Adversarial Paths

by Sven Nilsen, 2018

Based on ideas from a discussion with Adam Nemecek, in this paper I formalize what it means to make a choice `A ~ 0` in path semantics, without being able to remember how one ended up in `A ~ 1`. The notation is designed to easily work with higher order dependencies between choices.

A choice is a type `A` with an associated function `A::f : A → B`.

An adversarial path is a higher order equivalence path such that:

A ~ 0 : T → A ~ 1

A ~ 1 := \(g : B → bool) = g ⊆ ∪ x : T { (A::f ~ 1)'(x) }

Where `A ~ 0` consumes `A` as a resource, and `A ~ 1` produces a new resource.

The rest of this paper explains the notation above.

I will derive it from the notation used in the paper "Equivalence Paths": An equivalence path is a function `~f` created from some function `f`. The `~` unary operator is called the "universal equivalence path". One can think of `~f` as crossing out all input-output pairs that intersect, such that for all outputs, there exist a unique input. To access all equivalence paths of a function, the transformation is controlled by manipulating the input domain constraint `~f{∀f}`. A higher order trivial path means that `∀f'` depends on some quantified variable. This means that:

∀f : A → bool

f : A → B

Is replaced by:

∀f' : T → A → bool

Since the trivial path `f ~ 0` of the equivalence path `~f{∀f}` is defined by:

f ~ 0 <=> ∀~f{∀f}

It follows that the higher order trivial path `(f ~ 0)'` of the higher order equivalence path `~f{∀f'}`:

(f ~ 0)' <=> ∀~f{∀f'}

(f ~ 0)' : T → A → bool

Since the existential path of `f` constrained to `f ~ 0` determines `f ~ 1`:

$$\exists f\{f \sim 0\} \iff f \sim 1$$

It follows that the higher order existential path of `f` constrained to `(f ~ 0)'` determines `(f ~ 1)'`:

$$\exists f\{(f \sim 0)'\} \iff (f \sim 1)'$$

In Adverserial Path Semantics, one exploits the following properties:

$$\exists f\{(f \sim 0)'\} \iff (f \sim 1)'$$

$$(f \sim 0)' : T \to A \to \text{bool}$$
$$(f \sim 1)' : T \to B \to \text{bool}$$
$$f : A \to B$$

Instead of defining an `f` for every `A`, it is associated with `A`, such that:

$$\exists A::f\{(A::f \sim 0)'\} \iff (A::f \sim 1)'$$

$$(A::f \sim 0)' : T \to A \to \text{bool}$$
$$(A::f \sim 1)' : T \to B \to \text{bool}$$
$$A::f : A \to B$$

A type `A` with an associated function `A::f` is called a “choice”.

An adversarial choice `A` has the following abstract judgemental properties:

$$\forall x \{ (A::f \sim 0)'(x) : \text{unknown} \}$$
$$\forall x \{ (A::f \sim 1)'(x) : \text{known} \}$$

This is meant as `A ~ 0` consumes `A` as a resource.

The higher order existential path `(∃A::f{(A::f ~ 0)'})(x) <=> (A::f ~ 1)'(x)` has a sub-type `A ~ 1`:

$$A \sim 1 := \backslash(g : B \to \text{bool}) = g \subseteq \cup\ x : T \{ (A::f \sim 1)'(x) \}$$

$$A \sim 1 : (B \to \text{bool}) \to \text{bool}$$

Because `(A::f ~ 0)'` is unknown for all inputs, this frees up the syntax `A ~ 0` to mean something else:

$$A \sim 0 : T \to A \sim 1$$

Since `A` is consumed by `A ~ 0`, one can interpret it as consuming `A` and producing `A ~ 1`. This notation is designed to easily work with higher order dependencies between choices.

`A ~ 0` is interpreted as making the choice itself, then feeding in some `x : T` to obtain `A ~ 1`. Notice that this can be thought of as “going through `A ~ 0` into `A ~ 1`”. One can also think of `A ~ 0` as committing to making a choice, even though, the concrete choice is delayed until the next step.