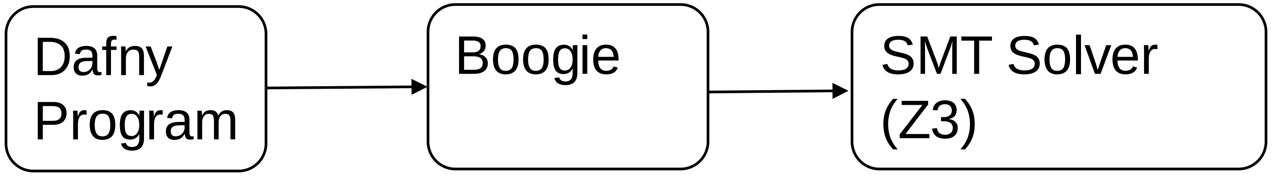
CMSC 433
Programming Language Technologies and Paradigms

SAT Solvers

Borrowed slides from Aarti Gupta, Sharad Malik, Emina Torlak

How Does Dafny work?



- Boogie is an intermediate verification language, intended as a layer on which to build program verifiers for other languages.

Boolean Satisfiability (SAT) Solvers

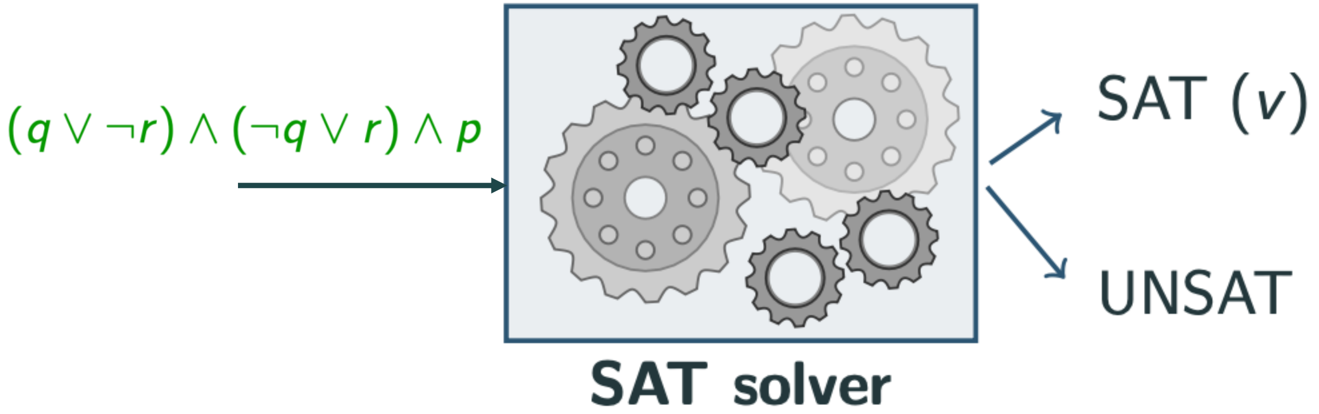
- Given a propositional logic (Boolean) formula,

$$F = (x1 \vee x2) \wedge (x3 \vee x4 \vee \neg x5)$$

- Find a variable assignment such that the formula evaluates to true, or prove that no such assignment exists.

SAT Solvers

- Engines for solving any problem reducible to propositional logic
  - Input: Propositional formula f
  - Output: SAT + valuation v such that v (f) = T if f satisfiable
    UNSAT: otherwise



