# Informatics 1 – Introduction to Computation
# Computation and Logic

Julian Bradfield
based on materials by
Michael P. Fourman

## Satisfying Assignments
## Boolean Algebra, Tseytin, Counting

Henry Scheffer, 1882–1964

Gregory Tseytin, 1936–2022

# Boolean operators – recap



The basic operators are $\neg$, $\wedge$, and $\vee$.

# Boolean operators – recap



The basic operators are $\neg$, $\wedge$, and $\vee$.

By now you have several times seen $\to$ and $\leftrightarrow$, and perhaps $\oplus$.

# Boolean operators – recap



The basic operators are $\neg$, $\wedge$, and $\vee$.

By now you have several times seen $\rightarrow$ and $\leftrightarrow$, and perhaps $\oplus$.

How many binary boolean operators are there?

# Boolean operators – recap



The basic operators are $\neg$, $\wedge$, and $\vee$.

By now you have several times seen $\rightarrow$ and $\leftrightarrow$, and perhaps $\oplus$.

How many binary boolean operators are there?

Hardware folks like NAND and NOR, which we'll write $\overline{\wedge}$ and $\overline{\vee}$.

# Boolean operators – recap



The basic operators are $\neg$, $\wedge$, and $\vee$.

By now you have several times seen $\rightarrow$ and $\leftrightarrow$, and perhaps $\oplus$.

How many binary boolean operators are there?

Hardware folks like NAND and NOR, which we'll write $\overline{\wedge}$ and $\overline{\vee}$.

| name | sym | t.t. | a.k.a. |
|---|---|---|---|
| true | $\top$ | 1 | 1, top |
| false | $\bot$ | 0 | 0, bottom |
| not | $\neg$ | 1 0 | complement, $\bar{\ }$ |
| and | $\wedge$ | 0 0<br>0 1 | &, ., $\times$ |
| or | $\vee$ | 0 1<br>1 1 | $\|$, $+$ |
| implies | $\rightarrow$ | 1 1<br>0 1 | $\leq$ |

| name | sym | t.t. | a.k.a. |
|---|---|---|---|
| implied by | $\leftarrow$ | 1 0<br>1 1 | $\geq$ |
| iff | $\leftrightarrow$ | 1 0<br>0 1 | $=$ |
| xor | $\oplus$ | 0 1<br>1 0 | $\neq$, parity |
| nand | $\overline{\wedge}$ | 1 1<br>1 0 | |
| nor | $\overline{\vee}$ | 1 0<br>0 0 | |

# Using boolean operators



Everything we've done with boolean operators can be extended to use $\to$, $\leftrightarrow$ and others.

In the optional question of tutorial 5, you were asked for sequent calculus rules for $\to$. They are:

$$\frac{\Gamma, b \vDash \Delta \quad \Gamma \vDash a, \Delta}{\Gamma, a \to b \vDash \Delta} \,(\to L) \qquad \frac{\Gamma, a \vDash b, \Delta}{\Gamma \vDash a \to b, \Delta} \,(\to R)$$

# Using boolean operators



Everything we've done with boolean operators can be extended to use $\rightarrow$, $\leftrightarrow$ and others.

In the optional question of tutorial 5, you were asked for sequent calculus rules for $\rightarrow$. They are:

$$\frac{\Gamma, b \vDash \Delta \quad \Gamma \vDash a, \Delta}{\Gamma, a \rightarrow b \vDash \Delta}\,(\rightarrow L) \qquad \frac{\Gamma, a \vDash b, \Delta}{\Gamma \vDash a \rightarrow b, \Delta}\,(\rightarrow R)$$

Rules for $\leftrightarrow$ are even more obvious:

$$\frac{\Gamma, a \rightarrow b, b \rightarrow a \vDash \Delta}{\Gamma, a \leftrightarrow b \vDash \Delta}\,(\leftrightarrow L) \qquad \frac{\Gamma \vDash a \rightarrow b, \Delta \quad \Gamma \vDash b \rightarrow a, \Delta}{\Gamma \vDash b \leftrightarrow a, \Delta}\,(\leftrightarrow R)$$

# Using boolean operators



Everything we've done with boolean operators can be extended to use $\rightarrow$, $\leftrightarrow$ and others.

