Testing a Computer Algebra System

Jacques Carette
McMaster University
Outline

- Introduction to Computer Algebra
- Testing - big picture
- Testing – details
- Summary
Introduction to Computer Algebra

- Maple will be shown
- Mathematica and MuPAD are largely similar
Testing – Big Picture

- **Fundamental Axiom:**
  - Testing $\equiv$ Automatic Testing

- **Axiom 2:**
  - Bugs are not closed until an automated test has been created
Statistics

- 11973 test files, ~40000 test cases
- Run on 14 `platforms’ nightly @Maplesoft
- +5 extra at research labs around world
  - Roughly 750K test cases run daily
- Takes ~8hrs on 2.4Ghz Linux PC
  - Uses 298.3 Gigs of memory
  - Allocates 33.4 Gigs
Infrastructure

- SCM
- Test suite DB
- Source code DB

- A = Test failure
- B = Recent changes

- (changes+author, sources)
- (tests, functions called)
- (sources, functions defined)

- A' = functions used
- B' = functions changed

- A' \cup \exists \cup \nabla \lor \exists \cup \exists B' = likely problem

- Email author. Email manager?
Infrastructure

SCM (changes+author, sources)
Test suite DB (tests, functions called)
Source code DB (sources, functions defined)

- Preventative

- rtest `rfindtest solve/rec`
Infrastructure

SCM  (changes+author, sources)
Test suite DB  (tests, functions called)
Source code DB  (sources, functions defined)
Test suite DB  (tests, time + memory used)

- 40 days of data
- Use z-score to get ‘real’ changes
  - Timings are noisy
  - Automated report
- Use student-t test for global trends
  - Can detect 0.5% slowdown with 1% noise
Robocop

- Given a test that fails,
  - Find most recent change that may be cause
  - Back out that change (locally)
  - Re-run test
  - Analyse results
- Repeat (backwards in time) if failure still present
- Works for failure as well as resource usage issues
Testing – Details

- Basic design

  problem := define_problem();
  answer := compute(problem);
  expected := expected_answer();
  verify(test#, problem, answer, expected);
Testing – Details

- Sample test

```plaintext
#test
with(inttrans):
  r1 := 'r1':
  TRY(1, assign(r1, laplace(arctan(-2/5*t), t, s)));
  TRY(2, eval(laplace(arctan(x*t), t, s), x=-2/5), r1) assuming Re(x)<0;
  r1 := 'r1':
  TRY(3, assign(r1, laplace(arctan((I-2/5)*t), t, s)));
  TRY(4, eval(laplace(arctan(x*t), t, s), x=I-2/5), r1) assuming Re(x)<0;
  r1 := 'r1':
  TRY(5, assign(r1, laplace(arctan(I*t), t, s)));
  TRY[verify,simplify](6, eval(laplace(arctan(x*t), t, s), x=I), r1)
      assuming Re(x)=0,Im(x)>0;
#end test
```
Testing - Failure reports

- Pass/Fail is only so useful
  - Need to know **why** a test failed
- First try: produce detailed output
  - Input, output, expected output
  - Problem: non-determinism + zero-testing
- Second try: produce script to reproduce
  - Input, output, command, expected output, all as a Maple script that can be re-executed
  - Very useful when testing long sequences
Testing - Selection

- For Unit tests:
  - + and not-not.
- Minimum wanted coverage:
  - Structural: all code, all data shapes
  - Semantic: all specification cases, book cases
- Hopeful coverage:
  - Structural: all paths, all data cases
  - Semantic: all book cases, functionally needed cases
Random Testing

- Done via generators
  - Generate random samples of a given length from data given by a grammar (CFG)
  - Grammar can describe data syntactically or semantically

- Good way to generate problems:
  - Generate answer
  - Invert computation to get problem
  - Solve problem forward
  - Compare
  - Think of testing an Eigenvalue solver for complex symmetric matrices
Summary

- Easier:
  - Mostly stateless
  - Automation
  - Integrated infrastructure
  - Introspection
Summary

- Harder:
  - Equivalence problem
  - Specification = `classical mathematics`
  - Non-determinism. Eigenbugs.
  - Testing `pdsolve` involves ~50% of the library
    - 72938 if statements in the library
  - Untyped