Satisfiability :
Applications and Backtracking

- *Definition*
- *Applications*
- *API and naive solver*
- *Backtracking*

http://algs4.cs.princeton.edu

Topics and objectives for the final three lecture

What else should a computer scientist know about algorithms?

- For many, this may be the last course on algorithms and complexity

What can be solved (by a computer)?

- The big questions of computer science

What can be solved efficiently?

- There are things that computers will provably not be able to do
- There are thousands of problems that *appear* to be intractable
  - The „NP-complete“ problems

What can we do if we can‘t find an efficient solution?

- Give up one of three things:
  - *Correctness* : Seek approximate answers
  - *Efficiency*: Use (good) exponential time algorithms
  - *Generality*: Solve (important) special cases

Outline of the lectures

Today: Combinatorial generation

- Generating all possible solutions – not too stupidly
- Backtracking : avoid searching hopeless corridors

Next lecture: Heuristics

- Adding some „smartness“ into backtracking search
- The „science of brute force“
    - Recent progress, extremely successful
- Solving special cases: 2-SAT

Third lecture: Heuristics

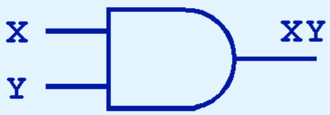
- Adding some „smartness“ into backtracking search: DPLL
- The „science of brute force“
- Recent progress, extremely successful

Satisfiability

Common theme: The Satisfiability Problem (SAT)

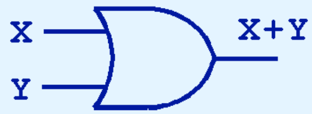
- Hugely practical „meta-problem“
  - Applications in AI, planning, circuit design, program verification...
- Fundamental theoretical importance

Logic gates and boolean operations



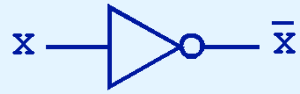
X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1



X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

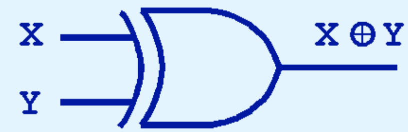


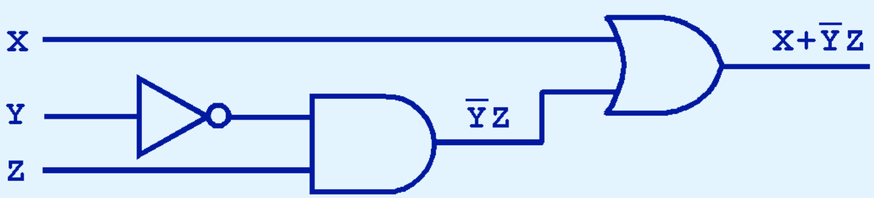
NOT X

X	$\overline{X}$
0	1
1	0

X XOR Y

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0





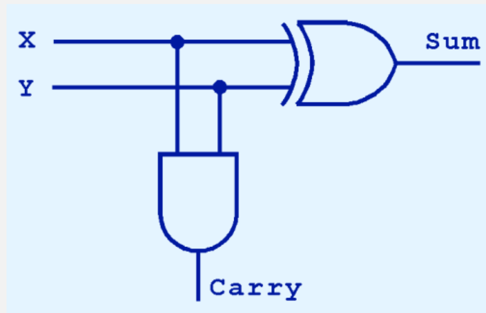
$F(x,y,z) = x\overline{z}+y$

x	y	z	$\overline{z}$	$x\overline{z}$	$x\overline{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Adders

(1-bit) Half adder

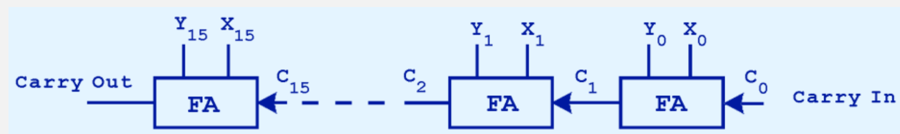
- 2 bit output

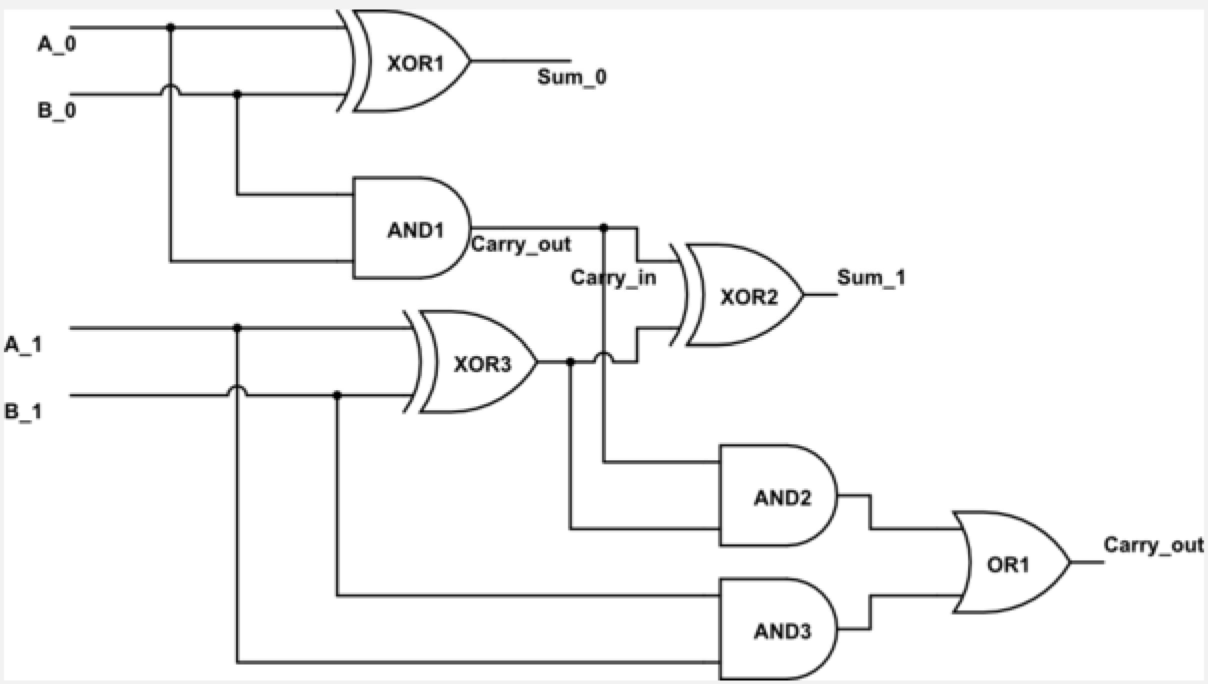


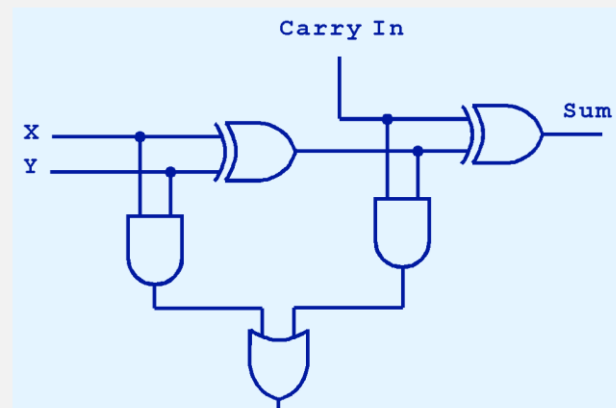
Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2-bit adder

- 3-bit output







Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Boolean Formulas

$\varphi = (x_1 \lor \neg x_2) \land (x_2 \lor x_1) \land (\neg x_1 \lor x_2)$

Truth values: **T** and **F**

Boolean **variable** (true or false): x_1

Literal: Variable x_1 or its negation $\neg x_1$

An **assignment** to the variables: e.g. x_1=T, x_2=F,

Evaluating the formula (with a given assignment)

X_1	X_2	φ
T	T	
T	F	
F	T	
F	F	

X_1	X_2	φ
1	1	
1	0	
0	1	
0	0	

Boolean formulas as a general purpose formulation

the ambassador wants to invite Peru, or exclude Quatar	$P \vee \neg Q$
the vice-ambassador wants Quatar, Romania, or both	$Q \vee R$
impossible to invite both Romania and Peru	$\neg(R \wedge P)$

Who do you invite?

$$\varphi = (P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg(R \wedge P))$$

$$= (P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P)$$

$$= (P \wedge Q \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge R)$$

A satisfying assignment (or model) of φ, is $\{P : T, Q : T, R : F\}$

Conjunctive Normal Form

$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_1)$

X_1	X_2	X_3	φ
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

Clause: Disjunction of literals $(x_1 \vee \neg x_2 \vee x_3)$

Conjunctive normal form (CNF) : conjunction of clauses

Basic fact: Every Boolean formula can be written in CNF.

Satisfiability problem: Given (CNF) boolean formula φ, find a truth assignment to the variables $x_1, x_2, \ldots x_n$, that *satisfies* φ, i.e., such that $\varphi(x_1, x_2, \ldots x_n) = 1$,

Conjunctive Normal Form

$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_1)$

X_1	X_2	X_3	φ
T	T	T	**T**
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Clause: Disjunction of literals $(x_1 \vee \neg x_2 \vee x_3)$

Conjunctive normal form (CNF) : conjunction of clauses

Shorthand notation for CNF formulas:

