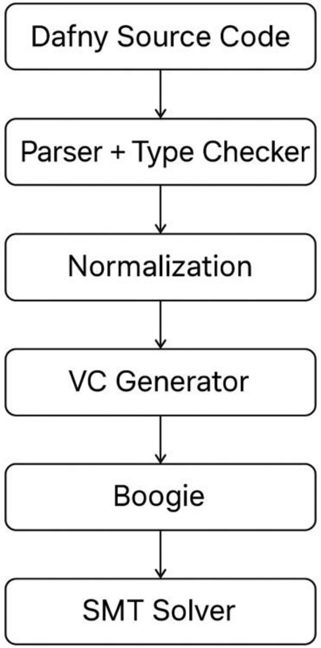
CMSC 433
Programming Language Technologies and Paradigms

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

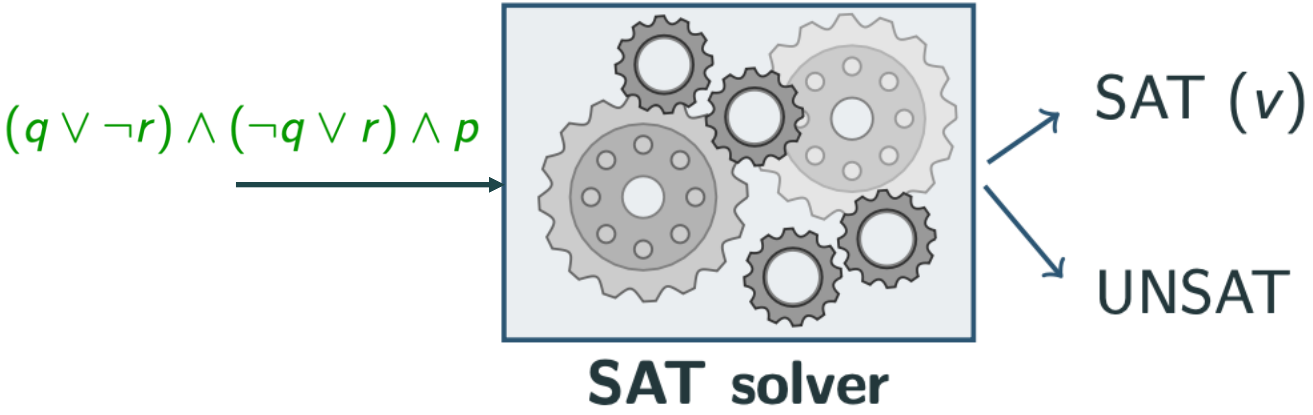
- Given a propositional logic (Boolean) formula,

