

New Programming Paradigms

Don Batory
Department of Computer Sciences
University of Texas at Austin



Setting the Stage...

Today's paradigm:

- think of programs as values
- modifications to programs are functions

`newProgram = function( program )`

Composing functions produce new programs or new versions of old programs

Of Course, the Problem is...

- # Effects of virtually all such functions are produced manually
 - costly, error prone, no productivity gains
 - ad hoc, ...
- # Future: **reusable** functions whose effects are computed **automatically**
 - current research identified two classes of reusable functions
 - generic and domain-specific

Examples: Generic Functions

Refactorings

- common OO program manipulations
 - move a method from a subclass to its superclass
 - automating application an OO design pattern
- tool support from vendors now appearing

Aspects (?)

- generic tools for extending, refining arbitrary programs

Generic because they work on all OO programs – they don't understand the **semantics** of the programs they effect

Examples: Domain-Specific

Feature-Oriented Programming

- relies on *premeditated designs*, product-lines
- function adds a *feature* to a program

Program = A(B(C))

- functions understand the **deep structure** and **semantics** of programs that they transform
- architecturally extensible – add and remove features at will
- benefits extend to **modular verification** (ex. model checking) as well

Example: Domain-Specific

Domain-Specific Languages

- raising the level of abstraction in programming

moreConcreteProgram = DSL(moreAbstractProgram)

- years of results show improved productivity, reduced maintenance, analyses, etc.
- oddly, most work on compilers deals with traditional issues (memory management, processor architecture optimizations)
- enormous world of DS compiler optimization problems awaiting compiler/language researchers

Conclusions

New paradigms satisfy same old model

- programs are values, functions are refinements
- **generic** or **DS** functions are **reusable** and **automatic**
- easy to recognize work that contributes to this paradigm
- **Common: move software design from art-form to science**
- Differences: approaches, implementations, problems addressed

Key issues for success:

- support for infrastructure (extensible languages, program transformations)
- funding research projects (tied to real problems)
- **“dating service” to link technology producers with technology customers**
 - increase effects of research ten-fold by getting ideas out to industry faster