CS270 - Week 6 Lecture Notes

Mark Boady

November 1, 2017

1 Contradiction Rules

Using negatives in Logical Deductions can be very powerful.

This section introduces a new symbol $\bot$. This symbol is used to represent a contradiction. For example, we would write

1.	$\sim A$	Premise
2.	A	Premise
3.	$\bot$	$\bot I1,2$

Once we have $A \wedge \sim A$ the expression is definitively false. We could just write F to say the proof is false, but this might be confused with a variable. The $\bot$ symbol is used to say that contradiction has been reached.

1.1 Introduction Rules

The contradiction symbol is most frequently used together with negative introduction.

1.	$(P \rightarrow (Q \vee R))$	Premise
2.	$\sim (Q \vee R)$	Premise
3.	P	Assume
4.	$(Q \vee R)$	$\rightarrow E1,3$
5.	$\bot$	$\bot I2,4$
6.	$\sim P$	$\sim I3-5$

There are two rules nested in this Proof. First we have Contradiction Introduction. $\bot I2,4$ says that line 2 and line 4 are opposites. This is a contradiction. Line 5 shows the contradiction was reached.

This contradiction is part of a subproof. We want to prove $\sim P$, but there is no way to move forward from our starting premises.

Instead, we take the goal and search for a contradiction. If P being true is impossible, then we can draw the conclusion that $\sim P$ must be true.

Line 6 says $\sim I3-5$. We introduced a negative sign in front of P because the subproof ending on line 5 proved making P true was impossible.

These two rules will frequently be used together.

1.2 Negative Elimination

Negative elimination allows for the opposite of negative introduction. We assume that value should be negative. If that is impossible, then we know the answer must be positive.

1.	$(\sim I \rightarrow B)$	Prem
2.	$(B \rightarrow I)$	Prem
3.	$\sim I$	Assume
4.	B	$\rightarrow E1,3$
5.	I	$\rightarrow E2,4$
6.	$\perp$	$\perp I3,5$
7.	I	$IP3-6$

The assumption on line 3 is used to justify line 7 because of the contradiction reached on line 6.

Line 7's note $IP3-6$ only lists the subproof that finds the contradiction.

1.3 Contradiction Elimination

If a subproof has reached a contradiction, then it is false. This is an absolute certainty. Added a new line to a proof is just "and"-ing it with all previous values. Once the subproof is definitively false, we can "and" new things without making it true since $F \wedge A = F$ regardless of the value of A.

This means we can use Contradiction Elimination to introduce anything into an expression. This allows for a shortcut. If we have reached a contradiction in a proof, we can use it to get anywhere we need to.

1.	M	Premise
2.	$\sim M$	Premise
3.	$\perp$	$\perp I1,2$
4.	$(A \vee B)$	$X3$

1.4 Proof That $P = \sim\sim P$

Proof that $P \rightarrow \sim\sim P$.

1.	P	Premise
2.	$\sim P$	Assume
3.	$\perp$	$\perp I1,2$
4.	$\sim\sim P$	$\sim I2-3$

Proof that $\sim\sim P \rightarrow P$

1.	$\sim\sim P$	Premise
2.	$\sim P$	Assume
3.	$\perp$	$\perp I 1, 2$
4.	P	$\sim IP 2-3$

Now we have $(\sim\sim P \to P) \wedge (P \to \sim\sim P)$ which proves that $P = \sim\sim P$.

1.5 Validity of the Cut Rule

The Cut Rule says $(P \vee Q) \wedge (\sim P \vee R) \to (Q \vee R)$.

1.	$(P \vee Q)$	Premise
2.	$(\sim P \vee R)$	Premise
3.	P	Assume
4.	$\sim P$	Assume
5.	$\perp$	$\perp I 3, 4$
6.	$(Q \vee R)$	$X 5$
7.	R	Assume
8.	$(Q \vee R)$	$\vee I 7$
9.	$(Q \vee R)$	$\vee E 2, 4-6, 7-8$
10.	Q	Assume
11.	$(Q \vee R)$	$\vee I 10$
12.	$(Q \vee R)$	$\vee E 1, 3-9, 10-11$

2 Practical Proofs

These are from the Book of Logic suggested reading.

2.1 Numeric Proof

Notation:
$\mathbb{Z}$ is the set of all integers $\mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$.
$\in$ is the symbol for in. It means a variable is selected from a set.
Saying $a \in \mathbb{Z}$ is saying that a is an integer.

Proposition 1. *If $a, b \in \mathbb{Z}$, then $a^2 - 4b \neq 2$.*

Proof. We prove this with a contradiction.

