



Foundations of Computing I

Pre-Lecture Problem

Create a Boolean Algebra expression for the following truth table (for the function F):

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Normal Forms

DNF: "OR of ANDs"

$$(\neg x \wedge \dots \wedge y) \vee \dots \vee (a \wedge \dots \wedge \neg b)$$

CNF: "AND of ORs"

$$(\neg x \vee \dots \vee z) \wedge \dots \wedge (a \vee \dots \vee \neg b)$$

In both of these, negations are "pushed" all the way in and must only appear directly next to a literal.

These forms are useful *computationally* because they are easy to work with (fewer cases, easier to simplify, ...).

Canonical Forms

Given a Truth Table...

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

How can we find normal forms?

If we use the same procedure, then we have a **canonical** form.

This means we can quickly check equality without relying on Boolean Simplification!

Sum-of-Products Canonical Form

AKA **Complete Disjunctive Normal Form (CDNF)**

① Read T rows off truth table

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

② Convert to Boolean Algebra

③ Add the minterms together

$$F = A'B'C + A'BC + AB'C + ABC' + ABC$$

Handwritten notes: $A'B$, $A'B(C'+C)$, $A'B$

Sum-of-Products Canonical Form

- ANDed product of literals - input combination for which output is true
- Each variable appears exactly once, true or inverted (but not both)

F in CDNF:

$$F(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC$$

Only this one is "CDNF" or Sum-of-Products Canonical Form

canonical form \neq minimal form

$$F(A, B, C) = A'B'C + A'BC + AB'C + ABC + ABC'$$

$$= (A'B' + A'B + AB' + AB)C + ABC'$$

$$= ((A' + A)(B' + B))C + ABC'$$

$$= C + ABC'$$

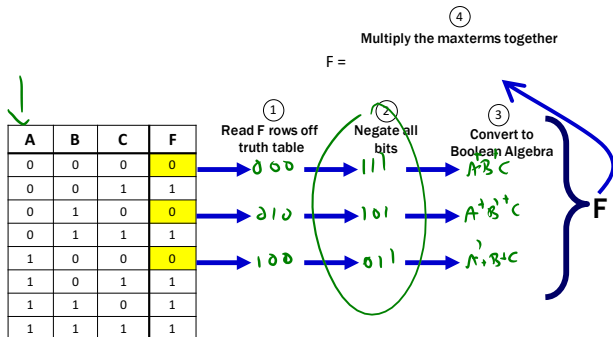
$$= ABC' + C$$

$$= AB + C$$

Both of these are in "DNF"

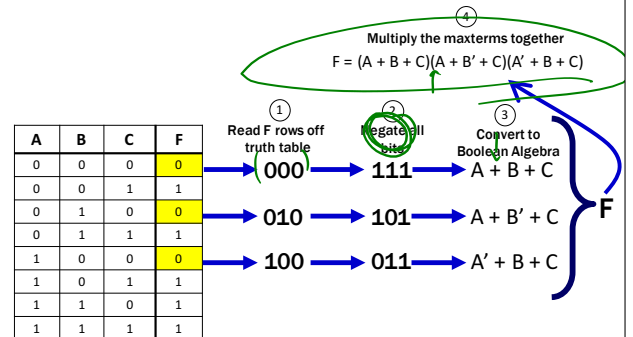
Product-of-Sums Canonical Form

AKA Canonical Conjunctive Normal Form (CCNF)



Product-of-Sums Canonical Form

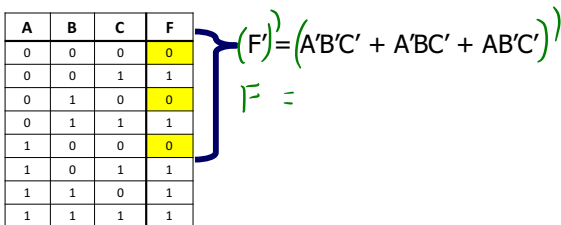
AKA Canonical Conjunctive Normal Form (CCNF)



Product-of-Sums: Why does this procedure work?

Useful Facts:

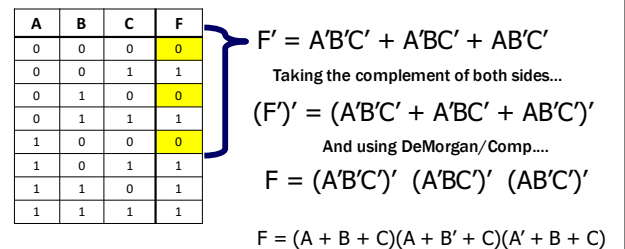
- We know $(F')' = F$
- We know how to get a DNF expansion for F'



Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know $(F')' = F$
- We know how to get a DNF expansion for F'



Some Administrivia

HW 1 Feedback Released

CSE 311 A

To see feedback, go to Canvas and click "View Feedback".