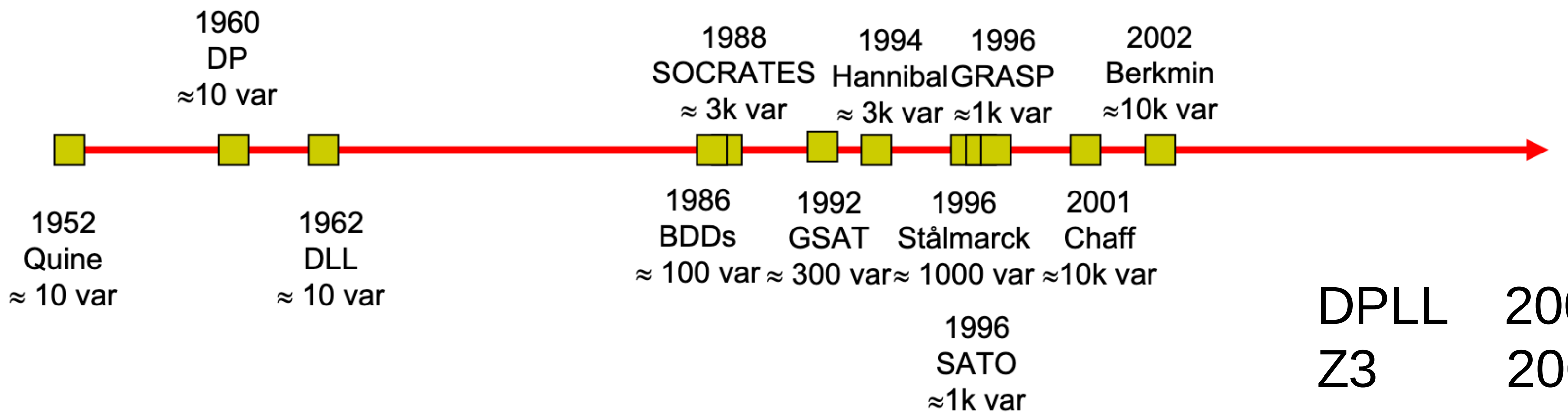
v(p)= T
v(q) = F
v(r) = F

SAT is NP-Complete

$$F = (x1 \vee x2) \wedge (x3 \vee x4 \vee \neg x5)$$

- For *n* variables, there are 2^n possible truth assignments to be checked.
- First established NP-Complete problem. (Stephen A. Cook 1971)

Sat Solvers Timeline



DPLL 2004
Z3 2008

Problem size: We went from 10 variables, 20 constraints (early 90's) to 1M+ variables and 5M+ constraints in 20 years.

Where are we today?

- Intractability of the problem no longer daunting
  - can regularly solve practical instances with ***millions*** of variables and constraints
- SAT has matured from theoretical interest to practical impact
  - Widely used in many aspects of chip design (Electronic Design Automation): equivalence checking, assertion verification, synthesis, debugging, post-silicon validation
  - Software verification
    - Commercial use at Microsoft, Amazon, Google, Facebook,...

Where are we today?

- Significant SAT community
  - SatLive Portal (http://www.satlive.org/)
  - Annual SAT competitions (http://www.satcompetition.org/)
  - SAT Conference (http://www.satisfiability.org/)

- Emboldened researchers to take on even harder problems related to SAT
  - Max-SAT: for optimization
  - Satisfiability Modulo Theories (SMT): for more expressive theories
  - Quantified Boolean Formulas (QBF): for more complex problems

Propositional Logic: Syntax

- **Atom**:
  - **truth symbols**: ⊤ (“true”), ⊥ (“false”)
  - **propositional variables**: p,q,r,…
- **Literal**
  - an atom α or its negation ¬α
- **Formula**:
  - an atom or the application of a **logical connective** to formulas F_1, F_2 :

$\neg F1$	*“not”*	(negation)
$F1 \wedge F2$	“and”	(conjunction)
$F1 \vee F2$	“or”	(disjunction)
$F1 \rightarrow F2$	“implies”	(implication)
$F1 \leftrightarrow F2$	“if and only if”	(iff)

Propositional Logic: Semantics

Given a Boolean formula F, and an *Interpretation I*, which maps variables to true/false

$$I : \{\, p \mapsto \text{true}, q \mapsto \text{false}, \ldots \}$$

- *I* is a **satisfying interpretation** of *F*, written as $I \models F$, if *F* evaluates to true under *I*.
  - A satisfying interpretation is also called a **model**.

- *I* is a **falsifying interpretation** of *F*, written as $I \not\models F$, if *F* evaluates to false under *I.*

Propositional Logic: Semantics

- Definition
  - Base case
    - $I \models \top$
    - $I \not\models \bot$
    - $I \models p$ iff $I[p]$=true
    - $I \not\models p$ iff $I[p]$=false

Propositional Logic: Semantics

- Definition
  - **Inductive cases:**

➢ $I \models \neg F$	iff $I \not\models F$
➢ $I \models F1 \wedge F2$	iff $I \models F1$ and $I \models F2$
➢ $I \models F1 \vee F2$	iff $I \models F1$ or $I \models F2$
➢ $I \models F1 \rightarrow F2$	iff $I \not\models F1$ or $I \models F2$
➢ $I \models F1 \leftrightarrow F2$	iff $I \models F1$ and $I \models F2$, or $I \not\models F1$ and $I \not\models F2$

Truth Table

A truth table shows whether a propositional formula is true or false for each possible truth assignment.

