

COSC 290: Discrete
Structures

Propositional Logic II

02/03/2026

Prof. Forrest

Warm up & Goals

Warm up

1. Talk to your neighbor
an event you attended on
campus recently that was
fun/interesting!

2. Demonstrate De Morgan's 2nd law holds

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$
T	T					
T	F					
F	T					
F	F					

Logistics

• Codelet 2 is due Friday

Satisfiability

$$\phi \Rightarrow P \wedge Q \quad \phi \text{ is Satisfiable} \quad \models$$

↓ ↓

T T

Definition: A truth assignment I satisfies a formula ϕ if $\phi[I] = T$; we write this as $I \models \phi$. A formula ϕ is satisfiable if there exists a truth assignment I s.t. $I \models \phi$; otherwise ϕ is unsatisfiable.

Definition: A formula ϕ is valid (or a tautology) if for all truth assignments I , $I \models \phi$.

$$\phi := P \wedge \neg P$$

$$(P \rightarrow Q) \vee (Q \rightarrow P)$$
$$P \vee \neg P$$

Program Verification

```
def f(x: int) → int:
    if x == 0:
        y = 1
    else:
        y = 2
    return y
```

```
def main():
    ...
    ...
```

assume x is some integer (pre condition)
 $y = f(x)$
 assert $y > 0$ (post condition)

Propositions

$P \quad x = 0$
 $Q \quad y = 1$
 $R \quad y = 2$
 $C \quad y > 0$

$P \rightarrow Q$ } if
 $\neg P \rightarrow R$ } else

$\neg(Q \wedge R)$ } only one
 $(Q \vee R)$ } has to be 1, 2

Will this program crash?

$$(P \rightarrow Q) \wedge (\neg P \rightarrow R) \wedge (\neg Q \vee \neg R) \wedge (Q \vee R) \wedge ((Q \vee R) \rightarrow C) \wedge \neg C$$

How do we use satisfiability to determine whether this program will crash?

S	u	$S \rightarrow u$
T	T	T
T	F	F
F	T	T
F	F	T

Slightly more interesting

```
def func(a: bool) → int:
```

```
    if a:
```

```
        x = 1
```

```
    else:
```

```
        x = 2
```

```
    if x == 2:
```

```
        y = 0
```

```
    else:
```

```
        y = 1
```

```
    return y
```

```
def main():
```

```
    x = 1
```

```
    x = 2
```

```
    # a is a valid bool
```

```
    y = func(a)
```

```
    assert y == 1
```

propositions

$p: a \text{ is True}$

$q: x = 1$

$r: x = 2$

$s: y = 0$

$u: y = 1$

$\text{func} \wedge \neg u$

will this program crash?

Satisfiability Applications

- program correctness
- Hardware verification (through logic gates)
covered in 2008
- Scheduling
- Sudoku
- robotics
- Software testing

Procedure

- ① Specifying a model in logic
(e.g., code does not create deadlocks)
- ② then describe system in logic
(e.g., piece of code, and the logic behind execution)
- ③ show $\text{system} \models \text{model}$

Rules of Inference

or how to show your argument is valid

if it is ^Psnowy, then class is canceled } premises
it is snowy }
therefore ^Pclass is canceled } conclusion

$$\frac{P \rightarrow Q}{P} \quad \left. \vphantom{\frac{P \rightarrow Q}{P}} \right\} \text{Syntax}$$
$$\therefore Q$$

$$\frac{((P \rightarrow Q) \wedge P) \rightarrow Q}{\text{tautology}}$$

if you live on Mars, then you are a trillionaire
you live on Mars
therefore you are a trillionaire

$$\underline{((p \rightarrow q) \wedge p) \rightarrow q}$$

tautology

p	q	$p \rightarrow q$	$(p \rightarrow q) \wedge p$	$((p \rightarrow q) \wedge p) \rightarrow q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

