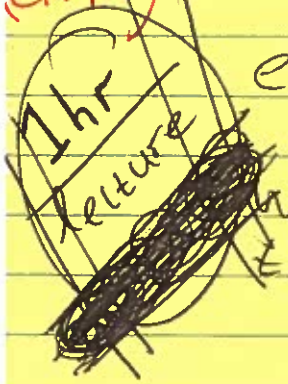
L10.
QUINN
CH4

CIRCUIT SATISFIABILITY



example of [redacted] an NP-Hard pb.

we do not [redacted] know of an algorithm to solve it in polynomial time $O(n^c)$
((JACK EDMONDS)) n = size of the input

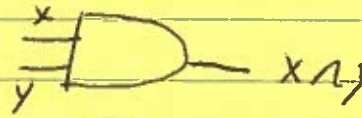
assumption: there are no loops in the circuit

INPUT: set of m boolean (T/F) values $x_1, \ldots, x_n$

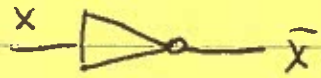
OUTPUT: a single boolean value

for specific input values, we can calculate the output of the circuit in polynomial (LINEAR) time using DFS (depth first search), since we can compute the output of a k-input gate in O(k) time

3 types of gates:

AND GATE 

OR GATE 

NOT GATE 

CircuitSat pb:

given a boolean circuit,
is there an input that
makes the circuit output T
(or the circuit always output F)

naïve solution: try all 2^n possible inputs exhaustively
~> exponential time

no other exact algorithm is available.
local search approaches are available

Cook-Levin THM

CircuitSat is NP-complete

case study: specific circuit with 16 inputs labeled a-p
each input takes 2 values 0, 1
~> $2^{16} = 65,536$ possible inputs

summary of the program design:

we will determine whether the circuit is satisfiable by considering all 65,536 possible input combinations

the combinations will be allocated to the p procs in a cyclic fashion

every proc will examine its combination that output T and print the ones

"cyclic fashion"

n tasks, 0, ..., n-1 are assigned to p procs, 0, ..., p-1 ~>

task k is assigned to proc k (mod p)

