HOW TO GIVE TALKS THAT PEOPLE CAN FOLLOW

Derek Dreyer
MPI for Software Systems

PLMW@POPL 2017
Paris, France
My job as a researcher

Do research
My job as a researcher

- Do research
- Write papers
- Give talks
My job as a researcher

- Do research
- Write papers
- Give talks
Entertain your audience!

- **Simon Peyton Jones.** *How to give a great research talk.* (MSR Summer School, 2016)
  - “Your mission is to **wake them up**!”
  - “Your most potent weapon, by far, is **your enthusiasm**!”

- **John Hughes.** *Unaccustomed as I am to public speaking.* (PLMW, 2016)
  - “**Put on a show**!”
Entertain your audience!

- Simon Peyton Jones. *How to give a great research talk*. (MSR Summer School, 2016)

  "Your mission is to wake them up!"

  "Your most potent weapon, by far, is your enthusiasm!"

Good advice, but I don’t know how to teach people to be entertaining…

- John Hughes. *Unaccustomed as I am to public speaking*. (PLMW, 2016)

  "Put on a show!"
Instead, we’ll focus on…

STRUCTURE
STRUCTURE

Jean-Luc Godard
STRUCTURE

Quentin Tarantino
A movie should have a beginning, a middle, and an end…

— Jean-Luc Godard
A movie should have
a beginning, a middle, and an end…
…but not necessarily in that order.

— Jean-Luc Godard
A movie should have a beginning, a middle, and an end… …but not necessarily in that order.

— Jean-Luc Godard

Derek Dreyer
Structure of 20-min. talk

- **Motivation** (~6 minutes)
  - What problem are you solving and why?
- **Contributions & key idea** (~3 minutes)
  - What did you actually do, and what is the key idea behind your solution?
- **Explanation of key idea** (~9 minutes)
- **Conclusion** (~2 minutes)
Motivation

- **Goal:**
  - Explain your problem and why the audience should care about it.

- **Pitfalls:** ???
Motivation

• **Goal:**
  - Explain your problem and why the audience should care about it.

• **Pitfalls:**
  - Fail to clearly state your problem.
  - Take too long to explain your problem.
Stage the motivation

- First, get to a problem.
  - Explain a **general** version of your problem (but not too general) **in the first 2 minutes**.

- Then, get to the problem.
  - Motivate and **explicitly state** your **specific** problem in the next 4 minutes.
  - Limit discussion of prior work only to what is needed to explain your problem.
Pilsner:

A Compositionally Verified Compiler for a Higher-Order Imperative Language

Georg Neis, Chung-Kil Hur, Jan-Oliver Kaiser, Craig McLaughlin, Derek Dreyer, Viktor Vafeiadis

MPI-SWS (Germany),
Seoul National University,
University of Glasgow

ICFP 2015
Vancouver
Pilsner:

A Compositionally Verified Compiler for a Higher-Order Imperative Language

Georg Neis, Chung-Kil Hur, Jan-Oliver Kaiser, Craig McLaughlin, Derek Dreyer, Viktor Vafeiadis

MPI-SWS (Germany), Seoul National University, University of Glasgow

ICFP 2015
Vancouver
Compiler Verification

- **Goal:** Formally guarantee that output of compiler “preserves semantics” of input
  - Successes: CompCert, CakeML

- Semantics preservation (traditionally):
  - If $P_T = \text{Compile}(P_S)$,
    then $\text{Behaviors}(P_T) \subseteq \text{Behaviors}(P_S)$. 
Compiler Verification

- **Goal**: Formally guarantee that output of compiler “preserves semantics” of input.

Success cases:
- CompCert, CakeML

- **Semantics preservation (traditionally)**:
  - If $P_T = \text{Compile}(P_S)$,
    then $\text{Behaviors}(P_T) \subseteq \text{Behaviors}(P_S)$.

Says nothing about separate compilation!
Compositional Compiler Verification

- Goal: Define semantics preservation for separately compiled modules

- Three key criteria (in our view):
  - Modularity
  - Flexibility
  - Transitivity
Modularity

Semantics preservation preserved by linking:
Behaviors(T₁ • T₂ • T₃) ⊆ Behaviors(S₁ • S₂ • S₃)
Can link results of different verified compilers, together with hand-optimized modules.
Transitivity

Can verify individual passes of a compiler, then link the results transitively.
Prior Work

A number of modular techniques proposed, but all are either:

1. **Not** flexible enough
2. **Not** transitive
Takeaway:
First, get to a problem.
Then, get to the problem.
Contributions & key idea

- **Goal:**
  - State what you did to solve the problem.
  - Briefly describe key idea of your solution.

