If the *Sound Modular Verification* paper [1] was fan fiction, this is a type-theory pastoral [3]. Suppose we kept type information all the way down to assembly. Not just `int` vs. `float` type information, but Reynolds-style syntactic discipline. Compilers could then optimize more aggressively and runtime systems could guarantee more safety properties. No more will the computer "accept almost any sequence of instructions and make sense of them at its own level" [2]; instead, stuck states will protect high-level invariants. Code will *always* come with the specification it must follow.

The article presents a 5-step, type-preserving translation from System F to a typed assembly language. Two of the passes are optimizations: hoisting and register allocation.

Strengths

- This sounds great—especially the safety properites that annotated code provide. What needs to happen for the world to start running typed assembly? Do we need a Xavier to crank out a commercially-viable implementation? A big security breach to motivate new hardware? To be patient and wait 20 more years, until that 30-year sweet spot? I wonder what the authors would say if we asked them today.
- The running example program was very useful.

Weaknesses

- Once we pick a TAL, we will be stuck with it, just as we're stuck with x86 and ARM. The authors don't seem to think the one here is right, based on their proposed research to eliminate array bounds checking. What then, is the right language? The paper doesn't give a strong opinion, nor does it talk about type checkers that accept more than one language (to ease migration from an implemented-now version to a better, future version)
- How does code link? I got that foreign binaries could be assumed type-correct, but the language didn't talk about modularity or foreign functions. Are those really straightforward with TAL? Do we need to re-typecheck all the code we link to, or is it impossible for an adversary to give a fake proof of type-correctness?
- The big question: how slow is TAL? How do the authors' experimental implementations run?

References

[1] Pieter Agten, Bart Jacobs, and Frank Piessens. Sound modular verification of C code executing in an unverified context. In *POPL*, 2015.

[2] C.A.R. Hoare. Hints on programming language design. In *Stanford AI Lab, MEMO AIM 224*, 1973.

[3] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. In *TOPLAS*, 1999.