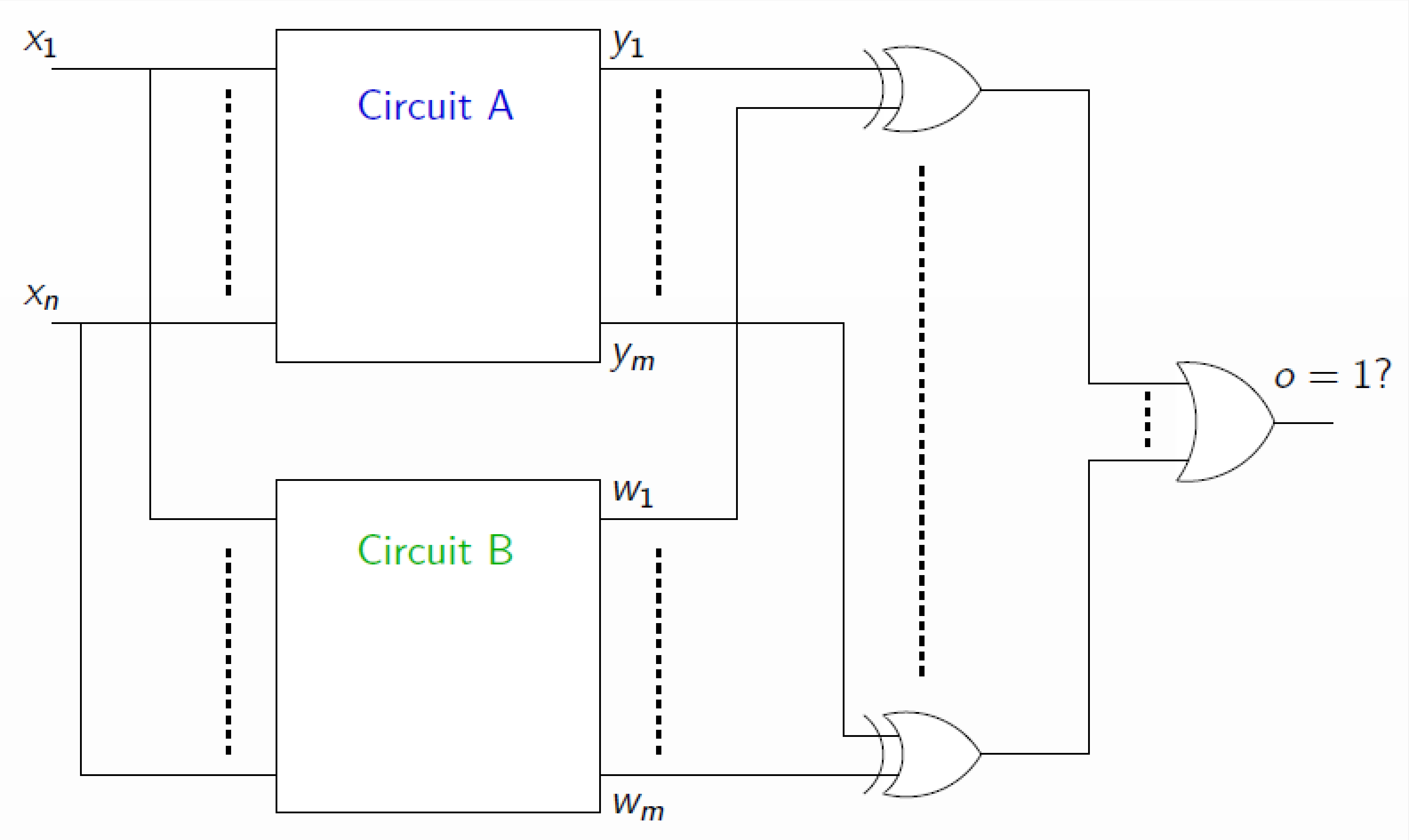
- Omit the ∨ between the literals
- Write $\overline{x_1}$ of $\neg x_1$
- Separate the clauses with a comma (not ∧)

Example: $x_1\overline{x_2}$, $x_1 x_2$, $\overline{x_1}x_2$, $\overline{x_2}x_3$, $\overline{x_3}x_1$

Combinational Equivalence Checking

Equivalence of combinational circuits

- Does Circuit A produce the same output as Circuit B, on every input?



Equivalent to the Satisfiability problem on the combined circuit

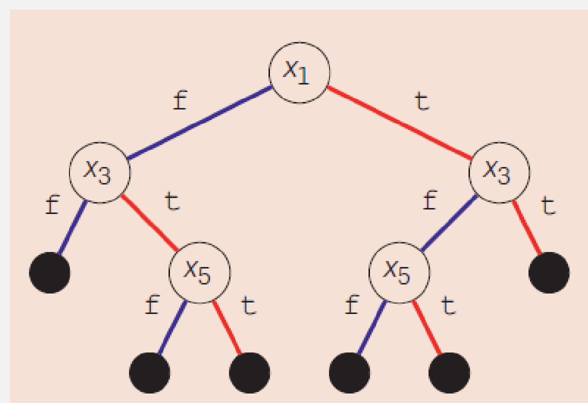
Boolean Schur Triples Problem

Mathematical problem (related to Ramsey theory):

- Does there exists a red/blue coloring of the numbers 1 to n, such that there is no monochromatic solution of $a + b = c$ with $a < b < c \leq n$.

$$
\begin{aligned}
&(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge \\
&(x_1 \vee x_4 \vee x_5) \wedge (\overline{x}_1 \vee \overline{x}_4 \vee \overline{x}_5) \wedge (x_2 \vee x_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5) \wedge \\
&(x_1 \vee x_5 \vee x_6) \wedge (\overline{x}_1 \vee \overline{x}_5 \vee \overline{x}_6) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\overline{x}_2 \vee x_4 \vee \overline{x}_6) \wedge \\
&(x_1 \vee x_6 \vee x_7) \wedge (\overline{x}_1 \vee \overline{x}_6 \vee \overline{x}_7) \wedge (x_2 \vee x_5 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_5 \vee \overline{x}_7) \wedge \\
&(x_3 \vee x_4 \vee x_7) \wedge (\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_7) \wedge (x_1 \vee x_7 \vee x_8) \wedge (\overline{x}_1 \vee \overline{x}_7 \vee \overline{x}_8) \wedge \\
&(x_2 \vee x_6 \vee x_8) \wedge (\overline{x}_2 \vee \overline{x}_6 \vee \overline{x}_8) \wedge (x_3 \vee x_5 \vee x_8) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee \overline{x}_8) \wedge \\
&(x_1 \vee x_8 \vee x_9) \wedge (\overline{x}_1 \vee \overline{x}_8 \vee \overline{x}_9) \wedge (x_2 \vee x_7 \vee x_9) \wedge (\overline{x}_2 \vee \overline{x}_7 \vee \overline{x}_9) \wedge \\
&(x_3 \vee x_6 \vee x_9) \wedge (\overline{x}_3 \vee \overline{x}_6 \vee \overline{x}_9) \wedge (x_4 \vee x_5 \vee x_9) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee \overline{x}_9)
\end{aligned}
$$

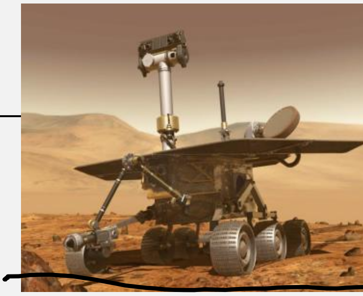
sch.txt

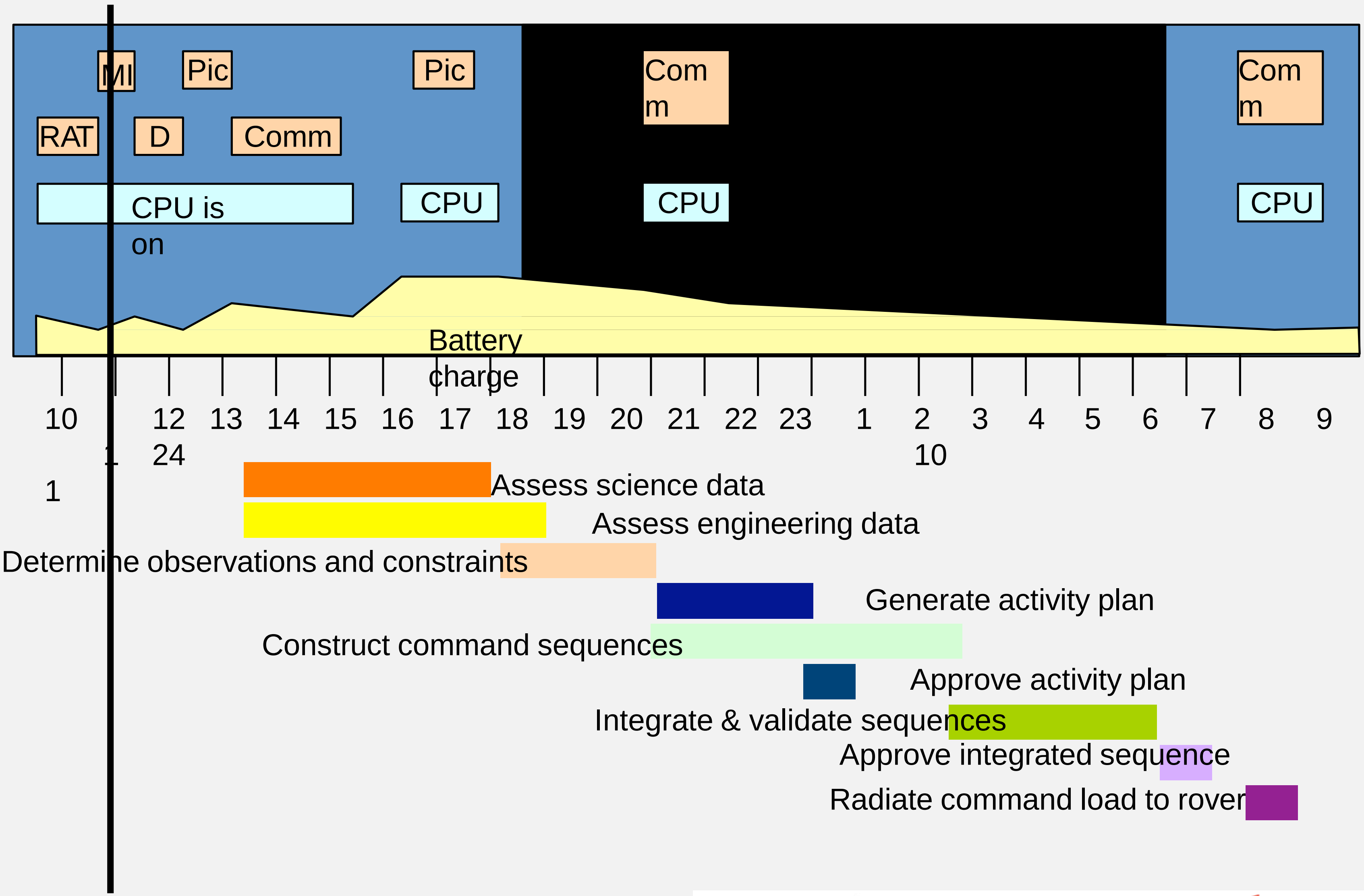


Source:

- Heule, Kullman, "The science of brute force", Communications of the ACM, Aug 2017.

