CSE 311 Lecture 05: Canonical Forms and Predicate Logic

Emina Torlak and Sami Davies

Topics

Boolean algebra
: A review of Lecture 04 with another end-to-end example.

Canonical forms
: Standard forms for a Boolean expression.

Predicate logic
: Extending propositional logic with predicates and quantifiers.

Boolean algebra

A review of Lecture 04 with another end-to-end example.

Boolean algebra is a notation for combinational logic

Think of it as a notation for propositional logic used in circuit design.

Boolean algebra consists of the following elements and operations:

- a set of elements $B = \{0, 1\}$,
- binary operations $\{+, \cdot\}$,
- a unary operation $\{\,'\,\}$.

These correspond to the truth values {F, T}, and the logical connectives $\vee$, $\wedge$, $\neg$.

Boolean operations satisfy the following axioms for any $a, b, c \in B$:

Closure

$$a + b \in B$$
$$a \cdot b \in B$$

Commutativity

$$a + b = b + a$$
$$a \cdot b = b \cdot a$$

Associativity

$$a + (b + c) = (a + b) + c$$
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

Distributivity

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$
$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Identity

$$a + 0 = a$$
$$a \cdot 1 = a$$

Complementarity

$$a + a' = 1$$
$$a \cdot a' = 0$$

Null

$$a + 1 = 1$$
$$a \cdot 0 = 0$$

Idempotency

$$a + a = a$$
$$a \cdot a = a$$

Involution

$$(a')' = a$$

Example: from spec to an adder circuit

Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	0
input A	101
input B	+ 001
sum	

Example: from spec to an adder circuit

Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	10
input A	101
input B	+ 001
sum	0

Example: from spec to an adder circuit

Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	010
input A	101
input B	+ 001
sum	10

Example: from spec to an adder circuit

Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	0010
input A	101
input B	+ 001
sum	110

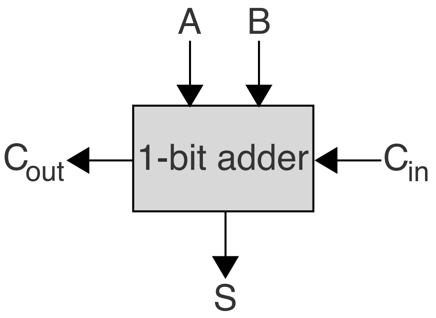
Example: from spec to an adder circuit

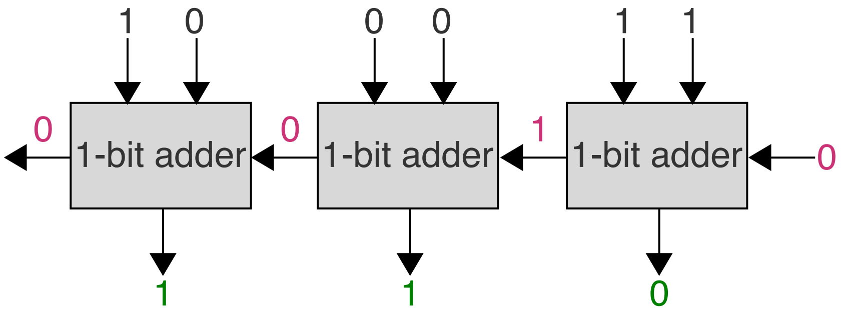
Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	0010
input A	101
input B	+ 001
sum	110

We can implement n-bit addition by chaining together n 1-bit adders:





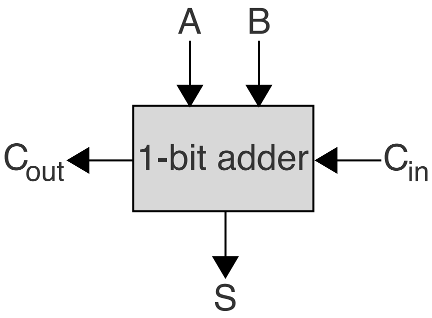
Example: from spec to an adder circuit

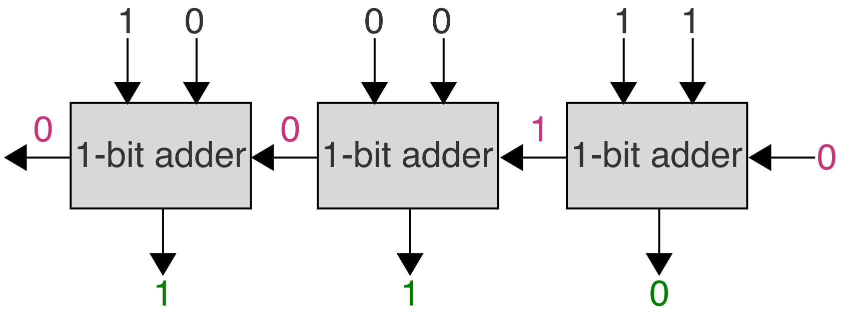
Binary addition is a basic operation implemented by every computer.

