Design by Contract
and
Static Verification

Bymer O’Hara

jameso@eecs.ku.edu
Organization

• Design by Contract (DbC)
• DbC implementations
• State-of-the-art in Java static verification
• Discussion questions
DbC

- Axiomatic semantics for imperative languages
- More recently applied to OO languages
- Behavioural specification
  - based on Hoare triple - \{P\} S \{Q\}
    - assertion Q will hold upon termination of S when executed in a state satisfying P
  - preconditions, postconditions, class invariants
- Short in syntax, rich in semantics
- Contracts checked at runtime
Four Levels of Contracts

1. Signature
2. Behaviour
3. Synchronization
4. Quality of Service (QoS)
Square Root Example

double sqrt(double x)
pre: x >= 0
post: sqrt(x) * sqrt(x) = x

* As a specification the postcondition is fine. If we used it for monitoring the code at runtime, we would not want to check equality between real numbers.
Benefits of Contracts

• More formal description than natural language description
• Aids reuse
• Monitor at runtime
• Eliminates “defensive programming”
Shortcomings of Contracts

• Monitored at runtime (not intended for static verification)
• Ad hoc practice of expressing contracts
Contract Language

• Behaviour Interface Specification Languages (BISLs)
• Functional
• Terms
  – quantification, frame axioms, model fields, access to prestate and poststate, exceptional cases, trace assertions, loop assertions, informal assertions
Organization

- Design by Contract (DbC)
- DbC implementations
- State-of-the-art in static verification
- Discussion questions
Eiffel

• First language with DbC built in
• OO
• IDE
• Commercial
• Features
  – functional
  – require, ensures, loop assertions, rescue, retry, assert
  – monitoring at different levels
JML

- More formal BISL for Java
- Rich specification language
- Preprocessor
- Features
  - model fields (API with purely functional classes), forall, exists, modifiable, old(expr), result, exceptional, informal assertions, loop assertions, etc.
  - lacks trace assertion
Contract Java

• Interfaces only
Organization

• Design by Contract (DbC)
• DbC implementations
• State-of-the-art in static verification
• Discussion questions
Static Verification

• In general, modular verification does not work
  – representation exposure (i.e. referential leakage)
  – behavioural subtypes (dynamic binding with subtyping)
  – frame problem with subtypes
Representation Exposure

• Definition
  – the transfer of modifiable components across abstract boundaries

• Problem
  – if there are multiple references to one object, changes to this object via one reference may affect the correctness of the objects holding other references.
  – not relevant if programs are constructed in such a way that they do not leak pointers.
Behavioural Subtypes

• Problem
  – behavioural subtypes could violate supertype contracts

• Solved!

  • Given: \{\text{Pre}_S\} S \{\text{Post}_S\}, \{\text{Pre}_T\} T \{\text{Post}_T\}, S <: T
    
    – strict solution
      • \{\text{Pre}_S \text{ OR } \text{Pre}_T\} S \{\text{Post}_S \text{ AND } \text{Post}_T\}

    – weaker solution
      • \{\text{Pre}_S \text{ OR } \text{Pre}_T\} S \{\text{Pre}_T \Rightarrow \text{Post}_T \text{ AND } \text{Pre}_S \Rightarrow \text{Post}_S\}
Frame Problem

• Frame axioms translate to extra postconditions which specify which variables may change.

• Definition
  – should behavioural subtypes be allowed to modify new fields?

• No satisfactory solution yet
Tools

• ESC/Java
  – not sound or complete

• LOOP
  – functional semantics in Hoare quintuples for non-classes
  – coalgebraic specification of classes
  – proof automation using predicate transformers

• JACK
WP Semantics

• wp(S,Q)
  – similar to Hoare semantics, but we do work writing predicate transformers which compute preconditions for free
  – predicate transformers mechanically compute the weakest precondition required for S to terminate in a state satisfying Q

• Example: assignment expression
  – wp(x := t, x = 5) = \{ t = 5 \}
LOOP Case Studies

- Java collections API
  - no referential leakage
- Java card application
  - found bug!
Conclusion

• Questions
  – If functional languages are easier to understand and allow for application of more powerful proof tools, why don't they enjoy more widespread use? Is the runtime performance of imperative alternatives that much better?
  – Are contracts useful for functional languages?