The SAT Problem

We are now going to take a slight detour, and focus on a particular computational problem, known as *Boolean satisfiability* or *SAT* for short.

Why study SAT?

1. A powerful technique for solving problems – translate a problem instance as an instance of SAT, and then use a *SAT-solver*
2. A way to understand hardness of problems. We will show that the SAT problem is "as hard" as any problem in the complexity class *NP* (*nondeterministic polynomial time*). (Cook-Levin Theorem)
   Then, to show another problem Π is hard, we just need to show that we could use an efficient solver for Π to build and efficient solver for SAT.

Wait a minute... if SAT is hard, what good are SAT-solvers? SAT solvers employ *heuristics* which seem to work well in practice on many classes of instances.

Boolean formulas

- A *Boolean variable* can take the values 1 (TRUE) or 0 (FALSE)
- A *Boolean formula* expression made up of Boolean variables using the *Boolean operations* AND, OR, and NOT, which are usually written $\wedge$,$\vee$ and $\neg$
- For a variable x, we often write $\overline{x}$ instead of $\neg x$
- A *truth assignment* T for a Boolean expression assigns each variable x the value 0 or 1. This is denoted $T(x)$ The value of an formula ϕ under a particular truth assignment T is the result of evaluating ϕ with each variable x replaced by its value $T(x)$, in the standard way
- E.g., Suppose $\phi = (\overline{x} \wedge y) \vee (\overline{y} \wedge z)$, and T is defined by $T(x) = 0$, $T(y) = 1$ and $T(z) = 0$.
- Then under T,

$$\begin{aligned}\phi &\equiv (\overline{0} \wedge 1) \vee (\overline{1} \wedge 0) \\ &\equiv (1 \wedge 1) \vee (0 \wedge 0) \\ &\equiv 1 \vee 0 \equiv 1\end{aligned}$$

Satisfiability of Boolean formulas

- A formula ϕ is *satisfiable* if it evaluates to 1 under some truth assignment
- A *truth table* summarizes the values given to a formula by all possible truth assignments.
- So a formula ϕ is satisfiable if one row of the table gives it the value 1
- SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean expression $\}$.

Conjunctive normal form

- A *literal* is either a variable or a negated variable. For each variable x, there are two literals, x and $\bar{x}$. So a literal is satisfied by T if it has the form x and $T(x) = 1$ or it has the form $\bar{x}$ and $T(x) = 0$.

- A *clause* is the OR of a set of one or more literals. It is *satisfied* by T if least one of its literals is satisfied by T

- A Boolean formula is in CNF form if it is the AND of clauses. It is *satisfiable* if there is some truth assignment that simultaneously satisfies all its clauses ("a satisfying truth assignment").

- CNF-SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable CNF formula $\}$

SAT Solvers

In later lectures we will see that there is good evidence against there being any *efficient* solution to the CNF-SAT problem. However, it turns out that there are *heuristic* approaches, based on back-tracking search, which seem to work *very* well in practice (also a topic for a later lecture.)

Right now, we want to consider the *application* of SAT to general problem solving. Suppose L is a language which represents a problem we would like to solve. Suppose we have a solver P for CNF-SAT, i.e., $P(\phi)$ returns 1 if $\langle\phi\rangle \in$ CNF-SAT and 0 otherwise. We do the following

1. Define an *efficient function* f with the property that for any string s, it is the case that $s \in L$ iff $f(s) \in$ CNF-SAT
2. Given an instance s of the problem L, compute $f(s)$, and give this as input to P
3. Return whatever answer is given by P

BONUS: If a formula ϕ is satisfiable, most SAT-solvers will also return the satisfying assignment – can be used to give a solution to the problem instance represented by s, if one exists.

Sudoku puzzle problem

Standard Sudoku – 9x9 grid (could be any size...)

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Easy to code a puzzle as a string, e.g.

25**3*9*1*1***4***4*7***2*8**52*********981***4***3******36**72*7******39*3***6*4

Solving Sudoku using SAT

We want a way to transform a Sudoku puzzle Π to a CNF formula ϕ.

For this to be useful

1. Π should have a solution iff ϕ is satisfiable
2. The size of ϕ should be be polynomial in the size of Π
3. The transformation should be efficient (e.g. it doesn't solve the puzzle itself...)

It would also be nice if

4. A satisfying assignment for ϕ should give us a solution for Π

The basic idea

How do we define the CNF corresponding to Π?

Most important decision: what are the *variables*?

x_{ijk} – represents that cell (i, j) contains value k

So for the example above the CNF should include the (single-variable) clauses

$x_{112}, x_{125}, x_{153}, \ldots, x_{314}, \ldots, x_{994}$

What else? There are many ways to encode constraints for a correct Sudoku solution. We now present a "minimal" encoding (why is this enough?)

