Set Properties

CS 231
Dianna Xu

Set Identities

- Basic laws on how set operations work
- Just like logical equivalence laws!
  - Replace U with ∨
  - Replace ∩ with ∧
  - Replace complement with ~
  - Replace ∅ with **c**
  - Replace ***U*** with **t**
- One additional on set differences

Set identities: De Morgan again

- These should look very familiar…

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Communicative	A U B = B U A	A ∩ B = B ∩ A
Associative	(A U B) U C = A U (B U C)	(A ∩ B) ∩ C = A ∩ (B ∩ C)
Distributive	A U (B ∩ C) = (A U B) ∩ (A U C)	A ∩ (B U C) = (A ∩ B) U (A ∩ C)
Identity	A U ∅ = A	A ∩ ***U*** = A
Complement	A U A^c = ***U***	A ∩ A^c = ∅
Double Complement	$(A^c)^c$ = A	
Idempotent	A U A = A	A ∩ A = A
Universal Bound	A U ***U*** = ***U***	A ∩ ∅ = ∅
De Morgan's	$(A \cup B)^c = A^c \cap B^c$	$(A \cap B)^c = A^c \cup B^c$
Absorption	A U (A ∩ B) = A	A ∩ (A U B) = A
Complement of ***U*** and ∅	$U^c = \varnothing$	$\varnothing^c = U$
Set Difference	A – B = A ∩ B^c	

Subset Relations

- A ∩ B ⊆ A, A ∩ B ⊆ B
- A ⊆ A U B, B ⊆ A U B
- A ⊆ B ∧ B ⊆ C → A ⊆ C

Proofs

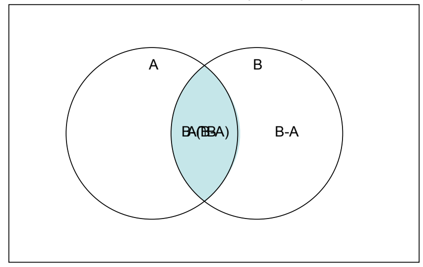
- To prove that A is a subset of B (A ⊆ B):
  - Assume that x∈A is a particular but arbitrarily chosen element of A
  - Show that x ∈ B
- To prove that two sets A and B are equal (A = B):
  - prove A ⊆ B, and
  - prove B ⊆ A

How to Prove a Set Identity

- For example: A∩B = B–(B–A)
- Methods:
  - The element method: Prove each set is a subset of each other, by showing any element that belongs to one also belongs to the other
  - Algebraic Proof: Use the set identity laws

What we are going to prove…

A∩B = B–(B–A)



Proof by Set Identity Laws

- Prove that A∩B=B–(B–A)

$B-(B-A)=B-(B\cap\overline{A})$	Definition of difference
$=B\cap\overline{(B\cap\overline{A})}$	Definition of difference
$=B\cap(\overline{B}\cup\overline{\overline{A}})$	De Morgan's law
$=B\cap(\overline{B}\cup A)$	Double Complement
$=(B\cap\overline{B})\cup(B\cap A)$	Distributive law
$=\varnothing\cup(B\cap A)$	Complement law
$=(B\cap A)$	Identity law
$=A\cap B$	Commutative law ■

Proof by Element Method

- Assume that an element is a member of one of the identities implies that it is a member of the other
- Repeat for the other direction
- We are trying to show:
  - (x ∈ A∩B → x ∈ B–(B–A))∧(x ∈ B–(B–A) → x ∈ A∩B)
  - This is the bi-conditional: x ∈ A∩B ↔ x ∈ B–(B–A)
- Not good for long proofs