$$F = (x1 \lor x2) \land (x3 \lor x4 \lor \neg x5)$$

- 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



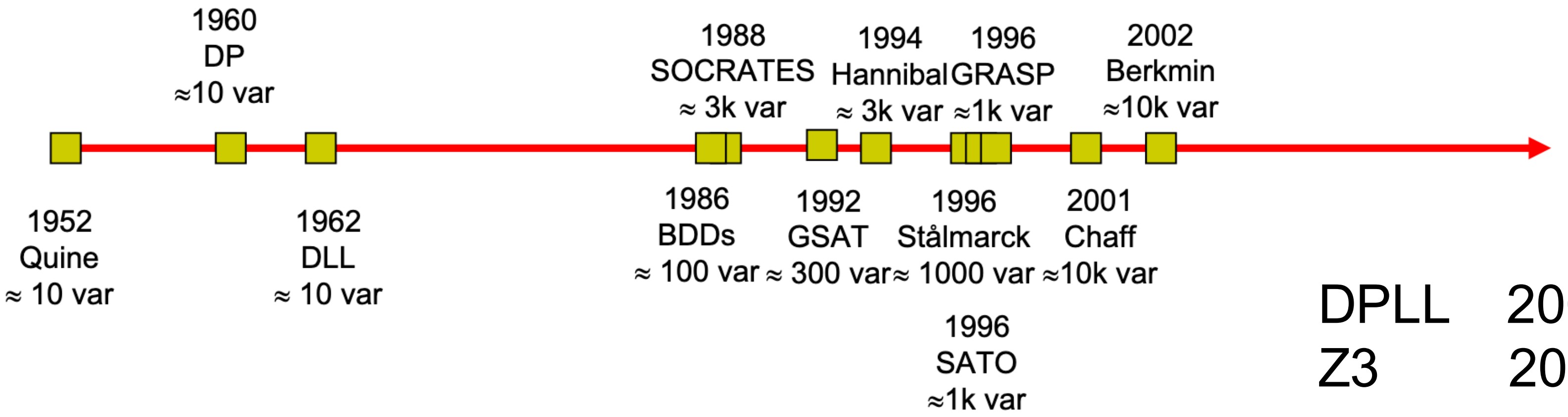
v(p) = T
v(q) = F
v(r) = F

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

- **Propositional logic** is a branch of logic that deals with **statements (propositions)** that can be **true or false** — but not both.
  - "It is raining." → can be **true** or **false**

- It focuses on how **truth values** combine and interact using **logical connectives**.
  - ¬P, P ∧ Q, P ∨ Q, P → Q, P ↔ Q

Propositional Logic: Syntax

- **Atom**:
  - **truth symbols**: $\top$ (“true”), $\bot$ (“false”)
  - **propositional variables**: $p,q,r,\ldots$
- **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**	**¬P**	**P→Q**	**¬P∧(P→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: $(p \land q) \rightarrow (p \lor \neg q)$

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

$I \models F$, 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 ¬*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

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

$I \models F1 \rightarrow F2$ iff $I \not\models F1$ or $I \models F2$

p	**q**	$p \land q$	$\neg q$	$p \lor \neg q$	**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

$$\frac{I \models F_1 \rightarrow F_2}{I \not\models F_1 \;\middle|\; I \models F_2}$$

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

$$\frac{I \models F_1 \leftrightarrow F_2}{I \models F_1 \wedge F_2 \;\middle|\; I \not\models F_1 \vee F_2}$$

$$\frac{I \not\models F_1 \leftrightarrow F_2}{I \models F_1 \wedge \neg F_2 \;\middle|\; I \models \neg F_1 \wedge F_2}$$

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↦false,q↦true} is a falsifying interpretation.

Proof by deduction: another example 2

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

$I \models F1 \to F2$ iff
$I \not\models F1$ or $I \models F2$

1. $I \not\models (p \wedge (p \to q)) \to q$
2. $I \not\models q$ (1,→)
3. $I \models (p \wedge (p \to q))$ (1,→)
4. $I \models p$ (3,∧)
5. $I \models p \to q$ (3,∧)
    1. $I \not\models p$ (5,→)
    2. $I \models q$ (5,→)

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* ⇔ *F2*, iff *F1* ↔ *F2* is valid.

- Formula *F1* **implies** *F2*, written *F1* ⟹ *F2*, iff *F1* → *F2* is valid.

- *F1* ⇔*F2* and *F1* ⟹*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**:

  $\neg(F \land G) \Leftrightarrow \neg F \lor \neg G$

  $\neg(F \lor G) \Leftrightarrow \neg F \land \neg G$

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 | ⊤ | ⊥

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:

$$(A \land \neg B \land \neg C) \lor (\neg D \land E \land F \land D \land F)$$
$$(A \land B) \lor (C)$$
$$(A \land B)$$
$$(A)$$

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

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

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

$A \lor (B \land (C \lor D))$, 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 ∧

$(F \lor (G \land H)) \Leftrightarrow (F \lor G) \land (F \lor H)$

$((G \land H) \lor F) \Leftrightarrow (G \lor F) \land (H \lor F)$

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

CNF Examples

- the following formulas are in conjunctive normal form:

$$(A \lor \neg B \lor \neg C) \land (\neg D \lor E \lor F \lor D \lor F)$$

$$(A \lor B) \land (C)$$

$$(A \lor B)$$

$$(A)$$

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

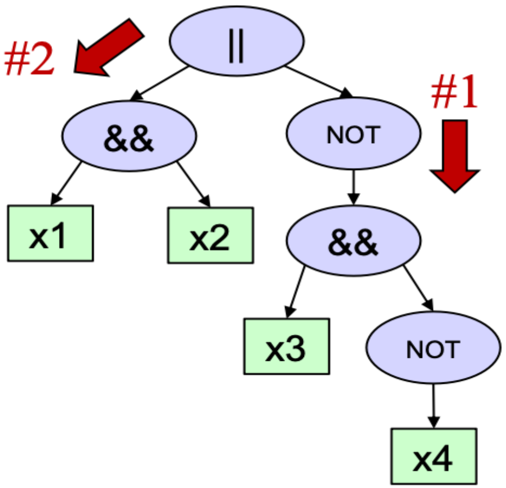
$\neg(A \land B)$, since an AND is nested within a NOT