Planning the operations of a Mars Rover



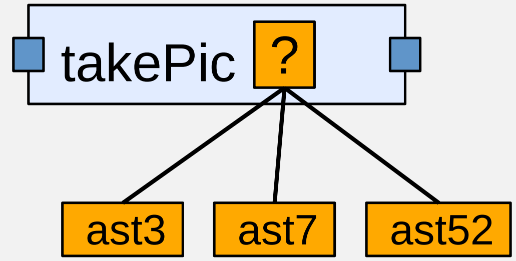


REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Constraint-Based Planning

Activities represented as intervals

- Each interval specifies activity
- Each interval has start and end
- Interval can have parameters

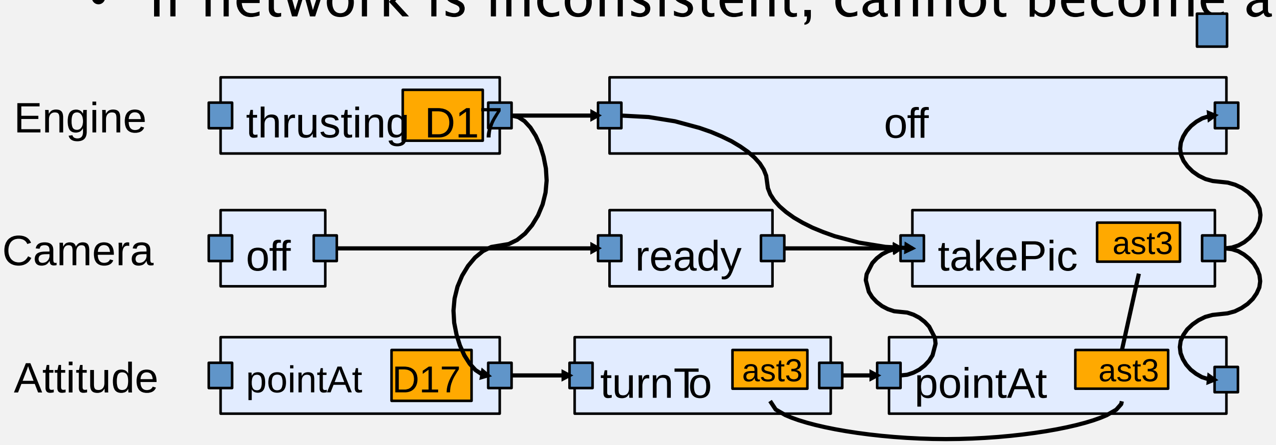


Candidate plan is a network of intervals

- Intervals linked by temporal constraints
- Interval parameters linked by constraints
- Gives rise to constraint network

Feasibility of candidate plan

- If network is inconsistent, cannot become a valid plan



The many (practical) uses of Satisfiability

- **Formal methods:**
  ◮ Hardware model checking;
  Software model checking;
  Termination analysis of term-rewrite systems;
  Test pattern generation (testing of software & hardware); etc.
- **Artificial intelligence:**
  ◮ Planning; Knowledge representation; Games (n-queens, sudoku, etc.)
- **Bioinformatics:**
  ◮ Haplotype inference; Pedigree checking; Maximum quartet consistency; etc.
- (Hardware) **Design automation:**
  ◮ Equivalence checking; Delay computation; Fault diagnosis; Noise analysis
- **Security:**
  ◮ Cryptanalysis; Inversion attacks on hash functions; etc.
- **Computationally hard problems:**
  ◮ Graph coloring; Traveling salesperson; etc.
- **Core engine for (many) other problem domains:**
  ◮ Theorem provers

Why Reduce to SAT

- Many important problems can be reduced to SAT instances
- Efficient algorithms can often decide large, interesting problems
- SAT is simple
- SAT solvers employ very sophisticated search techniques
- SAT solvers are the workhorses of AI

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Satisfiability

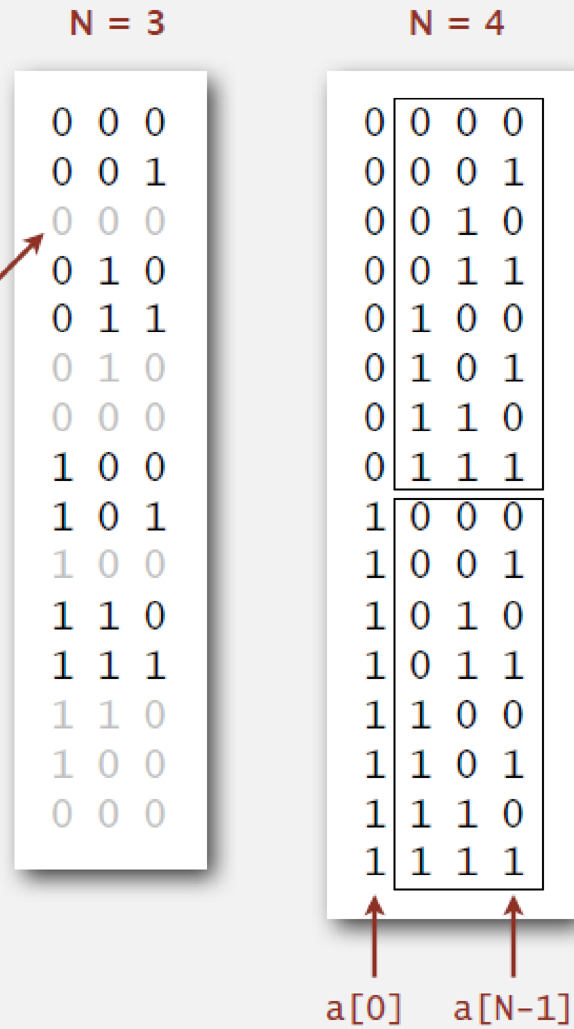
- *Naive (Truth-table) solver*

Warmup: enumerate N-bit strings

Goal. Process all 2^N bit strings of length N.

- Maintain array `a[]` where `a[i]` represents bit `i`.
- Simple recursive method does the job.

```
// enumerate bits in a[k] to a[N-1]
private void enumerate(int k)
{
  if (k == N)
  { process(); return; }
  enumerate(k+1);
  a[k] = 1;
  enumerate(k+1);
  a[k] = 0;          clean up
}
```



Remark. Equivalent to counting in binary from 0 to $2^N - 1$.

Warmup: enumerate N-bit strings

```
public class BinaryCounter
{
   private int N;   // number of bits
   private int[] a; // a[i] = ith bit

   public BinaryCounter(int N)
   {
      this.N = N;
      this.a = new int[N];
      enumerate(0);
   }

   private void process()
   {
      for (int i = 0; i < N; i++)
         StdOut.print(a[i]) + " ";
      StdOut.println();
   }

   private void enumerate(int k)
   {
     if (k == N)
     {  process(); return;  }
     enumerate(k+1);
     a[k] = 1;
     enumerate(k+1);
     a[k] = 0;
   }
}
```

```
public static void main(String[] args)
{
   int N = Integer.parseInt(args[0]);
   new BinaryCounter(N);
}
```

```
% java BinaryCounter 4
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

all programs in this lecture are variations on this theme

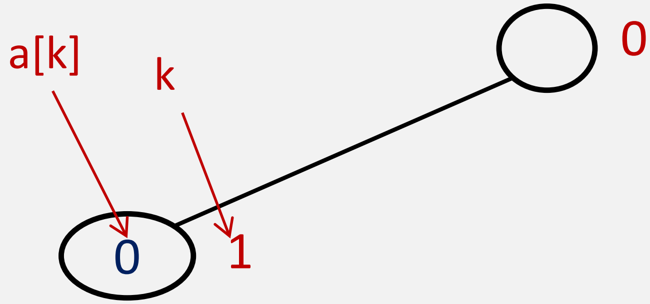
Generating All Assignments

```
public class AsgmtGen
{
  private int N; // number of bits
  private int[] a; // a[i] = ith bit

  public AsgmtGen(int N)
  {
    this.N = N;
    this.a = new int[N];
    enumerate(0);
  }

  private void process()
  {
    char[] tf = { 'F', 'T'};
    for (int i = 0; i < N; i++)
      StdOut.print(tf[a[i]] + " ");
    StdOut.println();
  }

  private void enumerate(int k)
  {
   if (k == N) { process(); return;}
   enumerate(k+1);
   a[k] = 1;
   enumerate(k+1);
   a[k] = 0;
  }
}
```

```
public static void main(String[] args)
{
   int N = Integer.parseInt(args[0]);
   new AsgmtGen(N);
}
```

```
% java AsgmtGen 4
F F F F
F F F T
F F T F
F F T T
F T F F
F T F T
F T T F
F T T T
T F F F
T F F T
T F T F
T F T T
T T F F
T T F T
T T T F
T T T T
```

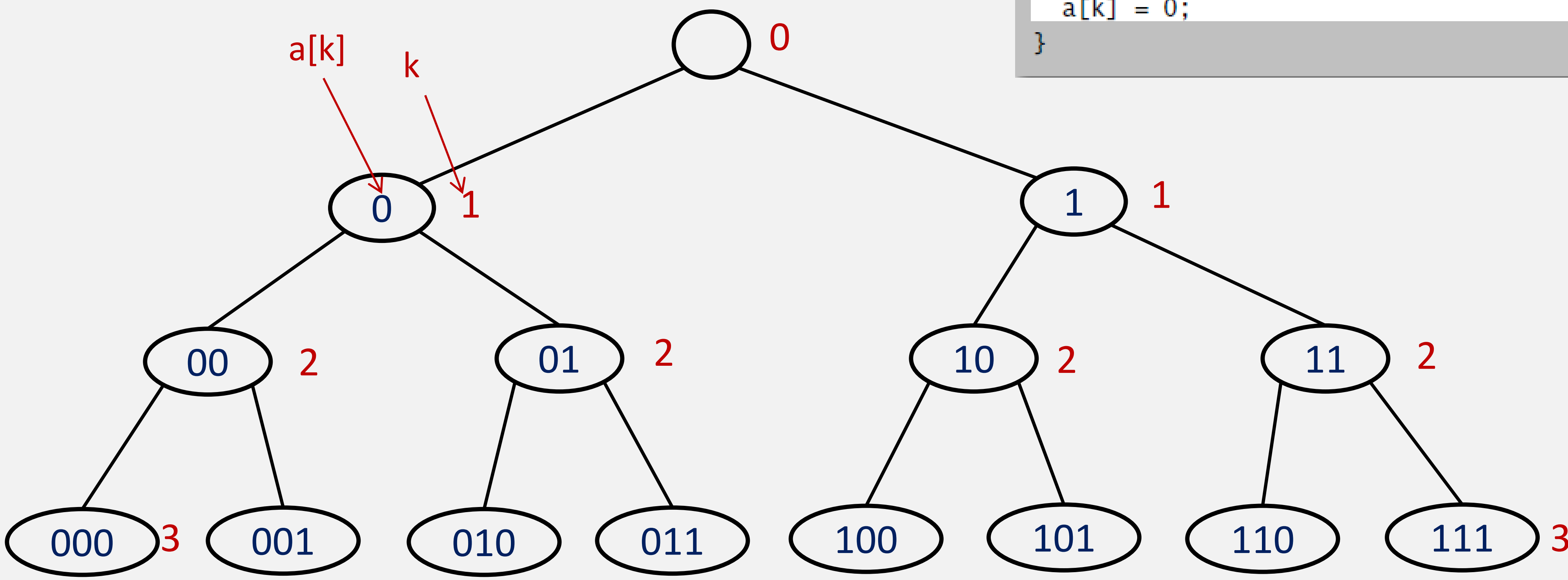
State space