- **Pitfall:** ???
Contributions & key idea

● **Goal:**
  - State what you did to solve the problem.
  - Briefly describe key idea of your solution.

● **Pitfall:**
  - Fail to have this section in your talk.
Don’t blow this golden opportunity!

- **Proudly state your contributions.**
  - After the motivation, the audience eagerly wants to hear what you did. Tell them!

- **Have a key idea.**
  - It will give audience a take-home message, and give focus to the rest of your talk.
Prior Work

A number of modular techniques proposed, but all are either:

1. **Not** flexible enough
2. **Not** transitive
Our Contributions

Parametric Inter-Language Simulations (PILS):

- New way to define semantics preservation
- Modular, flexible, and transitive

Pilsner:

- The first compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!
Our Contributions

Pilsner

ML: \( S_1 \)

Cps conversion

CPS: \( M_1 \) \text{ opts.}

Code generation

Asm: \( T_1 \)
Our Contributions

Pilsner

ML: $S_1$

Cps conversion

CPS: $M_1$ \(\text{opts.}\)

Code generation

Asm: $T_1$
Our Contributions

ML:
S₁ → Cps conversion → S₁

CPS:
M₁ → Code generation → M₁

Asm:
T₁ → T₁

Zwickel:
S₂ → Direct → S₂

SMC Example:
S₃ → Hand-Opt. → S₃
Our Contributions

**Parametric Inter-Language Simulations (PILS):**

- New way to define semantics preservation
- Modular, flexible, and transitive

**Pilsner:**

- The *first* compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!
Our Contributions

**Parametric Inter-Language Simulations (PILS):**

- New way to define semantics preservation
- Modular, flexible, and transitive

**Pilsner:**

- The *first* compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!
Compiler correctness

Step-indexing

Logical relations

POPL ’11

POPL ’12

PILS

Game semantics

Abramsky+ ’98

Bisimulations

Sumii+ ’05, ’09

Iassen+ ’05, ’07

Dreyer+ ’10

POPL ’11

POPL ’12

Ahmed ’06

Ahmed+ ’09

Benton-Hur ’09

Pitts-Stark ’98

Appel-McAllester ’01
A Kripke Logical Relation Between ML and Assembly

Chung-Kil Hur* Derek Dreyer
Max Planck Institute for Software Systems (MPI-SWS)
{gil,dreyer}@mpi-sws.org

1. Introduction

While compiler verification is an age-old problem, there has been remarkable progress in the last several years in proving the correctness of compilers for increasingly realistic languages with increasingly realistic runtime systems. Of particular note is Leroy’s CompCert project [18], in which he used the Coq proof assistant to both develop and verify a multi-pass optimizing compiler from Cenior (a C-like intermediate language), to PowerPC assembly. Daray [13] has adapted the CompCert framework to a compiler for a pure mini-ML language, and McCune et al. [19] have extended it to support interfacing with a garbage collector. Independently, Chlipala [10, 12] has developed verified compilers for both pure and impure functional core languages, the former garbage-collected, with a focus on using custom Coq tactics to provide significant automation of verification.

Abstract

There has recently been great progress in proving the correctness of compilers for increasingly realistic languages with increasingly realistic runtime systems. Most work on this problem has focused on proving the correctness of a particular compiler, leaving open the question of how to verify the correctness of assembly code that is hand-optimized or linked together from the output of multiple compilers. This has led Benton and other researchers to propose more abstract, compositional notions of when a low-level program correctly realizes a high-level one. However, the state of the art in so-called “compositional compiler correctness” has only considered relatively simple high-level and low-level languages.

In this paper, we propose a novel, extensional, compiler-independent notion of equivalence between high-level programs in an expressive, impure ML-like λ-calculus and low-level programs written in an expressive, impure assembly language via Step-indexed Kripke logical relations, biorthogonal projections, and the Marriage of Bisimulations and Kripke Logical Relations
Takeaway:

Proudly state your contributions.
Have a key idea.
Explanation of key idea

● **Goal:**
  
  - Explain your key idea in detail.

● **Pitfall:** ???
Explanation of key idea

● **Goal:**
  - Explain your key idea in detail.

● **Pitfall:**
  - Fail to properly structure a long section.
Talklets

● **Break explanation of key idea into talklets.**
  - More digestible units of story (2-4 min.)
  - But just having talklets is not enough…

● **Use transitions between talklets to remind the audience of the big picture.**
  - Summarize the point of the last talklet and how it connects to the next one.
Summary

- First, get to a problem.
- Then, get to the problem.
- Proudly state your contributions.
- Have a key idea.
- Break explanation of key idea into talklets.
- Use transitions between talklets to remind the audience of the big picture.