In the optional question of tutorial 5, you were asked for sequent calculus rules for $\rightarrow$. They are:

$$\frac{\Gamma, b \vDash \Delta \quad \Gamma \vDash a, \Delta}{\Gamma, a \rightarrow b \vDash \Delta}\,(\rightarrow L) \qquad \frac{\Gamma, a \vDash b, \Delta}{\Gamma \vDash a \rightarrow b, \Delta}\,(\rightarrow R)$$

Rules for $\leftrightarrow$ are even more obvious:

$$\frac{\Gamma, a \rightarrow b, b \rightarrow a \vDash \Delta}{\Gamma, a \leftrightarrow b \vDash \Delta}\,(\leftrightarrow L) \qquad \frac{\Gamma \vDash a \rightarrow b, \Delta \quad \Gamma \vDash b \rightarrow a, \Delta}{\Gamma \vDash b \leftrightarrow a, \Delta}\,(\leftrightarrow R)$$

Boring exercise: take all the stuff you've done in Haskell on WFFs etc., and extend it for these operators, if you haven't already.

Note that $(\rightarrow R)$ has the special case

$$\frac{a \vDash b}{\vDash a \rightarrow b}$$

which ties down the precise similarity between $\vDash$ and $\rightarrow$.

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*
- Identity: $a \vee 0 = a$ and $a \wedge 1 = a$

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*
- Identity: $a \vee 0 = a$ and $a \wedge 1 = a$
- Distributivity: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and *vice versa*

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*
- Identity: $a \vee 0 = a$ and $a \wedge 1 = a$
- Distributivity: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and *vice versa*
- Complements: $a \vee \neg a = 1$ and $a \wedge \neg a = 0$

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*
- Identity: $a \vee 0 = a$ and $a \wedge 1 = a$
- Distributivity: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and *vice versa*
- Complements: $a \vee \neg a = 1$ and $a \wedge \neg a = 0$

This set of axioms is far from minimal. Astonishingly, this single axiom suffices: $\neg(\neg(\neg(a \vee b) \vee c) \vee \neg(a \vee \neg(\neg c \vee \neg(c \vee d)))) = c$ https://doi.org/10.1023/A:1020542009983

# Boolean algebra



Algebra is about equations between things. These equations characterize boolean logic and operators:

- Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$ and sim. for $\wedge$
- Commutativity: $a \vee b = b \vee a$ and sim. for $\wedge$
- Absorption: $a \vee (a \wedge b) = a$ and *vice versa*
- Identity: $a \vee 0 = a$ and $a \wedge 1 = a$
- Distributivity: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and *vice versa*
- Complements: $a \vee \neg a = 1$ and $a \wedge \neg a = 0$

Other convenient derived equations include:

- Negation cancellation: $\neg\neg a = a$
- Zero/One: $\neg 1 = 0$ and $\neg 0 = 1$
- Simple absorption: $a \vee a = a$ and sim. for $\wedge$
- De Morgan: $\neg(a \vee b) = \neg a \wedge \neg b$ and *vice versa*

This set of axioms is far from minimal. Astonishingly, this single axiom suffices: $\neg(\neg(\neg(a \vee b) \vee c) \vee \neg(a \vee \neg(\neg c \vee \neg(c \vee d)))) = c$

https://doi.org/10.1023/A:1020542009983

# Deriving De Morgan



An example of proofs using boolean algebra: prove $\neg(a \vee b) = \neg a \wedge \neg b$ from the six axioms.

