**1. Intro.** This program generates data for a given function, in the form needed by the SAT-SYNTH program.

I hacked it in a big hurry, for one particular case. With a little more work I can of course parameterize it so that it will generate a reasonably wide class of examples from user-friendly input specs.

At the moment I have only one parameter, $t$: Each new data point is chosen so that each of its coordinates differs from the previous point with probability $2^{-t}$. For example, if $t = 3$ and if I've just output $x$ and $f(x)$, I will next output $x \oplus y$ and $f(x \oplus y)$, where every bit of $y$ is 1 with probability 1/8 and 0 with probability 7/8. On the other hand if $t = 1$, every data point is random.

(Well, there's also a second parameter, namely a random seed.)

The function $f$ here is assumed to be

$$f(x_1,\ldots,x_{20}) = \bar{x}_2\bar{x}_3\bar{x}_{10} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12} \vee x_8\bar{x}_{13}\bar{x}_{15} \vee \bar{x}_8x_{10}\bar{x}_{12},$$

because I'm featuring that particular function in my book. Other functions could easily be generated, however, by changing $M$ and the *term* table below.

I generate lots of data points (currently 1000). The program SAT-SYNTH-TRUNC will use only an initial segment of them.

```
#define M 4    /* this many terms */
#define N 20    /* this many variables */
#define tmax 5    /* maximum number of literals per term */
#define imax 1000    /* this many data points are generated */
#define O "%"    /* used for percent signs in format strings */
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"
  int term[M][tmax + 1] = {{−2, −3, −10, 0}, {−6, −10, −12, 0}, {8, −13, −15, 0}, {−8, 10, −12, 0}};
  int seed;    /* the random number seed */
  int t;    /* the number of times to AND bits together before use */
  char x[N + 1];    /* the current data point */
  unsigned int randbits;    /* yet-unused random bits, preceded by 1 */
  main(int argc, char *argv[])
  {
    register int a, b, i, j, k;
    ⟨Process the command line 2⟩;
    ⟨Set up the first data point 3⟩;
    for (i = 0; i < imax; i++) {
      ⟨Output the current x and f(x) 4⟩;
      ⟨Set up the next data point 5⟩;
    }
  }
```

**2.** ⟨Process the command line 2⟩ ≡

```
  if (argc ≠ 3 ∨ sscanf(argv[1], ""O"d", &t) ≠ 1 ∨ sscanf(argv[2], ""O"d", &seed) ≠ 1) {
    fprintf(stderr, "Usage:␣"O"s␣t␣seed\n", argv[0]);
    exit(−1);
  }
```

This code is used in section 1.

**3.** ⟨ Set up the first data point 3 ⟩ ≡

```
gb_init_rand(seed);
for (j = 1; j ≤ N; j++) x[j] = gb_next_rand() & 1;
randbits = 1;
```

This code is used in section 1.

**4.** ⟨ Output the current $x$ and $f(x)$ 4 ⟩ ≡

```
for (j = 1; j ≤ N; j++) printf(""O"d", x[j]);
for (a = 0, j = 0; j < M; j++) {
  for (b = 1, k = 0; term[j][k]; k++) b &= (term[j][k] > 0 ? x[term[j][k]] : 1 − x[−term[j][k]]);
  a |= b;
}
printf(":"O"d\n", a);
```

This code is used in section 1.

**5.** ⟨ Set up the next data point 5 ⟩ ≡

```
for (k = 0; k ≡ 0; ) {
  for (j = 1; j ≤ N; j++) {
    if (randbits ≡ 1) {
      randbits = gb_next_rand();    /* get 31 new random bits */
      for (k = 1; k < t; k++) randbits &= gb_next_rand();
      randbits |= #80000000;    /* prepend a 1 bit */
    }
    k |= randbits & 1;    /* set k nonzero if there was a change */
    x[j] ⊕= randbits & 1;
    randbits ≫= 1;
  }
}
```

This code is used in section 1.

## 6. Index.

⟨ Output the current $x$ and $f(x)$ 4 ⟩ Used in section 1.
⟨ Process the command line 2 ⟩ Used in section 1.
⟨ Set up the first data point 3 ⟩ Used in section 1.
⟨ Set up the next data point 5 ⟩ Used in section 1.

# SAT-SYNTH-DATA