CS-E4800 Artificial Intelligence

Jussi Rintanen

Department of Computer Science
Aalto University

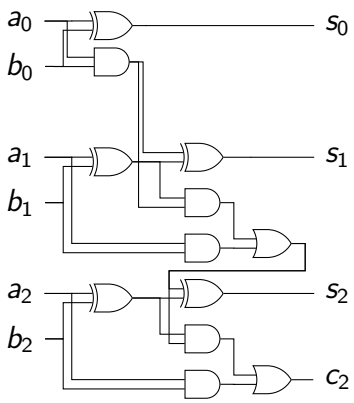
January 30, 2020

Logic

- Philosophy
  - leading theory of thinking from 350 BC until 1920s
  - ancient: India, Greece (especially axiomatic method, Aristoteles' syllogisms)
  - George Boole's "*Laws of Thought*" (1854)
- A.I.: leading theory of thinking from 1950s until 1980s
  - logic-based approaches to A.I.
  - Douglas Lenat's failed Cyc project (1984)
- Foundation of digital electronics (Boolean logic functions)
- Foundations of mathematics (only existing theory)
- Theory of mathematical proofs (only existing theory)
- Theory of correct inference and reasoning (only existing theory)
- A leading general theory of meaning in NLP (limited applicability)

Importance in Computer Technology

$$a_2 a_1 a_0 + b_2 b_1 b_0 = c_2 s_2 s_1 s_0$$



- $s_0 \leftrightarrow (a_0 \oplus b_0)$
- $c_0 \leftrightarrow (a_0 \wedge b_0)$
- $s_1 \leftrightarrow (c_0 \oplus (a_1 \oplus b_1))$
- $c_1 \leftrightarrow ((c_0 \wedge (a_1 \oplus b_1)) \vee (a_1 \wedge b_1))$
- $s_2 \leftrightarrow (c_1 \oplus (a_1 \oplus b_1))$
- $c_2 \leftrightarrow ((c_1 \wedge (a_2 \oplus b_2)) \vee (a_2 \wedge b_2))$

Importance in Computer Technology

Theorem

Any function $f : \{0,1\}^n \to \{0,1\}^m$ is expressible in terms of primitive Boolean functions (e.g. NOT&OR, NOT&AND, NAND, or NOR.)

Computations with digital electronics:

1. Bitvectors and bitmaps: arithmetics, bitmap graphics, ...
2. With clocks: sequential computations (CPUs, GPUs, ...)
3. Computation efficient: sub-nanosecond scale, highly parallel

Importance in SW Technology

Digital electronics: Given inputs $x_1, \ldots, x_n$, compute $f(x_1, \ldots, x_n) = ?$.

For high-level reasoning in AI and CS, *turn this around*:

- Given *output* $y = 1$, find $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n) = y$.
- Lots and lots of hard problems in AI and CS expressible this way
  - A.I.: This and the next lecture
  - SW, HW, protocols etc.: verification and validation
  - Many synthesis problems
  - Diagnosis, monitoring
- Best existing methods take exponential time to solve (worst case)
- This problem (SAT) is the best-known NP-complete problem

Importance in SW Technology

Digital electronics: Given inputs $x_1, \ldots, x_n$, compute $f(x_1, \ldots, x_n) = ?$.

For high-level reasoning in AI and CS, *turn this around*:

- Given *output* $y = 1$, find $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n) = y$.
- Lots and lots of hard problems in AI and CS expressible this way
  - A.I.: This and the next lecture
  - SW, HW, protocols etc.: verification and validation
  - Many synthesis problems
  - Diagnosis, monitoring
- Best existing methods take exponential time to solve (worst case)
- This problem (SAT) is the best-known NP-complete problem

Importance in SW Technology

Digital electronics: Given inputs $x_1, \ldots, x_n$, compute $f(x_1, \ldots, x_n) = ?$.

For high-level reasoning in AI and CS, *turn this around*:

- Given *output* $y = 1$, find $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n) = y$.
- Lots and lots of hard problems in AI and CS expressible this way
  - A.I.: This and the next lecture
  - SW, HW, protocols etc.: verification and validation
  - Many synthesis problems
  - Diagnosis, monitoring
- Best existing methods take exponential time to solve (worst case)
- This problem (SAT) is the best-known NP-complete problem

Importance in SW Technology

Digital electronics: Given inputs $x_1, \ldots, x_n$, compute $f(x_1, \ldots, x_n) = ?$.

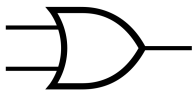
For high-level reasoning in AI and CS, *turn this around*:

- Given *output* $y = 1$, find $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n) = y$.
- Lots and lots of hard problems in AI and CS expressible this way
  - A.I.: This and the next lecture
  - SW, HW, protocols etc.: verification and validation
  - Many synthesis problems
  - Diagnosis, monitoring
- Best existing methods take exponential time to solve (worst case)
- This problem (SAT) is the best-known NP-complete problem

This Lecture

- Basics of propositional (Boolean) logic
- Decision procedures for satisfiability, logical consequence
- Constraint solving: Graph coloring, Minesweeper, Sudoku
- State-space search through SAT
- Normal forms

Disjunction / OR



set-theory: union

α	β	$\alpha \vee \beta$
0	0	0
0	1	1
1	0	1
1	1	1

α	β	$\alpha + \beta$
0	0	0
0	1	1
1	0	1
1	1	1

α	β	$\alpha \cup \beta$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{1\}$	$\{1\}$
$\{1\}$	$\emptyset$	$\{1\}$
$\{1\}$	$\{1\}$	$\{1\}$

Like integer addition for 0 and 1, except that 1+1=1!!