P	**Q**	**$\neg P$**	**$P \to Q$**	**$\neg P \wedge (P \to Q)$**
T	T	F	T	F
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Propositional Logic: Semantics

- Example

F: $(p \land q) \rightarrow (p \lor \neg q)$

I: $\{p \mapsto \text{true}, q \mapsto \text{false}\}$

Propositional Logic: Semantics

- Example

F: $(p \wedge q) \rightarrow (p \vee \neg q)$

I: $\{p \mapsto \text{true}, q \mapsto \text{false}\}$

$I \vDash F$, I is a **satisfying interpretation** of F

Satisfiability & Validity of Propositional Formulas

- F is **satisfiable** iff $I \models F$ for some I.

- F is **valid** iff $I \models F$ for all I.

- **Duality** of satisfiability and validity: F is valid iff $\neg F$ is unsatisfiable.
  - If we have a procedure for checking satisfiability, we can also check validity of propositional formulas, and vice versa.

Techniques for Deciding Satisfiability & Validity

- Search
  - Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

- Deduction
  - Assume the formula is invalid, apply proof rules, and check for contradiction in every branch of the proof tree.

Proof by Search: enumerating interpretations

$F : (p \wedge q) \rightarrow (p \vee \neg q)$

$I \vDash F1 \rightarrow F2$ iff $I \nvDash F1$ or $I \vDash F2$

p	**q**	$p \wedge q$	$\neg q$	$p \vee \neg q$	**F:**
F	F	F	T	T	T
F	T	F	F	F	T
T	F	F	T	T	T
T	T	T	F	T	T

Proof by Search: enumerating interpretations

$F : (p \wedge q) \rightarrow (p \vee \neg q)$

$I \vDash F1 \rightarrow F2$ iff $I \nvDash F1$ or $I \vDash F2$

p	**q**	$p \wedge q$	$\neg q$	$p \vee \neg q$	**F:**
F	F	F	T	T	T
F	T	F	F	F	T
T	F	F	T	T	T
T	T	T	F	T	T

Valid

Proof by Deduction: semantic arguments

- A **proof rule** consists of
  - *premise*: facts that must hold to apply the rule.
  - *conclusion*: facts derived from applying the rule.

- Commas indicate derivation of multiple facts; pipes indicate alternative facts (branches in the proof).

$$\frac{\text{Premise}}{\text{Conclusion}}$$

Proof by Deduction: semantic arguments

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F_1 \wedge F_2}{I \models F_1, I \models F_2} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \mid I \not\models F_2}$$

$$\frac{I \models F_1 \vee F_2}{I \models F_1 \mid I \models F_2} \qquad \frac{I \not\models F_1 \vee F_2}{I \not\models F_1, I \not\models F_2}$$

Proof by Deduction: semantic arguments

$$\frac{I \models F_1 \rightarrow F_2}{I \not\models F_1 \mid I \models F_2} \qquad \frac{I \not\models F_1 \rightarrow F_2}{I \models F_1,\ I \not\models F_2}$$

$$\frac{I \models F_1 \leftrightarrow F_2}{I \models F_1 \land F_2 \mid I \not\models F_1 \lor F_2} \qquad \frac{I \not\models F_1 \leftrightarrow F_2}{I \models F_1 \land \neg F_2 \mid I \models \neg F_1 \land F_2}$$

Proof by deduction: another example 1

- Prove $p \land \neg q$ *is valid* or find a falsifying interpretation.

1. $I \not\models p \land \neg q$ (assumed)
   a. $I \not\models p$ (1, ∧)
   b. $I \not\models \neg q$ (1, ∧)
      i. $I \models q$ (1b,¬)

The formula is invalid, and I = $\{p \mapsto \text{false}, q \mapsto \text{true}\}$ is a falsifying interpretation.

Proof by deduction: another example 2

- Prove $(p \wedge (p \rightarrow q)) \rightarrow q$ or find a falsifying interpretation.

$I \vDash F1 \rightarrow F2$ iff

$I \nvDash F1$ or $I \vDash F2$

1.	$I \nvDash (p \wedge (p \rightarrow q)) \rightarrow q$	
2.	$I \nvDash q$	$(1, \rightarrow)$
3.	$I \vDash (p \wedge (p \rightarrow q))$	$(1, \rightarrow)$
4.	$I \vDash p$	$(3, \wedge)$
5.	$I \vDash p \rightarrow q$	$(3, \wedge)$
	1. $I \nvDash p$	$(5, \rightarrow)$
	2. $I \vDash q$	$(5, \rightarrow)$

We have reached a contradiction in every branch of the proof, so the formula is valid.