It works just like decimal addition: we add numbers digit by digit, from least to most significant, keeping track of the current sum and carry.

carry	0010
input A	101
input B	+ 001
sum	110

We can implement n-bit addition by chaining together n 1-bit adders:

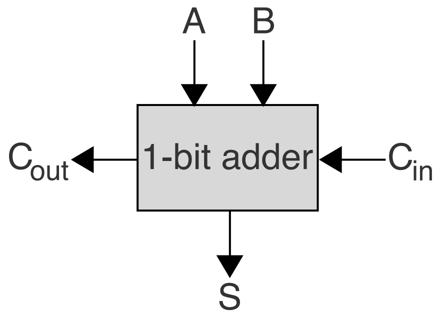




Let's implement the 1-bit adder circuit!

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



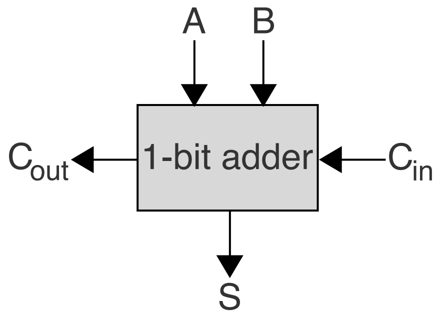
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S =$

$C_{out} =$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



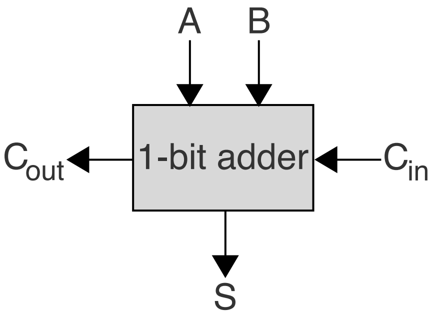
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in}$$

$$C_{out} =$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



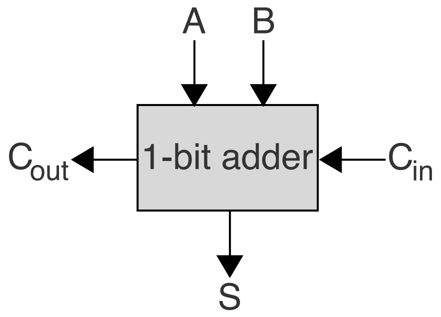
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in}$$

$$C_{out} =$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



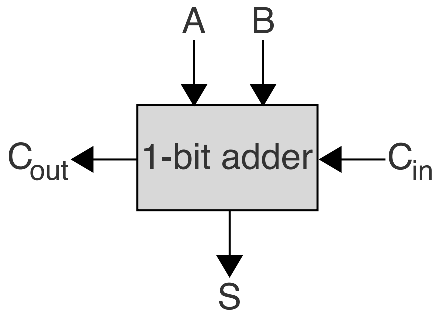
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in}$$

$$C_{out} =$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



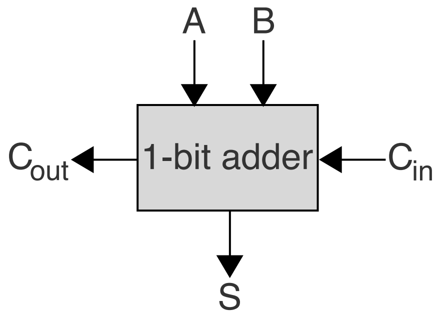
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$

$$C_{out} =$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



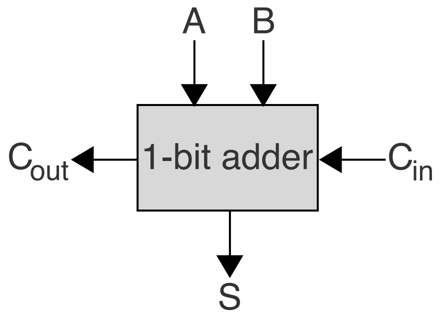
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$

$$C_{out} = A' \cdot B \cdot C_{in}$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



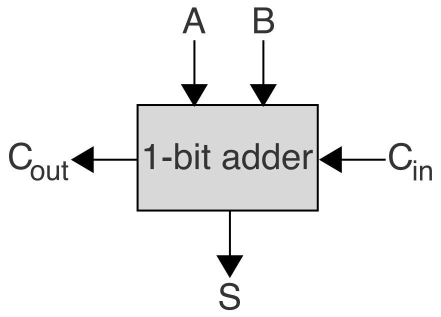
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$

$$C_{out} = A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in}$$

Example: from spec to logic via a truth table

- **Inputs:** A, B, C_{in} (input bits and carry-in)
- **Outputs:** S, C_{out} (sum and carry out)



