storage needs. In order to relax the hardware complexity, most of the proposed architectures implement the CYK algorithm, whose basic operations are much simpler than those of Earley's. The first FPGA implementation of Earley's algorithm was given in [16]. The approach proposed in [17] uses a combinatorial circuit for the fundamental operator of Earley's algorithm. In this manner, an decrease in time by a factor of one to two orders of magnitude is achieved, compared to previous hardware implementations [16], depending on the size of the grammar and on the input string length. The speed-up, compared to the pure software implementation, varies from two orders of magnitude for toy-scale grammars to six orders of magnitude for large real life grammars, a speed-up which is really important in embedded real-time systems.

Concerning the semantic analysis, there are various software approaches that apart from syntactic analysis, also tackle with the attribute evaluation. A well-known approach is that of Yacc [18], which generates a parser based on an analytic grammar written in a notation similar to BNF. The class of grammars accepted is LALR(1)[1] grammars with disambiguating rules and is based on the S-attributed approach for AGs [19]. Another similar approach, more current, is that of Bison [20], a general-purpose parser generator that converts a grammar description for an LALR(1) context-free grammar into a C program to parse that grammar. Bison [20] is based on the L-attributed [19] approach for AGs. A combined variety of standard tools that implement compiler construction strategies into a domain-specific programming environment is called Eli [21,22]. For the attribute evaluation, Eli makes usage of LIDO [23], a language for the specification of computations in trees, that supports both inherited and synthesized attributes. Furthermore, in the Utrecht University the tool UUAG [24] has been implemented that decorates a given parse tree, omitting the task of syntactic analysis, supporting both inherited and synthesized attributes.

Aside from the software approaches, recently hardware oriented approaches were reinvigorated by presenting implementations in reconfigurable FPGA boards and prior to that, VLSI approaches were presented. In [25] a hardware implementation of a shape grammar was proposed. Nevertheless, there were two main limitations. The first was that the attributes were standard and only for the proposed attribute grammar and the second was that the scale of the hardware was dependent on the size of the problem and the input string length. In 2005 Panagopoulos [26] presented a RISC microprocessor for attribute evaluation implemented on FPGA. It was the first effort to design a specialized microprocessor for attribute grammar evaluation and exploiting its merits in performance and programming simplicity in knowledge engineering applications. It was semantically driven, it supported dynamic parsing by exploiting tree-pruning techniques to increase efficiency and prevent the memory explosion problem of storing all possible parse trees while giving all possible solutions (nondeterministic), i.e. it could provide all possible parse trees for a specific input string. The main limitation is that the implementation was based on Floyd's sequential parser [27] and therefore was not efficient enough, compared to parsing algorithms of references [7,8]. Furthermore, the attribute evaluation was executed on an extended RISC microprocessor implemented on an FPGA, hence like a common software approach but running at a lower frequency, since FPGAs cannot achieve the high frequencies of current processors. The same year another approach was proposed [28] using Earley's parallel parsing algorithm [10] implemented in a hardware module [16] mapped on an FPGA collaborating with an

[1] Look Ahead Left to right, Rightmost derivation parser with 1 lookahead symbol.