Homework

1. Extend the definitions of $\mathrm{bb}_a(\cdot)$ and $\mathrm{bb}_t(\cdot)$ to cover the shift operators $\mathrm{bb}_t(a_k \ll b_k)$, $\mathrm{bb}_t(a_k \gg_u b_k)$ and $\mathrm{bb}_t(a_k \gg_s b_k)$ (2 P)

2. Consider the definition of $\mathrm{bb}(\cdot)$ on slide 18. We claim the formula $\mathrm{bb}(\varphi)$ and φ are equisatisfiable, but unfortunately this is not the case for the current encoding.

   (a) Demonstrate that the encoding $\mathrm{bb}(\cdot)$ does not produce equisatisfiable propositional formulas using the formula $\psi = \neg(a_2 \hat{} a_2 = 0_2)$. (1 P)

   (b) Can you repair the definition of $\mathrm{bb}(\cdot)$? If yes, how? (1 P)

3. In low level code bit manipulation can sometimes be used to lower the number of operations, or remove unnecessary branching. For the following C-functions use an SMT-solver (for example Z3[1]) with bit-vector arithmetic (for example QF_BV[2] of SMTLib) to prove that the functions have the intended behaviour. You may assume that an `int` has 32-bits, and a `char` has 8 bits.
   Hint: In C the `^` is a bitwise xor operator, `&` a bitwise and operator, and `>>` a logical right shift on unsigned integers and an arithmetic right shift on signed integers.

   (a) The following function should compute the absolute value of a 32-bit integer. (1 P)

   ```
   unsigned int abs(int v) {   // v is a 32-bit signed integer
     unsigned int r;           // the result
     int mask = v >> 31;
     r = (v + mask) ^ mask;
     return r;
   }
   ```

   (b) This function should reverse the bits in a (8-bit) byte. (e.g. `11101010` turns into `01010111`). (1 P)

   ```
   unsigned char reverse_bits(unsigned char b) { // b is an 8-bit value
     unsigned char r;                  // the result (char) is truncated to 8 bits
     // the trailing ULL makes the literal an unsigned 64-bit integer
     r = ((b * 0x0000000202020202ULL) &  0x0000010884422010ULL) % 1023;
     return r;
   }
   ```

   (c) The last function computes the parity of a 32-bit unsigned integer. The parity of an integer is 0 if the number of one bits is even and 1 otherwise (e.g. `1101` has parity 1, `0110` has parity 0). (2 P)

   ```
   unsigned int parity(unsigned int v) { // v is a 32-bit unsigned integer
     unsigned int r;                     // the result
     v = v ^ (v >> 1);
     v = v ^ (v >> 2);
     \\ the trailing U makes the literal an unsigned 32-bit integer
     v = (v & 0x11111111U) * 0x11111111U;
     r = (v >> 28) & 1;
     return r;
   }
   ```

[1] Z3 SMT-solve: `https://github.com/Z3Prover/z3`
[2] documentation of the QF_BV logic in SMTLib: `https://smt-lib.org/logics-all.shtml#QF_BV`

4. Let $\lfloor a_m.b_k \rfloor_n$ be the operation which takes as input a fixed-point number $\langle a_m.b_k \rangle$ with a fractional part of k bits and rounds it to a number with a fractional part of n bits, where $n < k$. Define the flattening $\mathrm{bb}_t(\lfloor a_m.b_k \rfloor_n)$ for the following two cases:

   (a) rounding towards $-\infty$ (e.g. $\lfloor 11.01 \rfloor_1 = 11.0$ in two's complement) (1 P)

   (b) rounding towards zero (e.g. $\lfloor 11.01 \rfloor_1 = 11.1$ in two's complement) (1 P)