Logic

Dr. Isaac Griffith Idaho State University

Outline

Idaho State University | Computer Science

After today's lecture you will:

-

Proof Checking by Computer

CS 1187

Proof Checking

- **Proof Checker** – a program that reads in a theorem and a proof, and determines whether the proof is valid and actually establishes the theorem
  - *Advantage* – can **always** determine whether a purported proof is valid or invalid. Useful to verify hand checked theorems.

- **Theorem Prover** – a program that reads a theorem and attempts to generate a proof from scratch.
  - *Advantage* – can sometimes save the user a lot of work
  - *Disadvantage* – does not always succeed

- **Proof Tools** – any computer software that helps with formal proofs, this includes both proof checkers and theorem provers.
  - Active area of research, but is gaining practical use
  - Most industrial strength poof tools are implemented in functional languages like Haskell

Example

Theorem: $\vdash Q \to ((P \wedge Q) \to (R \wedge Q))$

Proof Tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\cancel{P \wedge R}}{R}\{\wedge E_R\} \quad \cancel{Q}}{R \wedge Q}\{\wedge I}}{(P \wedge R) \to (R \wedge Q)}\{\to I\}}{Q \to ((P \wedge R) \to (R \wedge Q))}\{\to I\}$$

Code:

```
proof1 :: Proof
proof1 =
  ImplI
    (ImpI
      (AndI
        ((AndER
            (Assume (And P R))
            R),
          Assume Q)
        (And R Q))
      (Imp (And P R) (And R Q)))
    (Imp Q (Imp (And P R) (And R Q)))
```

Representing WFFs

- WFFs can be modeled in Haskell as an Algebraic Data Type
  - Boolean constants are represented by `TRUE` and `FALSE`
  - Propositional variables are represented as upper-case letters (except `T` and `F`)
  - We can make a string a propositional variable using `Pvar "name"`
  - Additional types are for the different operators

```
data Prop
  = FALSE
  | TRUE
  | A | B | C | D | E | G | H | I | J | K | L | M
  | N | O | P | Q | R | S | U | V | W | X | Y | Z
  | Pvar String
  | And Prop Prop
  | Or Prop Prop
  | Not Prop
  | Imp Prop Prop
  | Equ Prop Prop
  deriving (Eq, Show)
```

Exercises

- Load `Stdm` and use it to define each of the following in Haskell using the `Prop` type
  - $Q \vee \text{False}$
  - $Q \rightarrow (P \rightarrow (P \vee Q))$

- Translate the following from Haskell into logical notation:
  - `Imp (Not P) (Or R S)`
  - `Equ (Imp P Q) (Or (Not P) Q)`

Exercises

- Load `Stdm` and use it to define each of the following in Haskell using the `Prop` type
  - $Q \vee \text{False} \Rightarrow$ `Or Q FALSE`
  - $Q \rightarrow (P \rightarrow (P \vee Q)) \Rightarrow$ `Imp Q (Imp P (Or P Q))`

- Translate the following from Haskell into logical notation:
  - `Imp (Not P) (Or R S)` $\Rightarrow \neg P \rightarrow (R \vee S)$
  - `Equ (Imp P Q) (Or (Not P) Q)` $\Rightarrow (P \rightarrow Q) \leftrightarrow (\neg P \vee Q)$

Representing Proofs

Idaho State University | Computer Science

- Just like with WFFs we can also model a proof as a Haskell algebraic data type
- A Proof contains both a proposition and a formal argument for the truth of that proposition
- We then have the following (1st Arg are the assumptions, 2nd the conclusion)
  - `Assume Prop`
  - `AndI (Proof, Proof) Prop`
  - `AndEL Proof Prop`
  - `AndER Proof Prop`
  - `OrIL Proof Prop`
  - `OrIR Proof Prop`
  - `OrE (Proof, Proof, Proof) Prop`
  - `ImpI Proof Prop`
  - `ImpE (Proof Proof) Prop`
  - `ID Proof Prop`
  - `CTR Proof Prop`
  - `RAA Proof Prop`

Representing Proofs

- We can also represent a theorem as a list of assumptions and a single conclusion

```
data Theorem = Theorem [Prop] Prop
```

- **Example:** Represent Theorem: $a_1, a_2, a_3 \vdash c$

```
Theorem [a1, a2, a3] c
```

- All of these data types and the means to check them are provided in the `Stdm` module

