Combining Proofs and Programs

Stephanie Weirich

University of Pennsylvania

September 2011

Dependently Typed Programming

Shonan Meeting Seminar 007

The TRELLYS project

The TRELLYS project

University of Pennsylvania	The University of Iowa	Portland State University
Stephanie Weirich	Aaron Stump	Tim Sheard
Chris Casinghino	Harley Eades	Ki Yung Ahn
Vilhelm Sjöberg	Peng (Frank) Fu	Nathan Collins

Garrin Kimmell

A collaborative project to design a statically-typed functional programming language based on dependent type theory.

The TRELLYS project

University of Pennsylvania	The University of Iowa	Portland State University
Stephanie Weirich	Aaron Stump	Tim Sheard
Chris Casinghino	Harley Eades	Ki Yung Ahn
Vilhelm Sjöberg	Peng (Frank) Fu	Nathan Collins

Garrin Kimmell

A collaborative project to design a statically-typed functional programming language based on dependent type theory.

Work-in-progress

Growing a new language

Trellys Design strategy: Start with general purpose, call-by-value, functional programming language and strengthen its type system.

Why call-by-value?

- Have to choose something. With nontermination, the order of evaluation makes a difference

Why call-by-value?

- Have to choose something. With nontermination, the order of evaluation makes a difference
- Good cost model. Programmers can predict the running time and space usage of their programs

Why call-by-value?

- Have to choose something. With nontermination, the order of evaluation makes a difference
- Good cost model. Programmers can predict the running time and space usage of their programs
- Distinction between values and computations built into the language. Variables stand for values, not computations

Programming language vs. logic

Even in the presence of nontermination, a call-by-value dependently-typed *programming language* provides *partial* correctness.

Theorem (Syntactic type soundness)

If $\vdash^{\mathsf{P}} a : A$ *then either* a *diverges or* $a \leadsto^{*} v$ *and* $\vdash^{\mathsf{P}} v : A$.

Programming language vs. logic

Even in the presence of nontermination, a call-by-value dependently-typed *programming language* provides *partial* correctness.

Theorem (Syntactic type soundness)

If $\vdash^{\mathsf{P}} a : A$ *then either* a *diverges or* $a \leadsto^* v$ *and* $\vdash^{\mathsf{P}} v : A$.

A dependently-typed *logic* provides *total* correctness.

Theorem (Termination)

If $\vdash^{\mathsf{L}} a : A$ *then* $a \leadsto^* v$ *and* $\vdash^{\mathsf{L}} v : A$.

Partial correctness

Type soundness alone gives a logical interpretation for *values*.

$$\vdash^{\mathsf{P}} a : \Sigma x\!:\!\mathsf{Nat}.\mathit{even}\ x = \mathit{true}$$

If a terminates, then it *must* produce a pair of a natural number and a *proof* that the result is even.
Canonical forms says the result must be (i, join), where $\mathit{even}\ i \leadsto^{*} \mathit{true}$ by inversion.

Partial correctness

Type soundness alone gives a logical interpretation for *values*.

$$\vdash^{\mathsf{P}} a : \Sigma x\!:\!\mathsf{Nat}.\mathit{even}\ x = \mathit{true}$$

If a terminates, then it *must* produce a pair of a natural number and a *proof* that the result is even.
Canonical forms says the result must be (i, join), where $\mathit{even}\ i \leadsto^* \mathit{true}$ by inversion.

But, implication is bogus.

$$\vdash^{\mathsf{P}} a : \Sigma x\!:\!\mathsf{Nat}.(\mathit{even}\ x = \mathit{true}) \to (x = 3)$$

Total correctness

Partial correctness is not enough.

- Implication is useful
- Can't compile this language efficiently (have to run "proofs")
- "Proof" irrelevance is fishy
- Users are willing to work harder for stronger guarantees

A logical language

- But, some programs do terminate. There is a terminating, logically-consistent logic hiding in a dependently-typed programming language.

A logical language

- But, some programs do terminate. There is a terminating, logically-consistent logic hiding in a dependently-typed programming language.
- How do we identify it?

A logical language

- But, some programs do terminate. There is a terminating, logically-consistent logic hiding in a dependently-typed programming language.
- How do we identify it?
- We use the type system!

A logical language

- But, some programs do terminate. There is a terminating, logically-consistent logic hiding in a dependently-typed programming language.
- How do we identify it?
- We use the type system!

