1. Intro. This program generates DLX data that finds all polymorphisms of given relations. I've tried to make it fairly general, so that I can use it for experiments. But I haven't tried to make it especially efficient.

The first command-line parameter is d, the domain size. It is followed by k, the arity of the polymorphism. Then come the tuples of a relation. And the next parameter might then be '`/`', in which case another relation (or sequence of relations) follows.

```
#define maxk 7    /* maximum arity of the polymorphism */
#define maxm 10   /* maximum arity of the relations */
#define maxr 10   /* maximum number of relations */
#define maxt 16   /* maximum number of tuples per relation */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
  int d, k;    /* command-line parameters */
  char tup[maxr][maxt][maxm];    /* tuples of the relations */
  char siz[maxr];    /* the number of tuples in each relation */
  char arity[maxr];    /* the arity of each relation */
  char a[maxk];    /* controlling digits */
  int nam[maxk];    /* hexadecimal names of arguments */
  main(int argc, char *argv[])
  {
    register i, j, l, m, p, r, s, t, v;
    ⟨Process the command line 2⟩;
    ⟨Echo the command line 5⟩;
    ⟨Print the item-name line 6⟩;
    for (i = 0; i < r; i++) ⟨Print the options for relation i 7⟩;
  }
```

2. ⟨Process the command line 2⟩ ≡

```
  if (argc < 3 ∨ sscanf(argv[1], "%d", &d) ≠ 1 ∨ sscanf(argv[2], "%d", &k) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣d␣k␣<tuples>␣[/␣<tuples>]*\n", argv[0]);
    exit(−1);
  }
  if (k ≤ 0 ∨ k > maxk) {
    fprintf(stderr, "Sorry,␣k␣must␣be␣positive␣and␣at␣most␣%d!\n", maxk);
    exit(−2);
  }
  for (r = 0, p = 3; r < maxr; r++) ⟨Input relation r 3⟩;
  if (r ≡ maxr) {
    fprintf(stderr, "Too␣many␣relations␣(maxr=%d)!\n", maxr));
    exit(−9);
  }
  ⟨Report successful command line 4⟩;
```

This code is used in section 1.

3. ⟨Input relation r 3⟩ ≡

```
{
  for (s = 0; argv[p] ∧ argv[p][0] ≠ '/'; p++, s++) {
    if (s ≡ 0) m = strlen(argv[p]);
    else if (m ≠ strlen(argv[p])) {
      fprintf(stderr, "tuple␣%s␣should␣have␣length␣%d,␣not␣%d!\n", argv[p], m,
          (int) strlen(argv[p]));
      exit(−3);
    }
    if (s ≡ maxt) {
      fprintf(stderr, "too␣many␣tuples␣(maxt=%d)!\n", maxt);
      exit(−4);
    }
    for (j = 0; j < m; j++) {
      v = argv[p][j] − '0';
      if (v < 0 ∨ v ≥ d) {
        fprintf(stderr, "value␣in␣tuple␣%s␣is␣out␣of␣range!\n", argv[p]);
        exit(−4);
      }
      tup[r][s][j] = v;
    }
  }
  if (s ≡ 0) {
    fprintf(stderr, "Empty␣relation␣(no␣tuples)!\n");
    exit(−5);
  }
  siz[r] = s, arity[r] = m;
  if (¬argv[p++]) break;
}
```

This code is used in section 2.

4. ⟨Report successful command line 4⟩ ≡

```
r++;
fprintf(stderr, "OK,␣I've␣input␣%d␣relation%s␣of␣size%s!arit%s", r, r ≡ 1 ? "" : "s",
    r ≡ 1 ? "" : "s", r ≡ 1 ? "y" : "ies");
for (j = 0; j < r; j++) fprintf(stderr, "␣%d!%d", siz[j], arity[j]);
fprintf(stderr, ".\n");
```

This code is used in section 2.

5. ⟨Echo the command line 5⟩ ≡

```
printf("|");
for (j = 0; j < argc; j++) printf("␣%s", argv[j]);
printf("\n");
```

This code is used in section 1.

6. Each relation r of size s has s^k primary items, `r`$a_1 \ldots a_k$, one for each constraint between a particular combination of m-tuples in that relation. (Relation r is identified by its code letter `'a'` $+\, r$.)

There are d^k secondary items $x_1 \ldots x_k$, one for each combination of arguments. The color of $x_1 \ldots x_k$ is the value of the polymorphism at those arguments.

⟨ Print the item-name line 6 ⟩ ≡

```
for (i = 0; i < r; i++) {
  for (j = 0; j < k; j++) a[j] = 0;
  while (1) {
    printf("%c", 'a' + i);
    for (j = 0; j < k; j++) printf("%x", a[j]);
    printf("␣");
    for (j = k − 1; j ≥ 0 ∧ a[j] ≡ siz[i] − 1; j−−) a[j] = 0;
    if (j < 0) break;
    a[j]++;
  }
}
printf("|");
for (j = 0; j < k; j++) a[j] = 0;
while (1) {
  printf("␣");
  for (j = 0; j < k; j++) printf("%x", a[j]);
  for (j = k − 1; j ≥ 0 ∧ a[j] ≡ d − 1; j−−) a[j] = 0;
  if (j < 0) break;
  a[j]++;
}
printf("\n");
```

This code is used in section 1.

7. ⟨Print the options for relation *i* 7⟩ ≡

```
{
  for (j = 0; j < k; j++) a[j] = 0;
  while (1) {
    for (j = 0; j < arity[i]; j++) {
      for (v = p = 0; p < k; p++) v = (v ≪ 4) + tup[i][a[p]][j];
      nam[j] = v;
    }
    for (t = 0; t < siz[i]; t++) {
      for (j = 0; j < arity[i]; j++)
        for (l = 0; l < j; l++)
          if (nam[l] ≡ nam[j] ∧ tup[i][t][l] ≠ tup[i][t][j]) goto next_t;
      printf("%c", 'a' + i);
      for (j = 0; j < k; j++) printf("%x", a[j]);
      for (j = 0; j < arity[i]; j++) {
        for (l = 0; l < j; l++)
          if (nam[l] ≡ nam[j]) break;
        if (l < j) continue;
        printf("␣%0*x:%x", k, nam[j], tup[i][t][j]);
      }
      printf("\n");
    next_t: continue;
    }
    for (j = k − 1; j ≥ 0 ∧ a[j] ≡ siz[i] − 1; j−−) a[j] = 0;
    if (j < 0) break;
    a[j]++;
  }
}
```

This code is used in section 1.

8. Index.

⟨ Echo the command line 5 ⟩ Used in section 1.
⟨ Input relation r 3 ⟩ Used in section 2.
⟨ Print the item-name line 6 ⟩ Used in section 1.
⟨ Print the options for relation i 7 ⟩ Used in section 1.
⟨ Process the command line 2 ⟩ Used in section 1.
⟨ Report successful command line 4 ⟩ Used in section 2.

INDICATOR-DLX