# Deriving De Morgan



An example of proofs using boolean algebra: prove $\neg(a \vee b) = \neg a \wedge \neg b$ from the six axioms.

**Lemma** If $x \vee y = 1$ and $x \wedge y = 0$, then $x = \neg y$ (and $y = \neg x$).

**Proof** $\neg y = 1 \wedge \neg y = (x \vee y) \wedge \neg y = (x \wedge \neg y) \vee (y \wedge \neg y) = (x \wedge \neg y) \vee 0 = (x \wedge \neg y) \vee (x \wedge y) = x \wedge (\neg y \vee y) = x \vee 1 = x$.

# Deriving De Morgan



An example of proofs using boolean algebra: prove $\neg(a \vee b) = \neg a \wedge \neg b$ from the six axioms.

**Lemma** If $x \vee y = 1$ and $x \wedge y = 0$, then $x = \neg y$ (and $y = \neg x$).

**Proof** $\neg y = 1 \wedge \neg y = (x \vee y) \wedge \neg y = (x \wedge \neg y) \vee (y \wedge \neg y) = (x \wedge \neg y) \vee 0 = (x \wedge \neg y) \vee (x \wedge y) = x \wedge (\neg y \vee y) = x \vee 1 = x$.

By the lemma, to prove $\neg(a \vee b) = \neg a \wedge \neg b$, it suffices to prove $(a \vee b) \vee (\neg a \wedge \neg b) = 1$ and $(a \vee b) \wedge (\neg a \wedge \neg b) = 0$. Both these follow easily by distributivity, complement, associativy and commutativity: e.g. the first is $(a \vee b) \vee (\neg a \wedge \neg b) = ((a \vee b) \vee \neg a) \wedge ((a \vee b) \vee \neg b) = \cdots = 1 \wedge 1 = 1$.

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
- Implication: $a \rightarrow b = \neg a \vee b$

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
- Implication: $a \rightarrow b = \neg a \vee b$

We can use equations to convert a formula to CNF – but how do we know what to do?

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \to b) \wedge (b \to a)$
- Implication: $a \to b = \neg a \vee b$

We can use equations to convert a formula to CNF – but how do we know what to do? This strategy works:

1. Get rid of $\to, \leftrightarrow$ using Implication and Bi-implication.

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
- Implication: $a \rightarrow b = \neg a \vee b$

We can use equations to convert a formula to CNF – but how do we know what to do? This strategy works:

1. Get rid of $\rightarrow$, $\leftrightarrow$ using Implication and Bi-implication.
2. Using the laws about $\neg$, push negations inwards to get negation normal form.

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
- Implication: $a \rightarrow b = \neg a \vee b$

We can use equations to convert a formula to CNF – but how do we know what to do? This strategy works:

1. Get rid of $\rightarrow$, $\leftrightarrow$ using Implication and Bi-implication.
2. Using the laws about $\neg$, push negations inwards to get negation normal form.
3. Use distributivity to push $\vee$ inside $\wedge$.

# Boolean algebra: implies and iff



We can add derived rules, as we did in sequent calculus:

- Bi-implication: $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
- Implication: $a \rightarrow b = \neg a \vee b$

We can use equations to convert a formula to CNF – but how do we know what to do? This strategy works:

1. Get rid of $\rightarrow$, $\leftrightarrow$ using Implication and Bi-implication.
2. Using the laws about $\neg$, push negations inwards to get negation normal form.
3. Use distributivity to push $\vee$ inside $\wedge$.

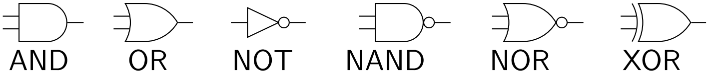
Progamming this was in FP tutorial 6! If you didn't try the optional and challenge parts, go back and try them now.

Doing this by hand tends to be boring: see textbook chapter 22 for worked examples.