```
// enumerate bits in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }
   enumerate(k+1);
   a[k] = 1;
   enumerate(k+1);
   a[k] = 0;
}
```

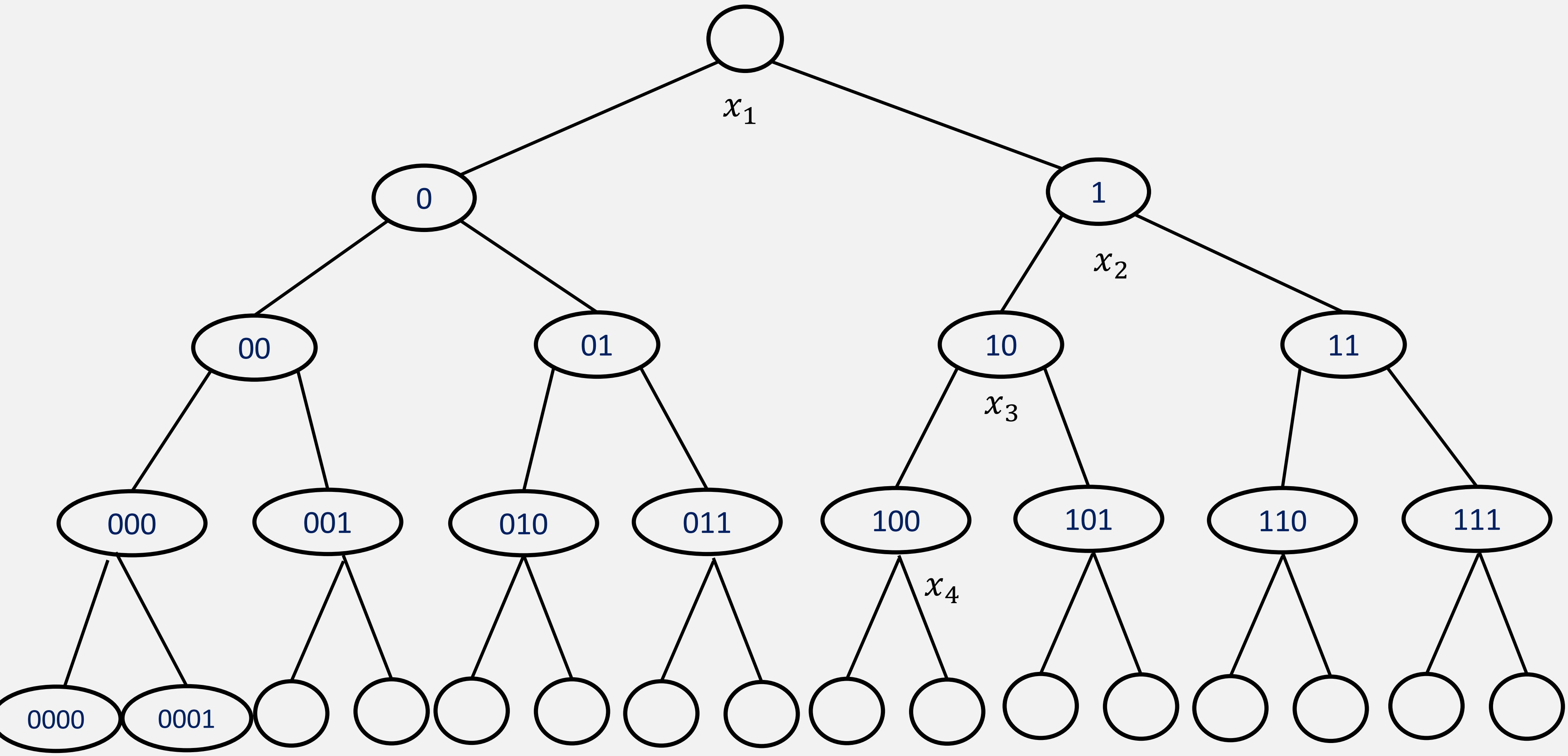
State space

```
// enumerate bits in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }
   enumerate(k+1);
   a[k] = 1;
   enumerate(k+1);
   a[k] = 0;
}
```



Example

$\mathcal{F}: \overline{x_1},\ x_1\overline{x_2},\ x_1x_2x_3,\ \ \overline{x_3}x_4,\ x_1\overline{x_3x_4}$



Satisfiability: Naive exponential time solver

```
public boolean naiveSatisfiability(Formula F)
{
    assignment = new Asgmt();
    return naiveSatisfiability(F, assignment);
}
```

```
private boolean naiveSatisfiability(Formula F, Asgmt asg)
{
    if (asg.size() == F.nVars()) return F.isSatisfied(asg);
    int v = nextVariable(F,asg);
    asg.add(v, true);
    if (naiveSatisfiability(F,asg)) return true;
    asg.add(v, false);
    if (naiveSatisfiability(F,asg)) return true;
    asg.remove(v);
    return false;
}
```

```
private void enumerate(int k)
{
 if (k=N) { process(); return;}
 enumerate(k+1);
 a[k] = 1;
 enumerate(k+1);
 a[k] = 0;
}
```

Algorithms

Robert Sedgewick | Kevin Wayne

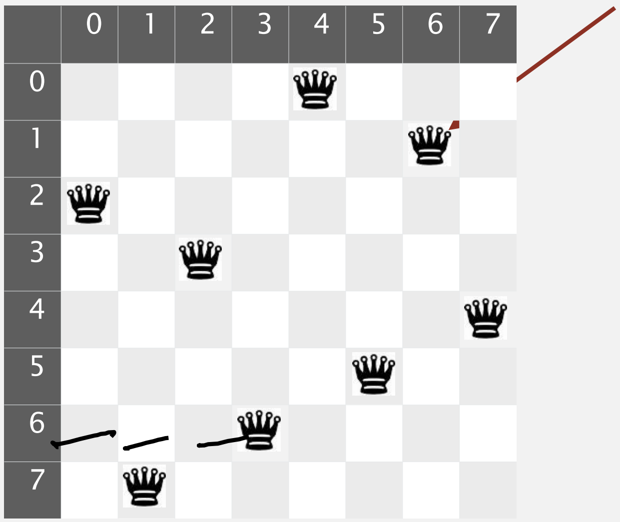
http://algs4.cs.princeton.edu

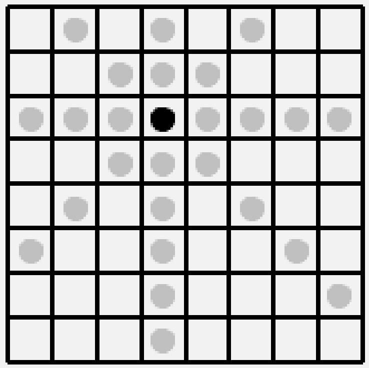
Combinatorial Search

- ‣ *introduction*
- ‣ *permutations*
- ‣ *backtracking*

N-queens problem

Q. How many ways are there to place N queens on an N-by-N board so that no queen can attack any other?





