# CLASSICAL REALIZABILITY

DANNY GRATZER

## 1. An Overview of Realizability

In this note I'd like to talk about a realizability model for classical logic, but it's worth writing down a few thoughts on how realizability in general works to make sure that we're on the same page.

Realizability is a technique for understanding a logic from a computational point of view. Rather than studying the proofs in a logic (proof theory) we define a system of computation and model propositions using collections of terms in that system. This technique dates back to Kleene's model of intuitionistic logic. The primary benefit of a realizability treatment is that it's an explanation of what the computational content of a theorem is. One interesting consequence for doing this for classical logic is to see that a computational interpretations aren't limited to constructive logics (just nice ones).

## 2. A Judgmental Formulation of Classical Logic

In order to formalize a computational interpretation of classical logic, we first have to formally define classical logic. I'll do this with the normal proof theoretic tools for doing this. We start by fixing some set of propositions we're interested in studying.

$$P \quad ::= \quad \top \mid \bot \mid P \wedge P \mid P \vee P \mid P \supset P \mid \neg P$$

We'd like to formulate the judgment $P$ **true**. Unlike in an intuitionistic logic, classical logic also needs the judgment $P$ **false** since proofs in this logic switch freely between searching for contradictions and building evidence. We also have a judgment for denoting a contradiction, **contra**. This gives us some way to mediate between $P$ **true** and $P$ **false** and also let's us state double negation in a cleaner way.

$$\frac{}{\top\ \mathbf{true}} \qquad \frac{}{\bot\ \mathbf{false}} \qquad \frac{A\ \mathbf{true} \qquad B\ \mathbf{true}}{A \wedge B\ \mathbf{true}} \qquad \frac{A\ \mathbf{false}}{A \wedge B\ \mathbf{false}} \qquad \frac{B\ \mathbf{false}}{A \wedge B\ \mathbf{false}}$$

$$\frac{A\ \mathbf{true}}{A \vee B\ \mathbf{true}} \qquad \frac{B\ \mathbf{true}}{A \vee B\ \mathbf{true}} \qquad \frac{A\ \mathbf{false} \qquad B\ \mathbf{false}}{A \vee B\ \mathbf{false}} \qquad \frac{B\ \mathbf{true}\ (A\ \mathbf{true})}{A \supset B\ \mathbf{true}}$$

$$\frac{A\ \mathbf{true} \qquad B\ \mathbf{false}}{A \supset B\ \mathbf{false}} \qquad \frac{A\ \mathbf{true}}{\neg A\ \mathbf{false}} \qquad \frac{A\ \mathbf{false}}{\neg A\ \mathbf{true}}$$

These rules govern the connective specific aspects of $P$ **true** and $P$ **false** but the real essence of the classical logic comes from how we let these two judgments interact through **contra**,

$$\frac{A\ \mathbf{true} \qquad A\ \mathbf{false}}{\mathbf{contra}} \qquad \frac{\mathbf{contra}\ (A\ \mathbf{false})}{A\ \mathbf{true}} \qquad \frac{\mathbf{contra}\ (A\ \mathbf{true})}{A\ \mathbf{false}}$$

Notice that I haven't defined any elimination rules for this system. It turns out that we can derive all of the by heavy use of contradiction.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A\ \mathbf{false}\ (A\ \mathbf{false})}}{A \wedge B\ \mathbf{false}\ (A\ \mathbf{false})} \qquad \dfrac{D}{A \wedge B\ \mathbf{true}}}{\mathbf{contra}\ (A\ \mathbf{false})}}{A\ \mathbf{true}}$$

Now the middle rule governing contradiction gives us double negation easily enough

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vee \neg A\ \mathbf{false}\ (A \vee \neg A\ \mathbf{false})}}{A\ \mathbf{false}\ (A \vee \neg A\ \mathbf{false})}}{\neg A\ \mathbf{true}\ (A \vee \neg A\ \mathbf{false})} \qquad \dfrac{\dfrac{\dfrac{}{A \vee \neg A\ \mathbf{false}\ (A \vee \neg A\ \mathbf{false})}}{\neg A\ \mathbf{false}\ (A \vee \neg A\ \mathbf{false})}}{\mathbf{contra}\ (A \vee \neg A\ \mathbf{false})}}{A \vee \neg A\ \mathbf{true}}$$

The remainder of the elimination rules as well as the metatheoretic properties of this system aren't going to be described.

## 3. A Computation System

Before we proceed with our realizability model we have to describe a computation system to use to model our programming language. Normally some variant of the lambda calculus suffices but for classical logic it's more convenient if our computation system includes continuations. As is typical with realizability, we won't specify the entirety of the language. Instead we'll describe several inferences we require

to be valid in the system and leave ourselves open to the possibility of more rules being added.

The trouble is that continuations are hard to give nice operational semantics to. To handle this, I've opted to give a stack machine presentation of the language. So our semantics will have two judgments, $k \rhd e$ and $k \lhd e$. In the first judgment we're working by examining some term $e$ and building up some surrounding context/continuation $k$. This judgment doesn't really do any work, it sets us up for $k \lhd e$ which takes some $e$ that we don't know how to reduce any further and works back through $k$ performing reductions as it goes. For example, the inferences governing how lambda abstraction and application in our language work are

$$\frac{}{k \rhd e\ e' \mapsto k;\ \mathtt{ap}(-;\ e') \rhd e} \qquad \frac{}{k \rhd \lambda x.\ e \mapsto k \lhd \lambda x.e}$$

$$\frac{}{k;\ \mathtt{ap}(-;\ e') \lhd \lambda x.e \mapsto k \rhd [e'/x]e}$$

When evaluating an application we push a new frame onto our stack of continuations and proceed in evaluation the function. Whenever we reach a $\lambda$ (or any value as we'll see) we stop trying to use $\rhd$ because we have no more expressions to decompose and we begin to work on $k$. In particular, whenever we try to $\lhd$ a lambda through a continuation representing application we actually perform the relevant substitution.

## 4. The Realizability Model

## 5. Soundness and Completeness