Equational Reasoning

CS 1187

Boolean Algebra

- Takes an *axiomatic* approach to logic based on mathematical reasoning
- Originally developed by George Boole
- A form of *equational reasoning* embodying two crucial ideas:
  1. We show two values are the same by building up *chains of equalities*
  2. We can *substitute equals for equals* in order to add a new link to the chain

Boolean Algebra

- A *chain of equalities* relies on the fact that if we know $a = b$ and $b = c$, then we can deduce $a = c$
- *substituting equals for equals* means that if we know $x = y$ and we have a big expression containing x, we can replace x with y without changing the value of the expression.
  - **Example:**

$$\begin{aligned} x &= 2 + p \\ y &= 5x + 3 \end{aligned}$$

$$y = 5(2 + p) + 3$$

  - **Note:** You should remember to apply parentheses to ensure substitutions act as a single value.
- It is good practice to provide justification for each substitution in the chain

Laws of Boolean Algebra

- **Laws:** set of equations that describe the basic algebraic properties of propositions
  - A law is a proposition that is always true for all assignments to its variables

- The essence of Boolean Algebra is:
  - We can use equations to state axioms on a set of operators
  - We can then use equational reasoning to explore the resulting system

Operations w/ Constants

- These laws describe how $\wedge$ and $\vee$ interact with the Boolean constants True and False

Laws:

$$
\begin{array}{rcll}
a \wedge \text{False} & = & \text{False} & \{\wedge \text{ null}\} \\
a \vee \text{True} & = & \text{True} & \{\vee \text{ null}\} \\
a \wedge \text{True} & = & a & \{\wedge \text{ identity}\} \\
a \vee \text{False} & = & a & \{\vee \text{ identity}\}
\end{array}
$$

Example:

Proof: $(P \wedge \text{True}) \vee \text{False} = P$

$$
\begin{array}{ll}
(P \wedge \text{True}) \vee \text{False} & \\
= P \wedge \text{True} & \{\vee \text{ identity}\} \\
= P & \{\wedge \text{ identity}\}
\end{array}
$$

Exercises

- Simplify $(P \wedge \text{False}) \vee (Q \wedge \text{True})$
- Prove the Equation: $(P \wedge \text{False}) \wedge \text{True} = \text{False}$

Exercises

- Simplify $(P \wedge \text{False}) \vee (Q \wedge \text{True})$

$$
\begin{aligned}
&(P \wedge \text{False}) \vee (Q \wedge \text{True}) \\
&= \text{False} \vee (Q \wedge \text{True}) && \{\wedge \text{ null}\} \\
&= \text{False} \vee Q && \{\vee \text{ identity}\} \\
&= Q && \{\vee \text{ null}\}
\end{aligned}
$$

- Prove the Equation: $(P \wedge \text{False}) \wedge \text{True} = \text{False}$

$$
\begin{aligned}
&(P \wedge \text{False}) \wedge \text{True} \\
&= \text{False} \wedge \text{True} && \{\wedge\text{null}\} \\
&= \text{False} && \{\wedge\text{identity}\}
\end{aligned}
$$

Basic Properties of $\wedge$ and $\vee$

Idaho State University | Computer Science

- These laws describe the basic properties of the $\wedge$ and $\vee$ operations
  - **idempotent** - property which allows us to reduce expressions like $a \wedge a \wedge a$ to a
  - **commutative** - property in which the order of operands can be reversed
  - **associative** - property wherein the grouping of parentheses may be changed without changing the meaning

Laws:

$$
\begin{array}{rcll}
a & \rightarrow & a \vee b & \{\text{disjunctive implication}\} \\
a \wedge b & \rightarrow & a & \{\text{conjunctive implication}\} \\
a \wedge a & = & a & \{\wedge \text{ idempotent}\} \\
a \vee a & = & a & \{\vee \text{ idempotent}\} \\
a \wedge b & = & b \wedge a & \{\wedge \text{ commutative}\} \\
a \vee b & = & b \vee a & \{\vee \text{ commutative}\} \\
(a \wedge b) \wedge c & = & a \wedge (b \wedge c) & \{\wedge \text{ associative}\} \\
(a \vee b) \vee c & = & a \vee (b \vee c) & \{\vee \text{ associative}\}
\end{array}
$$