```
int[] a = { 2, 7, 3, 6, 0, 5, 1, 4 };
```

Representation. No 2 queens in the same row or column $\Rightarrow$ permutation.

Additional constraint. No diagonal attack is possible.

Challenge. Enumerate (or even count) the solutions. ← unlike N-rooks problem, nobody knows answer for N > 30

N - queens is a Satisfiability problem

An N-queens placement is <u>valid</u>

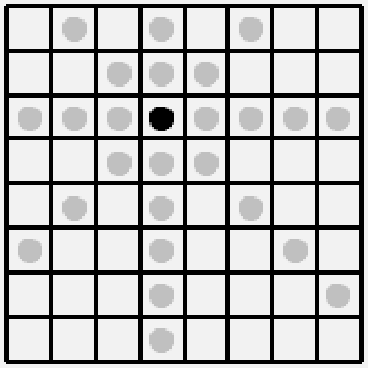
iff

a. Every column has a queen $(\wedge_i \, x_{1i} \vee x_{2i} \vee x_{3i} \vee x_{4i})$, AND

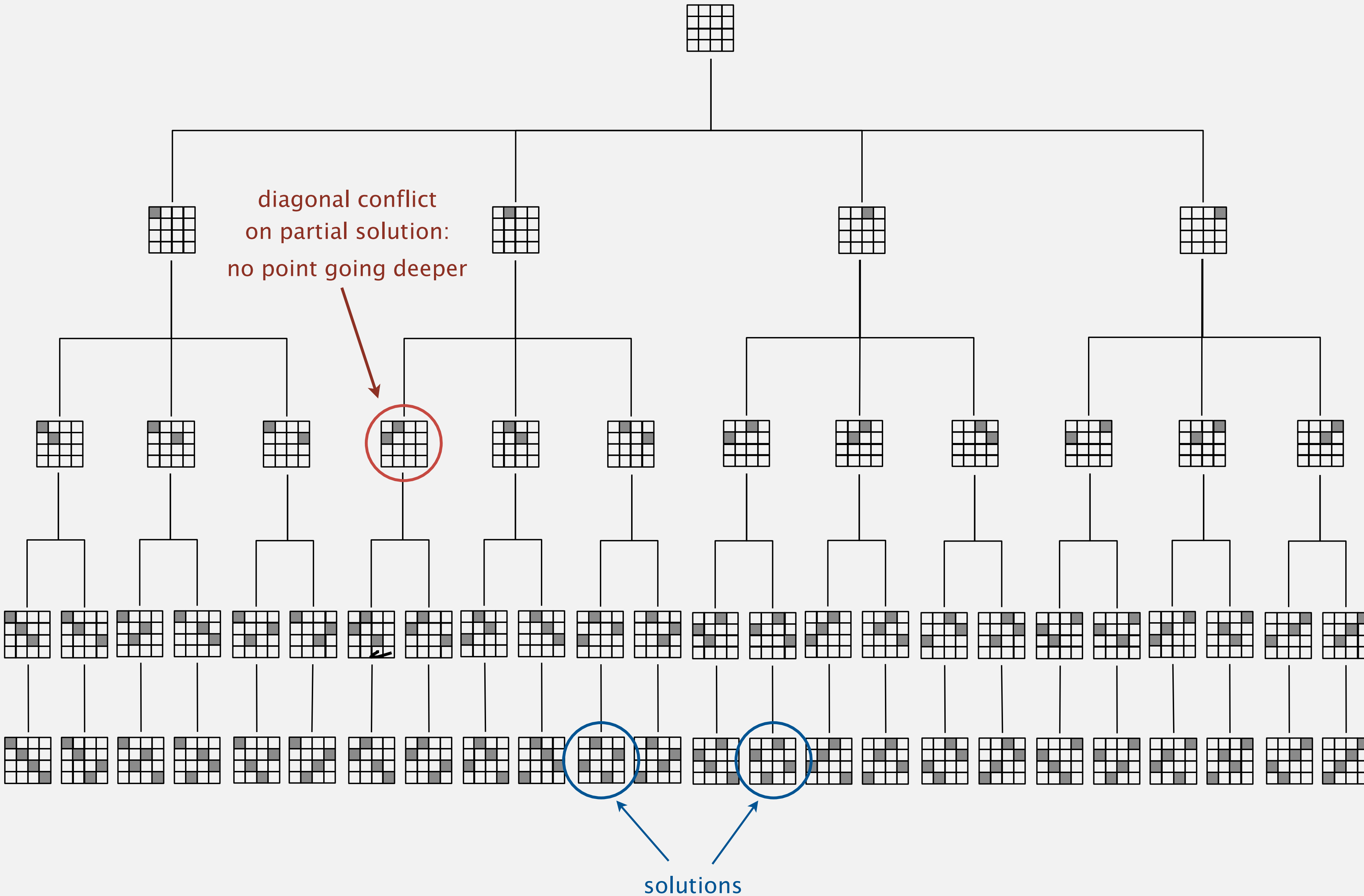
b. Every row has a queen $(\wedge_i \, x_{i1} \vee x_{i2} \vee x_{i3} \vee x_{i4})$ AND

c. No two attacking locations both contain queens

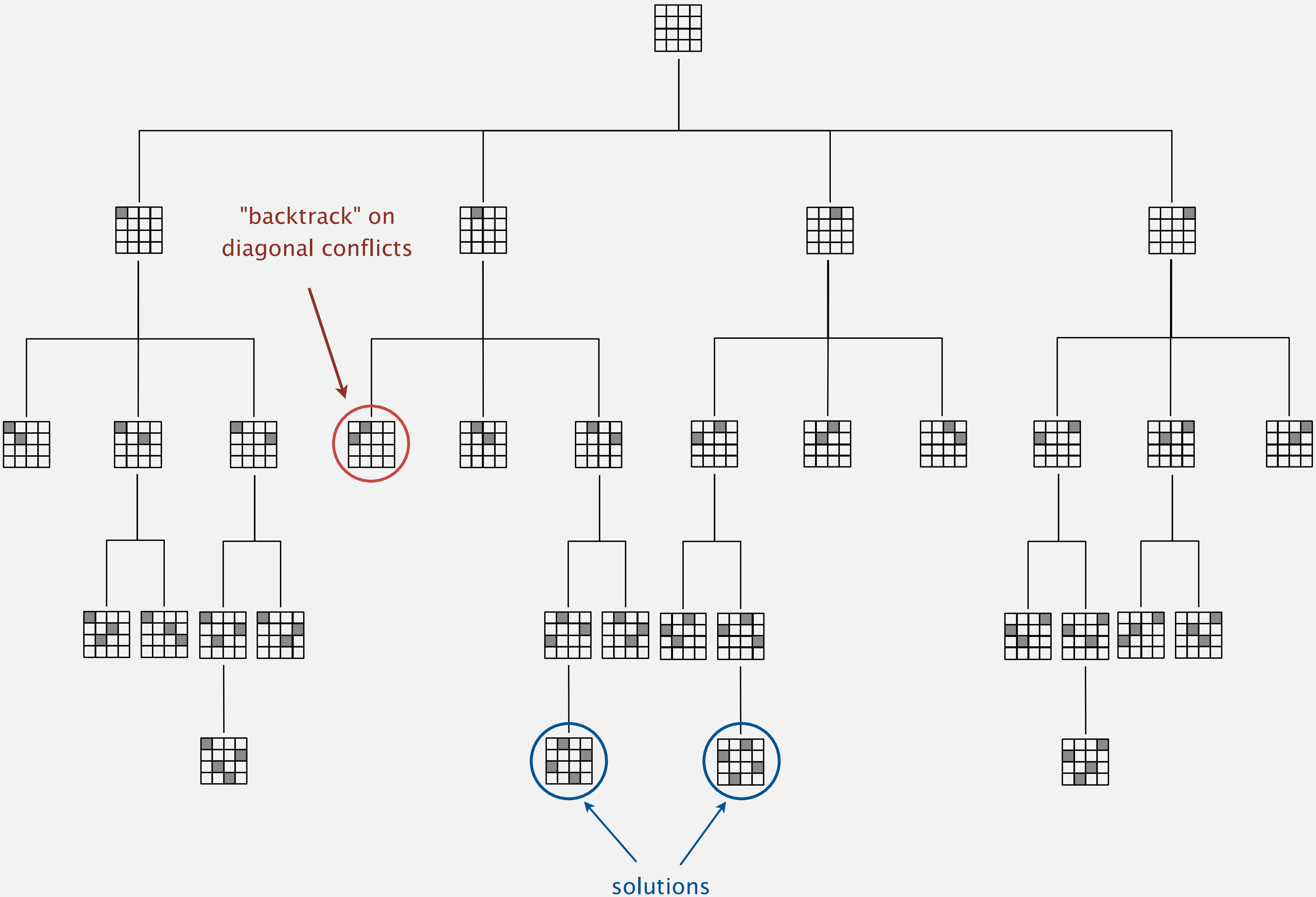
(For every two attacking locations ij and kl, $\overline{x_{ij}} \vee \overline{x_{kl}}$



4-queens search tree



4-queens search tree (pruned)



Backtracking paradigm. Iterate through elements of search space.

- When there are several possible choices, make one choice and recur.
- If the choice is a dead end, backtrack to previous choice, and make next available choice.

Benefit. Identifying dead ends allows us to prune the search tree.

Ex. [backtracking for N-queens problem]

- Dead end: a diagonal conflict.
- Pruning: backtrack and try next column when diagonal conflict found.

Applications. Puzzles, combinatorial optimization, parsing, ...

N-queens problem: Backtracking solution

```
private boolean canBacktrack(int k)
{
   for (int i = 0; i < k; i++)
   {
      if ((a[i] - a[k]) == (k - i)) return true;
      if ((a[k] - a[i]) == (k - i)) return true;
   }
   return false;
}

// place N-k queens in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;   }

   for (int i = k; i < N; i++)
   {
      exch(k, i);
      if (!canBacktrack(k)) enumerate(k+1);
      exch(i, k);
   }
}
```

stop enumerating if adding queen k leads to a diagonal violation

```
% java Queens 4
1 3 0 2
2 0 3 1

% java Queens 5
0 2 4 1 3
0 3 1 4 2
1 3 0 2 4
1 4 2 0 3
2 0 3 1 4
2 4 1 3 0
3 1 4 2 0
3 0 2 4 1
4 1 3 0 2
4 2 0 3 1

% java Queens 6
1 3 5 0 2 4
2 5 1 4 0 3
3 0 4 1 5 2
4 2 0 5 3 1
```

a[0]   a[N-1]

N-queens problem: effectiveness of backtracking

Pruning the search tree leads to enormous time savings.

N	Q(N)	N !	time (sec)
8	92	40,320	–
9	352	362,880	–
10	724	3,628,800	–
11	2,680	39,916,800	–
12	14,200	479,001,600	1.1
13	73,712	6,227,020,800	5.4
14	365,596	87,178,291,200	29
15	2,279,184	1,307,674,368,000	210
	14,772,512	20,922,789,888,000	

Conjecture. $Q(N) \sim N! \ / \ c^N$, where c is about 2.54.

Hypothesis. Running time is about $(N! \ / \ 2.5^N) \ / \ 43{,}000$ seconds.

http://algs4.cs.princeton.edu

Satisfiability and NP-completeness

- *Naive truth-table solver*
- *Backtracking*

Satisfiability: Backtracking solver