ex. n=20, p=6

proc 0 ~> 0, 6, 12, 18

proc 1 ~> 1, 7, 13, 19

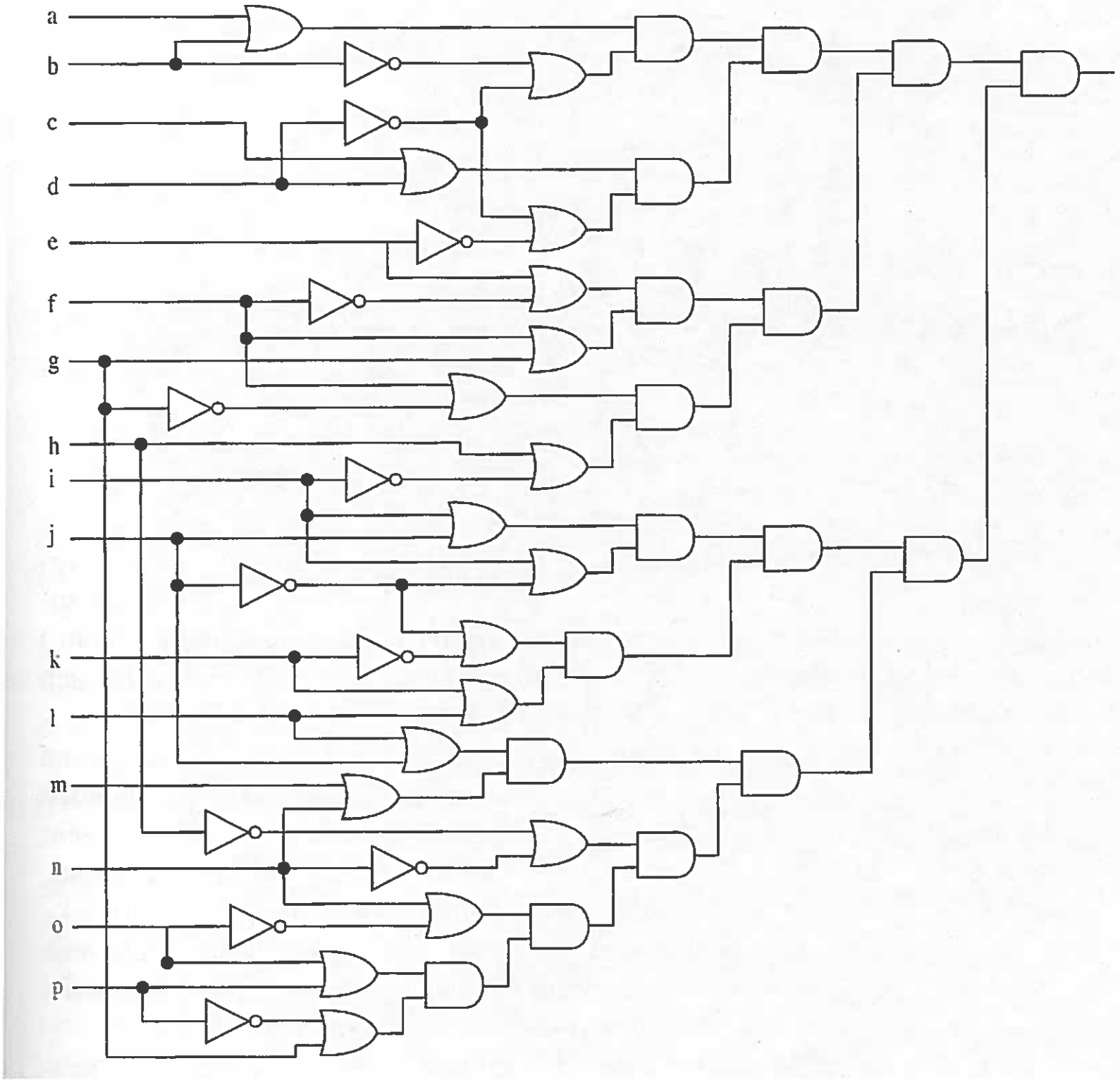
proc 2 ~> 2, 8, 14

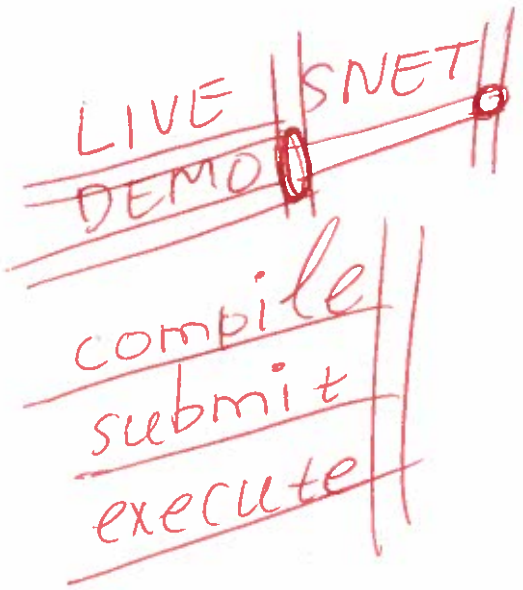
proc 3 ~> 3, 9, 15

proc 4 ~> 4, 10, 16

proc 5 ~> 5, 11, 17

proactical application of CircuitSat: design and verification of logical devices


a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p




```
/*
 *  Circuit Satisfiability, Version 1
 *
 *  This MPI program determines whether a circuit is
 *  satisfiable, that is, whether there is a combination of
 *  inputs that causes the output of the circuit to be 1.
 *  The particular circuit being tested is "wired" into the
 *  logic of function 'check_circuit'. All combinations of
 *  inputs that satisfy the circuit are printed.
 *
 *  Programmed by Michael J. Quinn
 *
 *  Last modification: 3 September 2002
 */

#include "mpi.h"
#include <stdio.h>

int main (int argc, char *argv[]) {
  int i;
  int id;           /* Process rank */
  int p;            /* Number of processes */
  void check_circuit (int, int);

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &id);
  MPI_Comm_size (MPI_COMM_WORLD, &p);

  for (i = id; i < 65536; i += p)
    check_circuit (id, i);

  printf ("Process %d is done\n", id);
  fflush (stdout);
  MPI_Finalize();
  return 0;
}

/* Return 1 if 'i'th bit of 'n' is 1; 0 otherwise */
```

```
#define EXTRACT_BIT(n,i) ((n&(1<<i))?1:0)

void check_circuit (int id, int z) {
  int v[16];      /* Each element is a bit of z */
  int i;

  for (i = 0; i < 16; i++) v[i] = EXTRACT_BIT(z,i);
  if ((v[0] || v[1]) && (!v[1] || !v[3]) && (v[2] || v[3])
    && (!v[3] || !v[4]) && (v[4] || !v[5])
    && (v[5] || !v[6]) && (v[5] || v[6])
    && (v[6] || !v[15]) && (v[7] || !v[8])
    && (!v[7] || !v[13]) && (v[8] || v[9])
    && (v[8] || !v[9]) && (!v[9] || !v[10])
    && (v[9] || v[11]) && (v[10] || v[11])
    && (v[12] || v[13]) && (v[13] || !v[14])
    && (v[14] || v[15])) {
    printf ("%d) %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n", id,
      v[0],v[1],v[2],v[3],v[4],v[5],v[6],v[7],v[8],v[9],
      v[10],v[11],v[12],v[13],v[14],v[15]);
    fflush (stdout);
  }
}
```