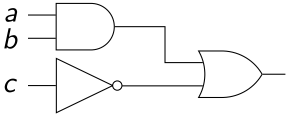
# (Not a) Short Digression: Circuits



Ultimately, logic is implemented in silicon via transistors, referred to as logic gates. Circuit designers draw gates like this:




Gates (boolean operators) are connected by drawing wires:

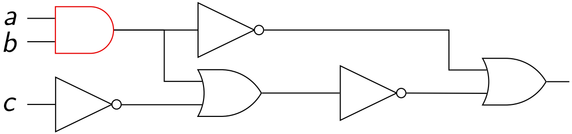



is the circuit for $(a \wedge b) \vee \neg c$.

# Circuits can duplicate expressions



A circuit can use the same output more than once:



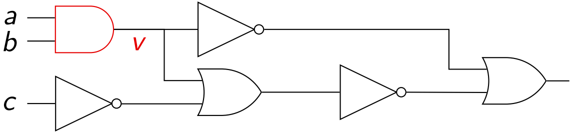

is $\phi = \neg(a \wedge b) \vee \neg((a \wedge b) \vee \neg c))$

How can we simulate re-use using only logic?

# Circuits can duplicate expressions



A circuit can use the same output more than once:




is $\phi = \neg(a \wedge b) \vee \neg((a \wedge b) \vee \neg c))$

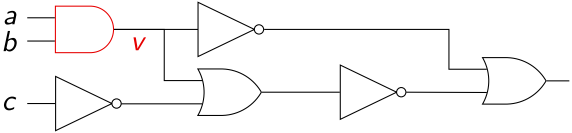
How can we simulate re-use using only logic?

$\psi = (\neg v \vee \neg(v \vee \neg c)) \wedge (v \leftrightarrow a \wedge b)$ (or think: $\neg v \vee \neg(v \vee \neg c)$ **where** $v = a \wedge b$)

# Circuits can duplicate expressions



A circuit can use the same output more than once:




is $\phi = \neg(a \wedge b) \vee \neg((a \wedge b) \vee \neg c))$

How can we simulate re-use using only logic?

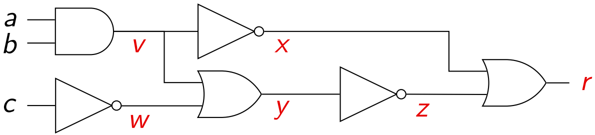
$\psi = (\neg v \vee \neg(v \vee \neg c)) \wedge (v \leftrightarrow a \wedge b)$ (or think: $\neg v \vee \neg(v \vee \neg c)$ **where** $v = a \wedge b$)

$\phi$ and $\psi$ are not *equal*, but they are equisatisfiable: $\phi$ has a satisfying assignment iff $\psi$ does, because any sat. asst. for $\phi$ gives one for $\psi$ and *vice versa*.

# Circuits can duplicate expressions



A circuit can use the same output more than once:




is $\phi = \neg(a \wedge b) \vee \neg((a \wedge b) \vee \neg c))$

How can we simulate re-use using only logic?

$\psi = (\neg v \vee \neg(v \vee \neg c)) \wedge (v \leftrightarrow a \wedge b)$ (or think: $\neg v \vee \neg(v \vee \neg c)$ **where** $v = a \wedge b$)

$\phi$ and $\psi$ are not *equal*, but they are equisatisfiable: $\phi$ has a satisfying assignment iff $\psi$ does, because any sat. asst. for $\phi$ gives one for $\psi$ and *vice versa*.

We can do this for all the intermediate values, and forget the original formula.

$r$
$r \leftrightarrow x \vee z$
$x \leftrightarrow \neg v$
$v \leftrightarrow a \wedge b$
$z \leftrightarrow \neg y$
$y \leftrightarrow v \vee w$
$w \leftrightarrow \neg c$

# The Tseytin transformation



does with formulae what we've just done with gates.

