1. Intro. I'm testing the clauses for nondeterministic finite-state automata in exercise 7.2.2.2–436.

[“Historical notes”: I wrote that exercise in October 2014, but didn't keep any copy of the draft that led to it. In April 2016, after discovering problems in the printed solution, I found a proofsheet dated 22 Oct 2014, in my file of scrap MS pages, which contained the earliest version. That version was more complicated than the present one, using states called $t_{kqaq'}$; it was inspired by the construction of Bacchus for deterministic automata. On that sheet I had pencilled in the new version, which was based on using the Quimper–Walsh construction of exercise 7.2.2.2–440 and specializing it to the case of regular grammars. Those changes were incorporated into the file `7.2.2.2.tex` on 22 October.]

The specifications for the automaton are given entirely on the command line. Each state is represented by a single ASCII character, between `!` and `}` inclusive, other than `0` or `1`; I recommend using letters. The first argument specifies the string length, n. The next argument specifies one or more input states. The next argument specifies one or more output states. The remaining arguments are the transitions.

For example, to get all n-bit strings of the form $0*110*$, the command-line arguments

```
n a c a0a a1b b1c c0c
```

ought to work.

Variables for the SAT clauses are: k for x_k; `q`k for q_k; and `qa`k for t_{kaq}. (Here k is given in decimal.)

I apologize for writing this in a huge hurry.

```
#define badstate(k) ((k) < '!' ∨ (k) ≥ '~' ∨ (k) ≡ '0' ∨ (k) ≡ '1')
#include <stdio.h>
#include <stdlib.h>
  int n;    /* command-line argument */
  char isinput[128], isoutput[128], isstate[128];
  char istrans[128][2];
  main(int argc, char *argv[])
  {
    register int a, j, k, q;
    ⟨Process the command line 2⟩;
    for (k = 1; k ≤ n; k++) {
      ⟨Generate the clauses of type (i) 3⟩;
      ⟨Generate the clauses of type (ii) 4⟩;
      ⟨Generate the clauses of type (iii) 5⟩;
      ⟨Generate the clauses of type (iv) 6⟩;
      ⟨Generate the clauses of type (v) 7⟩;
    }
    ⟨Generate the clauses of type (vi) 8⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
if (argc < 4 ∨ sscanf(argv[1], "%d", &n) ≠ 1) {
  fprintf(stderr, "Usage:␣%s␣n␣I␣O␣{qaq'}*\n", argv[0]);
  exit(−1);
}
for (j = 0; argv[2][j]; j++) {
  k = argv[2][j];
  if (badstate(k)) {
    fprintf(stderr, "Improper␣input␣state␣`%c'!\n", k);
    exit(−2);
  }
  isinput[k] = 1;
}
for (j = 0; argv[3][j]; j++) {
  k = argv[3][j];
  if (badstate(k)) {
    fprintf(stderr, "Improper␣input␣state␣`%c'!\n", k);
    exit(−3);
  }
  isoutput[k] = 1;
}
for (j = 4; j < argc; j++) {
  if (badstate(argv[j][0]) ∨ badstate(argv[j][2]) ∨ argv[j][1] < '0' ∨ argv[j][1] > '1' ∨ argv[j][3]) {
    fprintf(stderr, "Improper␣transition␣`%s'!\n", argv[j]);
    exit(−4);
  }
  isstate[argv[j][0]] = 1;
  isstate[argv[j][2]] = 1;
  istrans[argv[j][2]][argv[j][1] − '0'] = 1;
}
printf("~");
for (k = 0; k < argc; k++) printf("␣%s", argv[k]);
printf("\n");    /* mirror the command line as the first line of output */
```

This code is used in section 1.

3. ⟨ Generate the clauses of type (i) 3 ⟩ ≡

```
for (q = '!'; q < '~'; q++)
  if (isstate[q]) {
    if (istrans[q][0]) {
      printf("~%c0%d␣~%d\n", q, k, k);
      printf("~%c0%d␣%c%d\n", q, k, q, k);
    }
    if (istrans[q][1]) {
      printf("~%c1%d␣%d\n", q, k, k);
      printf("~%c1%d␣%c%d\n", q, k, q, k);
    }
  }
```

This code is used in section 1.

4. ⟨ Generate the clauses of type (ii) 4 ⟩ ≡

```
for (q = '!'; q < '~'; q++)
  if (isstate[q]) {
    printf("~%c%d", q, k − 1);
    for (j = 4; j < argc; j++)
      if (argv[j][0] ≡ q) printf("␣%c%c%d", argv[j][2], argv[j][1], k);
    printf("\n");
  }
```

This code is used in section 1.

5. ⟨ Generate the clauses of type (iii) 5 ⟩ ≡

```
for (q = '!'; q < '~'; q++)
  if (isstate[q]) {
    printf("~%c%d", q, k);
    for (j = 4; j < argc; j++)
      if (argv[j][2] ≡ q) printf("␣%c%c%d", argv[j][2], argv[j][1], k);
    printf("\n");
  }
```

This code is used in section 1.

6. ⟨ Generate the clauses of type (iv) 6 ⟩ ≡

```
printf("%d", k);
for (j = 4; j < argc; j++)
  if (argv[j][1] ≡ '0') printf("␣%c0%d", argv[j][2], k);
printf("\n");
printf("~%d", k);
for (j = 4; j < argc; j++)
  if (argv[j][1] ≡ '1') printf("␣%c1%d", argv[j][2], k);
printf("\n");
```

This code is used in section 1.

7. ⟨ Generate the clauses of type (v) 7 ⟩ ≡

```
for (q = '!'; q < '~'; q++)
  if (isstate[q]) {
    for (a = '0'; a < '2'; a++)
      if (istrans[q][a − '0']) {
        printf("~%c%c%d", q, a, k);
        for (j = 4; j < argc; j++)
          if (argv[j][2] ≡ q ∧ argv[j][1] ≡ a) printf("␣%c%d", argv[j][0], k − 1);
        printf("\n");
      }
  }
```

This code is used in section 1.

8. ⟨ Generate the clauses of type (vi) 8 ⟩ ≡

```
for (q = '!'; q < '~'; q++)
  if (isstate[q]) {
    if (¬isinput[q]) printf("~%c0\n", q);
    if (¬isoutput[q]) printf("~%c%d\n", q, n);
  }
```

This code is used in section 1.

9. Index.

SAT-NFA