Visual COKO: A Debugger for Query Optimizer Development

Daniel J. Abadi
Brandeis University
dna@cs.brandeis.edu

Mitch Cherniack
Brandeis University
mfc@cs.brandeis.edu

1. INTRODUCTION

Query optimization generates plans to retrieve data requested by queries. Query rewriting, which is the first step of this process, rewrites a query expression into an equivalent form to prepare it for plan generation. COKO-KOLA introduced a new approach to query rewriting that enables query rewrites to be formally verified using an automated theorem prover [1]. KOLA is a language for expressing term rewriting rules that can be "fired" on query expressions. COKO is a language for expressing query rewriting transformations that are too complex to express with simple KOLA rules [2].

COKO is a programming language designed for query optimizer development. Programming languages require debuggers, and in this demonstration, we illustrate our COKO debugger: Visual COKO. Visual COKO enables a query optimization developer to visually trace the execution of a COKO transformation. At every step of the transformation, the developer can view a tree-display that illustrates how the original query expression has evolved.

2. THE TOOL

A Visual COKO provides a similar interface to the standard debugging tool, gdb. Like gdb, Visual COKO includes the standard program execution control commands: *step*, *next*, *continue*, *break*, and *clear*. These commands parallel the C debugging commands in that, for the *step* and *next* commands, a call to another COKO transformation is treated like a function call.

Aside from providing control of transformation execution, Visual COKO also displays the current state of the rewriting computation after each step. This is accomplished by displaying the parse tree for the query expression being rewritten (expressed in KOLA). Visual COKO has three display options. The first option is to display the entire query tree. The second option is to display only the current branch that the transformation is working on. The final option is to display a branch of the tree that is bound to a temporary variable that will be later used in pattern-matching.

Like the term rewriting rules that they generalize, COKO transformations are either successful or unsuccessful upon firing. Similarly, every statement within a COKO transformation also succeeds or fails. These success values determine the flow of control in a COKO program. For example, entire sections of COKO code might be contingent on a successful pattern-match statement. The *success* option will, when activated, display the success value of every completed statement as they occur. By reporting the success values as they are generated, Visual COKO assists the user in locating logical errors in control flow.

Paramount in COKO transformations are the KOLA rules that they fire. The only way that COKO can alter a query is by firing a KOLA rule on that query (it is for this reason that to prove a COKO transformation correct, all one has to do is to prove the correctness of all its component KOLA rules). Visual COKO contains a *rule* option that, when activated, alerts the user every time a rule is fired on a query, whether or not the rule firing succeeded, and the input and output KOLA expressions.

The debugger also provides a conditional break command. Unlike gdb where conditions are based on data values, Visual COKO bases conditions on the success or failure of statement execution. That is, the debugger will stop on a statement with a conditional break only if the execution of that statement succeeded. This is useful for monitoring rule firings, because most attempts to fire rules fail, and therefore execution would stop infrequently. Conditional breaks are inserted by way of the *condition* command. Visual COKO also provides a transformation stack window through which breaks (conditional or standard) can be placed on any statement from any transformation that had called the current transformation.

Visual COKO enables a query optimization developer to visually trace the execution of a COKO transformation, one step at a time. In functioning both as a debugger and as a visual aid, Visual COKO facilitates the query optimizer development process.

3. THE DEMONSTRATION

Visual COKO and the COKO development process were demonstrated. Visual COKO's utility as a visual aid in tracing the execution flow of a transformation was demonstrated by stepping though various widely-used transformations. Visual COKO's utility as a debugger was shown with example transformations with subtle bugs, and by demonstrating development sessions that identified and removed them.

4. REFERENCES

[1] Cherniack, M. and Zdonik, S.B. Rule languages and internal algebras for rule-based optimizers. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, June, 1996.

[2] Cherniack, M. and Zdonik, S. Changing the Rules: Transformations for rule-based optimizers. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June, 1998

ACM SIGMOD'2002, June 4-6, Madison, Wisconsin, USA.