A logical language

- But, some programs do terminate. There is a terminating, logically-consistent logic hiding in a dependently-typed programming language.
- How do we identify it?
- We use the type system!

New typing judgement form:

$$\Gamma \vdash^{\theta} a : A \quad \textit{where} \quad \theta ::= \mathsf{L} \mid \mathsf{P}$$

Subsumption

Many rules are shared.

$$\frac{\vdash \Gamma \quad x :^{\theta} A \in \Gamma}{\Gamma \vdash^{\theta} x : A} \qquad \frac{\Gamma \vdash^{\theta} b : \mathsf{Nat}}{\Gamma \vdash^{\theta} \mathsf{S}\ b : \mathsf{Nat}}$$

Subsumption

Many rules are shared.

$$\frac{\vdash \Gamma \quad x :^{\theta} A \in \Gamma}{\Gamma \vdash^{\theta} x : A} \qquad \frac{\Gamma \vdash^{\theta} b : \mathsf{Nat}}{\Gamma \vdash^{\theta} \mathsf{S}\ b : \mathsf{Nat}}$$

Programmatic language allows features (general recursion, type-in-type, abort etc.) that do not type check in the logical language.

$$\frac{}{\Gamma \vdash^{\mathsf{P}} \star : \star}$$

Subsumption

Many rules are shared.

$$\frac{\vdash \Gamma \quad x :^{\theta} A \in \Gamma}{\Gamma \vdash^{\theta} x : A} \qquad \frac{\Gamma \vdash^{\theta} b : \mathsf{Nat}}{\Gamma \vdash^{\theta} \mathsf{S}\ b : \mathsf{Nat}}$$

Programmatic language allows features (general recursion, type-in-type, abort etc.) that do not type check in the logical language.

$$\frac{}{\Gamma \vdash^{\mathsf{P}} \star : \star}$$

Logical language is a *sublanguage* of the programmatic language.

$$\frac{\Gamma \vdash^{\mathsf{L}} a : A}{\Gamma \vdash^{\mathsf{P}} a : A}$$

Mixing proofs and programs

These two languages are not independent.

- Should be able to allow programs to manipulate proofs, and proofs to talk about programs.
- Data structures (in both languages) should have both logical and programmatic components.

The @ Modality

New type form $A@\theta$ internalizes the judgement

$$\Gamma \vdash^{\theta} v : A$$

The @ Modality

New type form $A@\theta$ internalizes the judgement

$$\Gamma \vdash^{\theta} v : A$$

Introduction form embeds values from one language into the other.

$$\frac{\Gamma \vdash^{\theta} v : A}{\Gamma \vdash^{\theta'} \mathsf{box}\, v : A@\theta}$$

The @ Modality

New type form $A@\theta$ internalizes the judgement

$$\Gamma \vdash^{\theta} v : A$$

Introduction form embeds values from one language into the other.

$$\frac{\Gamma \vdash^{\theta} v : A}{\Gamma \vdash^{\theta'} \mathsf{box}\, v : A@\theta}$$

Elimination form derived from modal type systems.

$$\frac{\Gamma \vdash^{\theta} a : A@\theta' \quad \Gamma, x :^{\theta'} A, z :^{\mathsf{L}} \mathsf{box}\, x = a \vdash^{\theta} b : B \quad \Gamma \vdash^{\theta} B : s}{\Gamma \vdash^{\theta} \mathsf{unbox}_z\ x = a\ \mathsf{in}\ b : B}$$

Datastructructures

Components of a pair are from the same language by default.

$$\frac{\Gamma \vdash^{\theta} a : A \quad \Gamma \vdash^{\theta} b : [a/x]B \qquad \Gamma \vdash^{\theta} [a/x]B : s \quad \Gamma \vdash^{\theta} \Sigma x\!:\!A.B : s}{\Gamma \vdash^{\theta} (a, b) : \Sigma x\!:\!A.B}$$

Programs can embed proofs about data.

$$\vdash^{\mathsf{P}} (0, \mathsf{box}\, v) : \Sigma x\!:\!\mathsf{Nat}.((y\!:\!\mathsf{Nat}) \to (x \leq y))@\mathsf{L}$$

Data structures are parametric in their logicality. The same datatype can store a list of proofs as well as a list of program values.

