Diagnosis of Harmful Aspects Using Regression Verification

Shmuel Katz

Computer Science Department

The Technion, Haifa, Israel

Do aspects applied to an original system cause harm?

- Assume the original system has a specification of its essential properties
- Show that the aspects maintain those properties (but can change others)
- Ignore the properties added by the aspects—at least "Do No Harm"
- Limits the obliviousness of the system to aspects applied over it; if "harm is caused", at least be aware of it.

Possible Approaches

- Regression testing
- Static code type analysis
- Deductive verification
- Model checking

Aspect code analysis: consider only the aspect code, (a) for families of systems or (b) for one instance

Augmented code analysis: consider the combination of the original and the aspects

Why not regression testing?

- Aspects make many changes at many points and can redirect control and results
- Entire computation paths/methods/fields are not tested
- Inherently global, for augmented system, and can demand excessive resources

Previous tests are often insufficient/irrelevant

Static aspect code analysis: Example—spectative aspects

- If the binding of aspect code to a system is only through explicit parameters, can see that only aspect fields are modified, and original control is unaffected
- Use data-flow techniques (*define-use* pairs)
- Thrm: For any original system, properties only involving original fields, methods, are not harmed by applying a spectative aspect.
- But: New method exposing a hidden value could be even in a spectative aspect …

Deductive verification for aspect code: Invariant extension

- IF *I* is an invariant of the original system, and is inductive, we can just show that

  $\{I\}\ t\ \{I\}$

holds for each action *t* of the aspect code, without considering when t is applied, and conclude that *I* is an invariant of the entire augmented system.

Useful example of aspect code analysis for a particular application, using info on original.

Example of invariant extension for a particular instance

- (x>y>0) is an invariant of some system
- An aspect has the form

  <complex> → double (x,y)

Then check {x>y>0} double(x,y) {x>y>0} and conclude (x>y>0) is an invariant of the entire augmented system

(Note: no need to analyze <complex>)

Using Aspect Validation for augmented system analysis

For situations where original system has been proven correct for its specification using software model checking (e.g., Bandera)

- Reprove for augmented system without new manual setup (just push a button…)
- Reuse the specification and annotations, given as verification aspects
- Treats all new paths/methods….
- In many cases uses the same abstractions

Conclusions

- Aspect code analysis for large families of properties/original programs---is best
- Sometimes static data-flow and simple inductions suffice for aspect code
- Otherwise augmented system analysis is sometimes inevitable—and a “validation” technique is recommended.
- Diagnosis of harm is a valuable step towards routine application of formal methods for aspect-oriented systems