Semantic Judgement

- Formulas *F1* and *F2* are **equivalent**, written $F1 \Leftrightarrow F2$, iff $F1 \leftrightarrow F2$ is valid.

- Formula *F1* **implies** *F2*, written $F1 \Rightarrow F2$, iff $F1 \rightarrow F2$ is valid.

- $F1 \Leftrightarrow F2$ and $F1 \Rightarrow F2$ are **not** propositional formulas (not part of syntax). They are properties of formulas, just like validity or satisfiability.

Normal Form

- A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.
  - Assembly language for a logic.

- Three important normal forms for propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

- Atom := Variable | ⊤ | ⊥
- Literal := Atom | ¬Atom
  Formula := Literal | Formula op Formula
- op := ∧ | ∨

- The only allowed connectives are ∧, ∨, and ¬. ¬ can appear only in literals.

- Conversion to NNF performed using **DeMorgan's Laws**:
  ¬(F ∧ G) ⇔ ¬F ∨ ¬G
  ¬(F ∨ G) ⇔ ¬F ∧ ¬G

NNF Examples

- The following formulae are all in negation normal form:

$$(A \vee B) \wedge C$$
$$(A \wedge (\neg B \vee C) \wedge \neg C) \vee D$$
$$A \vee \neg B$$
$$A \wedge \neg B$$

- The following formulae are not in negation normal form:

$$A \Rightarrow B$$
$$\neg(A \vee B)$$
$$\neg(A \wedge B)$$
$$\neg(A \vee \neg C)$$

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | ¬Atom

Formula := Clause ∨ Formula

Clause := Literal | Literal ∧ Clause

- Disjunction of conjunction of literals.
- Deciding satisfiability of a DNF formula is trivial.

To convert to DNF, convert to NNF and distribute ∧ over ∨:

$(F \wedge (G \vee H)) \Leftrightarrow (F \wedge G) \vee (F \wedge H)$

$((G \vee H) \wedge F) \Leftrightarrow (G \wedge F) \vee (H \wedge F)$

DNF Examples

- The following formulas are in DNF:

$$(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F \wedge D \wedge F)$$
$$(A \wedge B) \vee (C)$$
$$(A \wedge B)$$
$$(A)$$

- The following formulas are **not** in DNF:

$\neg(A \vee B)$, since an OR is nested within a NOT

$\neg(A \wedge B) \vee C$, since an AND is nested within a NOT

$A \vee (B \wedge (C \vee D))$, since an OR is nested within an AND

Conjunctive Normal Form (CNF)

Atom := Variable | ⊤ | ⊥

Literal := Atom | ¬Atom

Formula := Clause ∧ Formula

Clause := Literal | Literal ∨ Clause

- Conjunction of disjunction of literals.
- Deciding the satisfiability of a CNF formula is hard.
- SAT solvers use CNF as their input language.

- To convert to CNF, convert to NNF and distribute ∨ over ∧

$(F \vee (G \wedge H)) \Leftrightarrow (F \vee G) \wedge (F \vee H)$

$((G \wedge H) \vee F) \Leftrightarrow (G \vee F) \wedge (H \vee F)$

However, this can result in an exponential increase in equation size.

CNF Examples

- the following formulas are in conjunctive normal form:

$$(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F \vee D \vee F)$$
$$(A \vee B) \wedge (C)$$
$$(A \vee B)$$
$$(A)$$

- The following formulas are **not** in conjunctive normal form:

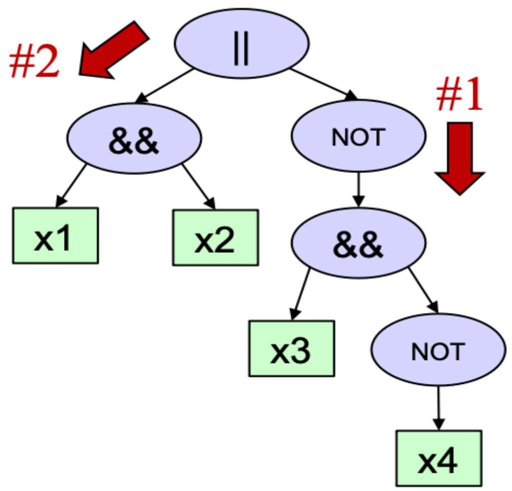
$\neg(A \wedge B)$, since an AND is nested within a NOT