Conjunction / AND

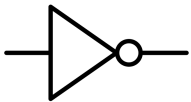
set-theory: intersection

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

α	β	$\alpha \cdot \beta$
0	0	0
0	1	0
1	0	0
1	1	1

α	β	$\alpha \cap \beta$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{1\}$	$\emptyset$
$\{1\}$	$\emptyset$	$\emptyset$
$\{1\}$	$\{1\}$	$\{1\}$

Negation / NOT



set-theory: complement

α	$\neg\alpha$
0	1
1	0

α	$\overline{\alpha}$
0	1
1	0

α	$\overline{\alpha}$
$\emptyset$	$\{1\}$
$\{1\}$	$\emptyset$

Implication

α	β	$\alpha \rightarrow \beta$
0	0	1
0	1	1
1	0	0
1	1	1

No logic gate symbol

α	β	$\overline{\alpha} \cup \beta$
$\emptyset$	$\emptyset$	$\{1\}$
$\emptyset$	$\{1\}$	$\{1\}$
$\{1\}$	$\emptyset$	$\emptyset$
$\{1\}$	$\{1\}$	$\{1\}$

Implication $A \rightarrow B$ only very roughly corresponds to "If A then B."

A and B do not have to related in anyway, e.g. causally.

$\alpha \rightarrow \beta$ is the same as $\neg\alpha \vee \beta$!

Equivalence

α	β	$(\alpha \leftrightarrow \beta)$
0	0	1
0	1	0
1	0	0
1	1	1

No logic gate symbol

α	β	$(\overline{\alpha} \cup \beta) \cap (\alpha \cup \overline{\beta})$
$\emptyset$	$\emptyset$	$\{1\}$
$\emptyset$	$\{1\}$	$\emptyset$
$\{1\}$	$\emptyset$	$\emptyset$
$\{1\}$	$\{1\}$	$\{1\}$

$\alpha \leftrightarrow \beta$ the same as $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

NAND Connective / Sheffer's Stroke

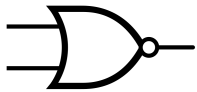
α	β	$(\alpha \mid \beta)$
0	0	1
0	1	1
1	0	1
1	1	0

α	β	$\overline{\alpha \cdot \beta}$
0	0	1
0	1	1
1	0	1
1	1	0

α	β	$\overline{\alpha \cap \beta}$
$\emptyset$	$\emptyset$	$\{1\}$
$\emptyset$	$\{1\}$	$\{1\}$
$\{1\}$	$\emptyset$	$\{1\}$
$\{1\}$	$\{1\}$	$\emptyset$

$\alpha \mid \beta$ is the same as $\neg(\alpha \wedge \beta)$!

NOR Connective / Peirce's arrow



α	β	$\alpha \downarrow \beta$
0	0	1
0	1	0
1	0	0
1	1	0

α	β	$\overline{\alpha + \beta}$
0	0	1
0	1	0
1	0	0
1	1	0

α	β	$\alpha \downarrow \beta$
$\emptyset$	$\emptyset$	$\{1\}$
$\emptyset$	$\{1\}$	$\emptyset$
$\{1\}$	$\emptyset$	$\emptyset$
$\{1\}$	$\{1\}$	$\emptyset$

$\alpha \downarrow \beta$ is the same as $\neg(\alpha \vee \beta)$!

Exclusive Or / XOR

α	β	$\alpha \underline{\vee} \beta$
0	0	0
0	1	1
1	0	1
1	1	0

α	β	$\alpha \oplus \beta$
0	0	0
0	1	1
1	0	1
1	1	0

α	β	$(\alpha \cap \overline{\beta}) \cup (\overline{\alpha} \cap \beta)$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{1\}$	$\{1\}$
$\{1\}$	$\emptyset$	$\{1\}$
$\{1\}$	$\{1\}$	$\emptyset$

Organizing a Party

ϕ_1	Alf will only come if Betty does	$\neg B \to \neg A$
ϕ_2	Betty and Cathy will argue (not both let in!)	$\neg B \vee \neg C$
ϕ_3	Cathy or Derek needed as DJ	$C \vee D$
ϕ_4	Alf is Ellie's ex: the two only with her hubby Derek	$(A \wedge E) \to D$
ϕ_5	Derek can't stand Betty	$\neg(B \wedge D)$

To figure out potential guests, find values for A, B, C, D, E so that $f(A, B, C, D, E) = 1$.
f is represented by the formula $\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5$.

Remark: We have a declarative specification of the problem, which is solved by general-purpose logical reasoning.