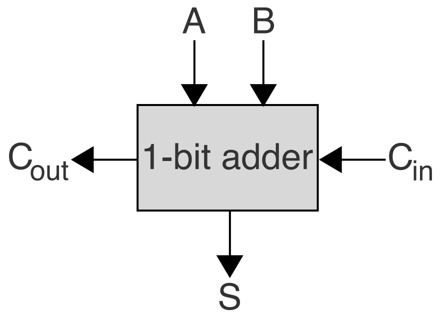
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$

$$C_{out} = A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in}$$

Example: from spec to logic via a truth table

- **Inputs**: A, B, C_{in} (input bits and carry-in)
- **Outputs**: S, C_{out} (sum and carry out)



A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$

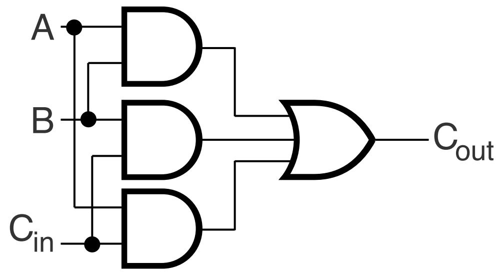
$$C_{out} = A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in}$$

Example: apply theorems to simplify the logic

$$
\begin{aligned}
C_{out} &= A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} \\
&= A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} + A \cdot B \cdot C_{in} && \text{Idempotence} \\
&= A \cdot B \cdot C_{in} + A' \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} && \text{Commutativity} \\
&= B \cdot C_{in} \cdot A + B \cdot C_{in} \cdot A' + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} && \text{Commutativity} \\
&= B \cdot C_{in} \cdot (A + A') + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} && \text{Distributivity} \\
&= B \cdot C_{in} \cdot 1 + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} && \text{Complementarity} \\
&= B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} && \text{Identity} \\
&= B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C'_{in} + A \cdot B \cdot C_{in} + A \cdot B \cdot C_{in} && \text{Idempotence} \\
&= B \cdot C_{in} + A \cdot B \cdot C_{in} + A \cdot B' \cdot C_{in} + A \cdot B \cdot C_{in} + A \cdot B \cdot C'_{in} && \text{Commutativity} \\
&= B \cdot C_{in} + A \cdot C_{in} \cdot B + A \cdot C_{in} \cdot B' + A \cdot B \cdot C_{in} + A \cdot B \cdot C'_{in} && \text{Commutativity} \\
&= B \cdot C_{in} + A \cdot C_{in} \cdot (B + B') + A \cdot B \cdot (C_{in} + C'_{in}) && \text{Distributivity} \\
&= B \cdot C_{in} + A \cdot C_{in} \cdot 1 + A \cdot B \cdot 1 && \text{Complementarity} \\
&= B \cdot C_{in} + A \cdot C_{in} + A \cdot B && \text{Identity}
\end{aligned}
$$

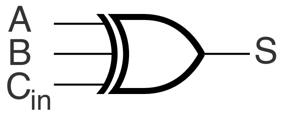
Example: map the logic to the available gates

$$C_{out} = B \cdot C_{in} + A \cdot C_{in} + A \cdot B$$



C_{out} mapped to AND, OR, and NOT gates.

$$S = A' \cdot B' \cdot C_{in} + A' \cdot B \cdot C'_{in} + A \cdot B' \cdot C'_{in} + A \cdot B \cdot C_{in}$$



S mapped to an XOR gate: $S \equiv A \oplus B \oplus C_{in}$.

Example: the recipe for translating a spec to a circuit

To translate a specification to a circuit:

1. Write the truth table (and, optionally, the program) for the spec.
2. Write the Boolean expression for the output bits.
3. Minimize the Boolean expressions for the output bits.
4. Map the minimized expressions to the available logic gates.

Example: the recipe for translating a spec to a circuit

To translate a specification to a circuit:

1. Write the truth table (and, optionally, the program) for the spec.
2. Write the Boolean expression for the output bits.
3. Minimize the Boolean expressions for the output bits.
4. Map the minimized expressions to the available logic gates.

①

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

Example: the recipe for translating a spec to a circuit

To translate a specification to a circuit:

1. Write the truth table (and, optionally, the program) for the spec.
2. Write the Boolean expression for the output bits.
3. Minimize the Boolean expressions for the output bits.
4. Map the minimized expressions to the available logic gates.

①

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

②③

$$
\begin{aligned}
F &= A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C \\
&= A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot C \cdot B' + A \cdot C \cdot B && \text{Commutativity} \\
&= A' \cdot B \cdot (C' + C) + A \cdot C \cdot (B' + B) && \text{Distributivity} \\
&= A' \cdot B \cdot (C + C') + A \cdot C \cdot (B + B') && \text{Commutativity} \\
&= A' \cdot B \cdot 1 + A \cdot C \cdot 1 && \text{Complementarity} \\
&= A' \cdot B + A \cdot C && \text{Identity}
\end{aligned}
$$

Example: the recipe for translating a spec to a circuit

To translate a specification to a circuit:

1. Write the truth table (and, optionally, the program) for the spec.
2. Write the Boolean expression for the output bits.
3. Minimize the Boolean expressions for the output bits.
4. Map the minimized expressions to the available logic gates.

