Uses logical relations to state and prove compiler correctness for *open* programs i.e. programs that reference code in other programs [1]. The logical relations leverage the type systems of source, target, and intermediate languages to guarantee the semantics of programs assembled from various components. The paper's verified compiler preserves type soundness across compiler and linker passes.

Strengths

- Formally states the problem of open-compiler correctness.
- The paper's guarantees hold even if the linked-to programs are compiled by an unverified compiler, a compiler for a different language, or exist only in the target language.
- The compiler correctness proof technique works for single- or multi-pass compilers.
- Examples include *more expressive* target languages and typed closure conversion.

Weaknesses

- Compiler correctness is for expressions and linking is modeled directly as substitution. But real compilers operate on programs and real linkers do more work than just substitution—linkers build new programs and are just as prone to errors as real compilers. Formalizing exactly what a linker like `gcc`'s does is an important and (somewhat) interesting problem.
- The admissibility requirements are pretty heavy. It's not totally unreasonable to have the user instantiate both ρ and $\mathcal{V}[\![\cdot]\!]\rho$, but it would be good to narrow the latter requirement down to specific properties.
- The verified compiler only exists on paper.

References

[1] James T. Perconti and Amal Ahmed. Verifying an open compiler using multi-language semantics. In *ESOP*, 2014.