Correct for all inputs vs. working on a large class of programs/problems?
narrow domain ==> correct by construction or provable
broader domain or larger programs ==> probably unprovable but still beneficial as a tool

How strong does the proof need to be to give you a reasonably correct program?
depends on the domain: banking web page vs. angry birds

Approximate computing, eg image decompression, web search
answer can be incorrect, to a degree
can we synthesize these programs?
the problem: tradeoff between resources and accuracy
needs a theory for reasoning about approximately correct programs

What artifact would be interesting to synthesize, and what's the motivation?
• inference of abstractions or invariants
• code and documentation
• generating use cases of APIs, as a documentation
• retargeting a program to a new version of an API (eg a bug was fixed under an API)
• retargeting a program to a new hardware, eg adjust matrix block sizes to adopt to new caches; especially for domains not handled by FFTW/Spiral
• synthesize a faster synthesizer: synthesize (the math for) schemas for a new domain
• automatically decompose a problem so that that outsource the computation
  • synthesize a spec, a design, an architecture
• synthesize a DSL compiler
• boilerplate code ==> patterns or libraries
• synthesis of coordination of two half-co$$ect programs
• synthesize of communication, to enable modularity

Schemas = syntactic constraints on artifacts (programs, invariants, abstractions)
• how to put all this on a common formal footing (semantics of schemas?)
• hierarchy of schemas
• how to combine schemas
• a common calculus of schemas
• a common informal language for researchers to communicate
• a common "SMTLIB" format for completing schemas?
• a formal language for combining logics

Modular synthesis?
• decompose a big problem into smaller problem
• customization/parametrization of an existing solution
Interactive synthesis
  • especially consider interfaces for end users (PBD)
  • talk to HCI folks

Where do specifications come from? What are they?
  • even more than in verification (cf. Harel quote) is a linguistic problem (language design, programming abstractions design)

What we should **not** synthesize because there are simpler/better solutions?
  • don't synthesize sophisticated algorithms (don’t replace Knuths)
  • synthesize boring or hard repetitive tasks like corner cases, initializations

Who are the users?
  • domain experts: scientists, statisticians
  • end users: excel, web browser,
  • programmers
  • managers and professors
  • education: teaching tools