1.	$a \in \mathbb{Z}$	Premise
2.	$b \in \mathbb{Z}$	Premise
3.	$a^2 - 4b = 2$	Assume
4.	$a^2 = 4b + 2$	Add $4b$ to both sides
5.	$a^2 = 2(2b + 1)$	Factor right side
6.	Let $a = 2c$	a must be even because it is 2(integer)
7.	$(2c)^2 = 2(2b + 1)$	Replacement
8.	$4c^2 = 2(2b + 1)$	Simplification
9.	$2c^2 = 2b + 1$	Divide both sides by 2
10.	$2c^2 - 2b = 1$	Subtract $2b$ from both sides
11.	$2(c^2 - b) = 1$	Factor out 2
12.	$c^2 - b = \frac{1}{2}$	Divide Both sides by 2
13.	$\perp$	$(c^2 - b) \in \mathbb{Z}$ but $\frac{1}{2}$ is not.
14.	$a^2 - 4b \neq 2$	By Contradiction Line 13

Q.E.D.

2.2 Primes Proof

Proposition 2 (Infinite Primes)**.** *There are infinitely many prime numbers.*

Proof. Assume there are a finite number of prime numbers. Let p_n be the largest prime number. All prime numbers can be iterated $p_1, p_2, \cdots, p_n$.

We create a new value, a, by multiplying all prime numbers together and adding 1.

$$a = (p_1 p_2 p_3 \cdots p_{n-1} p_n) + 1 \tag{1}$$

It is known that any integer greater than 1 must have at least 1 prime divisor. Therefore, we can say that there exists a prime p_k and integer c such that

$$a = c p_k \tag{2}$$

Let us assume that $1 \leq k \leq n$, that p_k is in the set of known prime numbers.

$$(p_1 p_2 p_3 \cdots p_{n-1} p_n) + 1 = c p_k \tag{3}$$

$$(p_1 p_2 p_3 \cdots p_{k-1} p_{k+1} \cdots p_{n-1} p_n) + \frac{1}{p_k} = c \tag{4}$$

This is a contradiction. We know that c is an integer and the product $(p_1 p_2 p_3 \cdots p_{k-1} p_{k+1} \cdots p_{n-1} p_n)$ is an integer. We also know that $\frac{1}{p_k}$ cannot be an integer. Since an integer added to a non-integer cannot be an integer, this equivalence is a contradiction.

This proves the p_k must be a prime number larger then p_n. This is a contradiction to the original assumption that p_n was the largest prime number. We have shown that there cannot be a largest prime number, therefore the set of prime numbers must be infinite.

Q.E.D.

3 Computer Science Proofs

This is a very brief introduction to a very complex theory. Forgive me for skipping a ton of details.

We can think of any program/algorithm as something that takes inputs and returns true or false.

For example, if we have a sorting algorithm quicksort(list). We would normally say this returns the list in a sorted form. We could also write the function as quicksort(input, output). The function returns true if is sorted correctly. As a side effect, the sorted values are places into output.

We can say that any program either accepts (returns true/success) or rejects (returns false/failure). All the rest of the programs actions can be considered side effects.

There is a third option. A program might go into an infinite loop and never give a result.

A decidable problem is one where a program will either accept or reject in a finite amount of time.

An undecidable problem is one where a program MIGHT loop infinitely. It is impossible for a program to always give an answer in finite time for an undecidable problem.

I recommend highly "Introduction to the Theory of Computation" by Michael Sipser for a formal introduction to these proofs. Also, take our Theory of Comp class.

3.1 The Halting Problem

Proposition 3 (Halting Problem)**.** *The problem of deciding if a program will halt is undecidable.*

Assume we have a program halt that takes a program P and the input to the program I. The program, halt(P, I) will return true if P returns a result in finite time for input I. If program P would enter an infinite loop, then halt(P,I) returns false.

If this program existed, we could write another program that uses its output.

cond() - Run halt on myself and do the opposite.

The program halt cannot decide what cond will do. It will always be wrong, but we assume it was correct. Therefore, the program cannot exist.

See related C++ files for a more detailed example.

3.2 Accept

Proposition 4 (Accepting Problem)**.** *The problem of deciding if a program will accept in undecidable.*

Assume we have a program accept(P,I) that returns true if P will accept input I in finite time.

We can use this to make program reject.

reject(P,I): make program M:(return not (P,I)). If accept(M) then return true else return false.

We can use accept to solve halt.

halt(P,I): if (accept(P,I) or reject(P,I)) return Will Halt else return Will Not Halt

This is a contradiction because halt cannot exist.

3.3 Empty

Proposition 5 (Empty Problem)**.** *The problem of deciding if a program will accept at least one input is undecidable.*

Assume we have a program Empty(P) that returns true if P accepts NO inputs and false otherwise.

We can use this to solve accept.

accept(P,I):

1. Make a new program M(w): if w==I run P(I) otherwise reject
2. Run Empty(M)
3. If M is empty, then P(I) rejects. If M is non-empty then it accepted exactly one string I.
4. return not Empty(M)

This program solves the accept problem, but that is contradiction. This must be undecidable.

3.4 Equal Programs

Proposition 6 (Equal)**.** *The problem of deciding if two programs accept the same inputs is undecidable.*

Assume we have a program Equal(A,B) that returns true if A and B only accept the same set of inputs. (The programs are functionally the same.)

We can use this to solve Empty

empty(A):

1. Make a TM R: return false
2. if Equal(A,R) return true else return False

This decides equal, that is a contradiction. This problem is undecidable.