Introduce a new variable $x$ for every subformula $\phi$, and add a clause saying $x \leftrightarrow \phi$. For example:

(see live demo)

# The Tseytin transformation



does with formulae what we've just done with gates.

Introduce a new variable $x$ for every subformula $\phi$, and add a clause saying $x \leftrightarrow \phi$. For example:

(see live demo)

Having done that, we can convert all the Tseytin formulae to CNF and conjoin them into one big CNF:

(see live demo)

# The Tseytin transformation



does with formulae what we've just done with gates.

Introduce a new variable $x$ for every subformula $\phi$, and add a clause saying $x \leftrightarrow \phi$. For example:

(see live demo)

Having done that, we can convert all the Tseytin formulae to CNF and conjoin them into one big CNF:

(see live demo)

That didn't look very impressive. But as $\phi$ gets bigger, $toCNF(\phi)$ may grow exponentially, while $tseytinCNF(\phi)$ grows linearly:

(see live demo)

# The Tseytin transformation



does with formulae what we've just done with gates.

Introduce a new variable $x$ for every subformula $\phi$, and add a clause saying $x \leftrightarrow \phi$. For example:

(see live demo)

Having done that, we can convert all the Tseytin formulae to CNF and conjoin them into one big CNF:

(see live demo)

That didn't look very impressive. But as $\phi$ gets bigger, $toCNF(\phi)$ may grow exponentially, while $tseytinCNF(\phi)$ grows linearly:

(see live demo)

Tseytin is an $O(n)$ conversion to an equisatisfiable CNF formula.

Unfortunately CNF-SAT can still be exponential – no free lunch.

Final question for you: how long does it take to check satisfiability of a DNF formula?

# 2-CNF-SAT



2-CNF-SAT (or just 2-SAT) is the special case where *every clause has at most two literals*, such as:

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

# 2-CNF-SAT



2-CNF-SAT (or just 2-SAT) is the special case where *every clause has at most two literals*, such as:

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

Any 2-SAT problem can be solved in *linear* time.

# 2-CNF-SAT

2-CNF-SAT (or just 2-SAT) is the special case where *every clause has at most two literals*, such as:

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

Any 2-SAT problem can be solved in *linear* time.

They arise naturally in problems involving may/must/must not relations between things: e.g. which courses you are able to take. Sometimes unfortunate consequences arise from simple rules . . .

# 2-variable clauses



Any two-variable clause can be written in terms of $\vee$ and $\neg$, and vice versa.

Rewriting the previous out of CNF gives:

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

which might represent the following rules:

1. You may not take both Astrology and Chiromancy
2. If you take Belomancy, you must take Chiromancy
3. You must take Astrology or Belomancy
4. If you take Chiromancy, you must take Dream Interpretation
5. You may not take both Dream Interpretation and Belomancy

What can you take?

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

This tells us a lot about satisfying assignments:

- If a literal is true, everything to the right must be true

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

This tells us a lot about satisfying assignments:

- If a literal is true, everything to the right must be true
- If it's false, everything to the left must be false

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

This tells us a lot about satisfying assignments:

- If a literal is true, everything to the right must be true
- If it's false, everything to the left must be false
- $B$ must be false

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

This tells us a lot about satisfying assignments:

- If a literal is true, everything to the right must be true
- If it's false, everything to the left must be false
- $B$ must be false
- if $C$ is true, so is $D$

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

What should I do with $A$, $\neg A$, and $\neg C$?

# Implication clauses



Any two-variable clause can *also* be written in terms of $\rightarrow$ and $\neg$:

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

This is useful because *implication is transitive*:
if $B \rightarrow C$ and $C \rightarrow D$, then $B \rightarrow D$

We can build a partial **graph** of implication between literals:

$$0 \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow \neg B \longrightarrow 1$$

This tells us a lot about satisfying assignments:

- If a literal is true, everything to the right must be true
- If it's false, everything to the left must be false
- $B$ must be false
- if $C$ is true, so is $D$

Satisfying assignments are got from cutting the line somewhere, which must be right of $B$. (And then dealing with the rest.)

