Applied Formal Methods for Software Product Line Research

Joseph Kiniry
University College Dublin

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Formal Methods
History and Update

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Misunderstandings in Formal Methods

- over-promising and under-delivering
  - a standard problem of some domains
- formalism without application
  - pure math for maths sake
- focus only on toy problems and languages
  - difficult problems plus limited resources
- no application in industry
  - if you believe that, give me back your CPU

Modern Applied Formal Methods

- we have moved from the “bronze age” to the “iron age” in applied formal methods
- a combination of factors
  - mathematical sophistication and powerful logical frameworks
  - powerful enough machines
  - a new generation of researchers egotistical enough to believe they could succeed where prior researchers have failed

“Submarine” Applied Formal Methods

- programming language semantics
  - type systems, compilers, static checkers
- CASE and RAD tools
  - UML (and OPEN, BON, etc.)
- hardware design languages and CAD tools
  - design, digital and analog simulation, automated and manual layout, etc.

SPL Projects Ideas

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Formalization for Visualization

- Project Idea #1
  - variability focus
  - efficient data traversal for visualization
- key requirements for formalization
  - parameterizable structure
  - partial structure-preserving operations
  - identification and preservation of key aspects of semantic interest

Conformance Analysis

- Project Idea #2
  - formalization of domain, application, and physical artifacts
- key requirements for formalization
  - parameterizable structure
  - structure-preserving operations
  - first-class refinement of artifacts
  - well-formedness condition testing

Formal Product Line Derivation

- Project Idea #3
  - automation focus
  - specs at many levels and granularities
- key requirements for formalization
  - parameterizable structure
  - structure-preserving operations
  - first-class refinement of artifacts
  - well-formedness condition testing

Software/Hardware Codesign

- Project Idea #4
  - dependency focus
- key requirements for formalization
  - parameterizable structure
  - well-formedness condition testing
  - use of pre-existing formal foundations

Requirements Analysis for Formalization

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Key Requirements for Formalization

- wide variety of artifacts
- parameterizable structure
- (partial) structure-preserving operations
- well-formedness condition testing
- first-class refinement of artifacts
- use of pre-existing formal foundations

Variety in Artifacts

- we must reason about and manipulate many kinds of artifacts
  - documentation, requirements, design, specification, program code, binaries, hardware
- the semantics of (some) artifacts are vague, wide-ranging, and contextual
- formalization must be hidden from the user---implies simplicity and automation

Parameterizable Structure

- parameterization is a first-class citizen
- classifiers with structure are necessary
- parameterization must be clear and represent variability in useful ways
- we must be able to reason with and about parameterization

Structure-preserving Operations

- operations on formal structures must represent operations on informal artifacts
- preserved structure and meaning must be well-understood
- we must be able to reason about operations
- operations must not be opaque and must be efficient

Well-formedness Condition Testing

- multiple abstraction levels in use
- ensure that models at different levels properly preserve structures of interest
- well-formedness testing must be automatic and precise

First-class Refinement

- multiple levels of granularity and abstraction are evident
- refinement in multiple dimensions
  - between levels (abstraction)
  - over time (evolution)

Incorporating Existing Formalisms

- reasoning about hardware and software
- several powerful, useful, and widely-used existing formalisms in hardware domain
- multi-domain integration means
  - informed choices in formalisms
  - appropriate tool choice for automation
  - consideration of social issues of domain

Potential Choices Derived from Requirements

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Choices in Formal Foundation

- classical logical formulation
  - e.g., first-order logic + set theory
- higher-order logic formulation
- type theory - develop new type systems for reasoning about structure
- many order-sorted algebras - develop and reason about algebraic models
- category theory - underlying theory

Choices in Tools for Mechanization

- first-order reasoning (e.g., SAT solvers, decision procedures, first-order provers)
- higher-order provers (e.g., PVS and Coq)
- algebraic frameworks (e.g., Maude)
- model checkers (e.g., SPIN and Bogor)

Choices in Languages for Specification

- UML
  - semantic morass
- well-defined analysis and design languages with semantics (e.g., BON, OPEN, etc.)
  - few existing tools => tool development
- semantic web-based techniques
  - avoiding the hard problems, tool issues
- new structured approaches a la Parnas (?)

Concrete Suggestions

- use PVS and/or Coq for formalization
- use (an evolution of) BON or a similar language for specification of artifacts' properties
- wholly rely upon existing theories of refinement and parameterization
- build rich tools while leveraging other Irish research and expertise

Irish Expertise

- DCU - program logics, semantics, testing and metrics, software architectures, crypto
- UCD - semantics, reuse, tool development, software engineering with AFM
- TCD - semantics, VDM/Z/Circus, hardware formalisms, concurrent and distributed systems, patterns, SOP, AOP
- UL - software process and methods, CSCW, software quality assurance

PVS and Coq

- interactive theorem provers
- both use higher-order logic
- rich type and module system
- first-class parameterization
- powerful reasoning capabilities, automation, reasonable scalability
- integration with new rich IDEs

EBON: Extended BON

- one language for structured specification
- ranges from informal to formal specs
- rich types and seamless refinement built-in
- pre-existing, extendible semantics
- approachable by non-programmers
- visual and textual syntax
- missing: no existing parameterization at most informal level of refinement

Underlying Formalism

- underlying theory specified in a concrete set of higher-order theories for the problem domain
- always keep in mind application of theory to problem and realization in tools
  - e.g., avoid higher-order reasoning as much as possible, use automated tools when appropriate (first-order provers and model checkers)

Underlying Formalism

- use algebraic and categorical foundations
- concrete domains of application
  - abstract interpretation
  - model checking
  - refinement calculus

Open Questions

- are executable specifications useful in the SPL domain?
- what are the appropriate toolbenches for integration?
- how important is tool integration?
- how much formalism to hide?
- how much formalization of existing toolsets and how much from-scratch work?

A Formal Product Line Engineering Framework

- need we define and build such a beast?
- would an Open Source framework be a significant Lero outcome?
- how small-but-formal could it be made?