We only have two classes left. We'll do a whirlwind tour of some more advanced applications of the techniques we've been studying. Next time, we'll talk about some work on bisimulations, one of Derek's original motivations for learning about this line of work.

There are several topics. Derek will try to give a high-level flavor of the ideas and how they relate to what we've learned.

• Hur-Dreyer, POPL'11. Idea: Generalizing Kripke Logical Relations to reasoning about equivalence between different languages. The paper is interesting in part because the languages are /very/ different. One is an ML-like language like the ones we've been studying. Another is assembly language.

• Krishnaswami et al, ICFP'12. Idea: Applying a variant of these Kripke Logical Relations to prove "semantics soundness" for a rather expressive dependent type system for state. KLRs had been applied in the past to reason about such things. This paper built on a line of work by Ahmed, Fluet, and Morrisett. The type system is richer and admits more interesting specifications than previous systems had. Derek will be explain very clearly the kind of richer possible-world structures supported by the system.

• Turon et al, POPL'13. Idea: Applying (and generalizing) these Kripke Logical Relations to account for fine-grained concurrency. That's a specific style of concurrent programming that's tricky to reason about. It uses KLRs as a starting point.

• Sumii-Pierce, 2004; Sangiorigi-Kobayashi-Sumii, 2007. A different technique called environmental bisimulations. Bisimulations have been around for a long time and used for reasoning about concurrent systems and process calculii. The generalization here was to reason about higher-order, stateful languages. It was a surprising and quite important development. Derek will explain the differences and argue that, basically, KLRs are "better". (Heh.) (Aside: Eventually, researchers realized that the two techniques were related and could be combined. We still don't have a precise correspondence between this approach and ours.)

• Hur et al., POPL'12. Parametric bisimulations. How to combine the benefits of KLRs and environmental bisimulations.

Aside: Both environmental bisimulations and parametric bisimulations seek to prove the kinds of things we've been proving in this class,

but in different ways.

• Krishnaswami-Dreyer, 2013. Work on a logical relation for dependent type theory supporting parametricity. (LRs without parametric reasoning had been done.) What makes dependent type theory different: Such languages model equality. Equality is essential to “full-blown” dependent type theory (à la Coq). Your model of equality has to be an equivalence relation. One thing we've touched on, but not really gone into, is the fact that the logical relations we've been developing are not necessarily transitive. (If all you care about is proving contextual equivalence, you can use ≡ctx's transitivity.) One thing this work supports but, say, CoC does not: You can prove that two terms are equivalent using parametricity and then rely on that equivalence inside the type theory.

— A Kripke Logical Relation Between ML and Assembly. (Hur-Dreyer, 2011)

Also worth reading (if somewhat outdated): Benton-Hur (ICFP'09). This was the first paper to suggest combining ⊤⊤-closure and step-indexing. They suggested it for somewhat technical reasons, but still. What made their work somewhat unsatisfying: They used a very simple, pure high-level language and an abstract machine. (Rather than an ML-like language and assembler.)

On rereading the paper, Derek thinks its quite good. If you want a paper that attempts to be understandable at a high-level, this is a good choice.

This work built on previous work Gil had done with Nick Benton on compositional compiler correctness. All the work on this topic is fairly preliminary: They've built no compilers.

The idea behind compositional compiler correctness: Proving compiler correctness in such a way that the notion of correctness is not tied to the compiler's implementation. Leroy, for example, develops a simulation relation that's very much tied to the phases of the compiler. If you want to, for example, reason about correctness of code linked together but coming from different compilers (or as hand-written assembly), that approach does not help. Put another way, you need some kind of compositional notion of correctness that's not tied to a specific compiler.

Benton-Hur's idea: Use logical relations. LRs say when two things are equivalent at a particular type.

One goal of this work was to scale the approach to more reaslistic langauges. Another was to show that the kinds of KLRs we saw last time are relevant to this project.

So, what's the goal when defining LRs between multiple languages? It's not contextual equivalence. That only makes sense when the terms are in the same language. (Each language gives rise to its own notion of context.)

To reason about equivalence between different languages, there's no completely satisfying standard notion of equivalence (like contextual equivalence). You can use the LR itself as the notion of equivalence. This begs the question: What have you accomplished? Adequacy is perfectly well-defined when working with multiple langauges. It's compatibility that's tricky.

In this work, they defined a very simple notion of how to implement ML-like constructs (eg, application) on the low-level side. For example, see Lemma 1 in the paper (compatibility for application). To show that this isn't just baking in a notion of compilation, they proved a very weird example relating, on the one hand, the Pitts-Stark Awkward example to some self-modifying assembler on the other hand.

Aside: The logical relation is defined inductively over types τ that are independent of the two languages whose "terms" are being related. One may instantiate the language parameters with, for example, two high-level languages (with different type systems) or two low-level languages.

What were the fundamental problems with this approach?
LRs are not generally transitive. For reasoning about compiler correctness, you want to argue that each compiler 'pass' is correct, then (transitively) compose the correctness proofs. This motivated the work on parametric bisimulations.

—

We looked at the following follow-on work to Sumii-Pierce.

> Eijiro Sumii.
> A Complete Characterization of Observational
> Equivalence in Polymorphic λ-Calculus with
> General References.
> http://www.kb.ecei.tohoku.ac.jp/~sumii/pub/poly-ref.pdf

Aside from Dave: Forgive these notes. The material was completely new to me. I couldn't both follow along and write things down.

In Sumii-Pierce, the handling of recursive types was not such a big deal. The handling of abstract types was interesting.

What are bisimulations? Something is a bisimulation if it's some relation that, roughly, is sound with respect to contextual equivalence. This is a coinductive method. It's got a different flavor from LRs which were, at least initially, inductively defined. With LR proofs, there's a definition saying what to prove (think how often we simply unroll the definition). With bisimulations, you come up with a whole set of related things. You say “this is a bisimulation”. The set contains, at least, the pair of things you actually care about.

Key thing: Environmental bisimulations are very syntactic. This $(\Delta,R)*$ thing is key to the whole approach (see the treatment of function arguments in the paper's Definition 6).

We'll return to this point in discussing Hur-Dreyer.