$\neg A \vee \neg C$ is symmetrical. Is $A \rightarrow \neg C$ symmetrical? (Remember back to sequents and contraposition. . . )

Remember *barbara*!

$0 \rightarrow$ anything, and anything $\rightarrow 1$

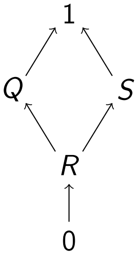
What should I do with $A$, $\neg A$, and $\neg C$?

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$



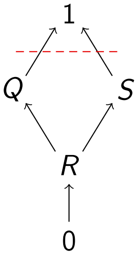

A cut is a set of edges which, when deleted, cut the graph in two.

A valid cut must separate 0 and 1.

# The Arrow Rule

says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$




A cut is a set of edges which, when deleted, cut the graph in two.

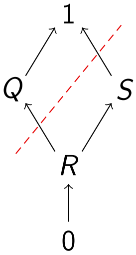
A valid cut must separate 0 and 1.

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$




A cut is a set of edges which, when deleted, cut the graph in two.

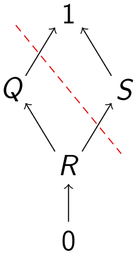
A valid cut must separate 0 and 1.

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \rightarrow Q) \wedge (R \rightarrow S)$$




A cut is a set of edges which, when deleted, cut the graph in two.

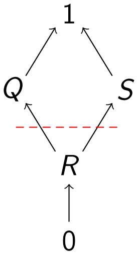
A valid cut must separate 0 and 1.

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$




A cut is a set of edges which, when deleted, cut the graph in two.

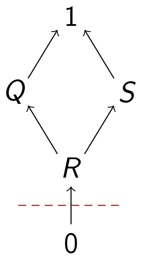
A valid cut must separate 0 and 1.

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$




A cut is a set of edges which, when deleted, cut the graph in two.

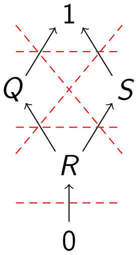
A valid cut must separate 0 and 1.

# The Arrow Rule



says that if we draw the full graph of implications, any valid cut through the graph gives a satisfying assignment: literals above the cut are true, those below are false. Another example:

$$(\neg R \vee Q) \wedge (\neg R \vee S) \quad \text{equiv} \quad (R \to Q) \wedge (R \to S)$$




There are five satisfying assignments, one for each valid cut.

A cut is a set of edges which, when deleted, cut the graph in two.

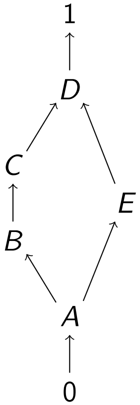
A valid cut must separate 0 and 1.

# Counting assignments



A more complex example:

$$(A \rightarrow B) \wedge (B \rightarrow C) \wedge (C \rightarrow D) \wedge (A \rightarrow E) \wedge (E \rightarrow D)$$

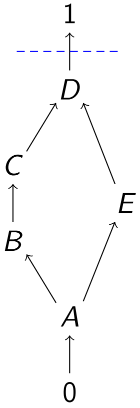


# Counting assignments



A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

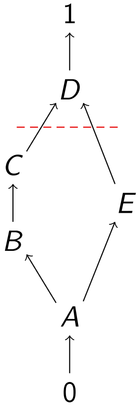


# Counting assignments



A more complex example:

$$(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (A \to E) \wedge (E \to D)$$

