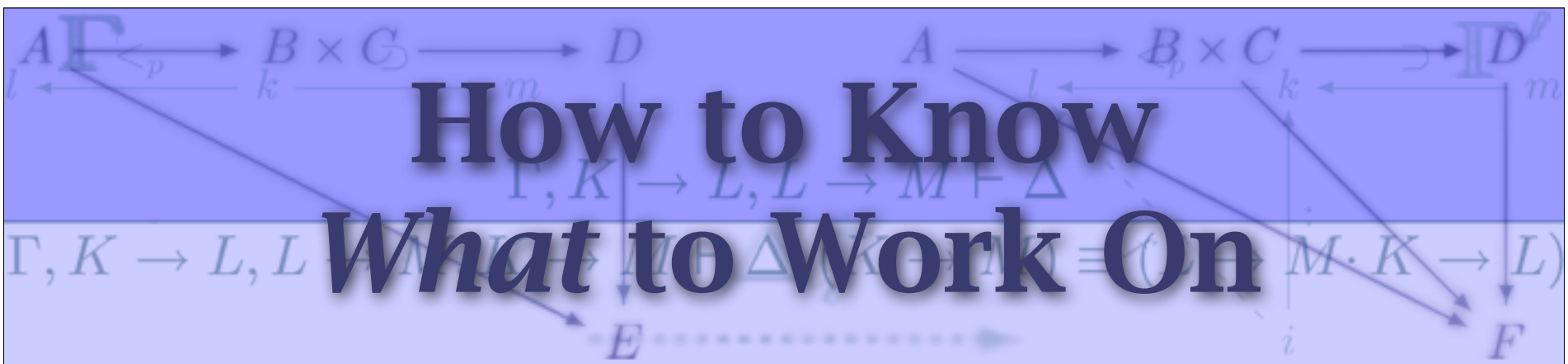


Usable Formal Tools

Joseph Kiniry

Writing Usable Software

- * creating usable, useful, attractive constructs is very difficult
- * a new, rare set of skills are necessary
- * aesthetics and psychology trump logic
- * all software is usable to its author
- * some software is usable to the neighbor
- * little software is usable to the punter
- * all software **can be** usable to the grandma



How to Know What to Work On

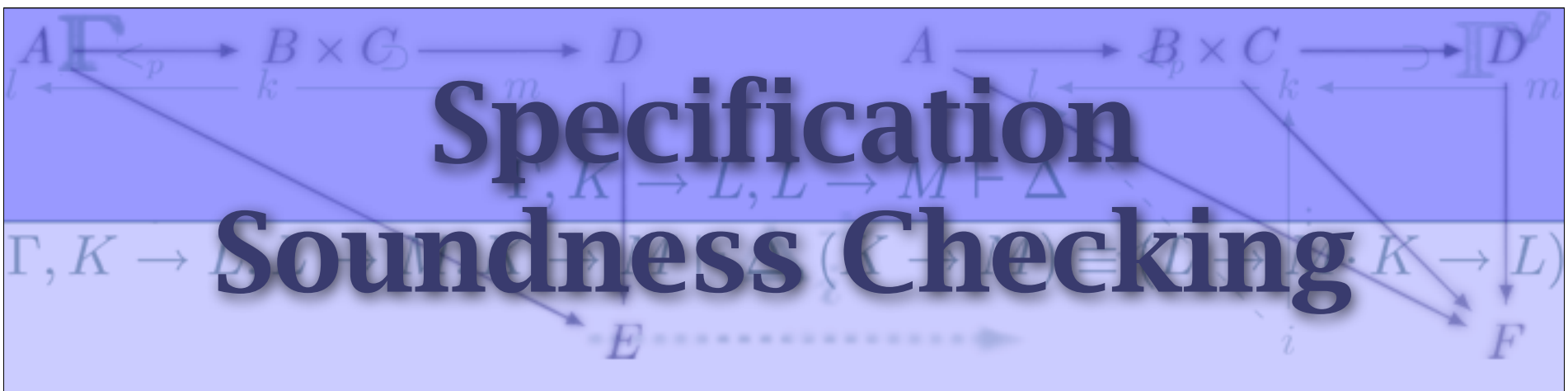
- ✧ find users that are thick-skinned but vocal
- ✧ find passionate developers to help
- ✧ provide quality user and developer support
- ✧ ask your users what **they** want
- ✧ identify, classify, describe, and **give away** interesting problems indiscriminately
- ✧ **use the tools yourself**