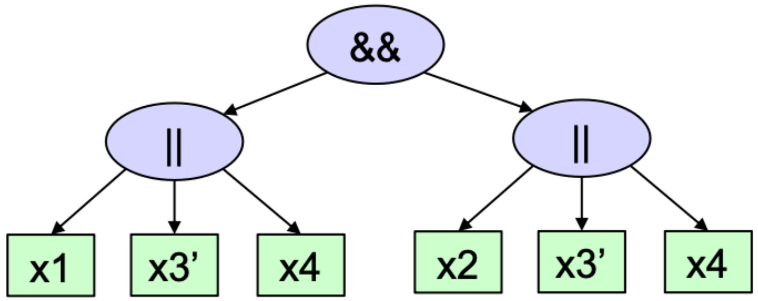
$\neg(A \vee B) \wedge C$, since an OR is nested within a NOT

$A \wedge (B \vee (D \wedge E))$, since an AND is nested within an OR

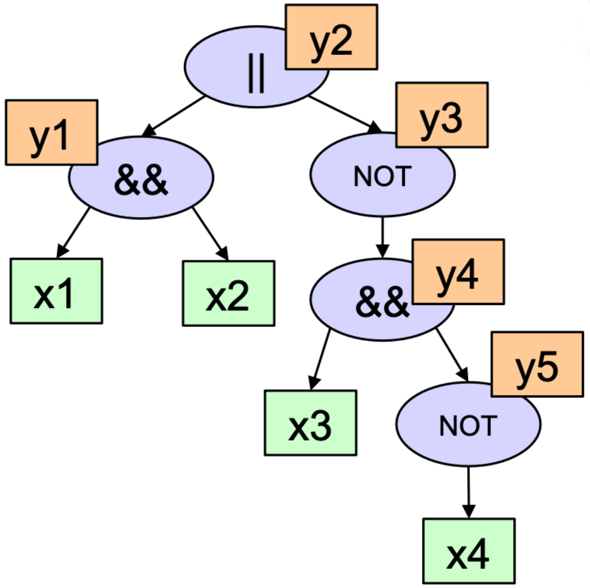
Translation to CNF: Example

(x1 ∧ x2) ∨ (¬ (x3 ∧ ¬ x4))
= (x1 ∧ x2) ∨ (¬ x3 ∨ ¬(¬ x4)) ... #de Mogans’s Law
= (x1 ∧ x2) ∨ (¬ x3 ∨ x4) ... ¬ simplification
=(x1 ∨ ¬ x3 ∨ x4)∧(x2 ∨ ¬ x3 ∨ x4) ...#Distribute (x1 ∧ x2)
= (x1 ∨ ¬ x3 ∨ x4) ∧ (x2 ∨ ¬ x3 ∨ x4)





Tseitin Transformation Example



- Main idea: Introduce fresh variable for each subformula and write ”equations”

New variables: y1, y2, y3, y4, y5

Equations

y1 = x1 ∧ x2

y2 = y1 ∨ y3

y3 = ¬ y4

y4 = x3 ∧ y5

y5 = ¬ x4

Equation	CNF to implement the Equation
z = ¬ x	(x ∨ z) ∧ (¬ x ∨ ¬ z)
z = x ∧ y	(x ∨ ¬z) ∧ (y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z)
z = x ∨ y	(¬x ∨ z) ∧ (¬y ∨ z) ∧ (x ∨ y ∨ ¬z)

CNF

(x1 ∨ ¬ y1) ∧ (x2 ∨ ¬ y1) ∧ (¬ x1 ∨ ¬ x2 ∨ y1) ∧ (¬ y1 ∨ y2) ∧ (¬ y3 ∨ y2) ∧ (y1 ∨ y3 ∨ ¬ y2) ∧ (y3 ∨ y4) ∧ (¬ y3 ∨ ¬ y4) ∧ (x3 ∨ ¬ y4) ∧ (y5 ∨ ¬ y4) ∧ (¬ x3 ∨ ¬ y5 ∨ y4) ∧ (x4 ∨ y5) ∧ (¬ x4 ∨ ¬ y5) ∧ (y2)

