

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

1* Intro. I’m hurriedly experimenting with a new(?) way to explore the complexity of 4-variable Boolean functions. Namely, I calculate the “footprint” of each function, the set of all first steps by which I know how to evaluate the function in k steps. Then, if the footprints of f and g overlap, I can compute $f \circ g$ in $\text{cost}(f) + \text{cost}(g)$ steps.

I can restrict consideration to the 2^{15} functions that take $(0, 0, 0, 0) \mapsto 0$.

This program extends FCHAINS4 by allowing several additional functions to be precomputed. Those functions appear on the command line, in hexadecimal form.

```
#define footsize 100
#include <stdio.h>
#include <stdlib.h>
typedef struct node_struct {
    unsigned int footprint[footsize];
    int parent;
    int cost;
    struct node_struct *prev, *next;
} node;
node func[1 << 15];
node head[9];
int x[100];
char buf[100]; /* lines of input */
char name[32 * footsize][16];
unsigned int tta, ttb; /* partial truth table found in input line */
unsigned int footp[footsize];
main(int argc, char *argv[])
{
    register int a, b, c, j, k, r, t, m, mm, s, ttt;
    register unsigned int u;
    register node *p, *q, *pp;
    <Read the initial functions 2>;
    <Initialize the tables 8>;
    for (r = 2; c; r++)
        for (k = (r - 1) >> 1; k >= 0; k--) <Combine all functions of costs k and r - 1 - k 3>;
    <Answer queries 12*>;
}
```

2. \langle Read the initial functions 2 $\rangle \equiv$

```

m = argc + 3;
for (k = 1; k ≤ m; k++) {
  if (k ≤ 4) x[k] = #ffff/((1 << (1 << (4 - k))) + 1);
  else if (sscanf(argv[k - 4], "%x", &x[k]) ≠ 1) {
    fprintf(stderr, "Parameter_%s_should_have_been_hexadecimal!\n", argv[k - 4]);
    exit(-1);
  }
  if (x[k] > #ffff) {
    fprintf(stderr, "Parameter_%s_is_too_big!\n", argv[k - 4]);
    exit(-1);
  }
  if (x[k] ≥ #8000) x[k] ⊕= #ffff;
}

```

This code is used in section 1*.

3. \langle Combine all functions of costs k and $r - 1 - k$ 3 $\rangle \equiv$

```

for (p = head[k].next; p-parent ≥ 0; p = p-next)
  for (q = head[r - 1 - k].next; q-parent ≥ 0; q = q-next) {
    for (j = 0; j < mm; j++)
      if (p-footprint[j] & q-footprint[j])  $\langle$  Try for breakthru and goto pqdone 6  $\rangle$ 
     $\langle$  Try for new function 4  $\rangle$ ;
    pqdone: continue;
  }

```

This code is used in section 1*.

4. **#define** fun(p) ((p) - func)

\langle Try for new function 4 $\rangle \equiv$

```

{
  t = fun(p) & fun(q);
  if (func[t].cost ≥ r)  $\langle$  Update the table for cost r 5  $\rangle$ ;
  t = fun(p) & (~fun(q));
  if (func[t].cost ≥ r)  $\langle$  Update the table for cost r 5  $\rangle$ ;
  t = (~fun(p)) & fun(q);
  if (func[t].cost ≥ r)  $\langle$  Update the table for cost r 5  $\rangle$ ;
  t = fun(p) | fun(q);
  if (func[t].cost ≥ r)  $\langle$  Update the table for cost r 5  $\rangle$ ;
  t = fun(p) ⊕ fun(q);
  if (func[t].cost ≥ r)  $\langle$  Update the table for cost r 5  $\rangle$ ;
}

```

This code is used in section 3.

5. $\langle \text{Update the table for cost } r \text{ 5} \rangle \equiv$

```

{
  pp = &func[t];
  if (pp-cost > r) {
    if (pp-cost ≡ 8) c--;
    pp-next-prev = pp-prev, pp-prev-next = pp-next;
    pp-cost = r, pp-parent = (fun(p) ≪ 16) + fun(q);
    for (j = 0; j < mm; j++) pp-footprint[j] = 0;
    pp-next = head[r].next, pp-prev = &head[r];
    pp-next-prev = pp, pp-prev-next = pp;
  }
  for (j = 0; j < mm; j++) pp-footprint[j] |= p-footprint[j] | q-footprint[j];
}

```

This code is used in section 4.