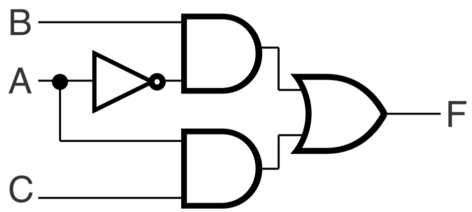
①

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

②③

$$
\begin{aligned}
F &= A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C \\
&= A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot C \cdot B' + A \cdot C \cdot B && \text{Commutativity} \\
&= A' \cdot B \cdot (C' + C) + A \cdot C \cdot (B' + B) && \text{Distributivity} \\
&= A' \cdot B \cdot (C + C') + A \cdot C \cdot (B + B') && \text{Commutativity} \\
&= A' \cdot B \cdot 1 + A \cdot C \cdot 1 && \text{Complementarity} \\
&= A' \cdot B + A \cdot C && \text{Identity}
\end{aligned}
$$

④



Canonical forms

Standard forms for a Boolean expression.

Why do we need canonical forms?

A truth table is the unique signature of a Boolean function.

- It captures the semantics (meaning) of the function.

The same truth table can have many realizations in Boolean algebra.

- One function can have many different syntactic representations.
- Depends on how good we are at Boolean simplification.

Canonical forms are standard form for a Boolean expression.

- We all come up with the same expression.
- Also used internally by theorem provers.

We will cover two useful canonical forms.

- Sum-of-products form.
- Product-of-sums form.

Sum-of-products canonical form

Also known as ...

Disjunctive Normal Form (DNF)

Minterm Expansion

To convert a truth table to sum-of-products:

① Read the rows with true (1) output.

② Convert to Boolean algebra.

③ Add the minterms together.

A	B	C	F	①	②
0	0	0	0		
0	0	1	0		
0	1	0	1	010	
0	1	1	1	011	
1	0	0	0		
1	0	1	1	101	
1	1	0	0		
1	1	1	1	111	

Sum-of-products canonical form

Also known as …

Disjunctive Normal Form (DNF)

Minterm Expansion

To convert a truth table to sum-of-products:

① Read the rows with true (1) output.

② Convert to Boolean algebra.

③ Add the minterms together.

A	B	C	F	①	②
0	0	0	0		
0	0	1	0		
0	1	0	1	010	$A'BC'$
0	1	1	1	011	$A'BC$
1	0	0	0		
1	0	1	1	101	$AB'C$
1	1	0	0		
1	1	1	1	111	ABC

Sum-of-products canonical form

Also known as ...

Disjunctive Normal Form (DNF)

Minterm Expansion

To convert a truth table to sum-of-products:

① Read the rows with true (1) output.

② Convert to Boolean algebra.

③ Add the minterms together.

A	B	C	F	①	②
0	0	0	0		
0	0	1	0		
0	1	0	1	010	$A'BC'$
0	1	1	1	011	$A'BC$
1	0	0	0		
1	0	1	1	101	$AB'C$
1	1	0	0		
1	1	1	1	111	ABC

③ $F = A'BC' + A'BC + AB'C + ABC$

Sum-of-products canonical form: properties

Product term (or minterm)

Conjunction of *literals*, which are variables or their negations.

Represents an input combination for which output is true.

Each variable appears exactly once, true or negated (but not both).

A	B	C	F	minterms
0	0	0	0	
0	0	1	0	
0	1	0	1	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	0	
1	0	1	1	$AB'C$
1	1	0	0	
1	1	1	1	ABC

F in canonical form

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

canonical form $\neq$ minimal form

$$\begin{aligned} F &= A'BC' + A'BC + AB'C + ABC \\ &= A'B(C + C') + AC(B + B') \\ &= A'B + AC \end{aligned}$$

Product-of-sums canonical form

Also known as …

Conjunctive Normal Form (CNF)

Maxterm Expansion

To convert a truth table to product-of-sums:

① Read the rows with false (0) output.

② Negate all bits.

③ Convert to Boolean algebra.

④ Multiply the maxterms together.

A	B	C	F	①	②	③
0	0	0	0	000		
0	0	1	0	001		
0	1	0	1			
0	1	1	1			
1	0	0	0	100		
1	0	1	1			
1	1	0	0	110		
1	1	1	1			

④

Product-of-sums canonical form

Also known as …

Conjunctive Normal Form (CNF)

Maxterm Expansion

To convert a truth table to product-of-sums:

① Read the rows with false (0) output.

② Negate all bits.

③ Convert to Boolean algebra.