Organizing a Party

A	B	C	D	E	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	Φ
0	0	0	0	0	1	1	0	1	1	0
0	0	0	0	1	1	1	0	1	1	0
0	0	0	1	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1
0	1	0	0	0	1	1	0	1	1	0
0	1	0	0	1	1	1	0	1	1	0
0	1	0	1	0	1	1	1	1	0	0
0	1	0	1	1	1	1	1	1	0	0
0	1	1	0	0	1	0	1	1	1	0
0	1	1	0	1	1	0	1	1	1	0
0	1	1	1	0	1	0	1	1	0	0
0	1	1	1	1	1	0	1	1	0	0

A	B	C	D	E	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	Φ
1	0	0	0	0	0	1	0	1	1	0
1	0	0	0	1	0	1	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0
1	0	0	1	1	0	1	1	1	1	0
1	0	1	0	0	0	1	1	1	1	0
1	0	1	0	1	0	1	1	0	1	0
1	0	1	1	0	0	1	1	1	1	0
1	0	1	1	1	0	1	1	1	1	0
1	1	0	0	0	1	1	0	1	1	0
1	1	0	0	1	1	1	0	0	1	0
1	1	0	1	0	1	1	1	1	0	0
1	1	0	1	1	1	1	1	1	0	0
1	1	1	0	0	1	0	1	1	1	0
1	1	1	0	1	1	0	1	0	1	0
1	1	1	1	0	1	0	1	1	0	0
1	1	1	1	1	1	0	1	1	0	0

$$\phi_1 = \neg B \to \neg A$$
$$\phi_2 = \neg B \vee \neg C$$
$$\phi_3 = C \vee D$$
$$\phi_4 = (A \wedge E) \to D$$
$$\phi_5 = \neg(B \wedge D)$$
$$\Phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5$$

Logic – Declarative Problem Solving

- Logic = General modeling language for solving hard combinatorial problems
- Separation Problem Modeling – Solution Method
- Hard part is the Solution Method, developed "once", re-usable
- Easier part is modeling problems in the specification language
- Propositional logic (this lecture) good match with a broad range of NP-complete problems

Formulas of the Propositional Logic

Definition

Given atomic formulas (propositional variables) $X = \{x_1, \ldots, x_n\}$, define propositional formulas over X:

- Every $x \in X$ is a formula.
- If α and β are formulas, then so are $\top$, $\bot$, $(\neg\alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \to \beta)$, and $(\alpha \leftrightarrow \beta)$.

Parentheses can be left out as long as no ambiguity arises.

$(a \vee (b \vee c))$ written as $a \vee b \vee c$
$(((\neg a) \vee b) \to c)$ written as $\neg a \vee b \to c$

Connectives' precedence:

1. $\neg$
2. $\vee$, $\wedge$
3. $\to$, $\leftrightarrow$

Valuations

Instead of Boolean functions $f(x_1, \ldots, x_n)$, we most talk about formulas ϕ and the values of the atomic formulas $x_1, \ldots, x_n$ in ϕ.

Definition

Let $X = \{x_1, \ldots, x_n\}$. A valuation is a function $v : X \to \{0, 1\}$.

Valuations are generalized to cover all formulas, not just atomic ones:

- $v(\top) = 1$
- $v(\bot) = 0$
- $v(\alpha \wedge \beta) = 1$ if and only if $v(\alpha) = 1$ and $v(\beta) = 1$
- $v(\alpha \vee \beta) = 1$ if and only if $v(\alpha) = 1$ or $v(\beta) = 1$
- $v(\alpha \to \beta) = 1$ if and only if $v(\alpha) = 0$ or $v(\beta) = 1$
- $v(\alpha \leftrightarrow \beta) = 1$ if and only if $v(\alpha) = v(\beta)$

We often write $v \models \phi$ for $v(\phi) = 1$

Satisfiability of Propositional Formulas (SAT)

Definition

A formula ϕ over X is satisfiable iff there is a valuation $v : X \to \{0, 1\}$ such that $v \models \phi$.

- In complexity theory, SAT is the set of *all satisfiable formulas*.
  Testing if $\phi \in$ SAT is the most famous NP-complete problem.
- There are $2^{|X|}$ valuations.
  All algorithms for testing $\phi \in$ SAT take $O(2^n)$ time (worst case).

Validity, Logical Consequence, Logical Equivalence

Definition

ϕ is valid iff $v \models \phi$ for all valuations v.

Definition

ϕ is a logical consequence of ψ (written $\psi \models \phi$) iff $v \models \phi$ for all valuations v such that $v \models \psi$.

Definition

Formulas ϕ and ψ are logically equivalent iff $v(\phi) = v(\psi)$ for all valuations v.

Examples

- The following formulas are satisfiable:
  - $a \vee b$
  - a
- The following formulas are unsatisfiable:
  - $a \wedge \neg a$
  - $a \wedge (b \vee c) \wedge (b \rightarrow \neg a) \wedge (c \rightarrow \neg a)$
- The following formulas are valid:
  - $a \vee \neg a$
  - $a \rightarrow a$
  - $a \rightarrow (b \rightarrow a)$
  - $(a \wedge b) \rightarrow a$
- The following logical consequences hold:
  - $a \wedge b \models a$
  - $a \models a$
  - $a \wedge b \models a \vee b$

Reductions to Satisfiability

Theorem

ϕ is valid if and only if $\neg\phi$ is not satisfiable.

Theorem

$\psi \models \phi$ if and only if $\psi \wedge \neg\phi$ is not satisfiable.

Theorem

$\psi \models \phi$ if and only if $\psi \rightarrow \phi$ is valid.

Consequence: Only need to consider satisfiability; other reasoning tasks reducible to it. (Consider *implementing* them!)

