Combining Formal Methods and Industrial Pragmatics

Eric L. McCorkle

November 2, 2016

Dependent Types (very briefly)

Dependently-typed languages are equipped with very powerful type systems

- Type systems are strong enough to express full specifications
- Type-checking amounts to proving implementations behave according to spec
- Must still get the specification right! (Who watches the watchers?)
- Proofs resemble code, must be developed and maintained like code
- Powerful tool for building security in!

"Industrial" Programming Languages

What makes a language successful in the "Real World"?

- Realities: very large codebases, code evolution, staff turnover, differing skill-levels, cost/benefit tradeoffs, compatibility, etc.
- Resist bit-rot, withstand inelegance, hold up under refactoring
- "Harm-reduction" often works better than "thou shalt"

What tends to work well?

- Optimize for least eventual cost (or pain)
- Modularization, encourage good practices, code reuse
- Present complex ideas in an accessible fashion
- API/library design is an art-form to be celebrated

Vision of "Industrial" Dependently-Typed Languages

How can we make depndent types and verification suitable for industrial programming?

Gradual Proof Checking (and Typing)

- Making people prove their entire program correct before running it is a non-starter for industry
- People develop software iteratively
- Cost/benefit tradeoffs, risk profiles, ROI differ over components
- "Two-phase" type/proof-checking: first phase is decidable, second phase does verification.
- Prove critical components correct, rely on testing for the rest, gradually work your way outward
- Provides a smooth transition from prototypes to verified systems
- Go a step further: do this with type checking in general (gradual typing)

Managing Large Verified Codebases

How would we manage large bodies of proofs about code?

- Proofs very closely resemble code
- Provide ability to automate proofs using the same language as the code
- Apply known techniques that work for code management: modularization, small units of functionality, API design
- Draw on historically successful language concepts (OO features, typeclasses, etc) to design constructs for managing verification
- Draw on (and perhaps refactor) parts of mathematics, particularly abstract algebra

A Vision of Industrial Dependent-Typed Languages

- View as a specification/reasoning system built into the language
- Proof obligations provided as an artifact of compilation, usable to other tools)
- Gradual "pay-as-you-go" typing and proof-checking
- Start with no spec, leave proof obligations unproven
- Develop spec iteratively, prove obligations where advantageous
- Proofs look like code, make use of traditional software engineering techniques
- A fully-mature module has specs, proofs, and facilities for automating proofs about the module

- Happy to discuss these ideas in greater detail
- I am actively working on a language to implement these ideas
- Particularly interested in how to improve infosec through better languages
- Email me (eric@metricspace.net) or come find me to talk more