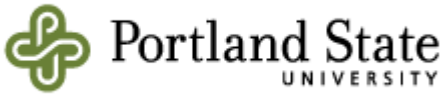
The Preliminary Design of the Trellys Core Language

PLPV 2011

Austin, TX

The Trellys Team



Stephanie Weirich

Aaron Stump

Tim Sheard

Chris Casinghino

Harley Eades

Ki Yung Ahn

Vilhelm Sjöberg

Peng (Frank) Fu

Nathan Collins

Garrin Kimmell

Trellys

- A collaborative project to design a new dependently typed programming language
- Emphasis on:
  - Writing practical programs
  - Exploring new points in the design space
- Today: 30 minute presentation, then 30 minutes of discussion

The Basics

- The Trellys core language is **dependently typed**.
- It is presented with a **collapsed syntax** (as in the lambda cube).
- The operational semantics is **call by value**.

Problem 1: Non-logical programs

- Many desired PL features aren't naturally logical.
  - General recursion, Type : Type, non-positive datatypes, nondeterminism, IO
- Techniques exist to cope with some of them.
  - Coinduction is popular now in Coq/Agda
- We wanted to try something different and more direct.

The Trellys Approach

- The core language is divided into **Logical** and **Programmatic** fragments by the typing judgement.

$$\theta := \mathsf{L} \mid \mathsf{P}$$

- We call θ the “consistency classifier”.

$$\Gamma \vdash_\theta A : B$$

The Two Fragments

- “L” looks like CIC with a universe hierarchy.
- “P” collapses the universes with Type:Type, adds general recursion, more datatypes, etc.
- Theorem:

$$\Gamma \vdash_{\mathsf{L}} A : B \;\Rightarrow\; \Gamma \vdash_{\mathsf{P}} A : B$$

  - So proofs are programs (but not all programs are proofs).

Freedom of Speech

- Terms in the logical fragment can contain programmatic subterms.
- This is important to write proofs *about* programs.
  - e.g., "$\omega = 3$" is a reasonable logical assertion
- The type system rules out such terms when they would diverge.

Problem 2: Efficiency and Proofs

- Computing with proofs is sometimes expensive and unnecessary.
- Coq has the Prop/Set distinction and program extraction.
  - But this forces some duplication and makes run-time relevance a property of terms themselves.
- In Trellys, we build on the ICC*/EPTS approach.

The Trellys Approach

- Whether a term is erased depends on how it is used.
  - $\lambda_+(x : A).b$ argument needed at run-time.
  - $\lambda_-(x : A).b$ argument is compile-time only.
- The type system checks that compile-time arguments are used only in erased positions.
- Erasure:

$$|\lambda_-(x : A).b| = |b|$$
$$|\lambda_+(x : A).b| = \lambda x.|b|$$

Trellys vs ICC*

- This erasure is similar to ICC* (Barras and Bernardo, 2008) and EPTS (Linger, 2008).
  - But we erase more type annotations.
- A similar idea also recently appeared in Agda.
- It has surprising interactions with the rest of Trellys (more on this in a bit).

Problem 3: Types and Equality

- In existing type theories, types gum up equality.
- For example, to prove

$$H \ : \ x = \mathsf{nat} \vdash \mathsf{Nil}\ x = \mathsf{Nil}\ \mathsf{nat}$$

  we must reason with the assumption.
- And to prove vector append associates, we have to reason about addition.
- We want something more computational.

The Trellys Approach

- In Trellys, "$a = b$" is a primitive type.
- We can prove $a = b$ when $|a| \leadsto^* c$ and $|b| \leadsto^* c$.
- Here, $|a|$ is erasure and $\leadsto^*$ is CBV reduction.
- Conversions require logical proofs and are erased.
  - i.e., $|\mathsf{conv}\ a\ \mathsf{by}\ b\ \mathsf{at}\ x.A| = |a|$

Non-termination vs. equality

- Since we have non-termination, a “normalize-and-compare” strategy won't work.
- Trellys equality proofs specify how many steps of reduction are needed.
- Hopefully elaboration from a source language could infer these.

How about proof irrelevance?

- In Trellys, not all proofs of a proposition are equal.
- But before comparing terms, we erase irrelevant arguments.
- Proofs in irrelevant positions don't get “in the way” of equality.

How about extensionality?

- Nope... actually it's inconsistent!
- With extensionality:

$$\lambda(x : 0 = 1).0 = \lambda(x : 0 = 1).1$$

- We erase all type annotations, so:

$$\lambda(x : \mathsf{nat}).0 = \lambda(x : 0 = 1).0$$

- Thus:

$$\lambda(x : \mathsf{nat}).0 = \lambda(x : \mathsf{nat}).1$$

Trellys vs. Coq/Agda vs. OTT

- Both Trellys and OTT attempt to extend the equality of traditional type theories.
- But they do so **differently.**
  - Trellys has a computational focus, while OTT supports type-directed principles like extensionality.
- The extensionality example suggests the two approaches may be incompatible**.**

Problem 4: Metatheory

- That last example was weird... is all hope lost?
- We'd like to prove the standard properties.
  - Type safety
  - Normalization, for the logical fragment
- Preservation, at least, should be easy.

Progress

- Progress is a little trickier:

  Theorem: If $\Gamma \vdash_\theta A : B$ then $|A| \rightsquigarrow |A'|$
  or $|A|$ is a value.

- As usual, this depends on normalization for canonical forms.
  - Suppose we have it, for now.

A problem for progress

- Consider some function

$$\lambda\_(x :^{\mathsf{L}} 0 = 1).b$$

- The contradictory assumption might be used as a coercion inside the body.
  - e.g., to index into an empty vector
- If the lambda is erased, this creates a stuck term.

A value restriction

- Solution: implicit lambda bodies must be values.

$$\lambda\_(x : A).v$$

- Now we know it doesn't get stuck.

Another value restriction

- Applications to non-logical terms are also restricted:

$$a\ v$$

- Two reasons
  - If it's a compile-time function, erasing a loop would be odd.
  - If it's a run-time function that produces logical results, a loop could break normalization.
- Logical non-values are allowed.

Is it inconvenient?

- Variables are values.
- So sequencing terms explicitly with “let” expressions helps.
- Some of this could be done automatically in the source language.

Normalization

- The programmatic fragment doesn't normalize.
- Nor do open terms in the logical fragment:
  - An assumption of $\mathtt{nat} = \mathtt{nat} \to \mathtt{nat}$ can be used to typecheck the Y combinator.
  - And the “proof” gets erased before evaluation.
- But closed logical terms should.
  - We've proved this for a smaller language with freedom of speech.

Summary

- Trellys supports non-logical features with a programmatic fragment.
- Trellys has ICC*-style erasure.
- Trellys equality has a computational flavor.
- Trellys exploits CBV reduction and value restrictions for type safety and soundness.

Some lingering questions

- Is there a consistent equality with this computational flavor that is compatible with extensionality?
- Can we eliminate any of the value restrictions?
- Will the L/C annotations cause significant duplication?
  - And how should datatypes be classified?