Truth-Tables for Satisfiability, Validity, ...

1. Construct a table with rows for all $2^{|X|}$ valuations of X
2. If testing validity or satisfiability of ϕ, have column for ϕ
3. If testing logical consequence $\psi \models \phi$, have columns for ψ and ϕ
4. Mark the truth values of ϕ (and ψ) on every row
5. ϕ is satisfiable iff at least one 1 in the ϕ column
6. ϕ is valid iff 1 in all rows in the ϕ column
7. $\psi \models \phi$ iff there is 1 in the ϕ column in all rows where there is 1 in the ψ column

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Example

a	b	c	$a \vee (b \wedge c)$	$a \wedge b$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

- Is $a \vee (b \wedge c)$ satisfiable? Yes. Valid? No!
- $a \wedge b \models a \vee (b \wedge c)$? Yes.

Decision Procedure

Let v be any valuation (e.g. an "empty" valuation).
Now, ϕ over $X = \{x_1, \ldots, x_n\}$ is in SAT iff testSAT(ϕ,1,v)$\neq$FALSE:

procedure testSAT(ϕ,i,v)
if $i > n$ **then** *(* All of $x_1, \ldots, x_n$ have value *)*
  if $v \models \phi$ **then return** v; *(* True, hence satisfiable *)*
  return FALSE;
v_0 := like v but with $v_0(x_i) = 0$; *(* Assign x_i false *)*
r := testSAT(ϕ,$i+1$,v_0);
if $r \neq$ FALSE **then return** r;
v_1 := like v but with $v_1(x_i) = 1$; *(* Assign x_i true *)*
return testSAT(ϕ,$i+1$,v_1);

If ϕ is unsatisfiable, whole binary tree with 2^n leaves is traversed.
Runtime is exponential (in the worst case).

Cardinality Constraints ≥ 1, ≤ 1

At Least One

$\text{atLeast1}(\phi_1, \ldots, \phi_n) = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$

At Most One

$\text{atMost1}(\phi_1, \ldots, \phi_n) = \bigwedge_{1 \leq i < j \leq n} (\neg(\phi_i \wedge \phi_j))$

Exactly One

$\text{exactly1}(\phi_1, \ldots, \phi_n) = \text{atLeast1}(\phi_1, \ldots, \phi_n) \wedge \text{atMost1}(\phi_1, \ldots, \phi_n)$

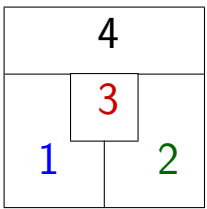
Example

$\text{exactly1}(a, b, c) \quad = \quad (a \vee b \vee c) \wedge \neg(a \wedge b) \wedge \neg(a \wedge c) \wedge \neg(b \wedge c)$

Map Coloring

Color map with (at most) three colors

Atoms $R_1, R_2, R_3, R_4, G_1, G_2, G_3, G_4, B_1, B_2, B_3, B_4$



Every region has exactly one color:
exactly1(R_i, G_i, B_i) for each $i \in \{1, \ldots, 4\}$

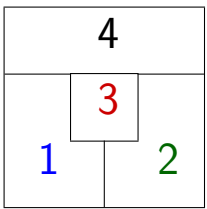
Neighboring regions i and j have different colors:
$\neg(R_i \wedge R_j)$ $\neg(G_i \wedge G_j)$ $\neg(B_i \wedge B_j)$

Unsatisfiable $\implies$ **3-coloring doesn't exist!**

Map Coloring

Color map with (at most) four colors

Atoms $R_1, R_2, R_3, R_4, G_1, G_2, G_3, G_4, B_1, B_2, B_3, B_4, W_1, W_2, W_3, W_4$

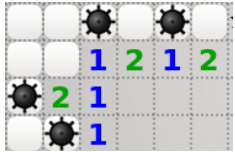


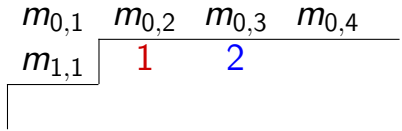
Every region has exactly one color:
exactly1(R_i, G_i, B_i, W_i) for each $i \in \{1, 2, 3, 4\}$

Neighboring regions i and j have different colors:
$\neg(R_i \wedge R_j)\ \neg(G_i \wedge G_j)\ \neg(B_i \wedge B_j)\ \neg(W_i \wedge W_j)$

Satisfiable $\Longrightarrow$ **4-coloring exists!**

Minesweeper

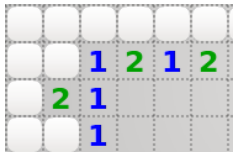


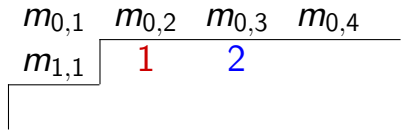


cell	at least, at most
1	$m_{1,1} \vee m_{0,1} \vee m_{0,2} \vee m_{0,3}$,
	$\neg(m_{1,1} \wedge m_{0,1}), \neg(m_{1,1} \wedge m_{0,2}), \neg(m_{1,1} \wedge m_{0,3}),$
	$\neg(m_{0,1} \wedge m_{0,2}), \neg(m_{0,1} \wedge m_{0,3}), \neg(m_{0,2} \wedge m_{0,3})$
2	$(m_{0,2} \wedge m_{0,3}) \vee (m_{0,2} \wedge m_{0,4}) \vee (m_{0,3} \wedge m_{0,4}),$
	$\neg(m_{0,2} \wedge m_{0,3} \wedge m_{0,4})$