Spring 2017

If prompted, log in with your UWNnetID—not your CSEnetID!

Home

View Feedback

Submit Homework

Then, click on the buttons to see your feedback!

Problem Name	Score	View Feedback
HW1		
0	X / 24	HW1-0
1	X / 16	HW1-1
2	X / 20	Submitted Online
3	X / 20	Submitted Online
4	X / 10	HW1-4
5	X / 10	HW1-5

Some Administrivia

- Workshops start today!!!!!!
- Every Wednesday, from 4pm – 6pm in OUG 136, TAs & I will be there to help you work on extra problems.
- We will have whiteboards, markers, and extra problems.
- You can show up to as little or as much of a workshop as you like. We recommend at least 20 minutes though.

Some Administrivia

Group Maker is now online:

If you would like us to help you find a group, go to:

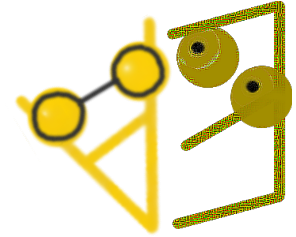
<https://grinch.cs.washington.edu/groups>

You will be asked some questions to help facilitate finding you a group.

We will re-make groups as necessary **every Friday at 8am**. So, if you want a group this week, make sure to sign up by then.

CSE 311: Foundations of Computing

Lecture 6: Predicate Logic



Predicate Logic

• Propositional Logic

– If the tortoise walks at a rate of one node per step, and the hare walks at a rate of two nodes per step, ...

• Predicate Logic

– If the tortoise is on node x , and the hare is on node $2x$, then ...

Predicate Logic

• Propositional Logic

– Break down a statement into pieces

• Predicate Logic

– Relates pieces of a statement to each other

What is a "Predicate"?

A **predicate** is a **method (function)** with arguments that returns a **boolean**.

Examples:

- `isPrime(x)`
- `isLessThan(x, y)`
- `hasSumOf(x, y, z)` $\{ \begin{matrix} x + y = z \\ \text{note } x + y = z \end{matrix} \}$

We will not give "implementations" of predicates. Instead, we'll assumed they're already defined "the way we want".

Defining a Predicate

`Cat(x)` ::= "x is a cat"

`Prime(x)` ::= "x is prime"

`HasTaken(x, y)` ::= "student x has taken course y"

`LessThan(x, y)` ::= " $x > y$ "

`Sum(x, y, z)` ::= " $x + y = z$ "

`GreaterThan5(x)` ::= " $x > 5$ "

`HasNChars(s, n)` ::= "string s has length n"

Notice that predicates can have varying numbers of arguments and input types.

Domain of Discourse

For ease of use, we define one “type”/“domain” that we work over. This set of objects is called the “**domain of discourse**”.

For each of the following, what might the domain be?

- (1) x is a cat, “x barks”, “x ruined my couch”
mammals “dogs”
- (2) “x is prime”, “x = 0”, “x < 0”, “x is a power of two”
- (3) “student x has taken course y” “x is a pre-req for z”

Domain of Discourse

For ease of use, we define one “type”/“domain” that we work over. This set of objects is called the “**domain of discourse**”.

For each of the following, what might the domain be?

- (1) “x is a cat”, “x barks”, “x ruined my couch”
“mammals” or “sentient beings” or “cats and dogs” or ...
- (2) “x is prime”, “x = 0”, “x < 0”, “x is a power of two”
“numbers” or “integers” or “integers greater than 5” or ...
- (3) “student x has taken course y” “x is a pre-req for z”
“students and courses” or “university entities” or ...

A Quick Note on “Variable Definition”

What’s wrong here?

isEven(x) ::= “y is even”

A Quick Note on “Variable Definition”

What’s wrong here?

isEven(x) ::= “**y** is even”

The definition doesn’t make sense, because y isn’t defined. It’s like writing the following code:

```
isEven(x) { return y % 2 == 0; }
```

Lessons:

- Be very careful with using “undefined variables”
- We need some way of introducing new variables...

Quantifiers

We use **quantifiers** to talk about collections of objects.

Universal Quantifier (“for all”): $\forall x P(x)$

P(x) is true for **every** x in the domain
read as “**for all x, P of x**”

Examples:

- $\forall x \text{ Odd}(x)$
- $\forall x \text{ LessThan5}(x)$

Quantifiers

We use **quantifiers** to talk about collections of objects.