Examples of Usability Improvements

- ✧ user-requested features for ESC/Java2
- ✧ some features are small, require no “deep thought,” nor are publishable
- ✧ other features are complex, require hard work (intellectual and physical), but are difficult to publish
- ✧ few are complex but easy to publish
- ✧ but hundreds of tool users is much more valuable than dozens of paper readers

Soundness and Completeness Warning System

- ✧ ESC/Java2 is neither sound nor complete
- ✧ one of the biggest complaints, but less from **users** and more from **FMers**
- ✧ characterizing S&C issues is complex:
 - ✧ relates to verification methodology, object logic, calculus, theorem prover, and the specification, source, and bytecode



Specification Soundness Checking


- ✧ naive users quickly check large programs
- ✧ specification soundness/consistency problems are usually quite subtle
- ✧ how to identify key problem terms in large, complex specifications is quite difficult
- ✧ mechanizing manual techniques is only part of the solution

Eclipse Integration

- ✧ bells & whistles and blinking lights gain more users than any formal property
- ✧ do not tie tools solely to one environment
 - ✧ command-line, Emacs, and the IDE are the holy trinity of tool adoption
- ✧ Eclipse is the current developer favorite
- ✧ all IDEs require a steep learning curve that flattens after a few months of immersion

Reporting Verification Condition Complexity

- * the many scaling problems in verification:
 - * e.g., # types, classes, programmers; proof state, specification, program, method size
- * **timely** tools are **useful** tools
- * non-interactive checking must be fast
- * naive programmers do not understand the intricacies of VC generation and proving
- * provide a simple complexity measure



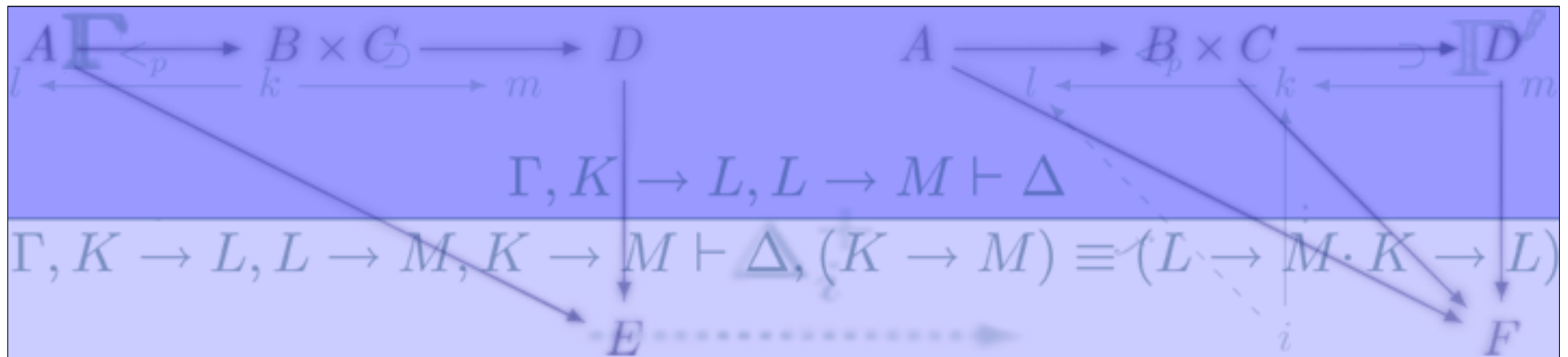
A Brainstorming Activity

- ✧ I lied.
- ✧ Reporting VC complexity is actually quite difficult to do right well.
- ✧ Let's brainstorm!
 - ✧ How might we display VC complexity to users, both in text and graphically?
 - ✧ How to provide quality feedback, given the variances of provers and object logics?



Challenges (for Tools and Tool Builders)

- ✧ finding and keeping users
- ✧ forcing oneself to become a user
- ✧ obtaining resources for development
- ✧ leveraging others' hard work
- ✧ integrating with other tools
- ✧ research community and university recognition for tool research



Questions?
Comments?