Logical consequences: $\neg m_{1,1}$ $\neg m_{0,1}$ $m_{0,4}$ $m_{0,2} \vee m_{0,3}$

Minesweeper





cell	at least, at most
1	$m_{1,1} \vee m_{0,1} \vee m_{0,2} \vee m_{0,3}$,
	$\neg(m_{1,1} \wedge m_{0,1}), \neg(m_{1,1} \wedge m_{0,2}), \neg(m_{1,1} \wedge m_{0,3}),$
	$\neg(m_{0,1} \wedge m_{0,2}), \neg(m_{0,1} \wedge m_{0,3}), \neg(m_{0,2} \wedge m_{0,3})$
2	$(m_{0,2} \wedge m_{0,3}) \vee (m_{0,2} \wedge m_{0,4}) \vee (m_{0,3} \wedge m_{0,4}),$
	$\neg(m_{0,2} \wedge m_{0,3} \wedge m_{0,4})$

Logical consequences: $\neg m_{1,1}$ $\neg m_{0,1}$ $m_{0,4}$ $m_{0,2} \vee m_{0,3}$

(Almost) Perfect Player of Minesweeper

1. Construct formula Φ for all cells with a number, as shown on the previous slide.
2. For each unknown cell (x, y), test if $\Phi \models \neg m_{x,y}$.
3. If yes, play (x, y).
4. If no such cell exists, pick a cell with number n and m unknown neighbors so that $\frac{n}{m}$ is the smallest possible, and play it. (Question: Is this the best that can be done?)
5. Repeat from step 1.

As long as step 4 can be avoided, the player never fails.

State-Space Search by Satisfiability: Example

Incrementation of 3-bit integers abc:
$(000, 001), (001, 010), (010, 011), (011, 100), (100, 101), (101, 110), (110, 111), (111, 000)$:

$$\begin{aligned} inc \;=\; & (\neg c \wedge c' \wedge (b \leftrightarrow b') \wedge (a \leftrightarrow a')) \\ & \vee (\neg b \wedge c \wedge b' \wedge \neg c' \wedge (a \leftrightarrow a')) \\ & \vee (\neg a \wedge b \wedge c \wedge a' \wedge \neg b' \wedge \neg c') \\ & \vee (a \wedge b \wedge c \wedge \neg a' \wedge \neg b' \wedge \neg c') \end{aligned}$$

Multiply by 2 (left shift, losing the most significant bit):

$$ml2 = (a' \leftrightarrow b) \wedge (b' \leftrightarrow c) \wedge \neg c'$$

Here $X = \{a, b, c\}$ represent the *current* values of the three bits, and $X' = \{a', b', c'\}$ are the corresponding *next state* values.

State-Space Search by Satisfiability: Example

Reach 101 from in four steps, by using actions *inc* and *ml*2?

$$
\begin{aligned}
\Phi_4 \quad = \quad & \neg a@0 \wedge \neg b@0 \wedge \neg c@0 \\
& \wedge (inc[X@0/X, X@1/X'] \vee ml2[X@0/X, X@1/X']) \\
& \wedge (inc[X@1/X, X@2/X'] \vee ml2[X@1/X, X@2/X']) \\
& \wedge (inc[X@2/X, X@3/X'] \vee ml2[X@2/X, X@3/X']) \\
& \wedge (inc[X@3/X, X@4/X'] \vee ml2[X@3/X, X@4/X']) \\
& \wedge a@4 \wedge \neg b@4 \wedge c@4
\end{aligned}
$$

Valuation that satisfies Φ_4:

i	$a@i$	$b@i$	$c@i$	action
0	0	0	0	*inc*
1	0	0	1	*inc* or *ml*2
2	0	1	0	*ml*2
3	1	0	0	*inc*
4	1	0	1	

State-Space Search by Satisfiability

Procedure:

1. Generate the formulas $\Phi_0, \Phi_1, \Phi_2, \ldots$ for $0, 1, 2, \ldots$ steps.
2. Test satisfiability one by one. Stop when satisfiable formula found.

Significance:

- Alternative to algorithms like BFS, GBFS, A*
- Scalability can be very good
- Applicable to important problems in AI and CS
  - Decision-making and problem-solving
  - Computer-Aided Verification (*"Bounded Model-Checking"* Turing Award 2007)

Solving Sudoku with Propositional Logic

- Fill in 9×9 grid with numbers $1, ..., 9$ so that
  - each row contains all nine,
  - each column contains all nine, and
  - each of the marked 3×3 sub-grids contains all nine.
- Example: initial grid with "clues" (left) and the solution (right)

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

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

The solution is unique.

Step of Declarative Problem Solving

- Specialized search algorithm for Sudoku? Too much work!
- When solving problem declaratively, we
  1. encode the problem in a specification language, and
  2. use a general-purpose solver (e.g. SAT) to find a solution.

- Similar chain of steps in much of declarative problem solving

(CNF (DIMACS format) is assumed by some SAT solvers, but not all.)