LIVE DEMO
snet
compile
submit
execute

```
/*
 *  Circuit Satisfiability, Version 2
 *
 *  This enhanced version of the program prints the
 *  total number of solutions.
 */

#include "mpi.h"
#include <stdio.h>

int main (int argc, char *argv[]) {
  int count;        /* Solutions found by this proc */
  int global_count;    /* Total number of solutions */
  int i;
  int id;           /* Process rank */
  int p;            /* Number of processes */
  int check_circuit (int, int);

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &id);
  MPI_Comm_size (MPI_COMM_WORLD, &p);

  count = 0;
  for (i = id; i < 65536; i += p)
    count += check_circuit (id, i);

  MPI_Reduce (&count, &global_count, 1, MPI_INT, MPI_SUM, 0,
    MPI_COMM_WORLD);
  printf ("Process %d is done\n", id);
  fflush (stdout);
  MPI_Finalize();
  if (!id) printf ("There are %d different solutions\n",
    global_count);
  return 0;
}

/* Return 1 if 'i'th bit of 'n' is 1; 0 otherwise */
#define EXTRACT_BIT(n,i) ((n&(1<<i))?1:0)
```

```
int check_circuit (int id, int z) {
  int v[16];      /* Each element is a bit of z */
  int i;

  for (i = 0; i < 16; i++) v[i] = EXTRACT_BIT(z,i);
  if ((v[0] || v[1]) && (!v[1] || !v[3]) && (v[2] || v[3])
    && (!v[3] || !v[4]) && (v[4] || !v[5])
    && (v[5] || !v[6]) && (v[5] || v[6])
    && (v[6] || !v[15]) && (v[7] || !v[8])
    && (!v[7] || !v[13]) && (v[8] || v[9])
    && (v[8] || !v[9]) && (!v[9] || !v[10])
    && (v[9] || v[11]) && (v[10] || v[11])
    && (v[12] || v[13]) && (v[13] || !v[14])
    && (v[14] || v[15])) {
    printf ("%d) %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n", id,
      v[0],v[1],v[2],v[3],v[4],v[5],v[6],v[7],v[8],v[9],
      v[10],v[11],v[12],v[13],v[14],v[15]);
    fflush (stdout);
    return 1;
  } else return 0;
}
```

program "sat2.c" ~> count the total #sol's

collective communication:
a group of proc's works together to distribute or gather a set of one or more values

REDUCTION is a collective comm/tion operation

~> modify check_circuit so that it returns { 0 combination does not satisfy; 1 combination does satisfy }

~> modify loop to accumulate the #valid sol.'s for each proc.

after a proc has completed its work it will participate in the REDUCTION operation.

MPI_Reduce performs one or more reduction operations on values submitted by all the proc's in a comm/tor.

~~MPI_Reduce~~

int MPI_Reduce (7 params

```
void *operand,      /* addr of 1st reduction elm */
void *result,       /*  "  "  "  reduction result */
int count,          /* reductions to perform */
MPI_Datatype type,  /* type of elms */
MPI_Op operator,    /* reduction operator */
int root            /* process getting result */
MPI_Comm comm )     /* comm/ctor */
```

list of other built-in REDUCTION ops ~> RTM MPI_PROD etc