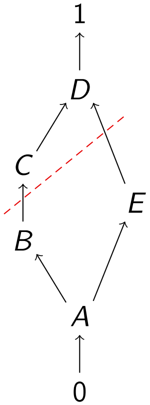


# Counting assignments



A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

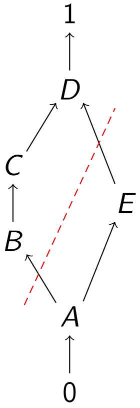


# Counting assignments



A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

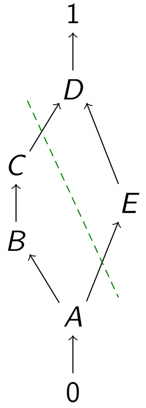


# Counting assignments



A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

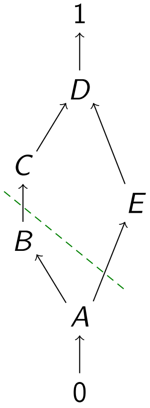


# Counting assignments



A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

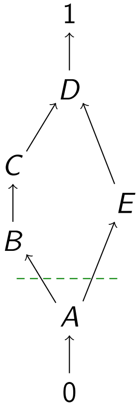


# Counting assignments



A more complex example:

$$(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (A \to E) \wedge (E \to D)$$

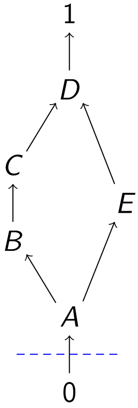


# Counting assignments

A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

There are eight ways to cut this.



# Counting assignments



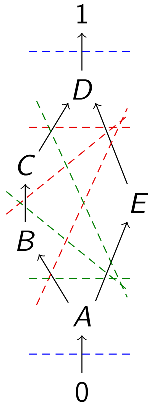
A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

There are eight ways to cut this.

We can count cuts thus:

- one cut above $D$



# Counting assignments

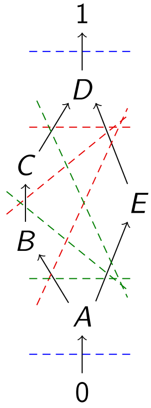
A more complex example:

$$(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (A \to E) \wedge (E \to D)$$

There are eight ways to cut this.

We can count cuts thus:

- one cut above $D$
- cuts across the pentagon: 2 ways to cut the right side, 3 ways to cut the left, so 6



# Counting assignments



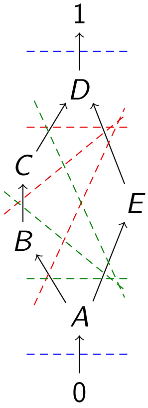
A more complex example:

$$(A \to B) \land (B \to C) \land (C \to D) \land (A \to E) \land (E \to D)$$

There are eight ways to cut this.

We can count cuts thus:

- one cut above $D$
- cuts across the pentagon: 2 ways to cut the right side, 3 ways to cut the left, so 6
- one cut below $A$



# Counting assignments



A more complex example:

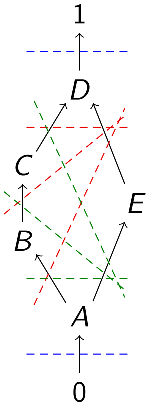
$$(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (A \to E) \wedge (E \to D)$$

There are eight ways to cut this.

We can count cuts thus:

- one cut above $D$
- cuts across the pentagon: 2 ways to cut the right side, 3 ways to cut the left, so 6
- one cut below $A$

For an even more complicated example, see the textbook (Chapter 23, p. 252).

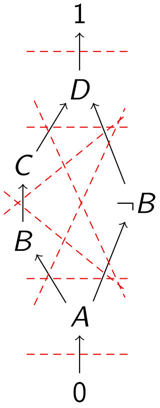


# Complementary literals



What happens with formulae that have $A$ and $\neg A$ (like the very first one)? Such as:

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



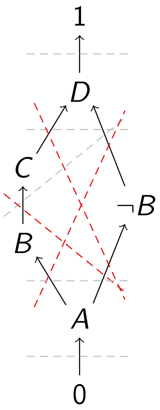
# Complementary literals



What happens with formulae that have $A$ and $\neg A$ (like the very first one)? Such as:

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

A valid cut must *separate complementary literals*, so only 3 cuts survive.