"Minimal" Encoding

- Every cell contains at least one number

$$\bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigvee_{k=1}^{9} x_{ijk}$$

- Each number appears at most once in every row

$$\bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{j=1}^{8} \bigwedge_{\ell=j+1}^{9} (\neg x_{ijk} \vee \neg x_{i\ell k})$$

- Each number appears at most once in every column

$$\bigwedge_{j=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{i=1}^{8} \bigwedge_{\ell=i+1}^{9} (\neg x_{ijk} \vee \neg x_{\ell jk})$$

- Each number appears at most one in every 3x3 sub-grid

$$\bigwedge_{k=1}^{9} \bigwedge_{a=0}^{2} \bigwedge_{b=0}^{2} \bigwedge_{u=1}^{3} \bigwedge_{v=1}^{2} \bigwedge_{w=v+1}^{3} (\neg x_{(3a+u)(3b+v)k} \vee \neg x_{(3a+u)(3b+w)k})$$

$$\bigwedge_{k=1}^{9} \bigwedge_{a=0}^{2} \bigwedge_{b=0}^{2} \bigwedge_{u=1}^{2} \bigwedge_{v=1}^{3} \bigwedge_{w=u+1}^{3} \bigwedge_{t=1}^{3} (\neg x_{(3a+u)(3b+v)k} \vee \neg x_{(3a+w)(3b+t)k})$$

Making sense of this

OK, first let's introduce Boolean implication $\Rightarrow$. Remember that $\phi \Rightarrow \psi$ is equivalent to $\neg\phi \vee \psi$.

Another thing to remember is that $\phi \Rightarrow (\psi \wedge \chi)$ is equivalent to $(\phi \Rightarrow \psi) \wedge (\phi \Rightarrow \chi)$.

Let's consider the formula that says each number appears at most once in every row. This means that for every $1 \leq i \leq 9$ (the row), every $1 \leq k \leq 9$ (the number), and every $1 \leq j \leq 8$ (column with something to the right of it)

$$x_{ijk} \Rightarrow \bigwedge_{\ell=j+1}^{9} \neg x_{i\ell k}$$

I.e., if k appears in cell (i, j) then it can't appear anywhere to the right in row i. This is enough to ensure that no number appears more than once in a row (think about it...)

Making more sense of it

For a given i, j, k we have

$$x_{ijk} \Rightarrow \bigwedge_{\ell=j+1}^{9} \neg x_{i\ell k}$$

This is equivalent to

$$\bigwedge_{\ell=j+1}^{9} (x_{ijk} \Rightarrow \neg x_{i\ell k})$$

which is equivalent to

$$\bigwedge_{\ell=j+1}^{9} (\neg x_{ijk} \vee \neg x_{i\ell k})$$

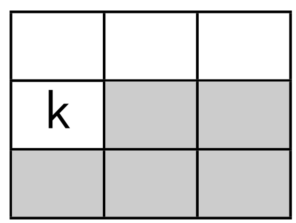
Then, to say that this is true for all i, j, k (with the appropriate bounds) we have

$$\bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{j=1}^{8} \bigwedge_{\ell=j+1}^{9} (\neg x_{ijk} \vee \neg x_{i\ell k})$$

Other constraints

The constraint for 3x3 sub-grids says that if k appears in a cell of a subgrid, then it cannot appear anywhere to the right in the same row, or in any row below.

In other words, think of the cells of a subgrid being numbered from 1 to 9, starting at the top left and ending at the bottom right. If a number appears in a cell, then it can't appear in any cell with a greater number:



DIMACS Format

Remember that a SAT solver takes a CNF formula as input, and decides whether or not the formula is satisfiable (and my also return a satisfying assignment.)

Many SAT solvers use DIMACS format to represent CNF formulas:

```
p cnf <# variables> <# clauses>

<list of clauses>
```

DIMACS Format

Each clause is given by a list of non-zero numbers terminated by a `0`. Each number represents a literal. Positve numbers `1,2,...` are unnegated variables. Negative numbers are negated variables. Comment lines preceded by a `c` are allowed. For example the CNF formula

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

would be given by the following file:

```
c A sample file
p cnf 4 3
1 3 4 0
-1 2 0
-3 -4 0
```

The Sudoku Clauses in DIMACS Format

Need to convert x_{ijk} variables to unique integers ≥ 1.

Natural way to do this: think of ijk as a base-9 number, and convert to decimal

$ijk \rightarrow 81 \times (i-1) + 9 \times (j-1) + (k-1) + 1$

Not quite converting to decimal – have to add 1 due to the restriction that variables are encoded as *strictly positive* natural numbers.

Also, note we subtract 1 from all of the indices to get them into the range $0, \ldots, 8$, which correspond to the base-9 digits

The Sudoku Clauses in DIMACS Format

So "every cell contains at least one number" is encoded by

```
c Cell 0,0 contains at least one number
1 2 3 4 5 6 7 8 9 0
c Cell 0,1 contains at least one number
10 11 12 13 14 15 16 17 18 0
...
c Cell 9,9 contains at least one number
721 722 723 724 725 726 727 728 729 0
```