Paper

Simon van Hus
6147879
s.vanhus@students.uu.nl

Abstract

Abstract.

1 Tracing

In the broadest terms, when we trace a program, we track the most basic steps the program takes provided some input. This is relevant for many applications in Computer Science. Certain automatic differentiation (AD) implement the forward-pass as effectively tracing, and then perform the reverse pass on the trace[cite me]. Tracing is also used to speed up program execution by making assumptions on the program's execution path from the trace[cite me]. Another example? Rule of threes!

However, despite its ambivalence, tracing is rarely properly defined, or defined only for a specific use case. So, in this section we will set out to create a more general definition of tracing.

To start, it will help us along to set clear expectations for what we expect a tracing function to do. In the simplest terms, we expect a tracing program to take an input program with a set of inputs, and output a "trace". This output trace is generally defined as a set of operations the input program performed on the inputs. A term often used for an output trace is a "single-line program"[cite me]: a program without control flow. Clearing control flow like if-then-else statements is only natural: after all, provided some input the program will only walk down one variation of this branching path.

Furthermore, it is also generally accepted that the trace consists of a subset of the syntax of the input program. More precisely, if our input program has the types as defined in Equation 1. Here we have sum-types as $\sigma + \tau$, product types as $\sigma \times \tau$, functions as $\sigma \to \tau$, literal real numbers, and Booleans as $\top$ or $\bot$.

$$\sigma, \tau \coloneqq \sigma + \tau \mid \sigma \times \tau \mid \sigma \to \tau \mid \mathbb{R} \mid \{\top, \bot\} \tag{1}$$

When we trace on a program using the types in Equation 1, we can select which types to keep intact, and which to "trace away". A common option is to keep only "grounded

types", where we defined a grounded type as a type that isn't constructed of other types. Looking at our example in Equation 1, a trace keeping only these grounded types would keep only the real numbers and the Booleans as they are not built of other types. Another common option is to keep only continuous types, tracing away all ungrounded and discrete types. Do that on our type set in Equation 1 would leave us with only the real numbers.

The main take-away here is that there is some freedom of choice in what to trace away. What parts we keep and what parts we trace away is very dependent on what information we want to keep in our trace. Which in turn is dependent on what our exact goal is for the tracing in the first place.

We can also choose to keep some of our ungrounded types, but then we run into a problem. Say we keep only functions ($\sigma \to \tau$) and real numbers, but our input program contains a function with type $\sigma \to (\tau_1 + \tau_2)$. This typing is valid in our input program, but no longer valid in our trace, so we find ourselves in a bind. It will be impossible to trace away the sum-type in the output of the function without tracing away the function itself. Why exactly? Of course we could define a subset $\sigma', \tau' \coloneqq \mathbb{R}$ and then redefine (or add a definition for) our function so that it becomes $\sigma' \to \tau'$ making it safe to trace. This then underlines the rule at work here: we can only keep types that do cannot be constructed of types that are traced away. This is why the grounded types are a natural set of types to keep, as they are never constructed from other types.

It seems that our tracing definition comes down to a function that takes in a program and an input to that program, and outputs the steps taken by the program run on the input. Where the input program takes uses some set of types, of which only a subset is kept in the trace, where the types in this subset may not be constructed using types from outside of the subset.

What now remains is a concrete definition of the output of the tracing program. We've already set that it should somehow contain the steps done in the