Probabilistic Paths

by Sven Nilsen, 2017

A probabilistic path is similar to an existential path, but for reasoning about uncertainty. For example, the following two functions `$f_0$` and `$f_1$` have the same existential path:

f_0 := \(x : nat) = if (x % 2) == 0 { 1 } else { 0 }
f_1 := \(x : nat) = if (x % 10) == 0 { 1 } else { 0 }

$\exists f_0$ <=> $\exists f_1$ <=> \(x : nat) = x == 0 ∨ x == 1

Yet, the function `$f_0$` returns `1` five times as often compared to `$f_1$`:

f_0 : 1 0 1 0 1 0 1 0 1 0 1 0 …
f_1 : 1 0 0 0 0 0 0 0 0 0 1 0 0 …

So, their probabilistic paths are different:

$\exists_p f_0$:= \(_ : nat) = if x == 0 ∨ x == 1 { 1/2 } else { 0 }
$\exists_p f_1$:= \(x : nat) = if x == 0 { 9/10 } else if x == 1 { 1/10 } else { 0 }

All boolean functions has an assigned probability of returning `true`, by using a higher order `if` function:

$P(if(A, B)(x)) = P(x) \cdot P(A) + (1-P(x)) \cdot P(B)$

if := \(a, b) = \(c) = if c { a } else { b }

Just like the existential path can not inherit knowledge about the input, the probability path uses a uniform random distribution for the input. This means the probability of input `P(x)` gets replaced by `1/2` after deriving the probability formula for a boolean function. The probabilistic path of every boolean function can be constructed by the `probx` function:

probx := \(k : real ∧ [prob] true) = \(x : bool) = if x { k } else { 1 – k }

prob := \(x : real) = x >= 0 ∧ x <= 1

For example, for the function `and`:

$P(and(x_0, x_1)) = P(x_0) \cdot P(x_1) = 1/2 \cdot 1/2 = 1/4$
$\exists_p and$ <=> probx(1/4)

In probability theory there are conditional probabilities. The equivalent concept for probabilistic paths is constructed by adding sub-type constraints to the input before taking the probabilistic path:

$\exists_p$and{(= true), _} <=> $\exists_p$id <=> probx(1/2)

Translating back and forth between the two notations is straight forward:

P(and(a, b) | a) = ($\exists_p$and{(= true), _})(true)
P(¬and(a, b) | a) = ($\exists_p$and{(= true), _})(false)

In proofs, inject the sub-type constraints so you get only free inputs and then take the probabilistic path:

P(and(a, b) | a) = ($\exists_p$id)(true)
P(and(a, b) | a)
P(and(a : (= true), b))
P(and{(= true), _}(a, b))
($\exists_p$and{(= true), _})(true)
    and{(= true), _} <=> id
($\exists_p$id)(true)

P(¬and(a, b) | a) = ($\exists_p$id)(false)
P(¬and(a, b) | a)
P(¬and(a : (= true), b))
P(¬and{(= true), _}(a, b))
    not · and <=> nand
P(nand{(= true), _}(a, b))
($\exists_p$nand{(= true), _})(true)
    Here I do a trick by inverting the output.
($\exists_p$and{(= true), _})(false)
    and{(= true), _} <=> id
($\exists_p$id)(false)