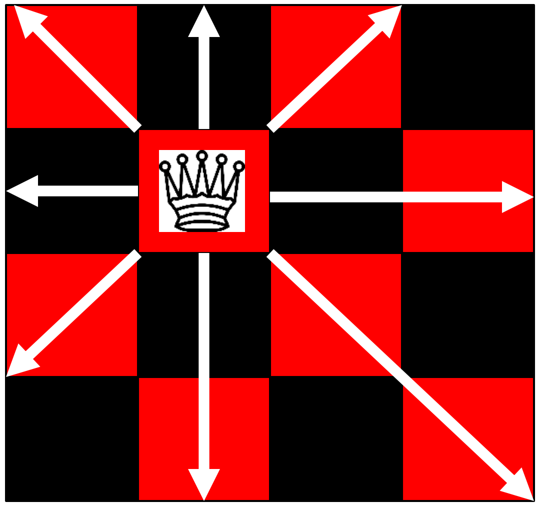
Tseitin Transformation

- Main idea: Introduce fresh variable for each subformula and write ”equations”

- Correctness of Tseitin transformation
  - For a given formula f, let Tseitin(f) denote the generated CNF formula
  - Size of Tseitin(f) is *linear* in the size of f
  - Tseitin(f) is *equi-satisfiable* with f
    - i.e., Tseitin(f) is satisfiable *if and only if* f is satisfiable

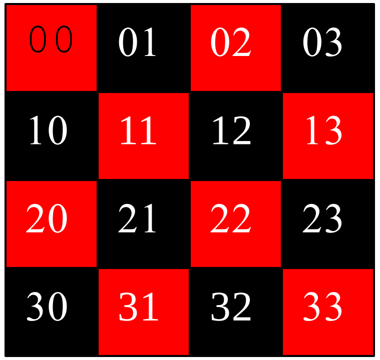
Solving real problems with SAT

- N-Queens Problem
  - Given an N x N chess board, find a placement of N queens such that no two queens can take each other



N-Queens as a SAT

- Introduce variables $x_{i\,j}$ for $0 \le i,j < N$,
  - x_{ij} = T if queen at position (i,j) F otherwise
- Constraints
  - Exactly one queen per row
    - $\text{Row}_i = x_{ij}$, $j=0\ldots N-1$
  - Exactly one queen per column
    - $\text{Column}_j = x_{ij}$, $i=0\ldots N-1$
  - At most one queen on diagonal
    - $\text{Diagonal}_{k-} = x_{ij}$, $i-j = k = -N+1\ldots,N-1$
    - $\text{Diagonal}_{k+} = x_{ij}$, $i+j = k = 0\ldots,2N-2$



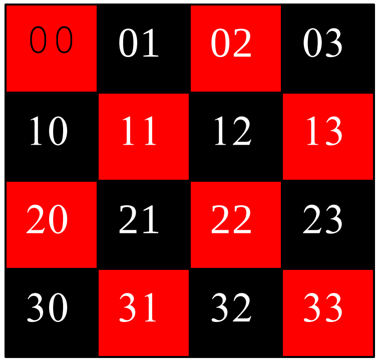
4-Queens SAT input

- Exactly one queen in row I
  - $x_{i0} \vee x_{i1} \vee x_{i2} \vee x_{i3}$
  - $x_{i0} \rightarrow \neg x_{i1} \wedge \neg x_{i2} \wedge \neg x_{i3}$
  - $x_{i1} \rightarrow \neg x_{i2} \wedge \neg x_{i3}$
  - $x_{i2} \rightarrow \neg x_{i3}$

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

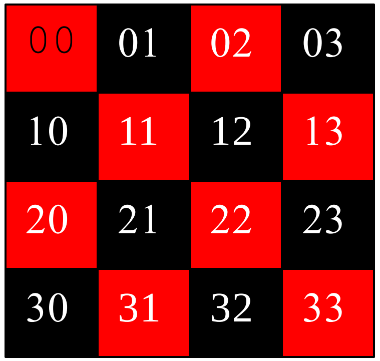
4-Queens SAT input

- Exactly one queen in column j
  - $x_{0j} \vee x_{1j} \vee x_{2j} \vee x_{3j}$
  - $x_{0j} \rightarrow \neg x_{1j} \wedge \neg x_{2j} \wedge \neg x_{3j}$
  - $x_{1j} \rightarrow \neg x_{2j} \wedge \neg x_{3j}$
  - $x_{2j} \rightarrow \neg x_{3j}$



4-Queens SAT input

- At most one queen in diagonal k-
  - $x_{20} \rightarrow \neg x_{31}$
  - …
  - $x_{00} \rightarrow \neg x_{11} \wedge \neg x_{22} \wedge \neg x_{33}$
  - $x_{11} \rightarrow \neg x_{22} \wedge \neg x_{33}$
  - $x_{22} \rightarrow \neg x_{33}$
  - …
  - $x_{02} \rightarrow \neg x_{13}$



N-queens Demo