Boolean Algebra Using If-Else

by Sven Nilsen, 2017

In path semantics you might need to convert between Boolean algebra and If-Else forms.

First defining the syntax of If-Else using slot lambda calculus:

({ _ }) = (_ _)(“{“)(_ _)(?)(“}”)(\)

(if _ { _ } else { _ }) = (_ _)(“if”)(_ _)(?)(_ _)({ _ })(?)(_ _)(“else”)({ _ })(\)

Slot lambda calculus is used to prove equivalence or transfer values between two expressions, without relying on a complex grammar language or a parser. It is a simple language that contains its own grammar and parser.

Setting up logical equivalences:

not <=> \(a: bool) = if a { false } else { true }
and <=> \(a: bool, b: bool) = if a { b } else { false }
or <=> \(a: bool, b: bool) = if a { true } else { b }

Constructing the slot lambdas:

not: \(a: bool) = (if _ { _ } else { _ })(a)(false)(true)
and: \(a: bool, b: bool) = (if _ { _ } else { _ })(a)(b)(false)
or: \(a: bool, b: bool) = (if _ { _ } else { _ })(a)(true)(b)

When the application rule evaluates according to normal if-else rule, this is equivalent to Boolean algebra.

For example, assume the following function:

\(a: nat, b: nat) = a + b

Examining the path:

is_zero(a: nat) = a == 0

\([is_zero] za, [is_zero] zb) = [is_zero] if za { zb } else { false }

Since it is known that this is logically equivalent to `and`, one can write:

\([is_zero] za, [is_zero] zb) = [is_zero] and(za, zb)

This proves there is a symmetric path:

add[is_zero] <=> and