Encoding Sudoku in the Propositional Logic

- Given an initial Sudoku marking, build a formula Φ such that
  - *the puzzle has a solution iff* Φ *is satisfiable*, and
  - from a satisfying valuation for Φ we can extract a solution.
- To obtain such a formula, we first introduce variables

$$S_{c,r,v}$$

  for each row $r = 1, ..., n$, column $c = 1, ..., n$ and value $v = 1, ..., n$
- Meaning: $S_{c,r,v}$ is true iff the grid element (c, r) has value v
- Next 2 slides present two variants of Φ:
  - I Basic formulation; Sufficient to characterize solutions correctly
  - II Enhanced formulation; Far faster to solve

Requirements for Solutions in Sudoku I

The formula $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ consists of:

1. Every grid cell has exactly one value:
   $C_1 = \bigwedge_{1 \le r \le n, 1 \le c \le n} \mathit{exactly1}(S_{c,r,1}, S_{c,r,2}, \ldots, S_{c,r,n})$
2. Every row has all numbers:
   $C_2 = \bigwedge_{1 \le r \le n, 1 \le v \le n} (S_{1,r,v} \vee S_{2,r,v} \vee \cdots \vee S_{n,r,v})$
3. Every column has all numbers:
   $C_3 = \bigwedge_{1 \le c \le n, 1 \le v \le n} (S_{c,1,v} \vee S_{c,2,v} \vee \cdots \vee S_{c,n,v})$
4. All sub-grids have all numbers:
   $C_4 = \bigwedge_{0 \le r' < \sqrt{n}, 0 \le c' < \sqrt{n}, 1 \le v \le n} (\bigvee_{(r,c) \in B_n(r',c')} S_{c,r,v})$
   where $B_n(r', c') = \{(r'\sqrt{n} + i, c'\sqrt{n} + j) | 1 \le i \le \sqrt{n}, 1 \le j \le \sqrt{n}\}$
5. The solution respects the given *CLUES* (the initial grid):
   $C_5 = \bigwedge_{(r,c,v) \in CLUES} (S_{c,r,v})$

Requirements for Solutions in Sudoku II

The formula $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ consists of:

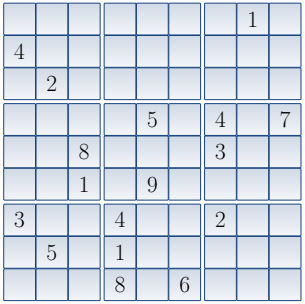
1. Every grid cell has exactly one value:
   $C_1 = \bigwedge_{1 \le r \le n, 1 \le c \le n} \mathit{exactly1}(S_{c,r,1}, S_{c,r,2}, \ldots, S_{c,r,n})$
2. Every row has all numbers exactly once:
   $C_2 = \bigwedge_{1 \le r \le n, 1 \le v \le n} \mathit{exactly1}(S_{1,r,v}, S_{2,r,v}, \ldots, S_{n,r,v})$
3. Every column has all numbers exactly once:
   $C_3 = \bigwedge_{1 \le c \le n, 1 \le v \le n} \mathit{exactly1}(S_{c,1,v}, S_{c,2,v}, \ldots, S_{c,n,v})$
4. All sub-grids have all numbers exactly once:
   $C_4 = \bigwedge_{0 \le r' < \sqrt{n}, 0 \le c' < \sqrt{n}, 1 \le v \le n} \mathit{exactly1}(\{S_{c,r,v} | (r,c) \in B_n(r',c')\})$
   where $B_n(r',c') = \{(r'\sqrt{n} + i, c'\sqrt{n} + j) | 1 \le i \le \sqrt{n}, 1 \le j \le \sqrt{n}\}$
5. The solution respects the given *CLUES* (the initial grid):
   $C_5 = \bigwedge_{(r,c,v) \in CLUES} (S_{c,r,v})$

Example Solution

The formulas Φ (I & II) have a size polynomial in n, consisting of (version I)

- $|CLUES| \leq n^2$ atomic formulas,
- $n^2 \frac{n(n-1)}{2}$ disjunctions with 2 disjuncts, and
- $3n^2$ disjunctions of length n

A Sudoku puzzle:



The formula:

$$(x_{1,1,1} \vee x_{1,1,2} \vee \ldots \vee x_{1,1,9}) \wedge \ldots$$
$$\ldots \wedge (x_{9,9,1} \vee x_{9,9,2} \vee \ldots \vee x_{9,9,9}) \wedge$$
$$(\neg x_{1,1,1} \vee \neg x_{1,1,2}) \wedge \ldots \wedge (\neg x_{9,9,8} \vee \neg x_{9,9,9}) \wedge$$
$$(x_{1,1,1} \vee x_{1,2,1} \vee \ldots \vee x_{1,9,1}) \wedge \ldots$$
$$\ldots \wedge (x_{9,1,9} \vee x_{9,2,9} \vee \ldots \vee x_{9,9,9}) \wedge$$
$$(x_{1,1,1} \vee x_{2,1,1} \vee \ldots \vee x_{9,1,1}) \wedge \ldots$$
$$\ldots \wedge (x_{1,9,9} \vee x_{2,9,9} \vee \ldots \vee x_{9,9,9}) \wedge$$
$$(x_{1,1,1} \vee x_{1,2,1} \vee x_{1,3,1} \vee x_{2,1,1} \vee \ldots \vee x_{3,3,1}) \wedge \ldots$$
$$\ldots \wedge (x_{7,7,9} \vee x_{7,8,9} \vee x_{7,9,9} \vee x_{8,7,9} \vee \ldots \vee x_{9,9,9}) \wedge$$
$$(x_{8,9,1}) \wedge (x_{1,8,4}) \wedge (x_{2,7,2}) \wedge \ldots \wedge (x_{6,1,6})$$