```
private boolean bbSatisfiability(Formula F, Asgmt asg) {
  if (F.isValuated(asg)) return F.isSatisfied(asg);
   int v = nextVariable(F,asg);
   asg.add(v, true);
   if (bbSatisfiability(F,asg)) return true;
   asg.setValue(v, false);
   if (bbSatisfiability(F,asg)) return true;
   asg.unset(v);
   return false;
}
```

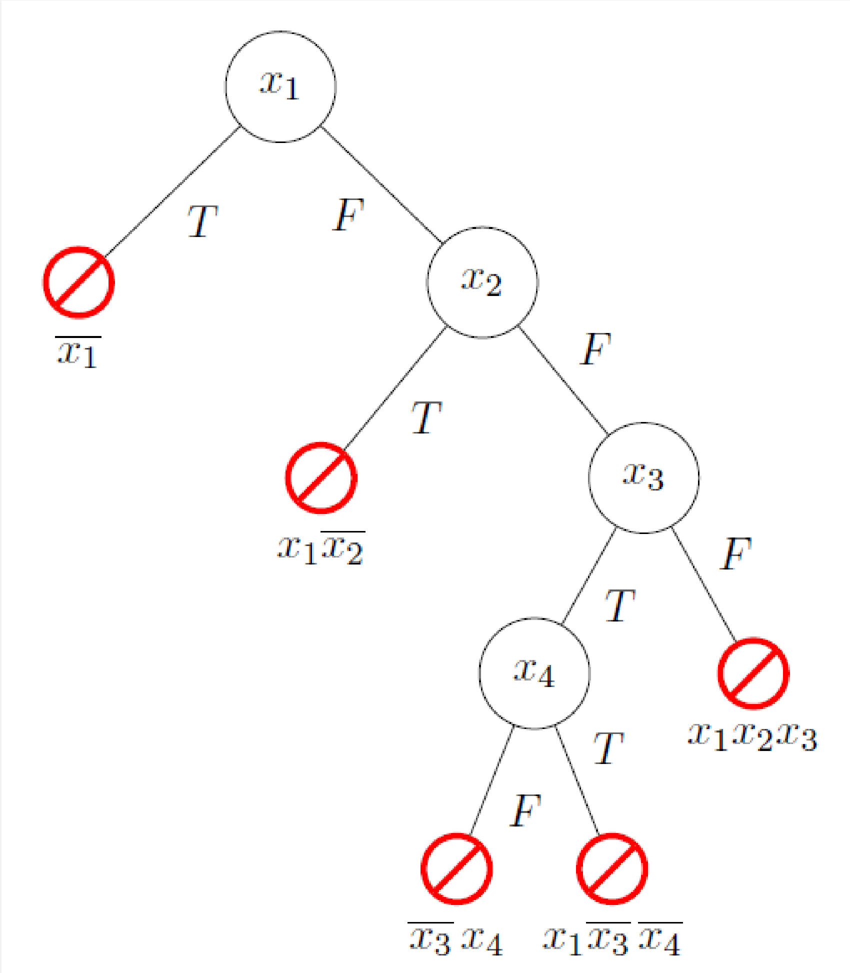
```
private void enumerate(int k)
{
 if (k=N) { process(); return;}
 enumerate(k+1);
 a[k] = 1;
 enumerate(k+1);
 a[k] = 0;
}
```

Example

$$\mathcal{F} : \overline{x_1},\ x_1\overline{x_2},\ x_1x_2x_3,\ \ \overline{x_3}x_4,\ x_1\overline{x_3x_4}$$

Example

$$\mathcal{F}:\overline{x_1},\ x_1\overline{x_2},\ x_1x_2x_3,\ \ \overline{x_3}x_4,\ x_1\overline{x_3x_4}$$



An example CNF formula

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$

34-clause formula

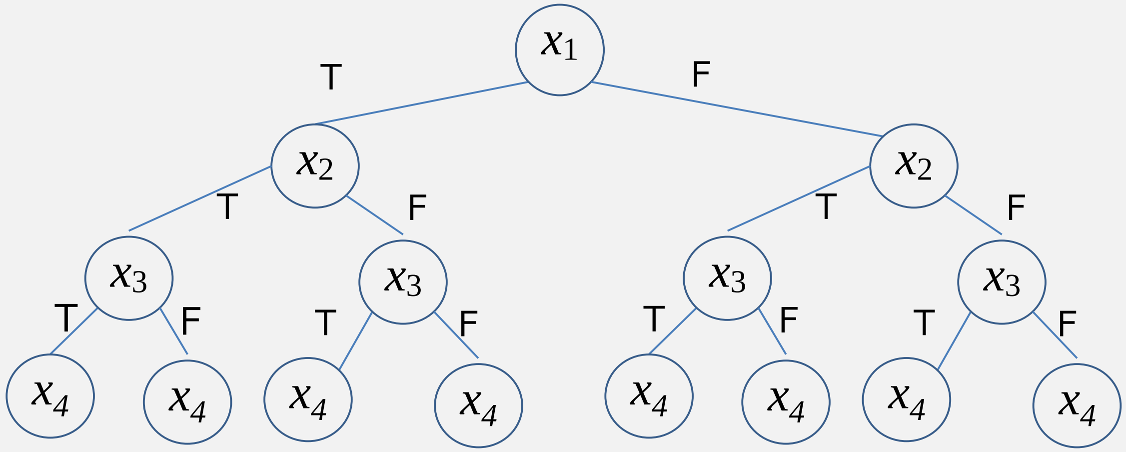
$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$

(Naive) Brute-force solution

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$

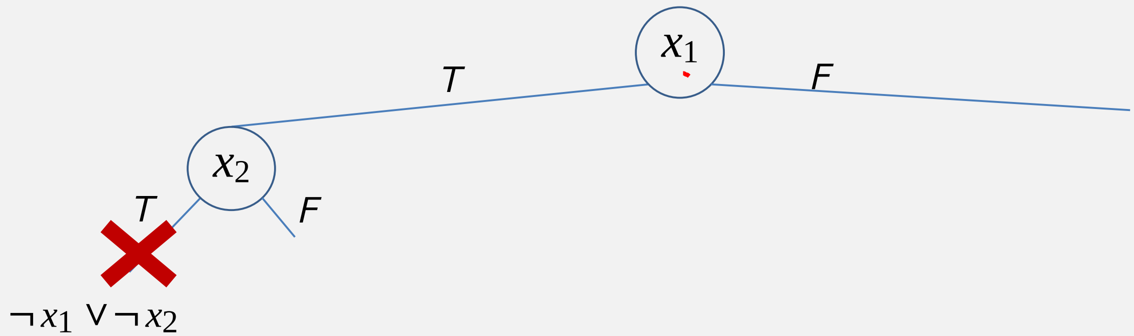


This continues 8 more levels, to level 12, for a complete binary tree with 4096 leaves.

Basic backtracking heuristic

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$



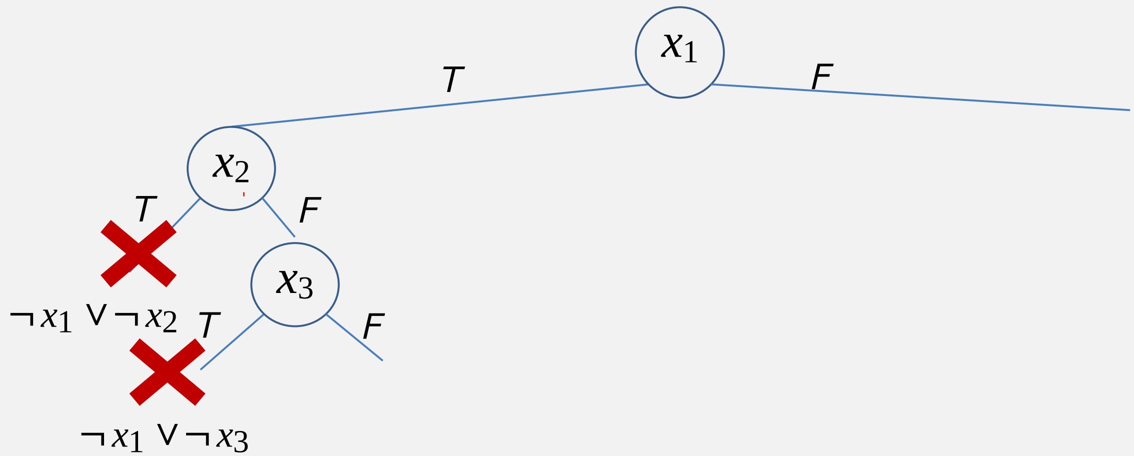
When x_1 receives the value T (= true), then x_2 must necessarily receive the value F(= false), because of the clause $\neg x_1 \vee \neg x_2$.

We label the dead-end (the red X) with the clause that conflicts.

Basic backtracking heuristic

$x_1 \lor x_2 \lor x_3$
$\neg x_1 \lor \neg x_2$
$\neg x_1 \lor \neg x_3$
$\neg x_2 \lor \neg x_3$
$x_4 \lor x_5 \lor x_6$
$\neg x_4 \lor \neg x_5$
$\neg x_4 \lor \neg x_6$
$\neg x_5 \lor \neg x_6$
$x_7 \lor x_8 \lor x_9$
$\neg x_7 \lor \neg x_8$
$\neg x_7 \lor \neg x_9$
$\neg x_8 \lor \neg x_9$
$x_{10} \lor x_{11} \lor x_{12}$
$\neg x_{10} \lor \neg x_{11}$
$\neg x_{10} \lor \neg x_{12}$
$\neg x_{11} \lor \neg x_{12}$
$\neg x_8 \lor \neg x_{11}$

$\neg x_1 \lor \neg x_4$
$\neg x_2 \lor \neg x_5$
$\neg x_3 \lor \neg x_6$
$\neg x_1 \lor \neg x_7$
$\neg x_2 \lor \neg x_8$
$\neg x_3 \lor \neg x_9$
$\neg x_1 \lor \neg x_{10}$
$\neg x_2 \lor \neg x_{11}$
$\neg x_3 \lor \neg x_{12}$
$\neg x_4 \lor \neg x_7$
$\neg x_5 \lor \neg x_8$
$\neg x_6 \lor \neg x_9$
$\neg x_4 \lor \neg x_{10}$
$\neg x_5 \lor \neg x_{11}$
$\neg x_6 \lor \neg x_{12}$
$\neg x_7 \lor \neg x_{10}$
$\neg x_9 \lor \neg x_{12}$

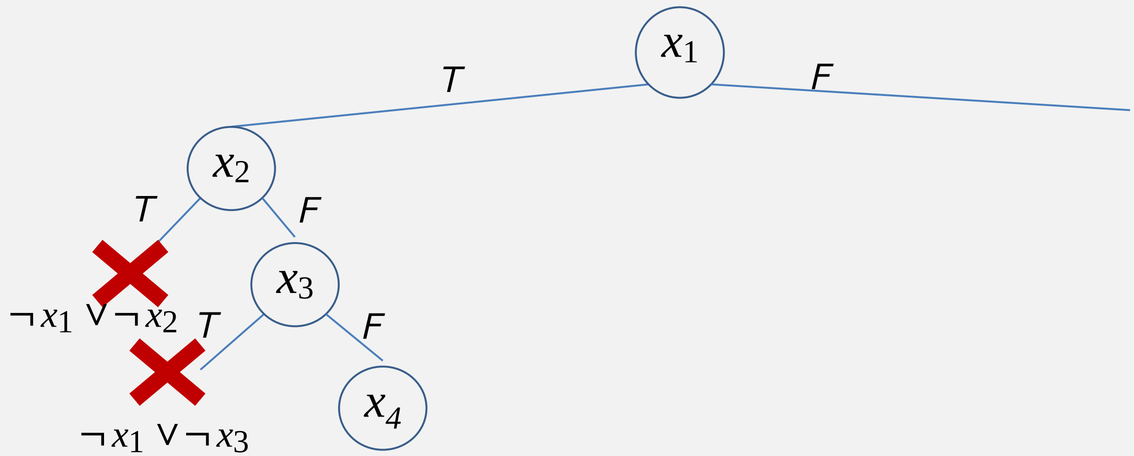


Similarly, x_3 must necessarily receive the value F(= false), because of the clause $\neg x_1 \lor \neg x_3$.
The same holds for x_4.

Basic backtracking heuristic

$\cancel{x_1 \vee x_2 \vee x_3}$
$\cancel{\neg x_1 \vee \neg x_2}$
$\cancel{\neg x_1 \vee \neg x_3}$
$\cancel{\neg x_2 \vee \neg x_3}$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

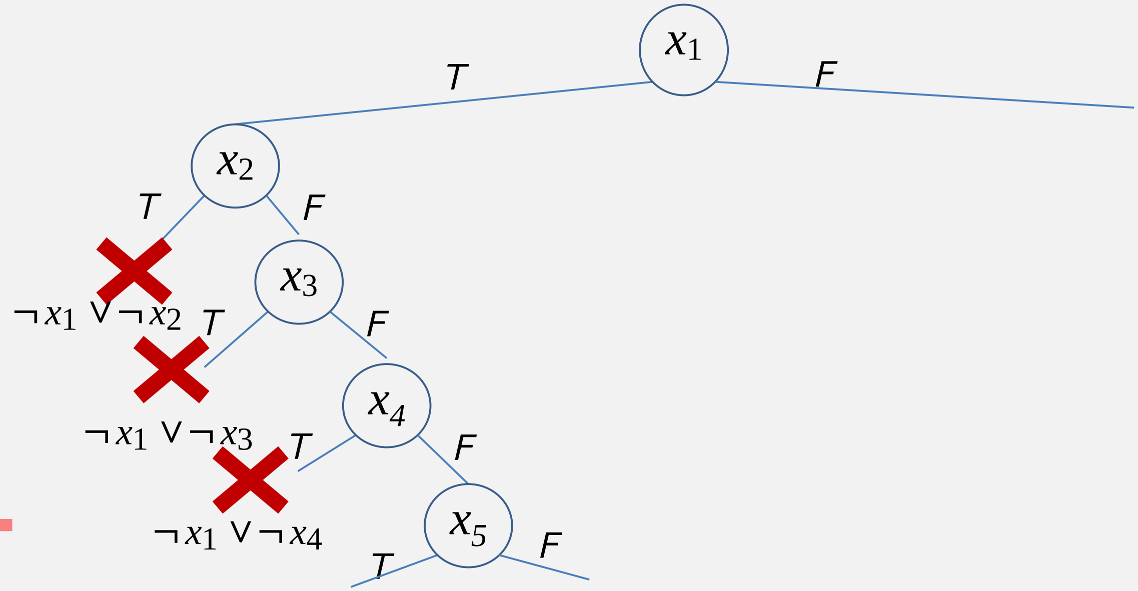
$\cancel{\neg x_1} \vee \neg x_4$
$\cancel{\neg x_2 \vee \neg x_5}$
$\cancel{\neg x_3 \vee \neg x_6}$
$\cancel{\neg x_1} \vee \neg x_7$
$\cancel{\neg x_2 \vee \neg x_8}$
$\cancel{\neg x_3 \vee \neg x_9}$
$\cancel{\neg x_1} \vee \neg x_{10}$
$\cancel{\neg x_2 \vee \neg x_{11}}$
$\cancel{\neg x_3 \vee \neg x_{12}}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$



Basic backtracking heuristic

$\cancel{x_1 \vee x_2 \vee x_3}$
$\cancel{\neg x_1 \vee \neg x_2}$
$\cancel{\neg x_1 \vee \neg x_3}$
$\cancel{\neg x_2 \vee \neg x_3}$
$x_4 \vee x_5 \vee x_6$
$\cancel{\neg x_4 \vee \neg x_5}$
$\cancel{\neg x_4 \vee \neg x_6}$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\cancel{\neg x_1 \vee \neg x_4}$
$\cancel{\neg x_2 \vee \neg x_5}$
$\cancel{\neg x_3 \vee \neg x_6}$
$\neg x_1 \vee \neg x_7$
$\cancel{\neg x_2 \vee \neg x_8}$
$\cancel{\neg x_3 \vee \neg x_9}$
$\neg x_1 \vee \neg x_{10}$
$\cancel{\neg x_2 \vee \neg x_{11}}$
$\cancel{\neg x_3 \vee \neg x_{12}}$
$\cancel{\neg x_4 \vee \neg x_7}$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\cancel{\neg x_4 \vee \neg x_{10}}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$



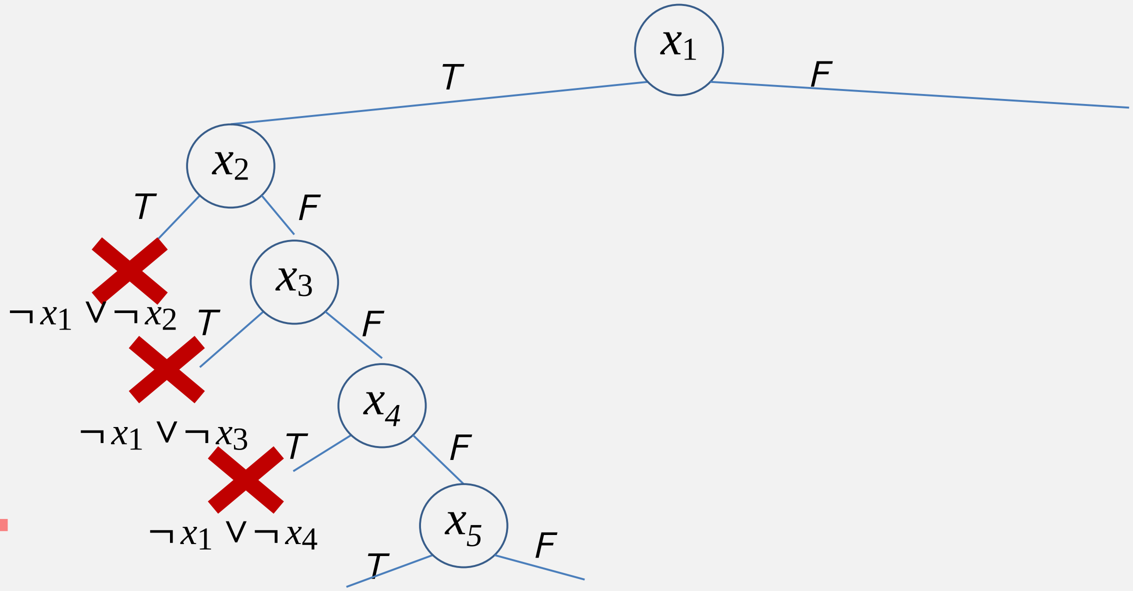
Both values are possible for the variable x_5 now. We shall only explore here the case when we set x_5 true.

We will show that the formula cannot be satisfied in this case (when both x_1 and x_5 are true).

Basic backtracking heuristic

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