④ Multiply the maxterms together.

A	B	C	F	①	②	③
0	0	0	0	000	111	
0	0	1	0	001	110	
0	1	0	1			
0	1	1	1			
1	0	0	0	100	011	
1	0	1	1			
1	1	0	0	110	001	
1	1	1	1			

④

Product-of-sums canonical form

Also known as …

Conjunctive Normal Form (CNF)

Maxterm Expansion

To convert a truth table to product-of-sums:

① Read the rows with false (0) output.

② Negate all bits.

③ Convert to Boolean algebra.

④ Multiply the maxterms together.

A	B	C	F	①	②	③
0	0	0	0	000	111	$A + B + C$
0	0	1	0	001	110	$A + B + C'$
0	1	0	1			
0	1	1	1			
1	0	0	0	100	011	$A' + B + C$
1	0	1	1			
1	1	0	0	110	001	$A' + B' + C$
1	1	1	1			

④

Product-of-sums canonical form

Also known as ...

Conjunctive Normal Form (CNF)

Maxterm Expansion

To convert a truth table to product-of-sums:

① Read the rows with false (0) output.

② Negate all bits.

③ Convert to Boolean algebra.

④ Multiply the maxterms together.

A	B	C	F	①	②	③
0	0	0	0	000	111	$A+B+C$
0	0	1	0	001	110	$A+B+C'$
0	1	0	1			
0	1	1	1			
1	0	0	0	100	011	$A'+B+C$
1	0	1	1			
1	1	0	0	110	001	$A'+B'+C$
1	1	1	1			

④ $$F = (A+B+C)(A+B+C')(A'+B+C)(A'+B'+C)$$

Product-of-sums canonical form: why does it work?

What we know …

$(F')' = F$ by Involution.

How to get a **minterm** expansion for F'.

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

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

Product-of-sums canonical form: why does it work?

What we know …

$(F')' = F$ by Involution.

How to get a **minterm** expansion for F'.

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

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

Taking the complement of both sides

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

Product-of-sums canonical form: why does it work?

What we know ...

$(F')' = F$ by Involution.

How to get a **minterm** expansion for F'.

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

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

Taking the complement of both sides

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

Using Involution and DeMorgan Laws

$$\begin{aligned} F &= (A'B'C')'(A'B'C)'(AB'C')'(ABC')' \\ &= (A+B+C)(A+B+C')(A'+B+C)(A'+B'+C) \end{aligned}$$

Product-of-sums canonical form: properties

Sum term (or maxterm)

Disjunction of *literals*, which are variables or their negations.
Represents an input combination for which output is false.
Each variable appears exactly once, true or negated (but not both).

A	B	C	F	maxterms
0	0	0	0	$A + B + C$
0	0	1	0	$A + B + C'$
0	1	0	1	
0	1	1	1	
1	0	0	0	$A' + B + C$
1	0	1	1	
1	1	0	0	$A' + B' + C$
1	1	1	1	

F in canonical form

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

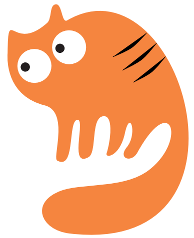
canonical form $\neq$ minimal form

$$\begin{aligned} F &= (A + B + C)(A + B + C')(A' + B + C)(A' + B' + C) \\ &= (A + B + CC')(A' + C + BB') \\ &= (A + B)(A' + C) \end{aligned}$$

Predicate logic

Extending propositional logic with predicates and quantifiers.

Predicate logic versus propositional logic



Propositional logic

“If Garfield is an orange cat and likes lasagna, then he has black stripes.”

Predicate logic

“All positive integers x, y, z satisfy $x^3 + y^3 \neq z^3$.”

Propositional logic lets us express complex propositions in terms of their constituent parts (atomic propositions) joined by connectives. Predicate logic lets us express how propositions depend on the objects they mention.

Key notions in predicate logic

Syntax

Predicate logic extends propositional logic with two key constructs: *predicates* and *quantifiers* ($\exists$, $\forall$).

Semantics

We define the meaning of formulas in predicate logic with respect to a *domain of discourse*.

Predicates

Predicate is a function that returns a truth value.

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"

Predicates can have varying arity (numbers of arguments).

Domain of discourse

To give meaning to predicates in a formula, we define a set of objects that those predicates can take as input.

This set of objects is called the *domain of discourse* for a formula.

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

“x is a cat”, “x barks”, “x ruined my couch”

“x is prime”, “x = 0”, “x < 0”, “x is a power of two”

“student x has taken course y” “x is a pre-req for z”

Domain of discourse

To give meaning to predicates in a formula, we define a set of objects that those predicates can take as input.

This set of objects is called the *domain of discourse* for a formula.