Valid Logical Equivalences

double negation	$\neg\neg\alpha$	$\equiv$	α
associativity $\vee$	$\alpha \vee (\beta \vee \gamma)$	$\equiv$	$(\alpha \vee \beta) \vee \gamma$
associativity $\wedge$	$\alpha \wedge (\beta \wedge \gamma)$	$\equiv$	$(\alpha \wedge \beta) \wedge \gamma$
commutativity $\vee$	$\alpha \vee \beta$	$\equiv$	$\beta \vee \alpha$
commutativity $\wedge$	$\alpha \wedge \beta$	$\equiv$	$\beta \wedge \alpha$
distributivity $\wedge$ $\vee$	$\alpha \wedge (\beta \vee \gamma)$	$\equiv$	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
distributivity $\vee$ $\wedge$	$\alpha \vee (\beta \wedge \gamma)$	$\equiv$	$(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
idempotence $\vee$	$\alpha \vee \alpha$	$\equiv$	α
idempotence $\wedge$	$\alpha \wedge \alpha$	$\equiv$	α
absorption 1	$\alpha \wedge (\alpha \vee \beta)$	$\equiv$	α
absorption 2	$\alpha \vee (\alpha \wedge \beta)$	$\equiv$	α
De Morgan's law 1	$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha) \wedge (\neg\beta)$
De Morgan's law 2	$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha) \vee (\neg\beta)$
contraposition	$\alpha \to \beta$	$\equiv$	$\neg\beta \to \neg\alpha$

negation $\top$	$\neg\top$	$\equiv$	$\bot$
negation $\bot$	$\neg\bot$	$\equiv$	$\top$
constant $\bot$	$\alpha \wedge \neg\alpha$	$\equiv$	$\bot$
constant $\top$	$\alpha \vee \neg\alpha$	$\equiv$	$\top$
elimination $\top$ $\vee$	$\top \vee \alpha$	$\equiv$	$\top$
elimination $\top$ $\wedge$	$\top \wedge \alpha$	$\equiv$	α
elimination $\bot$ $\vee$	$\bot \vee \alpha$	$\equiv$	α
elimination $\bot$ $\wedge$	$\bot \wedge \alpha$	$\equiv$	$\bot$
elimination $\bot$ $\to$	$\bot \to \alpha$	$\equiv$	$\top$
elimination $\bot$ $\to$	$\alpha \to \bot$	$\equiv$	$\neg\alpha$
elimination $\top$ $\to$	$\top \to \alpha$	$\equiv$	α
elimination $\top$ $\to$	$\alpha \to \top$	$\equiv$	$\top$
commutativity $\leftrightarrow$	$\alpha \leftrightarrow \beta$	$\equiv$	$\beta \leftrightarrow \alpha$
elimination $\top$ $\leftrightarrow$	$\top \leftrightarrow \alpha$	$\equiv$	α
elimination $\bot$ $\leftrightarrow$	$\bot \leftrightarrow \alpha$	$\equiv$	$\neg\alpha$

ϕ_1 and ϕ_2 are logically equivalent if ϕ_2 is obtained from ϕ_1 by replacing a subformula by an equivalent one.

Normal Forms

- Formulas in a normal form are easier to handle.
- The best-known normal forms are
  - Conjunctive Normal Form (CNF, product-of-sums), and
  - Disjunctive Normal Form (DNF, sum-of-products).
- Leading methods for solving SAT assume CNF
- Systematic methods for turning any formula to DNF and CNF

Normal Forms

- For atomic formulas a, the formulas a and $\neg a$ literals.
- The complement $\overline{l}$ of a literal l is defined by $\overline{a} = \neg a$ and $\overline{\neg a} = a$.

Definition

A formula α is in Conjunctive Normal Form (CNF) iff α is a conjunction $\beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n$ where every conjunct β_i is a disjunction $l_1 \vee l_2 \vee \cdots \vee l_{m_i}$ where $l_1, l_2, \ldots, l_{m_i}$ are literals.

Definition

A formula α is in Disjunctive Normal Form (DNF) iff α is a disjunction $\beta_1 \vee \beta_2 \vee \cdots \vee \beta_n$ where every disjunct β_i is a conjunction $l_1 \wedge l_2 \wedge \cdots \wedge l_{m_i}$ where $l_1, l_2, \ldots, l_{m_i}$ are literals.

Clauses and Clause Sets

Definition

For a CNF formula $\alpha = \beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n$ the formulas $\beta_i = l_{i,1} \vee \cdots \vee l_{i,n_i}$ are clauses.

