Proving The Absence of Bugs Using Tests

DSLab

Can Cebeci

George Candea

Symbolic testing enables developers without verification expertise to formally prove correctness using a common testing-based paradigm

Current methods for building high-assurance software

Testing

- Can only discover bugs, not prove correctness
- High-coverage test suites are costly
- Widespread industry adoption

Verification

- Either restricts the target system (no loops, pointers, or environment calls)
- Or requires (prohibitively costly) manual proofs
- Minimal industry adoption

Reconciling the two approaches: tests encode developer insight

- Developers write (symbolic) tests that check functional correctness
- Candidate invariants inferred from test suites guide automated verification

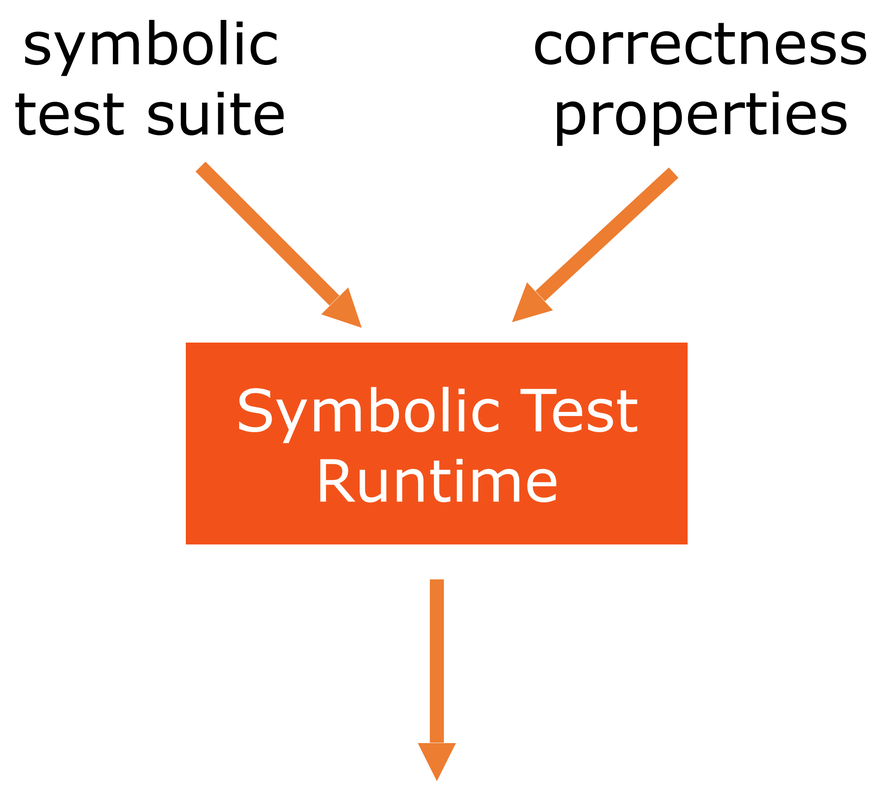
```
def symbtest_dup_success():
    # pid is the id of the current process
    pid = state.current_proc

    # process has a free file descriptor
    assumeExists(lambda x: fd_valid(x) and
        not fd_in_use(state.procs[pid], x))

    # let fd be an arbitrary used file
    # descriptor in pid's open file table
    fd = get_symbolic_var("fd", int)
    assume(fd >= 0 and fd < FD_MAX)
    assume(fd_in_use(state.procs[pid], fd))

    res = syscall("dup", pid, fd)

    # the syscall should not return an error
    assert(res >= 0)
    # res should be a used file descriptor
    assert(fd_in_use(state.procs[pid], res))
```



- test coverage (lines, paths)
- failing assertions (concrete)
- proof or counterexample for each property

Higher coverage for the same testing effort

Proofs of safety and correctness without proof-writing effort

Want to work on something related? Talk to us!