1. **Intro.**  This program generates clauses that are satisfiable if and only if there’s a Boolean chain \( x_1, \ldots, x_{n+r} \) in \( n \) variables \( x_1, \ldots, x_n \) that computes the functions whose truth tables are \( T_1, \ldots, T_m \). The parameters are given on the command line. I assume that \( n \leq 6 \), so that each truth table has at most 64 bits. The truth tables are specified in hexadecimal notation, using \( 2^{n-2} \) hex digits each.

The chains are assumed to be “normal”; that is, each function on each step takes \((0, \ldots, 0) \mapsto 0\). (If a parameter \( T_j \) isn’t normal, we compute its complement.)

Steps are indicated in clause-variable names by a single character, beginning with \( 1, 2, \ldots, 9, a, b, \ldots \); the first \( n \) steps are reserved for the projection functions \( x_1 \) through \( x_n \).

The clauses involve several kinds of variables:

- \( F_{kbb} \) means that the Boolean binary function at step \( k \) has \( F_k(b, b') = 1 \); here \( n < k \leq n + r \) and \( 0 \leq b, b' \leq 1, b + b' > 0 \).
- \( K_{kji} \) means that \( x_k = F_k(x_j, x_i) \); here \( n < k \leq n + r \) and \( k > j > i > 0 \).
- \( Z_{ik} \) means that the \( i \)th output \( z_i \) is \( x_k \); here \( 1 \leq i \leq m \) and \( n < k \leq n + r \).
- \( X_{kb} \) means that the Boolean function \( x_k \) takes \((b_1, \ldots, b_n) \mapsto 1 \); here \( n < k \leq n + r \) and \( 0 \leq b_1, \ldots, b_n \leq 1, b_1 + \cdots + b_n > 0 \).

```c
#define maxn 6 /* at most this many variables */
#define maxk 36 /* at most this many steps */
#include <stdio.h>
#include <stdlib.h>

int n, r; /* command-line parameters */
unsigned long long t[maxk]; /* truth tables on the command line */
unsigned long long x[maxn + 1]; /* truth tables for the projections */

(Subroutines 8);

main(int argc, char *argv[])
{
    register int b, bb, bbb, h, i, j, k, m;
    register unsigned long long mask;
    (Process the command line 2);
    for (k = n + 1; k \leq n + r; k++) /* Generate the clauses for step k */
     for (i = 1; i \leq m; i++) /* Generate the clauses for output i */
```
2. \( \{ \) Process the command line \( \text{2} \) \( \equiv \)

\[
\text{if} \ (\text{argc} < 4 \lor \text{sscanf}(\text{argv}[1], \text{"%d", \\&n}) \neq 1 \lor \text{sscanf}(\text{argv}[2], \text{"%d", \\&r}) \neq 1) \{ \\
\text{fprintf}(\text{stderr}, \text{"Usage: \%s n x_1 t_1 \ldots t_m \n", argv[0]}); \\
\text{exit}(-1); \\
\}
\]

\[
\text{if} \ (n < 2 \lor n > \text{maxn}) \{ \\
\text{fprintf}(\text{stderr}, \text{"n should be \ between 2 and \%d, \ not \%d! \n", maxn, n}); \\
\text{exit}(-2); \\
\}
\]

\[
\text{if} \ (n + r > \text{maxk}) \{ \\
\text{fprintf}(\text{stderr}, \text{"n+r should be \ at most \%d, \ not \%d! \n", maxk, n + r}); \\
\text{exit}(-3); \\
\}
\]

\[
\text{mask} = (n \equiv 6 \ ? -1 : (1 \ll (1 \ll n)) - 1); \\
x[1] = \text{mask} \gg (1 \ll (n - 1)); \\
\text{for} \ (i = 2; i \leq n; i++) \ x[i] = x[i - 1] \oplus (x[i - 1] \ll (1 \ll (n - i)))); \\
m = \text{argc - } 3; \\
\text{if} \ (m > r) \{ \\
\text{fprintf}(\text{stderr}, \text{"the number of outputs should be at most \%d, \ not \%d! \n", m}); \\
\text{exit}(-4); \\
\}
\]