Universal Quantifier (“for all”): $\forall x P(x)$

P(x) is true for **every** x in the domain
read as “**for all x, P of x**”

Examples: Are these true? It depends on the domain. For example:

- $\forall x \text{ Odd}(x)$
- $\forall x \text{ LessThan5}(x)$

{1, 3, -1, -27}	Integers	Odd Integers
True	False	True
True	False	False

Universal Quantifier ("forall") (Programmatically)

$\forall x P(x)$

```
forallP(x) {
  boolean result = true;
  for (x : DOMAIN) {
    result = result && P(x);
  }
  return result;
}
```

Quantifiers

We use **quantifiers** to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x P(x)$

There is an x in the domain for which $P(x)$ is true
read as "**there exists x , P of x** "

Examples:

- $\exists x \text{ Odd}(x)$
- $\exists x \text{ LessThan5}(x)$

Quantifiers

We use **quantifiers** to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x P(x)$

There is an x in the domain for which $P(x)$ is true
read as "**there exists x , P of x** "

Examples: Are these true? It depends on the domain. For example:

- $\exists x \text{ Odd}(x)$
- $\exists x \text{ LessThan5}(x)$

{1, 3, -1, -27}	Integers	Non-Zero Multiples of 10
True	True	False
True	True	False

Existential Quantifier ("exists") (Programmatically)

$\exists x P(x)$

```
existsP(x) {
  boolean result = false;
  for (x : DOMAIN) {
    result = result || P(x);
  }
  return result;
}
```

Statements with Quantifiers

Just like with propositional logic, we need to define variables (this time **predicates**) before we do anything else. We must also now define a **domain of discourse** before doing anything else.

Domain of Discourse
Positive Integers

Predicate Definitions
Even(x) ::= "x is even" Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd" Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime" Sum(x, y, z) ::= "x + y = z"

Statements with Quantifiers

Domain of Discourse
Positive Integers

Predicate Definitions
Even(x) ::= "x is even" Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd" Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime" Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x \exists y \text{ Greater}(y, x)$

For all pos. int. x , there exists a pos. int. y such that y is greater than x .

$\forall x \exists y \text{ Greater}(x, y)$
less than

$\forall x \exists y (\text{Greater}(y, x) \wedge \text{Prime}(y))$

$\forall x (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$

$\exists x \exists y (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$

Statements with Quantifiers (Literal Translations)

Domain of Discourse
Positive Integers

Predicate Definitions
 Even(x) ::= "x is even" Greater(x, y) ::= "x > y"
 Odd(x) ::= "x is odd" Equal(x, y) ::= "x = y"
 Prime(x) ::= "x is prime" Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

- $\forall x \exists y \text{ Greater}(y, x)$
 "For every pos. int. x, there is a pos. int. y, such that $y > x$."
 $\forall x \exists y \text{ Greater}(x, y)$
 "For every pos. int. x, there is a pos. int. y, such that $x > y$."
 $\forall x \exists y (\text{Greater}(y, x) \wedge \text{Prime}(y))$
 "For every positive integer x, there is a pos. int. y such that $y > x$ and y is prime."
 $\forall x (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$
 "For each pos. int. x, if x is prime, then $x = 2$ or x is odd."
 $\exists x \exists y (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$
 "There exist positive integers x and y such that $x + 2 = y$ and x and y are prime."

Statements with Quantifiers (Better Translations)

Domain of Discourse
Positive Integers

Predicate Definitions
 Even(x) ::= "x is even" Greater(x, y) ::= "x > y"
 Odd(x) ::= "x is odd" Equal(x, y) ::= "x = y"
 Prime(x) ::= "x is prime" Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

- $\forall x \exists y \text{ Greater}(y, x)$
 "There is no greatest integer."
 $\forall x \exists y \text{ Greater}(x, y)$
 "There is no least integer."
 $\forall x \exists y (\text{Greater}(y, x) \wedge \text{Prime}(y))$
 "There is always a prime number greater than any positive integer."
 $\forall x (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$
 "Every prime positive integer is either 2 or odd."
 $\exists x \exists y (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$
 "There exist prime positive integers that differ by two."

English to Predicate Logic

Domain of Discourse
Mammals

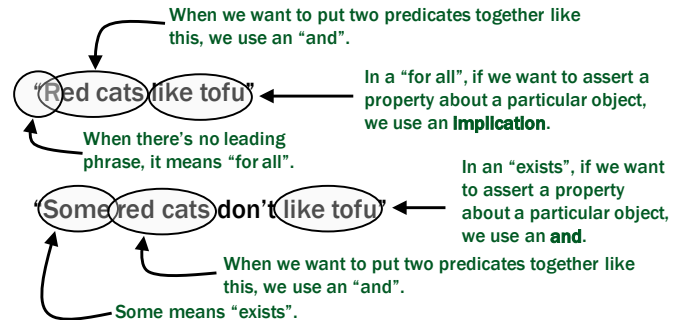
Predicate Definitions
 Cat(x) ::= "x is a cat"
 Red(x) ::= "x is red"
 LikesTofu(x) ::= "x likes tofu"