6. $\langle \text{Try for breakthru and goto } pqdone \text{ 6} \rangle \equiv$

```

{
  t = fun(p) & fun(q);
  if (func[t].cost ≥ r - 1)  $\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle$ ;
  t = fun(p) & (~fun(q));
  if (func[t].cost ≥ r - 1)  $\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle$ ;
  t = (~fun(p)) & fun(q);
  if (func[t].cost ≥ r - 1)  $\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle$ ;
  t = fun(p) | fun(q);
  if (func[t].cost ≥ r - 1)  $\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle$ ;
  t = fun(p) ⊕ fun(q);
  if (func[t].cost ≥ r - 1)  $\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle$ ;
  goto pqdone;
}

```

This code is used in section 3.

7. This code is not executed when $k = 0$, because q 's footprint is zero in that case.

$\langle \text{Update the table for cost } r - 1 \text{ 7} \rangle \equiv$

```

{
  pp = &func[t];
  if (pp-cost > r - 1) {
    if (pp-cost ≡ 8) c--;
    pp-next-prev = pp-prev, pp-prev-next = pp-next;
    pp-cost = r - 1, pp-parent = (fun(p) ≪ 16) + fun(q);
    for (j = 0; j < mm; j++) pp-footprint[j] = 0;
    pp-next = head[r - 1].next, pp-prev = &head[r - 1];
    pp-next-prev = pp, pp-prev-next = pp;
  }
  for (j = 0; j < mm; j++) pp-footprint[j] |= p-footprint[j] & q-footprint[j];
}

```

This code is used in section 6.

8. $\langle \text{Initialize the tables 8} \rangle \equiv$
for ($p = \&func[2]; p < \&func[\#8000]; p++$) ($p-1$) $\rightarrow next = p, p\rightarrow prev = p-1, p\rightarrow cost = 8;$
 $func[1].cost = 8;$
for ($k = 0; k \leq 8; k++$) $head[k].parent = -1, head[k].next = head[k].prev = \&head[k];$
 $head[0].next = head[0].prev = \&func[0];$
 $func[0].next = func[0].prev = \&head[0];$
 $head[8].next = \&func[1], func[1].prev = \&head[8];$
 $head[8].prev = \&func[\#7fff], func[\#7fff].next = \&head[8];$
 $\langle \text{Initialize the functions of cost 0 9} \rangle;$
 $\langle \text{Initialize the functions of cost 1 10} \rangle;$

This code is used in section 1*.

9. $\langle \text{Initialize the functions of cost 0 9} \rangle \equiv$
for ($k = 1; k \leq m; k++$) {
 $p = \&func[x[k]];$
if ($p\rightarrow cost \equiv 0$) **continue**;
 $p\rightarrow next\rightarrow prev = p\rightarrow prev, p\rightarrow prev\rightarrow next = p\rightarrow next;$
 $p\rightarrow cost = 0;$
 $p\rightarrow next = head[0].next, p\rightarrow prev = \&head[0];$
 $p\rightarrow next\rightarrow prev = p, p\rightarrow prev\rightarrow next = p;$
}
 $c = (1 \ll 15) - 1 - m;$

This code is used in section 8.

10. $\langle \text{Initialize the functions of cost 1 10} \rangle \equiv$
 $s = 0;$
for ($r = 2; r \leq m; r++$)
for ($k = 1; k < r; k++$) {
 $t = x[k] \& x[r], sprintf(name[s], "\%d\&\%d(\%04x)", k, r, t);$
 $\langle \text{Update for cost 1 11} \rangle;$
 $t = x[k] \& (\sim x[r]), sprintf(name[s], "\%d\>\%d(\%04x)", k, r, t);$
 $\langle \text{Update for cost 1 11} \rangle;$
 $t = (\sim x[k]) \& x[r], sprintf(name[s], "\%d\<\%d(\%04x)", k, r, t);$
 $\langle \text{Update for cost 1 11} \rangle;$
 $t = x[k] | x[r], sprintf(name[s], "\%d|\%d(\%04x)", k, r, t);$
 $\langle \text{Update for cost 1 11} \rangle;$
 $t = x[k] \oplus x[r], sprintf(name[s], "\%d^\sim\%d(\%04x)", k, r, t);$
 $\langle \text{Update for cost 1 11} \rangle;$
}
 $mm = (s + 31)/32;$

This code is used in section 8.

11. $\langle \text{Update for cost 1 } 11 \rangle \equiv$

```

p = &func[t];
if (p->cost > 1) {
    if (s ≥ 32 * footsize) {
        fprintf(stderr, "Too many special functions (footsize=%d)!\n", footsize);
        exit(-3);
    }
    p->next->prev = p->prev, p->prev->next = p->next;
    p->cost = 1, p->parent = (x[k] << 16) + x[r];
    p->footprint[s >> 5] = 1 << (s & #1f);
    p->next = head[1].next, p->prev = &head[1];
    p->next->prev = p, p->prev->next = p;
    s++;
    c--;
}

```

This code is used in section 10.