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

“x is a cat”, “x barks”, “x ruined my couch”

“mammals” or “sentient beings” or “cats and dogs” or …

“x is prime”, “$x = 0$”, “$x < 0$”, “x is a power of two”

“student x has taken course y” “x is a pre-req for z”

Domain of discourse

To give meaning to predicates in a formula, we define a set of objects that those predicates can take as input.

This set of objects is called the *domain of discourse* for a formula.

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

“x is a cat”, “x barks”, “x ruined my couch”

“mammals” or “sentient beings” or “cats and dogs” or …

“x is prime”, “x = 0”, “x < 0”, “x is a power of two”

“numbers” or “integers” or “integers greater than 5” or …

“student x has taken course y” “x is a pre-req for z”

Domain of discourse

To give meaning to predicates in a formula, we define a set of objects that those predicates can take as input.

This set of objects is called the *domain of discourse* for a formula.

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

“x is a cat”, “x barks”, “x ruined my couch”

“mammals” or “sentient beings” or “cats and dogs” or …

“x is prime”, “x = 0”, “x < 0”, “x is a power of two”

“numbers” or “integers” or “integers greater than 5” or …

“student x has taken course y” “x is a pre-req for z”

“students and courses” or “university entities” or …

Quantifiers

Quantifiers let us talk about *all* or *some* objects in the domain.

$\forall x.\ P(x)$

$P(x)$ is true **for every** x in the domain.

Read as “for all x, $P(x)$”.

Called the **universal quantifier**.

$\exists x.\ P(x)$

There is an x in the domain for which $P(x)$ is true.

Read as “there exists x, $P(x)$”.

Called the **existential quantifier**.

Universal quantifier ∀

$\forall x.\ P(x)$

$P(x)$ is true **for every** x in the domain.

Examples: are these true?

$\forall x.\ \mathrm{Odd}(x)$

$\forall x.\ \mathrm{LessThan5}(x)$

Universal quantifier $\forall$

$\forall x.\, P(x)$

$P(x)$ is true **for every** x in the domain.

Examples: are these true?

$\forall x.\, \mathrm{Odd}(x)$

$\forall x.\, \mathrm{LessThan5}(x)$

Depends on the domain.

	$\{-3, 3\}$	Integers	Odd Integers
$\forall x.\, \mathrm{Odd}(x)$	True	False	True
$\forall x.\, \mathrm{LessThan5}(x)$	True	False	False

Universal quantifier ∀

$\forall x.\ P(x)$

$P(x)$ is true **for every** x in the domain.

Examples: are these true?

$\forall x.\ \mathrm{Odd}(x)$

$\forall x.\ \mathrm{LessThan5}(x)$

Depends on the domain.

	$\{-3, 3\}$	Integers	Odd Integers
$\forall x.\ \mathrm{Odd}(x)$	True	False	True
$\forall x.\ \mathrm{LessThan5}(x)$	True	False	False

You can think of $\forall x.\ P(x)$ as conjunction over all objects in the domain.

- $\forall x.\ \mathrm{Odd}(x)$
  - over $\{-3, 3\}$ is the conjunction $\mathrm{Odd}(-3) \wedge \mathrm{Odd}(3)$
  - over integers is the infinite conjunction $\ldots \wedge \mathrm{Odd}(-1) \wedge \mathrm{Odd}(0) \wedge \mathrm{Odd}(1) \wedge \ldots$

Existential quantifier $\exists$

$\exists x.\ P(x)$

There is an x in the domain for which $P(x)$ is true.

Examples: are these true?

$\exists x.\ \mathrm{Odd}(x)$

$\exists x.\ \mathrm{LessThan5}(x)$

Existential quantifier $\exists$

$\exists x.\ P(x)$

There is an x in the domain for which $P(x)$ is true.

Examples: are these true?

$\exists x.\ \mathrm{Odd}(x)$

$\exists x.\ \mathrm{LessThan5}(x)$

Depends on the domain.

	$\{-3, 3\}$	Integers	Positive Multiples of 5
$\exists x.\ \mathrm{Odd}(x)$	True	True	True
$\exists x.\ \mathrm{LessThan5}(x)$	True	True	False

Existential quantifier $\exists$

$\exists x.\, P(x)$

There is an x in the domain for which $P(x)$ is true.

Examples: are these true?

$\exists x.\, \mathrm{Odd}(x)$

$\exists x.\, \mathrm{LessThan5}(x)$

Depends on the domain.

	$\{-3, 3\}$	Integers	Positive Multiples of 5
$\exists x.\, \mathrm{Odd}(x)$	True	True	True
$\exists x.\, \mathrm{LessThan5}(x)$	True	True	False

You can think of $\exists x.\, P(x)$ as disjunction over all objects in the domain.

