Automatic Verification of Computer Programs

What is verification

- Check correctness of the implementation given the specification
- Static verification
  - Check correctness without executing the program
  - E.g. static type systems, theorem provers
- Dynamic verification
  - Check correctness by executing the program
  - E.g. unit tests, automatic testing
- Automatic verification
  - Push-button verification

How to get the specification

- Need machine-readable specification for automatic verification (not just comments)
- Different variants:
  - Eiffel‘s „Design by Contract“
    - With new **across** construct even features quantifiers
  - .Net 4.0 „Code Contracts“
    - Implmeneted contracts as a library
  - JML „Java Modeling Language“
    - Dialect of Java featuring contracts with special comments
- Writing expressive specification is difficult

Dynamic Verification

- Execute program and check that execution satisfies specification
- Manual
  - Write unit tests (xUnit framework)
  - Execute program and click around
- Automatic
  - Random testing
    - Generate random objects
    - Execute random routines

Random Testing

- Select routine under test
- Precondition used for input validation
  - Test is valid if it passes precondition
- Postcondition used as test oracle
  - Test is successful if it passes postcondition

- Improvements to random testing by making input selection smarter (e.g. use linear solver)

Random Testing

- Create random objects
  - Call random creation procedure
  - Call random commands
  - For arguments, generate random input
- Random basic types
  - Interesting values: Void, (2^31-1), 1, 0, -1, …
- Build object pool

Static Verification

- Need a model of the programming language
  - What is the effect of an instruction
- Translate program to a mathematical representation
- Use an automatic or interactive theorem prover to check that specification is satisfied in every possible execution

Translation to Boogie

```
make
        local
                a: ACCOUNT
        do
                create a.make
        end
```

```
implementation APPLICATION.make {
  var temp_1;
entry:
  havoc temp_1;
  assume (temp_1 != Void) &&
                        (!Heap[temp_1, $allocated]);
  Heap[temp_1, $allocated] := true;
  Heap[temp_1, $type] := ACCOUNT;
  call create.ACCOUNT.make(temp_1);
}
```

Translation to Boogie

- Translates AST from EiffelStudio to Boogie
- Uses Boogie verifier to check Boogie files
- Traces verification errors back to Eiffel source

Verification Demo

- EiffelStudio: AutoTest
- EVE: AutoProof
- VisualStudio: CodeContracts

Automatic Fault Correction

- Build a test suite
  - E.g through automatic testing
- Find and localize faults
  - Failing test cases
  - Static analysis
- Try fixes
  - Apply fix templates and random code changes
- Validate fixes
  - Run test suite again, now all tests have to pass

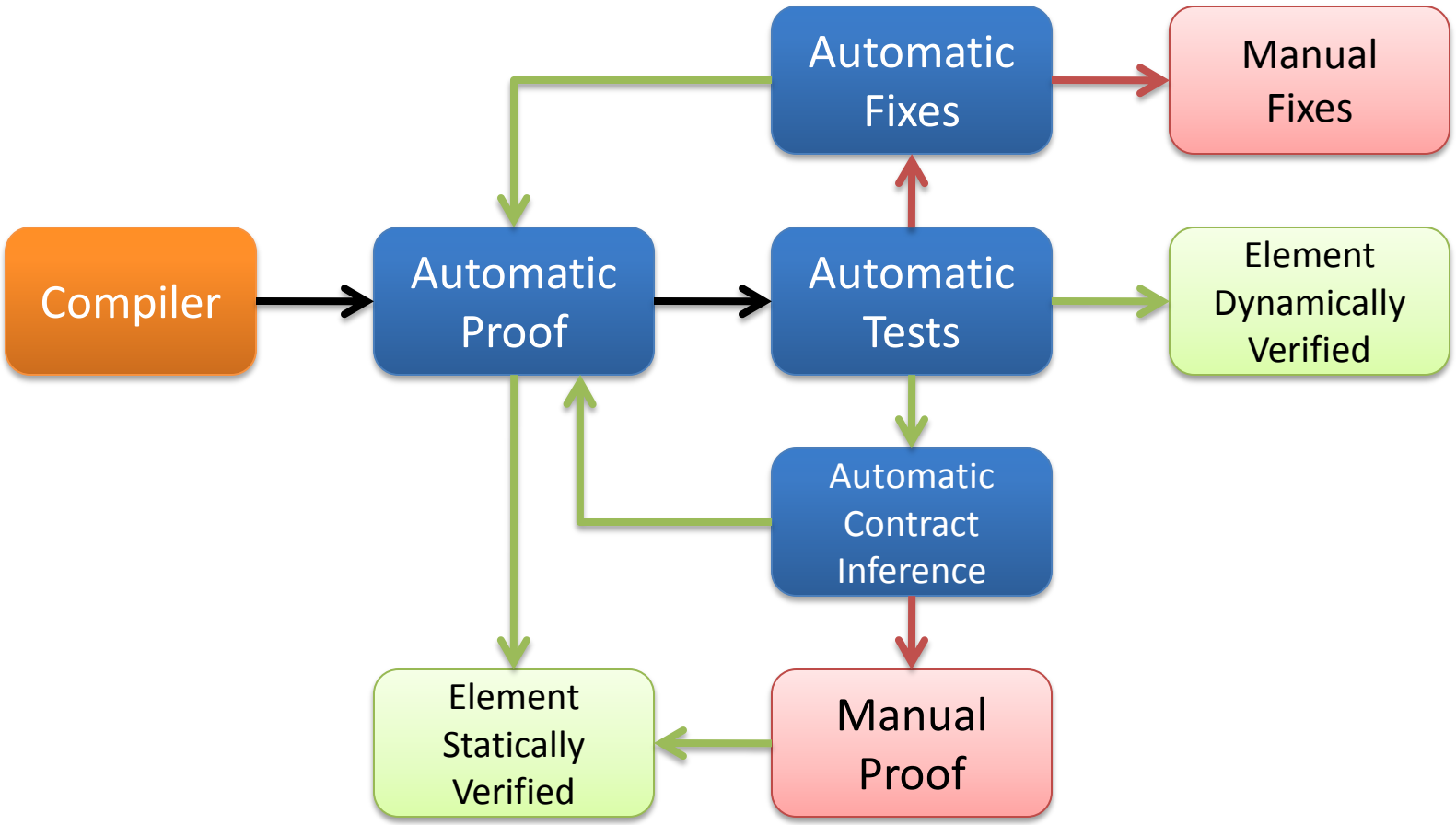
Dynamic Contract Inference

- Build a test suite
  - E.g through automatic testing
- Run program and store interesting values in interesting program points
  - E.g. argument values on feature calls
- Post-analyze values and recognize patterns
- Propose new contracts based on patterns

Static Contract Inference

- Infer precondition from postcondition or other assertions
  - Weakest precondition calculus
- Infer loop invariants from postcondition
  - Generate mutations from postcondition

Putting It All Together



Demo

- VAMOC video

References

- EVE: http://se.inf.ethz.ch/research/eve/
- AutoTest, AutoProof, AutoFix, CITADEL, …: http://se.inf.ethz.ch/research/
- CodeContracts: http://research.microsoft.com/en-us/projects/contracts/
- JML: http://www.cs.ucf.edu/~leavens/JML/