12* $\langle \text{Answer queries } 12^* \rangle \equiv$

```

while (1) {
    printf("Asterisks and bits (hex): ");
    fflush(stdout);
    if (!fgets(buf, 100, stdin)) break;
    if (sscanf(buf, "%x %x", &tta, &tth) ≠ 2) break;
    a = tta, b = tth;
    if (b & #8000) b ⊕= #ffff ⊕ a;
    for (j = b, k = 9999; j < #10000; ) {
        if (func[j].cost ≤ k) {
            if (func[j].cost < k)
                for (r = 0; r < mm; r++) footp[r] = 0;
            k = func[j].cost, ttt = j;
            for (r = 0; r < mm; r++) footp[r] |= func[j].footprint[r];
        }
        r = (j | (#ffff - a)) + 1;
        j = (r & (#10000 + a)) + b;
    }
    printf("%04x has cost ", ttt);
    if (ttt & #8000) ttt ⊕= #ffff;
    printf("%d, parents (%04x, %04x), and footprint", func[ttt].cost, func[ttt].parent >> 16,
        func[ttt].parent & #ffff);
    for (j = 0; j < mm; j++)
        if (footp[j]) {
            s = 32 * j;
            for (u = footp[j]; u; u >>= 1, s++)
                if (u & 1) printf(" %s", name[s]);
        }
    printf("\n");
}

```

This code is used in section 1*.

13* Index.

The following sections were changed by the change file: [1](#), [12](#), [13](#).

a: [1](#)*
argc: [1](#)*, [2](#).
argv: [1](#)*, [2](#).
b: [1](#)*
buf: [1](#)*, [12](#)*
c: [1](#)*
cost: [1](#)*, [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [11](#), [12](#)*
exit: [2](#), [11](#).
fflush: [12](#)*
fgets: [12](#)*
footp: [1](#)*, [12](#)*
footprint: [1](#)*, [3](#), [5](#), [7](#), [11](#), [12](#)*
footsize: [1](#)*, [11](#).
fprintf: [2](#), [11](#).
fun: [4](#), [5](#), [6](#), [7](#).
func: [1](#)*, [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [11](#), [12](#)*
head: [1](#)*, [3](#), [5](#), [7](#), [8](#), [9](#), [11](#).
j: [1](#)*
k: [1](#)*
m: [1](#)*
main: [1](#)*
mm: [1](#)*, [3](#), [5](#), [7](#), [10](#), [12](#)*
name: [1](#)*, [10](#), [12](#)*
next: [1](#)*, [3](#), [5](#), [7](#), [8](#), [9](#), [11](#).
node: [1](#)*
node_struct: [1](#)*
p: [1](#)*
parent: [1](#)*, [3](#), [5](#), [7](#), [8](#), [11](#), [12](#)*
pp: [1](#)*, [5](#), [7](#).
pqdone: [3](#), [6](#).
prev: [1](#)*, [5](#), [7](#), [8](#), [9](#), [11](#).
printf: [12](#)*
q: [1](#)*
r: [1](#)*
s: [1](#)*
sprintf: [10](#).
sscanf: [2](#), [12](#)*
stderr: [2](#), [11](#).
stdin: [12](#)*
stdout: [12](#)*
t: [1](#)*
tta: [1](#)*, [12](#)*
tth: [1](#)*, [12](#)*
ttt: [1](#)*, [12](#)*
u: [1](#)*
x: [1](#)*

- ⟨ Answer queries [12*](#) ⟩ Used in section [1*](#).
- ⟨ Combine all functions of costs k and $r - 1 - k$ [3](#) ⟩ Used in section [1*](#).
- ⟨ Initialize the functions of cost 0 [9](#) ⟩ Used in section [8](#).
- ⟨ Initialize the functions of cost 1 [10](#) ⟩ Used in section [8](#).
- ⟨ Initialize the tables [8](#) ⟩ Used in section [1*](#).
- ⟨ Read the initial functions [2](#) ⟩ Used in section [1*](#).
- ⟨ Try for breakthru and **goto** *pqdone* [6](#) ⟩ Used in section [3](#).
- ⟨ Try for new function [4](#) ⟩ Used in section [3](#).
- ⟨ Update for cost 1 [11](#) ⟩ Used in section [10](#).
- ⟨ Update the table for cost $r - 1$ [7](#) ⟩ Used in section [6](#).
- ⟨ Update the table for cost r [5](#) ⟩ Used in section [4](#).

FCHAINS4X-DONTCARES

	Section	Page
Intro	1	1
Index	13	6