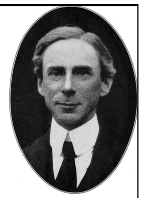
Proof by Element Method

- Assume that x ∈ B–(B–A)
  - By definition of set difference, x ∈ B ∧ x ∉ B–A
- Consider x ∉ B–A
  - x ∈ B–A = x ∈ B ∧ x ∉ A
  - x ∉ B–A = ~(x ∈ B ∧ x ∉ A) = x ∉ B ∨ x ∈ A
- So we have x ∈ B ∧ (x ∉ B ∨ x ∈ A)
  - x ∈ B ∧ x ∉ B = **c**
  - x ∈ B ∧ x ∈ A = x ∈ A∩B
  - Thus, x ∈ B–(B–A) → x ∈ A∩B
- B–(B–A) ⊆ A∩B

Proof by Element Method

- Assume that x ∈ A∩B
  - By definition of intersection, x ∈ A ∧ x ∈ B
- Thus, we know that x ∉ B–A
  - B–A includes all the elements in B but not in A
- Consider B–(B–A)
  - We know x ∈ B ∧ x ∉ B–A
  - By definition of difference, x ∈ B–(B–A)
- x ∈ A∩B → x ∈ B–(B–A)
- A∩B ⊆ B–(B–A) ■

Russell's Paradox



- Consider the set:
  - $S = \{ A \mid A \text{ is a set} \wedge A \notin A \}$
- Is S an element of itself?
- Consider:
  - $S \in S$
    - Then S can not be in itself, by definition
  - $S \notin S$
    - Then S is in itself by definition
  - Contradiction!

How Do We Fix It?

- Consider the set:
  - $S = \{ A \mid A \subseteq \boldsymbol{U} \wedge A \notin A \}$
- Similarly:
  - $S \in S \rightarrow S \subseteq \boldsymbol{U} \wedge S \notin S$
- But:
  - $S \notin S \rightarrow \sim(S \subseteq \boldsymbol{U} \wedge S \notin S) = S \nsubseteq \boldsymbol{U} \vee S \in S$
- In other words, S is not a proper set

The Halting Problem

- Given a program P, and input I, will the program P ever terminate?
  - Meaning will P(I) loop forever or halt?
- Can a computer program determine this?
  - Can a human?
- First shown by Alan Turing in 1936

Some Notes

- To "solve" the halting problem means we create a function CheckHalt(P,I)
  - P is the program we are checking for halting
  - I is the input to that program
- And it will return "loops forever" or "halts"
- Note it must work for *any* program, not just some programs, and *any* input

Perfect Numbers

- Numbers whose divisors (not including the number) add up to the number
  - 6 = 1 + 2 + 3
  - 28 = 1 + 2 + 4 + 7 + 14
- The list of the first 10 perfect numbers:
  6, 28, 496, 8128, 33550336, 8589869056, 137438691328, 2305843008139952128, 2658455991569831744654692615953842176, 191561942608236107294793378084303638130997321548169216
  - The last one was 54 digits!
- All known perfect numbers are even; it's an open (i.e. unsolved) problem if odd perfect numbers exist

Where Does That Leave Us?

- If a human can't figure out how to do the halting problem, we can't make a computer do it for us
- It turns out that it is impossible to write such a CheckHalt() function
  - But how to prove this?

CheckHalt()'s Non-existence

- Consider P(I): a program P with input I
- Suppose that CheckHalt(P,I) exists
  - prints either "loop forever" or "halt"
- A program is a series of bits
  - And thus can be considered data as well
- Thus, we can call CheckHalt(P,P)
  - It's using the bits of program P as the input to program P

CheckHalt()'s non-existence

- Consider a new function:
  Test(P):
  loops forever if CheckHalt(P,P) prints "halts"
  halts if CheckHalt(P,P) prints "loops forever"
- Now run Test(Test)
  - If Test(Test) halts…
    - Then CheckHalt(Test,Test) returns "loops forever"…
    - Which means that Test(Test) loops forever
    - Contradiction!
  - If Test(Test) loops forever…
    - Then CheckHalt(Test,Test) returns "halts"…
    - Which means that Test(Test) halts
    - Contradiction!

The Halting Problem

- It was the first algorithm that was shown to not be able to exist
  - You can prove an existential by showing an example (a correct program)
  - But it's much harder to prove that a program can *never* exist