# Complementary literals

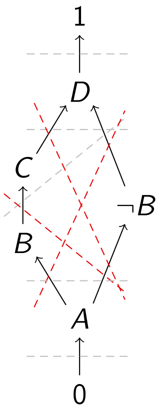


What happens with formulae that have $A$ and $\neg A$ (like the very first one)? Such as:

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

A valid cut must *separate complementary literals*, so only 3 cuts survive.

Note $A \rightarrow \neg B$ is the same as $B \rightarrow \neg A$ (contraposition), so sometimes you can remove complementary literals. This makes thing easier!



# Cycles in the graph



It's quite possible for the implication graph to contain *cycles*. For example:

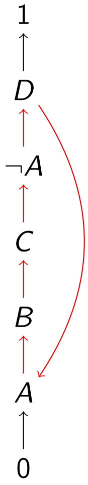
$$(A \to B) \wedge (B \to C) \wedge (C \to \neg A) \wedge (\neg A \to D) \wedge (D \to A)$$

# Cycles in the graph



It's quite possible for the implication graph to contain *cycles*. For example:

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



# Cycles in the graph

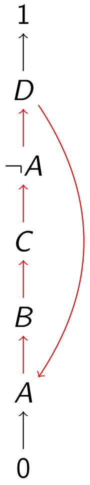


It's quite possible for the implication graph to contain *cycles*. For example:

$$(A \to B) \land (B \to C) \land (C \to \neg A) \land (\neg A \to D) \land (D \to A)$$

Every literal in a cycle must take the same value, so:

A valid cut *must not cut a cycle*.



# Cycles in the graph



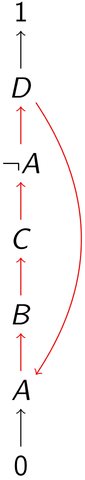
It's quite possible for the implication graph to contain *cycles*. For example:

$$(A \to B) \land (B \to C) \land (C \to \neg A) \land (\neg A \to D) \land (D \to A)$$

Every literal in a cycle must take the same value, so:

A valid cut *must not cut a cycle*.

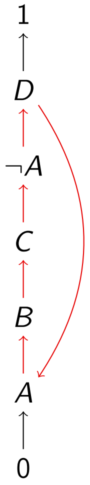
In this is example, the cycle contains complementary literals, so must be cut! There is **no satisfying assignment**.



# Cycles in the graph

It's quite possible for the implication graph to contain *cycles*. For example:

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




Every literal in a cycle must take the same value, so:

A valid cut *must not cut a cycle*.

In this is example, the cycle contains complementary literals, so must be cut! There is **no satisfying assignment**.

Sometimes cycles can be removed by taking the contrapositive. Go back to the first example (slide 12) and complete it both with and without a cycle.

# Summary



Drawing the implication graph and counting valid cuts lets us count satisfying assignments of 2-SAT formulae.

A valid cut must:

- separate 0 and 1
- separate complementary literals
- not cut a cycle

# Summary



Drawing the implication graph and counting valid cuts lets us count satisfying assignments of 2-SAT formulae.

A valid cut must:

- separate 0 and 1
- separate complementary literals
- not cut a cycle

Why do we care? It turns out that #2-SAT (as it is known) has application in statistical physics and artificial intelligence. It is also of theoretical interest in several ways.

# Summary



Drawing the implication graph and counting valid cuts lets us count satisfying assignments of 2-SAT formulae.

A valid cut must:

- separate 0 and 1
- separate complementary literals
- not cut a cycle

Why do we care? It turns out that #2-SAT (as it is known) has application in statistical physics and artificial intelligence. It is also of theoretical interest in several ways.

(There is one quirk we haven't considered. What if the implication graph is *non-planar*? See the book for how to deal with that.)