- Since CNF's structure is fixed, connective can be left implicit.
  - CNF formula $\beta_1 \wedge \cdots \wedge \beta_n$ as a set $\{\beta_1, \ldots, \beta_n\}$
  - with each $\beta_i = l_{i,1} \vee \cdots \vee l_{i,n_i}$ also represented as a set $\{l_{i,1}, \ldots, l_{i,n_i}\}$
- Example: $a \wedge (b \vee c) \wedge (\neg d \vee \neg e)$ as $\{\{a\}, \{b, c\}, \{\neg d, \neg e\}\}$

Normal Forms

Example

CNF: $(a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge (a_5 \vee a_6 \vee a_7)$
CNF & DNF: $a_1 \vee \neg a_2 \vee a_3$
DNF: $(\neg a_1 \wedge \neg a_2) \vee (a_3 \wedge a_1) \vee (a_1 \wedge \neg a_2 \wedge a_3)$

Definition

β is CNF (DNF) of α iff β is in CNF (DNF) and $\beta \equiv \alpha$.

Example

$a \vee (\neg b \wedge c)$ is logically equivalent to $(a \vee \neg b) \wedge (a \vee c)$ in CNF.

Every formula can be transformed to both CNF and DNF.

Normal Form Transformations

Procedure to obtain DNF and CNF:

1. Eliminate equivalences $\leftrightarrow$
2. Eliminate implications $\rightarrow$
3. Move negations inwards so that they only occur in front of atomic formulas. (reaching Negation Normal Form (NNF).)
4. Move conjunctions outside disjunctions (to reach CNF), or move disjunctions outside conjunctions (to reach DNF).

Every step preserves logical equivalence.

Transformation Rules for CNF and DNF

$$
\begin{array}{rcll}
\alpha \leftrightarrow \beta & \rightsquigarrow & (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) & (1) \\
\alpha \rightarrow \beta & \rightsquigarrow & \neg\alpha \vee \beta & (2) \\
\neg(\alpha \vee \beta) & \rightsquigarrow & \neg\alpha \wedge \neg\beta & (3) \\
\neg(\alpha \wedge \beta) & \rightsquigarrow & \neg\alpha \vee \neg\beta & (4) \\
\neg\neg\alpha & \rightsquigarrow & \alpha & (5) \\
\alpha \vee (\beta \wedge \gamma) & \rightsquigarrow & (\alpha \vee \beta) \wedge (\alpha \vee \gamma) & (6) \\
(\alpha \wedge \beta) \vee \gamma & \rightsquigarrow & (\alpha \vee \gamma) \wedge (\beta \vee \gamma) & (7) \\
\alpha \wedge (\beta \vee \gamma) & \rightsquigarrow & (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) & (8) \\
(\alpha \vee \beta) \wedge \gamma & \rightsquigarrow & (\alpha \wedge \gamma) \vee (\beta \wedge \gamma) & (9)
\end{array}
$$

All of these rules are instances of the equivalences given earlier!

Observations

- Rules 2-5 do not increase the number of occurrences of atoms.
- Rules 6-9 double the number of occurrences of α or γ. Hence transformation to CNF and DNF may increase size exponentially.

Many SAT solvers require input in CNF:

- In many applications formulas close to CNF (no size explosion!)
- Tseitin transformation of a non-CNF formula Φ:
  - Produces formula Φ' in CNF, size linear in $|\Phi|$
  - Introduces new variables that don't occur in Φ
  - Φ' is satisfiable if and only if Φ is satisfiable

Example

Let's find the normal forms for $a \vee b \to (b \leftrightarrow c)$.

$$\begin{aligned}
&\rightsquigarrow \quad a \vee b \to (b \to c) \wedge (c \to b) && (1)\\
&\rightsquigarrow \quad \neg(a \vee b) \vee ((\neg b \vee c) \wedge (\neg c \vee b)) && (2)\\
&\rightsquigarrow \quad (\neg a \wedge \neg b) \vee ((\neg b \vee c) \wedge (\neg c \vee b)) && (3)
\end{aligned}$$

NNF reached!

Continue from NNF to CNF:

$$\begin{aligned}
&\rightsquigarrow \quad (\neg a \vee ((\neg b \vee c) \wedge (\neg c \vee b))) \wedge (\neg b \vee ((\neg b \vee c) \wedge (\neg c \vee b))) && (7)\\
&\rightsquigarrow \quad (\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg c \vee b) \wedge (\neg b \vee ((\neg b \vee c) \wedge (\neg c \vee b))) && (6)\\
&\rightsquigarrow \quad (\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg c \vee b) \wedge (\neg b \vee \neg b \vee c) \wedge (\neg b \vee \neg c \vee b). && (6)
\end{aligned}$$

Continue from NNF to DNF:

$$\begin{aligned}
&\rightsquigarrow \quad (\neg a \wedge \neg b) \vee ((\neg b \vee c) \wedge \neg c) \vee ((\neg b \vee c) \wedge b) && (8)\\
&\rightsquigarrow \quad (\neg a \wedge \neg b) \vee (\neg b \wedge \neg c) \vee (c \wedge \neg c) \vee ((\neg b \vee c) \wedge b) && (9)\\
&\rightsquigarrow \quad (\neg a \wedge \neg b) \vee (\neg b \wedge \neg c) \vee (c \wedge \neg c) \vee (\neg b \wedge b) \vee (c \wedge b). && (9)
\end{aligned}$$

Summary

- Logics basics
- Fundamental reasoning tasks: satisfiability, logical consequence
- Solution of hard (NP-hard) constraint satisfaction problems

Next week:

- Logic and Natural language semantics
- Logic in Knowledge Representation, inference, reasoning