\[
\text{for} \ (i = 1; i \leq m; i++) \{ \\
\text{if} \ (\text{sscanf}(\text{argv}[2 + i], \text{"%llx", \\&t[i]) \neq 1) \{ \\
\text{fprintf}(\text{stderr}, \text{"I couldn't scan \ truth table \%d! \n", i}); \\
\text{exit}(-5); \\
\}
\]

\[
\text{if} \ (n \leq 6 \land (t[i] \gg (1 \ll n))) \{ \\
\text{fprintf}(\text{stderr}, \text{"Truth table \%d has too many bits! \n", i, t[i]}); \\
\text{exit}(-6); \\
\}
\]

\[
\text{if} \ (t[i] \gg ((1 \ll n) - 1)) \ t[i] = (\sim t[i]) \& \text{mask}; \\
\}
\]

\[
\text{printf}(\text{"\text{sat-chains}\%d, \%d", n, r}); \\
\text{for} \ (i = 1; i \leq m; i++) \ \text{printf}(\text{"\%llx", t[i]}); \\
\text{printf}(\text{"\n"}); \\
\]

This code is used in section 1.

3. \( \{ \) Generate the clauses for step \( \text{3} \) \( \equiv \)

\[
\text{(Generate clauses to say that operation \( k \) isn't trivial \( 4 \));} \\
\text{(Generate clauses to say that step \( k \) is based on two prior steps \( 5 \));} \\
\text{(Generate clauses to say that step \( k \) is used at least once \( 6 \));} \\
\text{(Generate clauses to exploit the completeness of the basis functions \( 7 \));} \\
\text{for} \ (i = 1; i < k; i++) \\
\text{for} \ (j = i + 1; j < k; j++) \ (\text{Generate the main clauses that involve \%k\%j\%i \( 9 \));} \\
\}
\]

This code is used in section 1.
4. \#define \texttt{e(t) \ (\langle t \leq 9 \ ? \ '0' + t : 'a' + t - 10 \rangle)}
\texttt{(Generate clauses to say that operation \(k\) isn’t trivial 4)} \equiv
\begin{align*}
 & \texttt{printf ("F%c01,F%c10,F%c11
", e(k), e(k), e(k));} \\
 & \texttt{printf ("F%c01,F%c10,F%c11
", e(k), e(k), e(k));} \\
 & \texttt{printf ("F%c01,F%c10,F%c11
", e(k), e(k), e(k));}
\end{align*}
This code is used in section 3.

5. \(\langle\text{Generate clauses to say that step } k \text{ is based on two prior steps 5}\rangle\) \equiv
\begin{align*}
 & \texttt{for (i = 1; i < k; i++)} \\
 & \texttt{printf ("K%c%c%c", e(k), e(j), e(i));} \\
 & \texttt{printf ("\n");}
\end{align*}
This code is used in section 3.

6. \(\langle\text{Generate clauses to say that step } k \text{ is used at least once 6}\rangle\) \equiv
\begin{align*}
 & \texttt{for (i = 1; i \leq n; i++)} \texttt{printf ("K%c%c", e(i), e(k));} \\
 & \texttt{for (j = k + 1; j \leq n + r; j++)} \\
 & \texttt{printf ("K%c%c", e(j), e(i));} \\
 & \texttt{for (j = k + 1; j \leq n + r; j++)} \\
 & \texttt{printf ("K%c%c", e(j), e(i));} \\
 & \texttt{printf ("\n");}
\end{align*}
This code is used in section 3.

7. If \(x_k\) depends only on \(x_j\) and \(x_i\), we can assume that no future step will combine \(x_k\) with either \(x_j\) or \(x_i\). (Because that future step might as well act directly on \(x_j\) and \(x_i\).)
\(\langle\text{Generate clauses to exploit the completeness of the basis functions 7}\rangle\) \equiv
\begin{align*}
 & \texttt{for (i = 1; i < k; i++)} \\
 & \texttt{for (j = i + 1; j < k; j++)} \\
 & \texttt{printf ("K%c%c", e(k), e(j), e(i));} \\
 & \texttt{printf ("K%c%c", e(k), e(j), e(i));} \\
\end{align*}
This code is used in section 3.