Abstraction

Standard abstraction rule conflicts with subsumption.

$$\frac{\Gamma, x :^{\theta} A \vdash^{\theta} a : B \quad \Gamma \vdash^{\theta} (x : A) \to B : s}{\Gamma \vdash^{\theta} \lambda x.a : (x : A) \to B}$$

Solution

Require every argument type to be an $A@\theta$ type, so subsumption has no effect.

$$\frac{\Gamma, x :^{\theta'} A \vdash^{\theta} b : B \quad \Gamma \vdash^{\theta} (x :^{\theta'} A) \to B : s}{\Gamma \vdash^{\theta} \lambda x.b : (x :^{\theta'} A) \to B}$$

Application implicitly boxes.

$$\frac{\Gamma \vdash^{\theta} a : (x :^{\theta'} A) \to B \quad \Gamma \vdash^{\theta} \mathsf{box}\, b : A@\theta' \quad \Gamma \vdash^{\theta} [b/x]B : s}{\Gamma \vdash^{\theta} a\ b : [b/x]B}$$

Logical preconditions

Programmatic functions can have logical parameters:

$$\Gamma \vdash^{\mathsf{P}} div : (n\ d :^{P} \mathsf{Nat}) \to (p :^{L} d \neq 0) \to \mathsf{Nat}$$

Such arguments are "proofs" that the preconditions of the function are satisfied.

Freedom of Speech

Logical functions can have programmatic parameters:

$$\Gamma \vdash^{\mathsf{L}} ds : (n\ d :^{P} \mathsf{Nat}) \to (p :^{L} d \neq 0) \to (\Sigma z : \mathsf{Nat}. z = div\ n\ d)$$

Freedom of Speech

Logical functions can have programmatic parameters:

$$\Gamma \vdash^{\mathsf{L}} ds : (n\ d :^{P} \mathsf{Nat}) \to (p :^{L} d \neq 0) \to (\Sigma z{:}\mathsf{Nat}.z = div\ n\ d)$$

ds is a proof that div terminates for nonzero arguments, even if div was originally defined with general recursion.

Shared values

- Some values are shared between the two languages.

Shared values

- Some values are shared between the two languages.
- For example, all natural numbers are values in the logical language as well as in the programmatic language.

Shared values

- Some values are shared between the two languages.
- For example, all natural numbers are values in the logical language as well as in the programmatic language.
- This means that it is sound to treat a variable of type Nat as logical, no matter what it is assumed to be in the context.

$$\frac{\Gamma \vdash^{\mathsf{P}} v : \mathsf{Nat}}{\Gamma \vdash^{\mathsf{L}} v : \mathsf{Nat}}$$

Uniform equality

- Equality proofs are also shared.

$$\frac{\Gamma \vdash^{\mathsf{P}} v : A = B}{\Gamma \vdash^{\mathsf{L}} v : A = B}$$

- This supports incremental verification. We can have a partial function return an equality proof and then use its result to satisfy logical preconditions.

Uniform equality

- Equality proofs are also shared.

$$\frac{\Gamma \vdash^{\mathsf{P}} v : A = B}{\Gamma \vdash^{\mathsf{L}} v : A = B}$$

- This supports incremental verification. We can have a partial function return an equality proof and then use its result to satisfy logical preconditions.
- However, we currently only know how to add this rule to logical languages with *predicative* polymorphism. Girard's trick interferes.

Uniform box

Challenge: the internalized type.

$$\frac{\Gamma \vdash^{\mathsf{P}} v : A@\theta}{\Gamma \vdash^{\mathsf{L}} v : A@\theta}$$

This allows proofs embedded in programs to be used when reasoning about those programs (not just as preconditions to other programs).

Promising initial results via step-indexed semantics, limitations necessary.

Related work

- Bar types in Nuprl - no admisibility required
- Partiality Monad
- F-star kinds
- ML5, distributed ML

Future work

- What can we add to the logical language? Large Eliminations?

Future work

- What can we add to the logical language? Large Eliminations?
- Interaction with classical reasoning: allow proofs to branch on whether a program halts or diverges

Future work

- What can we add to the logical language? Large Eliminations?
- Interaction with classical reasoning: allow proofs to branch on whether a program halts or diverges
- Elaboration to an annotated language

Summary

- Can have full-spectrum dependently-typed language with nontermination, effects, etc.
- Call-by-value semantics permits "partial correctness"
- Logical and programmatic languages can interact
  - All proofs are programs
  - Logic can talk about programs
  - Programs can contain proofs
  - Some values can be transferred from programs to logic