The Trellys Project

Tim Sheard
Aaron Stump
Stephanie Weirich
Trellys

• A multi-year, multi-institution, coordinated project to develop the dependently-typed programming language Trellys

• A Community-Based Design process – inspired by the Haskell Committee in the 1980s

• NSF funded project to coordinate the effort
  – Portland State University (Tim Sheard)
  – University of Iowa (Aaron Stump)
  – University of Pennsylvania (Stephanie Weirich)
Letters of Support

- Andreas Abel, Munich
- Thorsten Altenkirch, Nottingham
- Lennart Augustsson, Standard Chartered Bank
- Bruno Barras, INRIA Saclay
- Edwin Brady, St. Andrews
- Peter Dybjer, Chalmers
- Cormac Flanagan, UC Santa Cruz
- Conor McBride, Strathclyde
- Greg Morrisett, Harvard
- Ulf Norell, Chalmers
- Simon Peyton Jones, Microsoft Research
- Frank Pfenning, CMU
- Brigitte Pientka, McGill
- Philip Wadler, Edinburgh
- Hongwei Xi, Boston U.
Goals

• Build tools to support the cost-effective construction of functionally correct software systems – A dependently typed programming language
Why dependent types?

• Verification at the source level, part of the development process
• Verification is incremental, richer types means more properties. Pay as you go
• Verification is modular, unrelated changes do not invalidate proofs
Trellys Platform Architecture

- **Presentation Language**: Heuristic-based strong type & proof inference, perhaps tailored to specific domain.
- **Elaborator**: General purpose, fully explicit, simple type & proof checking.
- **Core Language**: Uses explicit information (about staging, purity, etc.) to produce efficient code.
- **Compiler**: Machine code.
- **Language Runtime and Standard Library**: Heterogeneously typed, support for integration with foreign code.
Core Language

• Focus of initial design effort Fall 2009
• Explicitly typed
• Call-by-value semantics
• Full-spectrum dependency
• Explicit erasure annotations
  – Equality defined in terms of implicit language
• Termination analysis to enforce logical consistency
• Compatible with classical reasoning
Presentation Language

• Text based universal language
• Programs are written, not extracted
• Typing annotations are minimal
• Many type casts are inferred in translation to core
Compilation

- Type-preserving compilation
- Goal: to reflect (observational) equalities deep into the compilation pipeline
- Programs marked explicitly as erasable are removed, leading to efficient code
Core Language: Design Questions

• What sort of termination analysis to use?
  – If any!
  – Not clear there is a sweet spot

• How expressive should we make the logic/language?
  – Can we make *:* compatible with termination analysis

• Should we separate proofs from programs?
  – Is a proof just a terminating, irrelevant expression, or do we need finer distinctions

• What notion of equality should we use for conversion?
  – Must be compatible with the operational semantics
Community based effort

• Small group of external experts (Simon Peyton Jones, Wouter Swierstra, Conor McBride, Bruno Barras) assisting with initial core language design
• A wider call for participants to review core-language effort in the Summer 2010
• A open call for participants in a Trellys workshop in January 2011
• Regular meetings throughout duration of project