- $\exists x.\, \mathrm{Odd}(x)$
  - over $\{-3, 3\}$ is the disjunction $\mathrm{Odd}(-3) \vee \mathrm{Odd}(3)$
  - over integers is the infinite disjunction $\ldots \vee \mathrm{Odd}(-1) \vee \mathrm{Odd}(0) \vee \mathrm{Odd}(1) \vee \ldots$

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\ \text{Even}(x)$
$\forall x.\ \text{Odd}(x)$
$\forall x.\ \text{Even}(x) \vee \text{Odd}(x)$
$\exists x.\ \text{Even}(x) \wedge \text{Odd}(x)$
$\forall x.\ \text{Greater}(x + 1, x)$
$\exists x.\ \text{Even}(x) \wedge \text{Prime}(x)$

Domain of discourse

Positive integers

Predicate definitions

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

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\, \mathrm{Even}(x)$ T

$\forall x.\, \mathrm{Odd}(x)$

$\forall x.\, \mathrm{Even}(x) \vee \mathrm{Odd}(x)$

$\exists x.\, \mathrm{Even}(x) \wedge \mathrm{Odd}(x)$

$\forall x.\, \mathrm{Greater}(x + 1, x)$

$\exists x.\, \mathrm{Even}(x) \wedge \mathrm{Prime}(x)$

Domain of discourse

Positive integers

Predicate definitions

Even(x) := “x is even”

Odd(x) := “x is odd”

Prime(x) := “x is prime”

Greater(x, y) := “x > y”

Equal(x, y) := “x = y”

Sum(x, y, z) := “z = x + y”

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\ \text{Even}(x)$	T
$\forall x.\ \text{Odd}(x)$	F
$\forall x.\ \text{Even}(x) \lor \text{Odd}(x)$	
$\exists x.\ \text{Even}(x) \land \text{Odd}(x)$	
$\forall x.\ \text{Greater}(x+1, x)$	
$\exists x.\ \text{Even}(x) \land \text{Prime}(x)$	

Domain of discourse

Positive integers

Predicate definitions

Even(x) := “x is even”

Odd(x) := “x is odd”

Prime(x) := “x is prime”

Greater(x, y) := “x > y”

Equal(x, y) := “x = y”

Sum(x, y, z) := “z = x + y”

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\, \text{Even}(x)$	T
$\forall x.\, \text{Odd}(x)$	F
$\forall x.\, \text{Even}(x) \lor \text{Odd}(x)$	T
$\exists x.\, \text{Even}(x) \land \text{Odd}(x)$	
$\forall x.\, \text{Greater}(x+1, x)$	
$\exists x.\, \text{Even}(x) \land \text{Prime}(x)$	

Domain of discourse

Positive integers

Predicate definitions

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

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\ \mathrm{Even}(x)$	T
$\forall x.\ \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Even}(x) \vee \mathrm{Odd}(x)$	T
$\exists x.\ \mathrm{Even}(x) \wedge \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Greater}(x+1, x)$	
$\exists x.\ \mathrm{Even}(x) \wedge \mathrm{Prime}(x)$	

Domain of discourse

Positive integers

Predicate definitions

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

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\ \mathrm{Even}(x)$	T
$\forall x.\ \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Even}(x) \lor \mathrm{Odd}(x)$	T
$\exists x.\ \mathrm{Even}(x) \land \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Greater}(x+1, x)$	T
$\exists x.\ \mathrm{Even}(x) \land \mathrm{Prime}(x)$	

Domain of discourse

Positive integers

Predicate definitions

Even(x) := "x is even"

Odd(x) := "x is odd"

Prime(x) := "x is prime"

Greater(x, y) := "x > y"

Equal(x, y) := "x = y"

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

Statements with quantifiers

Just like with propositional logic, we need to define variables (this time predicates). And we must also now define a domain of discourse.

What is the truth value of these statements?

$\exists x.\ \mathrm{Even}(x)$	T
$\forall x.\ \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Even}(x) \vee \mathrm{Odd}(x)$	T
$\exists x.\ \mathrm{Even}(x) \wedge \mathrm{Odd}(x)$	F
$\forall x.\ \mathrm{Greater}(x+1, x)$	T
$\exists x.\ \mathrm{Even}(x) \wedge \mathrm{Prime}(x)$	T

Domain of discourse

Positive integers

Predicate definitions

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

Predicate logic to English

Translate the following statements to English

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

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

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

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

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

Domain of discourse

Positive integers

Predicate definitions

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

Predicate logic to English

Translate the following statements to English

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

For every positive integer x, there is a positive integer y, such that $y > x$.

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

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

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

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

Domain of discourse

Positive integers

Predicate definitions

Even(x) := “x is even”

Odd(x) := “x is odd”

Prime(x) := “x is prime”

Greater(x, y) := “x > y”

Equal(x, y) := “x = y”

Sum(x, y, z) := “z = x + y”

Predicate logic to English

Translate the following statements to English

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

For every positive integer x, there is a positive integer y, such that $y > x$.

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

For every positive integer x, there is a positive integer y, such that $x > y$.