8. \(\langle\text{Subroutines 8}\rangle\) \equiv
\begin{align*}
 & \text{void \texttt{printX(char *s, int k, int t)}} \\
 & \{ \\
 & \quad \texttt{register int i;} \\
 & \quad \texttt{if (k > n)} \{ \\
 & \quad \quad \texttt{printf ("%d", s, e(k));} \\
 & \quad \quad \texttt{for (i = 1; i \leq n; i++) \ printf ("%(n - i) & 1);} \\
 & \quad \} \\
 & \}
\end{align*}
This code is used in section 1.
9. \texttt{#define bit\_h(i) (int)((x[i] \gg ((1 \ll n) - 1 - h)) \& 1)}

(Generate the main clauses that involve $Kkji$ 9) \equiv

\begin{verbatim}
for (h = 1; h < (1 \ll n); h++) {
    for (b = 0; b < 1; b++) {
        printf("~Z%c%c", e(k), e(j), e(i));
        printf("X", j, h);
        printf("X", i, h);
        printf("X", k, h);
        printf("\n"); /* (0,0) \rightarrow 0 */
    }
    else {
        printf("~X", j, h);
        printf("X", i, h);
        printf("X", k, h);
        printf("\n", bbb ? "", e(k), b, bb);
    }
}
\end{verbatim}

This code is used in section 3.

10. \{ Generate the clauses for output $i$ 10 \} \equiv

\begin{verbatim}
for (k = n + 1; k \leq n + r; k++) { (Generate the clauses that involve $Zik$ 12); }
\end{verbatim}

This code is used in section 1.

11. \{ Generate clauses to say that output $i$ is present 11 \} \equiv

\begin{verbatim}
for (k = n + 1; k \leq n + r; k++) printf("\n", e(i), e(k));
printf("\n");
\end{verbatim}

This code is used in section 10.

12. \{ Generate the clauses that involve $Zik$ 12 \} \equiv

\begin{verbatim}
for (h = 1; h < (1 \ll n); h++) {
    printf("Z\%c\%c", e(i), e(k));
    if (t[i] & (1_{LL} \ll ((1 \ll n) - 1 - h))) printf("X", k, h);
    else printf("X", k, h);
    printf("\n");
}
\end{verbatim}

This code is used in section 10.
13. Index.

argc: 1, 2.
argv: 1, 2.
b: 1.
bb: 1, 9.
bbb: 1, 9.
bith: 9.
e: 4.
exit: 2.
fprintf: 2.
h: 1.
i: 1, 8.
j: 1.
k: 1, 8.
m: 1.
main: 1.
mask: 1, 2.
maxk: 1, 2.
maxn: 1, 2.
n: 1.
printf: 2, 4, 5, 6, 7, 8, 9, 11, 12.
printX: 8, 9, 12.
r: 1.
s: 8.
sscanf: 2.
stderr: 2.
t: 1, 8.
x: 1.
(Generate clauses to exploit the completeness of the basis functions)  Used in section 3.
(Generate clauses to say that operation $k$ isn’t trivial)  Used in section 3.
(Generate clauses to say that output $i$ is present)  Used in section 10.
(Generate clauses to say that step $k$ is based on two prior steps)  Used in section 3.
(Generate clauses to say that step $k$ is used at least once)  Used in section 3.
(Generate the clauses for output $i$)  Used in section 1.
(Generate the clauses for step $k$)  Used in section 1.
(Generate the clauses that involve $Z_{ik}$)  Used in section 10.
(Generate the main clauses that involve $K_{kJi}$)  Used in section 3.
(Process the command line)  Used in section 1.
(Subroutines)  Used in section 1.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro</td>
<td>1</td>
</tr>
<tr>
<td>Index</td>
<td>13</td>
</tr>
</tbody>
</table>