Practical SAT Solving (ST 2024) | Assignment 4

Markus Iser, Dominik Schreiber, Tomáš Balyo
Algorithm Engineering (KIT) 2024-06-18 – 2024-07-09

1 Multiplier Encodings (10 Points)

The task is to output the encoding of a multiplier circuit in CNF. The circuit should take two unsigned 64-bit integers a and b as input and output their product. Variables 1 to 65 of the CNF should be the bits of the first input number a. Variables 66 to 129 should be the bits of the second input number b. Variables 130 to 193 should be the bits of the output number c. Variables 194 should indicate whether an overflow happened during addition. The program should output the CNF in DIMACS format (including a header with the number of variables and clauses). We will test your program by setting the bits of the input numbers by adding unit clauses and checking whether the output number is the product of the input numbers. You can find the encoding of the multiplier circuit in `https://satlecture.github.io/kit2024/references/2010%20-%20Bejar%20-%20Encoding%20Basic%20Arithmetic%20Operations%20for%20SAT-Solvers.pdf`.

2 Perfect Hashing Competition (16 Points)

Algorithm 1 shows the finalization step of the Murmur3 hash function.[1] This function f takes an unsigned 64-bit integer key k and an unsigned 64-bit integer seed s and returns an unsigned 64-bit integer. Given a bound $B = 2^x$ ($x \in \mathbb{N}$) and a set of $n \leq B$ unsigned 64-bit integers $K = \{k_1, \ldots, k_n\}$, write a program which determines a seed s in such a way that

$$|\{\ f(k_1, s) \bmod B, f(k_2, s) \bmod B, \ldots, f(k_n, s) \bmod B\ \}| = n,$$

i.e., all keys are mapped *bijectively* to a set of n unique integers in the range $[0, B)$. Encode the function depicted in Algorithm 1 as a Boolean circuit in CNF and use an incremental SAT solver to solve this problem. Your program should take the bound B and the set of keys K as input and output the seed s. You can reuse your encoding of the multiplier circuit from the previous task or optimize it further as it is done in `https://satlecture.github.io/kit2024/references/2024%20-%20Bierlee%20-%20Single%20Constant%20Multiplication%20for%20SAT.pdf`.

[1] `http://zimbry.blogspot.com/2011/09/better-bit-mixing-improving-on.html`

Algorithm 1: Function f: Murmur3Final

Input: Key k: uint64
Input: Seed s: uint64

$k \leftarrow k + s$
$k \leftarrow k \otimes (k >> \texttt{33})$
$k \leftarrow k \times \texttt{0xff51afd7ed558ccd}$
$k \leftarrow k \otimes (k >> \texttt{33})$
$k \leftarrow k \times \texttt{0xc4ceb9fe1a85ec53}$
$k \leftarrow k \otimes (k >> \texttt{33})$

return k;