$$p_1 \wedge p_2 \wedge p_3 \dots \wedge p_n \rightarrow q$$

tautology

Logical equivalence

- Step by step convert S to a specific form

Sentence
or
formula

- Simplify our S to equivalent φ
that are tautologies

CNF

Conjunctive normal
form

Something is CNF is easy to check

if it is the conjunction of
a bunch of disjunctions of literals

clauses

$(\neg p)$
 $\neg p$

1 clause

$p \vee q$
 $\uparrow$
CNF

$(p \vee q)$

$(\neg p)$
 $\neg p$

$(r \vee s)$

$(p \wedge q) \vee (r \wedge \neg p)$
 $\uparrow$
1 2

Not
CNF

$$\varphi := (p \vee q \vee \neg p) \wedge (r \vee q \vee p \vee \neg q) \wedge \neg r$$

$$\downarrow \text{ T}$$

$$(p \vee \neg p)$$

not tautology

Build CNF

① replace unnecessary operators like $\rightarrow$ $\leftrightarrow$ $\oplus$
 $\vee$ $\wedge$ $\neg$

2. push negatives down

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

3. or over and

Step 1: replace if

$$\underbrace{(p \wedge (p \rightarrow q))}_{\text{replace if}} \oplus q$$

$$p \rightarrow q \equiv \neg p \vee q$$

(1) replace $\rightarrow$

replace this
if

replace $(p) \rightarrow p$

$$\equiv \neg(p \wedge (p \rightarrow q)) \vee q$$

replace this one

$$\equiv \neg(p \wedge (\neg p \vee q)) \vee q$$

Step 2 Push down negation

$$\equiv \neg(p \wedge (\neg p \vee q)) \vee q$$

$$\equiv \neg p \vee \neg(\neg p \vee q) \vee q$$

$$\equiv (\neg p \vee (p \wedge \neg q)) \vee q$$

$$\neg(p \wedge q)$$

$$\neg p \vee \neg q$$

Step 3
or over and

$$\equiv \underbrace{(\neg p \vee (p \wedge \neg q)) \vee q}$$

or and or

$$((\neg p \vee p) \wedge (\neg p \vee \neg q)) \vee q$$

these are equivalent

$$r \vee (p \wedge q)$$

$$(r \vee p) \wedge (r \vee q)$$

$$(q \vee (\neg p \vee p)) \wedge (q \vee (\neg r \vee \neg q))$$

↑ ↑ ↑ ↑ ↑
 conjunction

$$2 \cdot (x + y)$$

$$2x + 2y$$

$$(\neg p \wedge (p \rightarrow q)) \rightarrow q \equiv (q \vee \neg p \vee p) \wedge (q \vee \neg p \vee \neg q)$$

Converted w/ three steps

1. replace $\rightarrow$ with $\vee$

2. move negation down

$$\text{eg. } \neg(p \wedge q) \equiv \neg p \vee \neg q$$

3. move or over and

$$s \vee (t \wedge r) \equiv (s \vee t) \wedge (s \vee r)$$

at the end we have a

CNF ☺

easy to check if a
tautology. we find in each
clause that there is a
literal $(p \vee q)$ and its
opposite $(\neg p \vee \neg q)$

Imagine we are defining a recursive function
or-over-and that takes a sentence φ
that is in NNF and returns a CNF.

Suppose $\varphi: \varphi_1 \vee \varphi_2$ and we make recursive calls

$$S_1 = \text{or-over-and}(\varphi_1)$$

$$S_2 = \text{or-over-and}(\varphi_2)$$

Suppose you inspect the returned propositions S_1 and
 S_2 , and it turns out that

$$S_1 \text{ is of the form } S_{11} \vee S_{12}$$

$$S_2 \text{ is of the form } S_{21} \vee S_{22}$$

Then, is $\varphi = S_1 \vee S_2$ in CNF?

(a) Yes

(b) Not necessarily