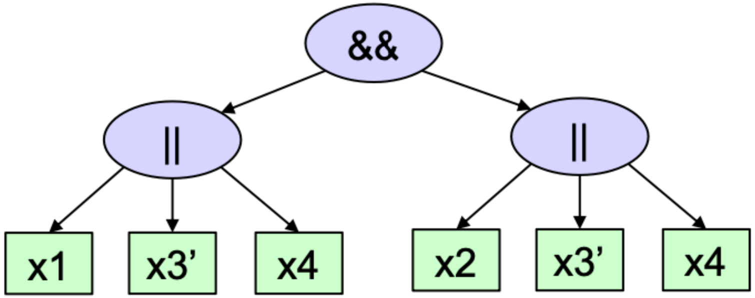
$\neg(A \lor B) \land C$, since an OR is nested within a NOT

$A \land (B \lor (D \land E))$, 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

- By introducing fresh variables, Tseitin transformation can translate every formula inro an equisatisfiable CNF formula.

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

- The CNF grows **linearly** with the size of the original formula.

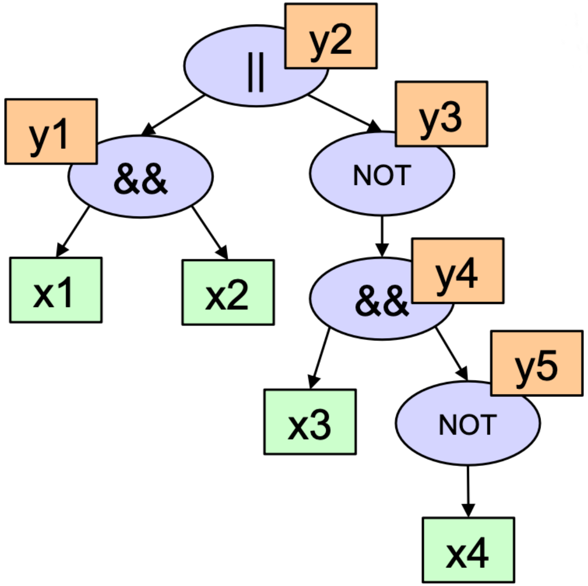
Tseitin Transformation Example

- $z = x \land y$ $(x \lor \neg z) \land (y \lor \neg z) \land (\neg x \lor \neg y \lor z)$

- $z \rightarrow (x \land y)$ Equivalently: $\neg z \lor (x \land y)$
- This gives us two clauses:
  - **$(\neg z \lor x)$**
  - **$(\neg z \lor y)$**
- $(x \land y) \rightarrow z$ Equivalently: $\neg(x \land y) \lor z$
- Using De Morgan's law: $(\neg x \lor \neg y \lor z)$

- $z = x \land y$ $(x \lor \neg z) \land (y \lor \neg z) \land (\neg x \lor \neg y \lor z)$

Tseitin Transformation Example



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

Equations

$y1 = x1 \land x2$

$y2 = y1 \lor y3$

$y3 = \neg\, y4$

$y4 = x3 \land y5$

$y5 = \neg\, x4$

CNF

$(x1 \lor \neg y1) \land (x2 \lor \neg y1) \land (\neg x1 \lor \neg x2 \lor y1) \land (\neg y1 \lor y2) \land (\neg y3 \lor y2) \land (y1 \lor y3 \lor \neg y2) \land (y3 \lor y4) \land (\neg y3 \lor \neg y4) \land (x3 \lor \neg y4) \land (y5 \lor \neg y4) \land (\neg x3 \lor \neg y5 \lor y4) \land (x4 \lor y5) \land (\neg x4 \lor \neg y5) \land (y2)$

