

Algebras For Automated Software Design

Don Batory and Jacob Neal Sarvela

Department of Computer Sciences

University of Texas at Austin

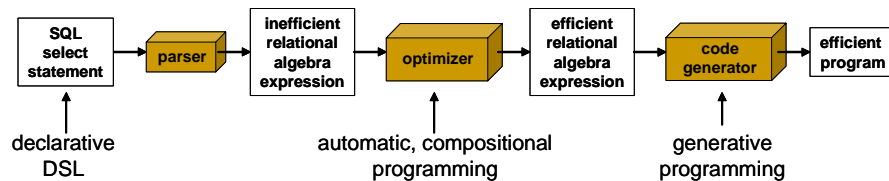
{batory, sarvela}@cs.utexas.edu

1 A Paradigm

The future of software engineering lies in automation. Critical to this vision are technologies that synthesize large-scale software systems — their designs, source code, and other related artifacts. Such technologies require advances in the following areas:

- *Generative Programming* — software domains are so well-understood that programs in the domain can be synthesized automatically;
- *Compositional Programming* — programs and their supporting tools and benchmarks that derive, verify, and analyze properties of these programs are assembled by composing prefabricated parts;
- *Domain-Specific Languages* — domain-specific notations simplify the specification and maintenance of programs;
- *Automatic Programming* — efficient programs are synthesized from declarative specifications.

To advance the above areas, let alone integrate them, is a daunting task. However, a spectacular example of their integration was realized over twenty-five years ago: *relational query optimization (RQO)*. A relational query is specified in SQL, a parser maps it to an inefficient relational algebra expression, a query optimizer optimizes the expression automatically, and an efficient query evaluation program is generated from the optimized expression.



SQL is a prototypical declarative *domain-specific language (DSL)*. Query evaluation programs are specified as compositions of relational algebra operators; relational algebra is a prototypical model of compositional programming. Automatic programming is achieved by query optimizers that rewrite an inefficient expression to a semantically equivalent but more efficient expression. Mapping a relational algebra expression to an efficient program is generative programming.

Query evaluation programs are notoriously hard-to-write, hard-to-optimize, and hard-to-maintain. RQO enabled these programs to be automatically generated. As a consequence, RQO had an enormous impact on database technology.

2 Some Results

A “holy grail” of Software Engineering is to replicate the RQO paradigm and success in other domains. Understanding how to do so has been the subject of our research for over twenty years. Among the key findings:

- Programs must be first-class entities in models of software design, where common modifications or extensions to programs correspond to operators on programs. The set of operators defines an algebra; compositions of these operators define particular programs that can be synthesized. The set of programs that can be synthesized is a *product-line*.
- Future programs will be very large; the scalability of algebraic representations is key to the scalability of compositional programming.
- Mixin-style inheritance refinement is insufficient [1]. Programming language support — far beyond mixins — is necessary to support algebraic composition.
- Compositions can be optimized by applying rewrite rules that are algebraic identities among operators. The resulting programs can be more efficient than that written by hand [3]. Algebraic identities are the basis for compositional reasoning about programs (software design) [4].
- Programs (software designs) have many different representations — source code, UML diagrams, makefiles, performance models, etc. Operators define a fundamental form of modularization, where modular representations of a program align along operator boundaries. For example, query optimizers maintain two different representations of each relational operator: code and cost model. Given a relational algebra expression:

```
E = join( select(...), select(...) )
```

an optimizer derives a cost model of that program by composing cost model representations of each operator:

```
Ecost = joincost( selectcost(...), selectcost(...) )
```

Optimizers use the above representation to identify the most efficient query evaluation program within a space of equivalent programs. Once this program is found, the code generator derives the source representation of the program by composing the code representations of these same operators:

```
Ecode = joincode( selectcode(...), selectcode(...) )
```

That is, the modularity and structure imposed on code is *exactly the same as that for cost models: they all align along operator boundaries*. In this way, consistent yet different representations of a design can be derived automatically from algebraic expressions. The same holds for other representations [2].

3 Goal

Application design must be transformed from an art into a science — a systematized body of knowledge that is organized around principles, ideally expressible as mathematics. This goal can be achieved by following the lead of relational database researchers in creating a science to specify, optimize, and synthesize programs by generalizing the RQO paradigm. It is a rich technical paradigm that integrates many areas of computer science.

4 References

- [1] D. Batory. “The Road to Utopia: A Future for Generative Programming”, *Dagstuhl on Domain-Specific Program Generation, March 2003*, <ftp://ftp.cs.utexas.edu/pub/predator/utopia.pdf>
- [2] D. Batory, J.N. Sarvela, and A. Rauschmayer. “Scaling Step-Wise Refinement”, *International Conference on Software Engineering (ICSE-2003)*, Portland, Oregon.
- [3] D. Batory, G. Chen, E. Robertson, and T. Wang. “Design Wizards and Visual Programming Environments for GenVoca Generators”, *IEEE Transactions on Software Engineering*, May 2000.
- [4] D. Batory and B.J. Geraci. “Composition Validation and Subjectivity in GenVoca Generators”, *IEEE Transactions on Software Engineering*, February 1997.