Manifest Adequacy*

Daniel K. Lee Robert J. Simmons

Carnegie Mellon University

1 First beer

"Wrong? I can't be wrong! I'm never wrong! I refuse to be wrong!"
– Anonymous sober type theorist

Out of a well-founded skepticism for the ability of paper to correct for the errors introduced by smudging and smearing caused by the coffee, beer, and blood spills that are standard hazards in the daily lives of programming language researchers, it has been decided that the future of programming languages research is in the formalization of languages as specifications for computer proof assistants. Commonly, the encoding of the language is as a script written in the programming language used to interact with the proof assistant. In the LF community, a great deal of sober thought has been given to the process in which the LF encoding of a language can be verified to be an *adequate* representation of the intended language. This process is performed by verifying there exists a *compositional bijection* between the LF encoding and the intended language. The failure of an adequacy proof generally implies someone wasted time working on a language he didn't even care about. Because adequacy proofs require time that could be more gainfully spent proving interesting theorems, reading Livejournals, marking the Wean blackboards with type theory, durnk dialing exgirlfriends, grading problem sets, eating tube meat, planning TGs, creating fonts, hunting penguin thieves, or playing internet poker, they are *never* performed in practice.

2 Second beer

We present a novel way of encoding the compositional bijection between the Twelf encodings of a language and the language it is meant to represent. We call this operator I, but in order to retain an air of mystery we will refuse to define the semantics of I here. However, we have a napkin proof that I is reflexive and idempotent. Under the I operator, every Twelf signature is manifestly an adequate representation of some language. Additionally, it is general knowledge that languages encoded in Twelf are superior to those never encoded in Twelf, anyway.

*This work is partially supported by the National Science Foundation under a Graduate Research Fellowship and D's Six Pax and Dogz.

```
tp : type

o : tp.
arrow : tp.

exp : type.

* : exp.
app : exp -> exp -> exp.
lam : tp -> (exp -> exp) -> exp.
```

Figure 1: `stlc.elf`

```
tp : type.

o : tp.
arrow : tp.

exp : type.

* : exp.
app : exp -> exp -> exp.
lam : tp -> (exp -> exp) -> exp.
```

Figure 2: I(`stlc.elf`)

3 Third and subsequent beers

We have discovered an implementation of I in the standard Unix toolset. Assuming the encoded language is in `durnken-logic.elf`, the following command generates the represented language in `durnken-logic-actual.elf`:

```
% cp durnken-logic.elf durnken-logic-actual.elf
```

Additionally, we have discovered a verifier for testing the correctness of an application of I. The successful completion of the following command with no output verifies that `durnken-logic-actual.elf` is $I($`durnken-logic.elf`$)$

```
% diff durnken-logic.elf durnken-logic-actual.elf
```

Anecdotal evidence states that the Unix implementations of I and its verifier are pretty damn fast. The extant pervasiveness of the implementation of the I operator and its verifier on standard computing environments demonstrate their fundamental and foundational importance.

4 Final beer

> *"Oh schnap, I've got it! Without a way to be wrong, I can't help but be right!"*
> – *Anonymous durnken type theorist*

In conclusion, we have suggested a formulation of the only adequacy argument we will ever bother to make.