Equation	CNF to implement the Equation
$z = \neg x$	$(x \lor z) \land (\neg x \lor \neg z)$
$z = x \land y$	$(x \lor \neg z) \land (y \lor \neg z) \land (\neg x \lor \neg y \lor z)$
$z = x \lor y$	$(\neg x \lor z) \land (\neg y \lor z) \land (x \lor y \lor \neg z)$

Tseitin Transformation

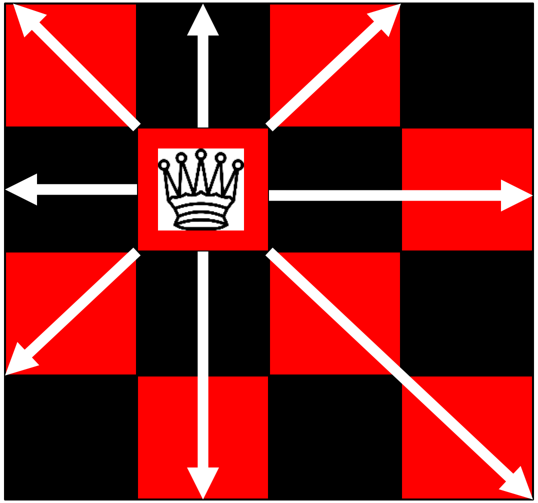
- For a given formula f, let Tseitin(f) denote the generated CNF formula

- 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
    - Row$_i$ = x_{ij}, j=0…N-1
  - Exactly one queen per column
    - Column$_j$ = x_{ij}, i=0…N-1
  - At most one queen on diagonal
    - Diagonal$_{k-}$ = x_{ij}, i-j = k = -N+1…,N-1
    - Diagonal$_{k+}$ = x_{ij}, i+j = k = 0…,2N-2

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

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

At least one queen by line:

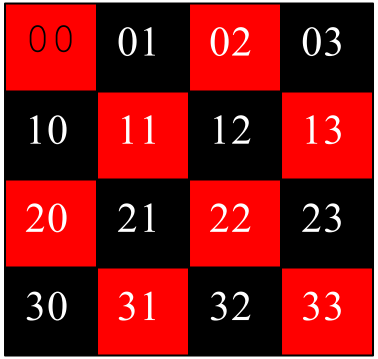
```
(assert (or x00  x01  x02 x03))
```

At most only one queen by line

```
(assert (not
   (or(and x01 x00)(and x02 x00)
      (and x02 x01)(and x03 x00)
      (and x03 x01)(and x03 x02))))
```

4-Queens SAT input

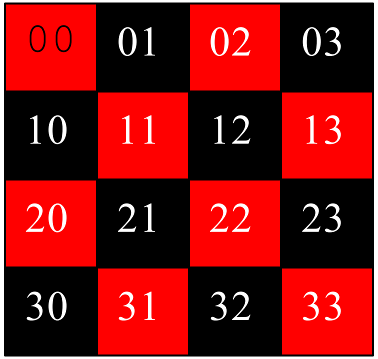
- Exactly one queen in column j
  - $\mathbf{x}_{0j} \lor \mathbf{x}_{1j} \lor \mathbf{x}_{2j} \lor \mathbf{x}_{3j}$
  - $\mathbf{x}_{0j} \rightarrow \neg\mathbf{x}_{1j} \land \neg\mathbf{x}_{2j} \land \neg\mathbf{x}_{3j}$
  - $\mathbf{x}_{1j} \rightarrow \neg\mathbf{x}_{2j} \land \neg\mathbf{x}_{3j}$
  - $\mathbf{x}_{2j} \rightarrow \neg\mathbf{x}_{3j}$



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

4-Queens SAT input

- At most one queen in diagonal k-
  - $\mathbf{x}_{20} \rightarrow \neg \mathbf{x}_{31}$
  - ...
  - $\mathbf{x}_{00} \rightarrow \neg \mathbf{x}_{11} \land \neg \mathbf{x}_{22} \land \neg \mathbf{x}_{33}$
  - $\mathbf{x}_{11} \rightarrow \neg \mathbf{x}_{22} \land \neg \mathbf{x}_{33}$
  - $\mathbf{x}_{22} \rightarrow \neg \mathbf{x}_{33}$
  - ...
  - $\mathbf{x}_{02} \rightarrow \neg \mathbf{x}_{13}$



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

N-queens Demo