- ↓
 "Red cats like tofu"
 $\forall x ((\text{Red}(x) \wedge \text{Cat}(x)) \rightarrow \text{LikesTofu}(x))$
 "Some red cats don't like tofu"
 $\exists x ((\text{Red}(x) \wedge \text{Cat}(x)) \rightarrow \neg \text{LikesTofu}(x))$

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
 Cat(x) ::= "x is a cat"
 Red(x) ::= "x is red"
 LikesTofu(x) ::= "x likes tofu"



English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
 Cat(x) ::= "x is a cat"
 Red(x) ::= "x is red"
 LikesTofu(x) ::= "x likes tofu"

- "Red cats like tofu"
 $\forall x ((\text{Red}(x) \wedge \text{Cat}(x)) \rightarrow \text{LikesTofu}(x))$
 "Some red cats don't like tofu"
 $\exists y ((\text{Red}(y) \wedge \text{Cat}(y)) \wedge \neg \text{LikesTofu}(y))$

Negations of Quantifiers

Predicate Definitions
 PF(x) ::= "x is a purple fruit"

$\forall x \text{ PF}(x)$

Imagine our domain is {plum, banana, apple}.
 Can you write the statement without any quantifiers?

$\text{PF}(\text{plum}) \wedge \text{PF}(\text{banana}) \wedge \text{PF}(\text{apple})$

What is the negation of that statement?

$\neg(\text{PF}(\text{plum}) \wedge \text{PF}(\text{banana}) \wedge \text{PF}(\text{apple}))$
 $\equiv \neg \text{PF}(\text{plum}) \vee \neg \text{PF}(\text{banana}) \vee \neg \text{PF}(\text{apple})$

"One of the fruits is not purple"

$\exists x \neg \text{P}(x)$

Negations of Quantifiers

Predicate Definitions

$PF(x) ::=$ "x is a purple fruit"

$$\forall x PF(x)$$

Imagine our domain is {plum, banana, apple}.

Can you write the statement without any quantifiers?

What is the negation of that statement?

De Morgan's Laws for Quantifiers

$$\neg \forall x P(x) \equiv \exists x \neg P(x)$$

$$\neg \exists x P(x) \equiv \forall x \neg P(x)$$

De Morgan's Laws for Quantifiers

$$\neg \forall x P(x) \equiv \exists x \neg P(x)$$

$$\neg \exists x P(x) \equiv \forall x \neg P(x)$$

"There is no largest integer"

$$\begin{aligned} \forall x (\neg(\forall y(x \geq y))) &\equiv \forall x (\exists y(\neg(x \geq y))) \\ &\equiv \forall x (\exists y(x < y)) \end{aligned}$$

"For every integer there is a larger integer"

Negations of Quantifiers

- not every positive integer is prime
- some positive integer is not prime
- prime numbers do not exist
- every positive integer is not prime

Bound and Free Variables

Consider the following program:

```
hello(x) {  
  return x + y;  
}
```

In this program, we say "x" is **bound** and "y" is **free**.

Bound and Free Variables

Consider the following program:

```
hello(x) {  
  return x + y;  
}
```

x is defined here

x is used here

y is never defined ☹

In this program, we say "x" is **bound** and "y" is **free**.

Scope of Quantifiers

It's the same idea with quantifiers.

$$\exists y \text{ Greater}(y, x)$$

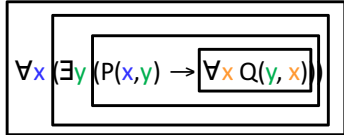
$$\forall x \exists y \text{ Greater}(y, x)$$

We figure out what a formula means "inside-out".

So, variables bind to the inner-most quantifier that tries to "capture" them.

$$\forall x \exists y \text{ Greater}(y, x)$$

This quantifier
does nothing!



Variable Renaming

$$\text{NotLargest1}(x) = \exists y \text{ Greater}(y, x) \quad \text{vs.} \quad \text{NotLargest2}(x) = \exists z \text{ Greater}(z, x)$$

```
notLargest1(x) {
  boolean result = false;
  for (y : DOMAIN) {
    result = result || y > x
  }
  return result;
}
```

```
notLargest2(x) {
  boolean result = false;
  for (z : DOMAIN) {
    result = result || z > x
  }
  return result;
}
```

These are the same program!
Variable names are irrelevant!

Scope of Quantifiers

$$\exists x (P(x) \wedge Q(x)) \quad \text{vs.} \quad \exists x P(x) \wedge \exists x Q(x)$$

Scope of Quantifiers

$$\exists x (P(x) \wedge Q(x)) \quad \text{vs.} \quad \exists x P(x) \wedge \exists x Q(x)$$

This one asserts P
and Q of the same x.

This one asserts P and Q
of potentially different x's.