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

$\forall x.\ \text{Prime}(x) \to (\text{Equal}(x, 2) \lor \text{Odd}(x))$

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

Domain of discourse

Positive integers

Predicate definitions

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

Predicate logic to English

Translate the following statements to English

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

For every positive integer x, there is a positive integer y, such that $y > x$.

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

For every positive integer x, there is a positive integer y, such that $x > y$.

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

For every positive integer x, there is a positive integer y, such that $y > x$ and y is prime.

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

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

Domain of discourse

Positive integers

Predicate definitions

Even(x) := "x is even"

Odd(x) := "x is odd"

Prime(x) := "x is prime"

Greater(x, y) := "x > y"

Equal(x, y) := "x = y"

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

Predicate logic to English

Translate the following statements to English

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

For every positive integer x, there is a positive integer y, such that $y > x$.

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

For every positive integer x, there is a positive integer 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 positive integer y, such that $y > x$ and y is prime.

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

For every positive integer x, if x is prime then $x = 2$ or x is odd.

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

Domain of discourse

Positive integers

Predicate definitions

Even(x) := "x is even"

Odd(x) := "x is odd"

Prime(x) := "x is prime"

Greater(x, y) := "x > y"

Equal(x, y) := "x = y"

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

Predicate logic to English

Translate the following statements to English

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

For every positive integer x, there is a positive integer y, such that $y > x$.

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

For every positive integer x, there is a positive integer y, such that $x > y$.

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

For every positive integer x, there is a positive integer y, such that $y > x$ and y is prime.

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

For every positive integer x, if x is prime then $x = 2$ or x is odd.

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

There exist positive integers x and y such that $x + 2 = y$ and x and y are prime.

Domain of discourse

Positive integers

Predicate definitions

Even(x) := "x is even"

Odd(x) := "x is odd"

Prime(x) := "x is prime"

Greater(x, y) := "x > y"

Equal(x, y) := "x = y"

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

Predicate logic to English: natural translations

Translate the following statements to English

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

There is no greatest positive integer.

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

There is no least positive integer.

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

For every positive integer there is a larger number that is prime.

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

Every prime number is 2 or odd.

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

There exist prime numbers that differ by two.

Domain of discourse

Positive integers

Predicate definitions

Even(x) := "x is even"

Odd(x) := "x is odd"

Prime(x) := "x is prime"

Greater(x, y) := "x > y"

Equal(x, y) := "x = y"

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

English to predicate logic

"Orange cats like lasagna."

"Some orange cats don't like lasagna."

Domain of discourse

Mammals

Predicate definitions

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

Orange(x) := "x is orange"

LikesLasagna(x) := "x likes lasagna"

English to predicate logic

"Orange cats like lasagna."

$$\forall x.\ ((\text{Orange}(x) \land \text{Cat}(x)) \rightarrow \text{LikesLasagna}(x))$$

"Some orange cats don't like lasagna."

Domain of discourse

Mammals

Predicate definitions

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

Orange(x) := "x is orange"

LikesLasagna(x) := "x likes lasagna"

English to predicate logic

"Orange cats like lasagna."

$$\forall x.\ ((\mathrm{Orange}(x) \land \mathrm{Cat}(x)) \to \mathrm{LikesLasagna}(x))$$

"Some orange cats don't like lasagna."

$$\exists x.\ ((\mathrm{Orange}(x) \land \mathrm{Cat}(x)) \land \neg\mathrm{LikesLasagna}(x))$$

Domain of discourse

Mammals

Predicate definitions

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

Orange(x) := "x is orange"

LikesLasagna(x) := "x likes lasagna"

English to predicate logic: translation hints

"Orange cats like lasagna."

$\forall x. ((\text{Orange}(x) \land \text{Cat}(x)) \to \text{LikesLasagna}(x))$

When there's no leading quantification, it means "for all".

When restricting to a smaller domain in a "for all", use implication.

"Some orange cats don't like lasagna."

$\exists x. ((\text{Orange}(x) \land \text{Cat}(x)) \land \neg\text{LikesLasagna}(x))$

"Some" means "there exists".

When restricting to a smaller domain in an "exists", use conjunction.

When putting predicates together, like orange cats, use conjunction.

Domain of discourse

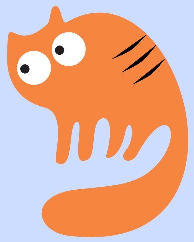
Mammals

Predicate definitions

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

Orange(x) := "x is orange"

LikesLasagna(x) := "x likes lasagna"



Summary

Canonical forms are standard form for a Boolean expression.

- Sum-of-products form.
- Product-of-sums form.

Predicate logic adds predicates and quantifiers to propositional logic.

- Predicate is a function that returns a truth value.
- Quantifiers let us talk about *all* ($\forall$) or *some* ($\exists$) objects in the domain.
- The domain of discourse is the set of objects over which the predicates and quantifiers in a formula are evaluated.