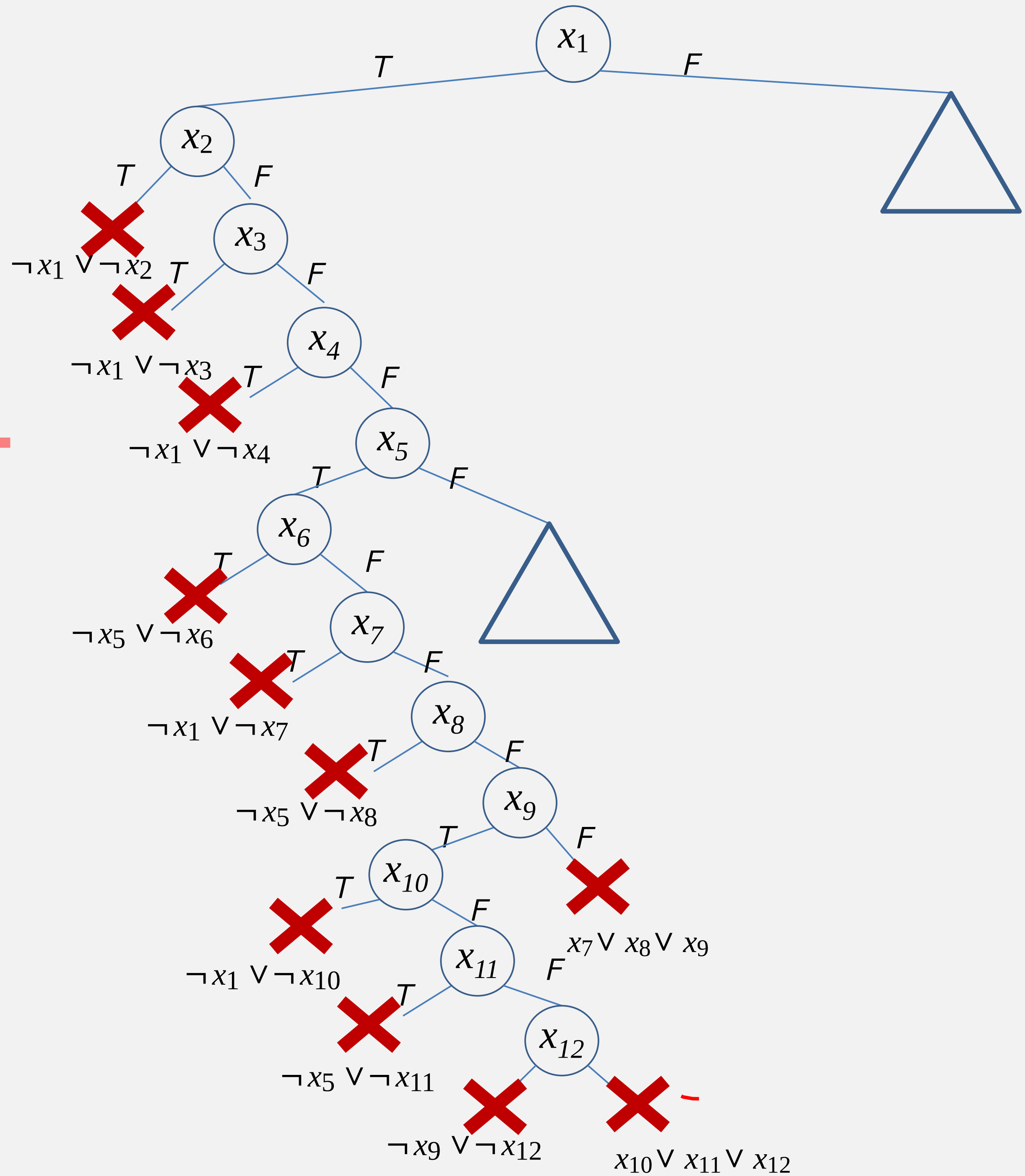
$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$



Basic backtracking heuristic

$x_1 \lor x_2 \lor x_3$
$\neg x_1 \lor \neg x_2$
$\neg x_1 \lor \neg x_3$
$\neg x_2 \lor \neg x_3$
$x_4 \lor x_5 \lor x_6$
$\neg x_4 \lor \neg x_5$
$\neg x_4 \lor \neg x_6$
$\neg x_5 \lor \neg x_6$
$x_7 \lor x_8 \lor x_9$
$\neg x_7 \lor \neg x_8$
$\neg x_7 \lor \neg x_9$
$\neg x_8 \lor \neg x_9$
$x_{10} \lor x_{11} \lor x_{12}$
$\neg x_{10} \lor \neg x_{11}$
$\neg x_{10} \lor \neg x_{12}$
$\neg x_{11} \lor \neg x_{12}$
$\neg x_8 \lor \neg x_{11}$

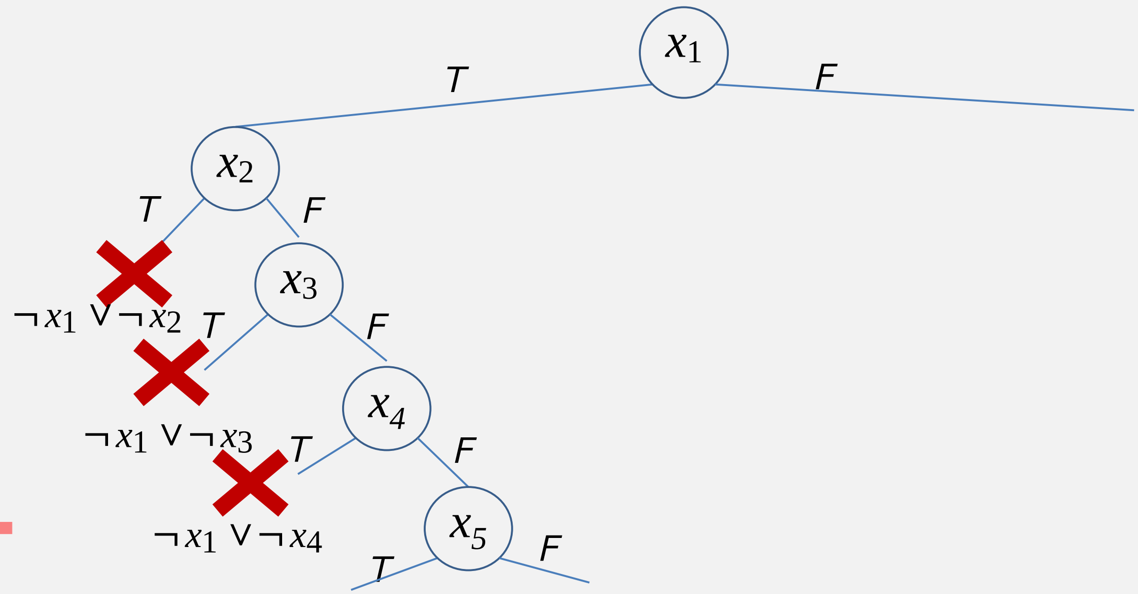
$\neg x_1 \lor \neg x_4$
$\neg x_2 \lor \neg x_5$
$\neg x_3 \lor \neg x_6$
$\neg x_1 \lor \neg x_7$
$\neg x_2 \lor \neg x_8$
$\neg x_3 \lor \neg x_9$
$\neg x_1 \lor \neg x_{10}$
$\neg x_2 \lor \neg x_{11}$
$\neg x_3 \lor \neg x_{12}$
$\neg x_4 \lor \neg x_7$
$\neg x_5 \lor \neg x_8$
$\neg x_6 \lor \neg x_9$
$\neg x_4 \lor \neg x_{10}$
$\neg x_5 \lor \neg x_{11}$
$\neg x_6 \lor \neg x_{12}$
$\neg x_7 \lor \neg x_{10}$
$\neg x_9 \lor \neg x_{12}$



Basic backtracking heuristic

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee \neg x_2$
$\neg x_1 \vee \neg x_3$
$\neg x_2 \vee \neg x_3$
$x_4 \vee x_5 \vee x_6$
$\neg x_4 \vee \neg x_5$
$\neg x_4 \vee \neg x_6$
$\neg x_5 \vee \neg x_6$
$x_7 \vee x_8 \vee x_9$
$\neg x_7 \vee \neg x_8$
$\neg x_7 \vee \neg x_9$
$\neg x_8 \vee \neg x_9$
$x_{10} \vee x_{11} \vee x_{12}$
$\neg x_{10} \vee \neg x_{11}$
$\neg x_{10} \vee \neg x_{12}$
$\neg x_{11} \vee \neg x_{12}$
$\neg x_8 \vee \neg x_{11}$

$\neg x_1 \vee \neg x_4$
$\neg x_2 \vee \neg x_5$
$\neg x_3 \vee \neg x_6$
$\neg x_1 \vee \neg x_7$
$\neg x_2 \vee \neg x_8$
$\neg x_3 \vee \neg x_9$
$\neg x_1 \vee \neg x_{10}$
$\neg x_2 \vee \neg x_{11}$
$\neg x_3 \vee \neg x_{12}$
$\neg x_4 \vee \neg x_7$
$\neg x_5 \vee \neg x_8$
$\neg x_6 \vee \neg x_9$
$\neg x_4 \vee \neg x_{10}$
$\neg x_5 \vee \neg x_{11}$
$\neg x_6 \vee \neg x_{12}$
$\neg x_7 \vee \neg x_{10}$
$\neg x_9 \vee \neg x_{12}$



http://algs4.cs.princeton.edu

Satisfiability

- *APIs and code*

Code

Software for Satisfiability: http://www.ru.is/~mmh/satisfiability/Satisfiability.html

Plotting.

- All classes contain standalone unit tests.
- Results are displayed graphically

Key to graphic output.

- Assignment on right
  - Blue: Set to true
  - Grey: Set to false
- Formula/clause on left
  - Blue literal: Satisfied (evaluates to true)
  - Grey literal: Evalutes to false
  - Black literal: Does not have a value
- Clauses:
  - Strikethrough: Clause evalutes to true
  - Cyan background: Clause evalutes to false (falsifies formula)

API: Assignment

public class	Asgmt	
	public Asgmt()	*Constructor*
void	add(int v, boolean val)	*Add one variable's value*
boolean	get(int v)	*Return the value of the variable*
boolean	contains(int v)	*Check if a variable has value*
void	remove(int v)	*Remove variable from asgmt*
Iterable<Integer>	vars()	*Return all the variables*
int	size()	*Return no. of literals*
void	unset(Asgmt asg2)	*Remove all variables appearing in asg2 from this assignment*
void	joinAsgmt(Asgmt asg2)	*Merge two assignments*
String	toString()	*string representation*

API: Clause

public class	Clause	
	`public Clause()`	*New clause*
`Iterable<Integer>`	`vars()`	*Return all the variables*
`int`	`size()`	*Return no. of literals*
`boolean`	`sign(int var)`	*Return the sign of variable*
`void`	`void add(int v, boolean val)`	*Insert literal*
`boolean`	`contains(int v)`	*Does variable appear in clause?*
`boolean`	`isTrue(Asgmt asg)`	*Evaluates to true with asg?*
`boolean`	`isFalse(Asgmt asg)`	*Evaluates to false with asg?*
`boolean`	`isUnitClause(Asgmt asg)`	*Are all but one literals of the clause false?*
`String`	`toString()`	*string representation*

API: Boolean formula

public class	Fmla	
	public Formula(int n)	*Initialize an empty formula*
	public Formula(In in)	*Read formula from file*
Iterable<Clause>	clauses()/vars()	*Iterate over the clauses/variables*
int	nClauses(); nVars();	*Return number of clauses/variables*
void	addClause(Clause cl)	*Adding clauses*
boolean	isValuated(Asgmt asg)	*Can we compute the value of the formula from the asgmt?*
boolean	isSatisfied(Asgmt asg)	*Does the assignment satisfy the formula?*
String	toString()	*string representation*

Library method in class FormulaGeneration:

```
public static Formula RandomCNF(int n, int m, int k)
```

Generate random k-CNF formula with n variables and m clauses

API: Solvers

public class	Solvers	
	public Solvers()	*New instance*
boolean	naiveSatisfiability()	*Naive solver, returning satisfiability*
boolean	bbSatisfiability()	*Backtracking solver, returning satisfiability*
boolean	dpSatisfiability()	*DPLL solver, returning satisfiability*
Asgmt	satAsgmt()	*Return the satisfying assignment*
int	nStates()	*Return the number of solver states*
Asgmt	unitLits(Formula F, Asgmt asg)	*Perform unit clause propagation*
Int	nextVariable(Formula F, Asgmt asg)	*Find the next variable to branch on*
String	toString()	*string representation*

Formula class implementation

```
public class Formula
{
 private final int n;              // # of variables
 private Bag<Clauses> clauses;
}
```

Formula I/O

```
public Formula(In in) {
  int n = in.readInt(), m = in.readInt();
  this.n = n; this.m = m;
  clauses = new Bag<Clauses>();
  in.readLine();
  for (int i = 0; i < m; i++) {
    String[] tokens = in.readLine().split(",");
    Clause newClause = new Clause();
    for (String lit : tokens) {
     int var = Math.abs(Integer.parseInt(lit));
     boolean sign = (lit.charAt(0) != '-');
     newClause.addLiteral(var,sign);
   }
   clauses.add(newClause);
   }
}
```

f1.txt

```
2
1
0,-1
```

$x_0 \vee \neg x_1 \wedge$

f2.txt

```
2
4
0,-1
0,1
-0,1
-0,-1
```

$(x_0 \vee \neg x_1) \wedge (x_0 \vee x_1)$
$\wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$

API: Boolean formula

public class	Fmla	
	public Fmla(In in)	*Read formula from file*
Iterable<Clause>	clauses()	*Return all the clauses*
int	nClauses();	*Return number of clauses*
void	addClause(Clause)	*Adding clauses*
boolean	isSatisfied(Assignment a)	*Does the partial assignment satisfy the formula?*
boolean	isValuated(Assignment a)	*Does the partial assignment result in a valued formula?*
String	toString()	*string representation*

Library method:

```
public static Formula RandomCNF(int n, int m, int k)
```

Generate random k-CNF formula with n variables and m clauses

Formula Satisfied by an Assignment

```
// Return true if clause evaluates to true
public boolean isTrue(Asgmt asg) {
   for (int var : vars())
      if (asg.isSet(var) && (asg.getValue(var) == getSign(var)))
                 return true;
        return false;
}

// Does the given (partial) assignment satisfy the formula?
public boolean isSatisfied(Asgmt asg) {
   for (Clause clause : clauses)
      if (!clause.isTrue(asg))
        return false;
   return true;
}
```

In class
`Clause`

In class
`Formula`

Can a formula be evaluated?

```
public boolean isValuated(Asgmt asg) {
  boolean allClausesTrue = true;
  for (Clause clause : clauses)
     if (clause.isFalse(clause,asg)) return true;
     else if (! Clause.isTrue(clause,asg)) allClausesTrue=false;
  return allClausesTrue;
}
```

In class
Fmla

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$
$\wedge (x_4 \vee x_5 \vee x_6)(\neg x_4 \vee \neg x_5) \wedge (\neg x_4 \vee \neg x_6)$

Partial assignment: $x_2 = T, x_3 = T, x_5 = T,$

Satisfiability: Backtracking solver

```
private boolean bbSatisfiability(Formula F, Asgmt asg) {
   nStates++;
   if (F.isValuated(asg)) return F.isSatisfied(asg);
   int v = nextVariable(F,asg);
   asg.add(v, true);
   if (bbSatisfiability(F,asg)) return true;
   asg.setValue(v, false);
   if (bbSatisfiability(F,asg)) return true;
   asg.unset(v);
   return false;
}
```

```
public boolean isValuated(Asgmt asg) {
  boolean allClausesTrue = true;
  for (Clause clause : clauses)
     if (clause.isFalse(clause,asg)) return true;
     else if (! clause.isTrue(clause,asg)) allClausesTrue=false;
  return allClausesTrue;
}
```

In class Formula

Attributions

- Joao Marques-Silva, Southampton
  - „Practical applications of Boolean Satisfiability“, talk at WODES’08
- Ari K. Jónsson, rektor
  - Presentation, „Practical Planning I“
- Federico Pecora, Örebro University
  - Lecture in Advanced AI, DT4019
- David Dill, Stanford
  - Lecture 2: Practical SAT solving, CS 357