Example: Prove $(\text{False} \wedge P) \vee Q = Q$

$$
\begin{array}{ll}
(\text{False} \wedge P) \vee Q & \\
= (P \wedge \text{False}) \vee Q & \{\wedge \text{ commutative}\} \\
= \text{False} \vee Q & \{\wedge \text{ null}\} \\
= Q \vee \text{False} & \{\vee \text{ commutative}\} \\
= Q & \{\vee \text{ identity}\}
\end{array}
$$

Exercises

- Prove: $(P \wedge ((Q \vee R) \vee Q)) \wedge S = S \wedge ((R \vee Q) \wedge P)$
- Prove: $P \wedge (Q \wedge (R \wedge S)) = ((P \wedge Q) \wedge R) \wedge S$

Exercises

- Prove: $(P \wedge ((Q \vee R) \vee Q)) \wedge S = S \wedge ((R \vee Q) \wedge P)$

$$
\begin{aligned}
&(P \wedge ((Q \vee R) \vee Q)) \wedge S = S \wedge ((R \vee Q) \wedge P) \\
&= (P \wedge ((R \vee Q) \vee Q)) \wedge S && \{\vee\text{commutative}\} \\
&= (P \wedge (R \vee (Q \vee Q))) \wedge S && \{\vee\text{associative}\} \\
&= (P \wedge (R \vee Q)) \wedge S && \{\vee\text{idempotent}\} \\
&= ((R \vee Q) \wedge P) \wedge S && \{\wedge\text{associative}\} \\
&= S \wedge ((R \vee Q) \wedge P) && \{\wedge\text{associative}\}
\end{aligned}
$$

- Prove: $P \wedge (Q \wedge (R \wedge S)) = ((P \wedge Q) \wedge R) \wedge S$

$$
\begin{aligned}
&P \wedge (Q \wedge (R \wedge S)) = ((P \wedge Q) \wedge R) \wedge S \\
&= (P \wedge Q) \wedge (R \wedge S) && \{\wedge\text{associative}\} \\
&= ((P \wedge Q) \wedge R) \wedge S && \{\wedge\text{associative}\}
\end{aligned}
$$

Distributive and DeMorgan's Laws

- These laws describe important properties of expressions containing both the $\vee$ and $\wedge$ operators

Laws:

$$
\begin{array}{rcll}
a \wedge (b \vee c) & = & (a \wedge b) \vee (a \wedge c) & \{\wedge \text{ distributes over } \vee\} \\
a \vee (b \wedge c) & = & (a \vee b) \wedge (a \vee c) & \{\vee \text{ distributes over } \wedge\} \\
\neg(a \wedge b) & = & \neg a \vee \neg b & \{\text{DeMorgan's law}\} \\
\neg(a \vee b) & = & \neg a \wedge \neg b & \{\text{DeMorgan's law}\}
\end{array}
$$

Laws of Negation

- These laws state the basic properties of negation

Laws:

$$
\begin{array}{rcll}
\neg \text{True} & = & \text{False} & \{\text{negate True}\} \\
\neg \text{False} & = & \text{True} & \{\text{negate False}\} \\
a \wedge \neg a & = & \text{False} & \{\wedge \text{ complement}\} \\
a \vee \neg a & = & \text{True} & \{\vee \text{ complement}\} \\
\neg(\neg a) & = & a & \{\text{double negation}\}
\end{array}
$$

Example: Simplify $P \wedge \neg(Q \vee P)$

$$
\begin{array}{ll}
P \wedge \neg(Q \vee P) & \\
= P \wedge (\neg Q \wedge \neg P) & \{\text{DeMorgan's law}\} \\
= P \wedge (\neg P \wedge \neg Q) & \{\wedge \text{ commutative}\} \\
= (P \wedge \neg P) \wedge \neg Q & \{\wedge \text{ associative}\} \\
= \text{False} \ \wedge \neg Q & \{\wedge \text{ complement}\} \\
= \neg Q \wedge \ \text{False} & \{\wedge \text{ commutative}\} \\
= \text{False} & \{\wedge \text{ null}\}
\end{array}
$$

