Parse Table Composition
Separate Compilation and Binary Extensibility of Grammars

Martin Bravenboer\textsuperscript{1} and Eelco Visser\textsuperscript{2}

\textsuperscript{1} University of Oregon, USA
\texttt{martin.bravenboer@acm.org}
\textsuperscript{2} Delft University of Technology, The Netherlands
\texttt{visser@acm.org}

Abstract. Module systems, separate compilation, deployment of binary components, and dynamic linking have enjoyed wide acceptance in programming languages and systems. In contrast, the syntax of languages is usually defined in a non-modular way, cannot be compiled separately, cannot easily be combined with the syntax of other languages, and cannot be deployed as a component for later composition. Grammar formalisms that do support modules use whole program compilation.

Current extensible compilers focus on source-level extensibility, which requires users to compile the compiler with a specific configuration of extensions. A compound parser needs to be generated for every combination of extensions. The generation of parse tables is expensive, which is a particular problem when the composition configuration is not fixed to enable users to choose language extensions.

In this paper we introduce an algorithm for \textit{parse table composition} to support separate compilation of grammars to \textit{parse table components}. Parse table components can be composed (linked) efficiently at runtime, i.e. just before parsing. While the worst-case time complexity of parse table composition is exponential (like the complexity of parse table generation itself), for realistic language combination scenarios involving grammars for real languages, our parse table composition algorithm is an order of magnitude faster than computation of the parse table for the combined grammars.

1 Introduction

Module systems, separate compilation, deployment of binary components, and dynamic linking have enjoyed wide acceptance in programming languages and systems. In contrast, the syntax of languages is usually defined in a non-modular way, cannot be compiled separately, cannot easily be combined with the syntax of other languages, and cannot be deployed as a component for later composition. Grammar formalisms that do support modules use whole program compilation and deploy a compound parser. In this paper we introduce an algorithm for \textit{parse table composition} to support separate compilation of grammars to \textit{parse table components}.

The lack of methods for deploying the definition and implementation of languages as components is harming programming practices. Languages are combined in an undisciplined and uncontrolled way, for example by using SQL, HQL, Shell commands, XPath,
import table person [ id INT, name VARCHAR, age INT ];
connection c = "jdbc:postgresql:mybook";
ResultSet rs = using c query { SELECT name FROM person WHERE age > {limit}};

Fig. 1. SQL extension of Java in ableJ (Silver)

$name = $_GET['name'];
$q = "SELECT * FROM users WHERE name = " . $name . "";
$q = <| SELECT * FROM users WHERE name = ${$name} |>
system("svn cat "file name" -r" . $rev);
system(<| svn cat "file name" -r${$rev} |>

Fig. 2. SQL and Shell extensions of PHP (StringBorg)

class FileEditor {
    void handle(Event e) when e@Open { ... }
    void handle(Event e) when e@Save { ... }
    void handle(Event e) {...} }

Fig. 3. Predicate dispatch in JPred (Polyglot)

Return(ConditionalExpression(e1, e2, e3)) -> If(e1, Return(e2), Return(e3))
[[ return e1 ? e2 : e3; ]] -> [[ if($e1) return $e2; else return $e3; ]]

Fig. 4. Transformation with concrete Java syntax (Stratego)

regular expressions, and LDAP in string literals. The compiler of the host language has no knowledge at all of these languages. Hence, the compiler cannot check if the programs are syntactically correct, nor can the compiler help protect the program against security vulnerabilities caused by user input that is not properly escaped. Extensible compilers such as ableJ [1] (based on Silver [2]), JastAddJ [3], and Polyglot [4] support the modular extension of the base language with new language features or embeddings of domain-specific languages. For example, the security vulnerabilities caused by the use of string literals can be avoided by extending the compiler to understand the syntax of the embedded languages. The extended compiler compiles the embedded fragments with the guarantee that user input is properly escaped according to the escaping rules of the embedded language. Figure 1 shows an application of the extensible compiler ableJ. This extension introduces constructs for defining database schemas and executing SQL queries. The implementation of the extension is modular, i.e. the source code of the base Java compiler is not modified. Silver and the ableJ compiler have also been applied to implement extensions for complex numbers, algebraic datatypes, and computational geometry. Similar to the SQL extension of ableJ, the StringBorg syntactic preprocessor [5] supports the embedding of languages in arbitrary base languages to prevent security vulnerabilities. Figure 2 shows applications of embeddings of SQL and Shell commands in PHP. In both cases, the StringBorg compiler guarantees that embedded sentences are syntactically correct and that strings are properly escaped, as opposed to the unhygienic string concatenation on the previous lines. The