Homework 9

Jiří Klepl

We want to define the standard lexical comparison on bit-vectors. It is equivalent to the standard unsigned comparison of the vectors that are created by shifting the operands left by their respective length of maximum prefixes consisting only of zeros (aka. the number of leading zeros) with an exception that on equality of the transformed vectors the algorithm decides the vector with fewer leading zeros is higher.

$a <_{lex} b := a << clz(a) <_u b << clz(b) \lor (a << clz(a) = b << clz(b) \land clz(a) >_u clz(b))$

Another, weaker, variation is just: $a <_{lex} b := a << clz(a) <_u b << clz(b)$

We already know how to implement $<_u$, $<<$ and $=$, so we need to describe only the unary operation clz.

One possible way to define clz is $clz_l(a) := \begin{cases} 0 & \text{if } (2^l \bigwedge_l a) > 0 \\ 1 + clz_{l-1}(a) & \text{otherwise} \end{cases}$

$(2^l \bigwedge_l a) > 0$ can be easily implemented using a single `and` gate, and $1 + clz_{l-1}$ is a simple increment.

Simple implementation for this linear clz algorithm, therefore, is a nested `if then else` circuit with hard-coded values (or we can use the addition - which is not easier):

$clz := \lambda a.\ (2^l \bigwedge_l a) > 0\ ?\ 0 : (2^{l-1} \bigwedge_{l-1} a) > 0\ ?\ 1 : \cdots l.$

This can be implemented using broadcasts and the bit-conditioned ternary operator:

broadcast: $set_l :=$ copy the given bit l times $= \lambda b.\ b \circ b \circ \cdots b$ (b repeating $l\times$)

bit-conditioned ternary: $\_?\_ : \_ := \lambda c.\ \lambda a.\ \lambda b.\ (set_l\ c \bigwedge_l a) \bigvee_l (set_l\ (\sim c) \bigwedge_l b).$