Proposes *witness generation* procedures for proving program transformations [1]. The main insight is that heuristic passes can produce evidence to help the verifier.

From high-level to low-level, the paper's connection to real life is:

- Compiler optimizations are program transformations.
- A program transformation is correct if running the result (target) gives an output that the source code could have produced.
- Formally, target code T *implements* source code S according to a relation α iff α relates all initial target states with initial source states and whenever the target reduces to a final state, the source can reduce to a final state related by α. Note that this definition ignores intermediate steps. An optimizing pass that produced T given S is correct if we can give a relation α with the above property.
- The paper argues that *stuttering simulations* are a useful class of relations because they are straightforward to prove and imply *implements* relations. If S and T are in a stuttering simulation, then T implements S.
- A stuttering simulation R must:
  - Relate initial target states to initial source states.
  - Provide a well-founded measure $<$ such that, for all transitions $t_1 \to t_2$ in T and all source states s_1 related by R to t_1, one of the following must hold:
    - There is a source state s_2 such that $s_1 \to s_2$ and $(t_2, s_2) \in R$
    - We stutter in T, which means $(t_2, s_1) \in R$ and $(t_2, s_1) < (t_1, s_1)$ holds.
    - We stutter in S; meaning is a s_2 such that $(t_2, s_2) \in R$ and $(t_2, s_2) < (t_1, s_2)$.

    The stuttering means we don't move to a new state along both axis, but instead decrease along $<$.
- Stuttering similations compose, just like passes compose into a whole compiler.
- Stuttering similation can prove interesting optimizations. The authors show examples of constant propogation, dead code elimination, CFG compression, loop hoisting, and loop reordering.

Strengths

- It's exciting to hope that we could validate all compiler passes in a uniform way. It's also exciting that we don't need to rewrite old optimizing transformations to do so (we only need to modify them to produce evidence).

Weaknesses

- I'm not certain how the wellfounded relations $<$ compose. Do we get to reset the measure between passes?
- The examples are explained well, but I'm looking forward to a more general way of producing these relations during an optimization pass.

References

[1] Kedar S. Namjoshi and Lenore D. Zuck. Witnessing program transformations. In *SAS*, 2013.