Set Identities

- Basic laws on how set operations work
- Just like logical equivalence laws!
  - Replace $\cup$ with $\vee$
  - Replace $\cap$ with $\wedge$
  - Replace complement with $\sim$
  - Replace $\emptyset$ with $c$
  - Replace $U$ with $t$
- One additional on set differences

Set identities: De Morgan again

- These should look very familiar…
  \[
  A \cap B = \bar{A} \cup \bar{B} \\
  A \cup B = \bar{A} \cap \bar{B}
  \]

Subset Relations

- $A \cap B \subseteq A, A \cap B \subseteq B$
- $A \subseteq A \cup B, B \subseteq A \cup B$
- $A \subseteq B \land B \subseteq C \rightarrow A \subseteq C$

Proofs

- To prove that $A$ is a subset of $B$ ($A \subseteq B$):
  - Assume that $x \in A$ is a particular but arbitrarily chosen element of $A$
  - Show that $x \in B$
- To prove that two sets $A$ and $B$ are equal ($A = B$):
  - prove $A \subseteq B$, and
  - prove $B \subseteq A$
How to Prove a Set Identity

- For example: \( A \cap B = B - (B - A) \)
- 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...

\[ A \cap B = B - (B - A) \]

Proof by Set Identity Laws

- Prove that \( A \cap B = B - (B - A) \)
- By definition of difference, \( B - (B - A) = B \cap (B \cup \overline{A}) \)
- By definition of difference, \( B \cap (B \cup \overline{A}) = B \cap \overline{B} \cup A \)
- By De Morgan's law, \( B \cap \overline{B} \cup A = B \cap (B \cup A) \)
- By Double Complement, \( B \cap (B \cup A) = (B \cap B) \cup (B \cap A) \)
- By Distributive law, \( (B \cap B) \cup (B \cap A) = \emptyset \cup (B \cap A) \)
- By Complement law, \( \emptyset \cup (B \cap A) = B \cap A \)
- By Identity law, \( B \cap A = A \cap B \)
- By 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:
  - \( (x \in A \cap B \rightarrow x \in B - (B - A)) \land (x \in B - (B - A) \rightarrow x \in A \cap B) \)
  - This is the bi-conditional: \( x \in A \cap B \iff x \in B - (B - A) \)
- Not good for long proofs

Proof by Element Method

- Assume that \( x \in A \cap B \)
  - By definition of intersection, \( x \in A \land x \in B \)
- Thus, we know that \( x \notin B - A \)
  - \( B - A \) includes all the elements in \( B \) but not in \( A \)
- Consider \( B - (B - A) \)
  - We know \( x \in B \land x \notin B - A \)
  - By definition of difference, \( x \in B - (B - A) \)
- \( x \in A \cap B \rightarrow x \in B - (B - A) \)
- \( A \cap B \subseteq B - (B - A) \)
Russell’s Paradox

• Consider the set:
  \[ S = \{ A \mid A \text{ is a set} \land A \notin A \} \]
• 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
  \[ \text{Contradiction!} \]

How Do We Fix It?

• Consider the set:
  \[ S = \{ A \mid A \subseteq U \land A \notin A \} \]
• Similarly:
  \[ S \in S \Rightarrow S \subseteq U \land S \notin S \]
• But:
  \[ S \notin S \Rightarrow (S \subseteq U \land S \notin S) \Rightarrow S \notin U \lor S \in S \]
• 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 \( \text{CheckHalt}(P,I) \)
  • \( P \) 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 \textit{any} program, not just some programs, and \textit{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, 265845991569831744654692615953842176, 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 \text{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