Laws of Implication

- These laws are frequently used in solving problems

Laws:

$$
\begin{array}{rcll}
a \wedge (a \rightarrow b) & \rightarrow & b & \{\text{Modus Ponens}\} \\
(a \rightarrow b) \wedge \neg b & \rightarrow & \neg a & \{\text{Modus Tollens}\} \\
(a \vee b) \wedge \neg a & \rightarrow & b & \{\text{disjunctive syllogism}\} \\
(a \rightarrow b) \wedge (b \rightarrow c) & \rightarrow & a \rightarrow c & \{\text{implication chain}\} \\
(a \rightarrow b) \wedge (c \rightarrow d) & \rightarrow & (a \wedge c) \rightarrow (b \wedge d) & \{\text{implication combination}\} \\
(a \wedge b) \rightarrow c & = & a \rightarrow (b \rightarrow c) & \{\text{Currying}\} \\
a \rightarrow b & = & \neg a \vee b & \{\text{implication}\} \\
a \rightarrow b & = & \neg b \rightarrow \neg a & \{\text{contrapositive}\} \\
(a \rightarrow b) \wedge (a \rightarrow \neg b) & = & \neg a & \{\text{absurdity}\}
\end{array}
$$

- Some laws are implications and some are equations
- The implications only work in the direction of the arrow and cannot be applied in the opposite direction.

Equivalence

- Equivalence is no necessarily needed, as shown in the following law:

$$a \leftrightarrow b \quad = \quad (a \rightarrow b) \wedge (b \rightarrow a) \quad \{\text{equivalence}\}$$

- Equivalence is similar to equality, but there is a subtle difference
  - $P \wedge P \leftrightarrow P$ is a proposition which has the value of **True**
  - $P \wedge P = P$ is an equation not a proposition
    - However, both the left- and right-hand sides of the equation are propositions

- An equation is a statement in the *meta-language*
  - It is a statement *about* propositions expressed using the *object language*

Logic in Computer Science

CS 1187

Formal Correctness of Software

- We can use English to Logic translation, in the specification of hardware and software systems
- We take natural language requirements and translate them to precise and unambiguous specifications
- These specifications can then be used as the basis for system development

Formal Correctness of Software

- We can use English to Logic translation, in the specification of hardware and software systems
- We take natural language requirements and translate them to precise and unambiguous specifications
- These specifications can then be used as the basis for system development
- **Example:**
  - Express the specification “The automated reply cannot be sent when the file system is full”
    - *p*: “The automated reply can be sent”
    - *q*: “The file system is full”

Formal Correctness of Software

- We can use English to Logic translation, in the specification of hardware and software systems
- We take natural language requirements and translate them to precise and unambiguous specifications
- These specifications can then be used as the basis for system development
- **Example:**
  - Express the specification “The automated reply cannot be sent when the file system is full”
    - p: “The automated reply can be sent”
    - q: “The file system is full”
    - Specification - $q \rightarrow \neg p$

Isomorphism and Type Systems

- Modern programming languages (especially functional ones) have very powerful and expressive type systems
- The rules of logic and those for type systems are deeply connected
- Thus, we need effective methods to deal with type systems for
  - Programmers
  - Compiler writers
  - Language designers

Translating English Sentences

- English (like other human languages) tends to be ambiguous
- Hence, translating to propositional logic removes this ambiguity
  - However, this does involve making certain assumptions
- Once translated, we must assign truth values to reason about the propositions

Boolean Searches

- **Boolean Searches** - Logical connectives are extensively used in information searches (such as Google)

Boolean Searches

- **Boolean Searches** - Logical connectives are extensively used in information searches (such as Google)

- **Example queries:**
  - “New Mexico” AND Universities
  - (New AND Mexico OR Arizona) AND Universities
  - (Mexico AND Universities) NOT New

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware
- **Logic Circuit** - receives input signals $p_1, p_2, \ldots, p_n$, each a bit and produces output signals $s_1, s_2, \ldots, s_n$, also each a bit.

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware
- **Logic Circuit** - receives input signals $p_1, p_2, \ldots, p_n$, each a bit and produces output signals $s_1, s_2, \ldots, s_n$, also each a bit.
- **Gates** - the basic circuits used to construct more complex logic circuits. The following are the basic gates:

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware
- **Logic Circuit** - receives input signals $p_1, p_2, \ldots, p_n$, each a bit and produces output signals $s_1, s_2, \ldots, s_n$, also each a bit.
- **Gates** - the basic circuits used to construct more complex logic circuits. The following are the basic gates:

Inverter

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware
- **Logic Circuit** - receives input signals $p_1, p_2, \ldots, p_n$, each a bit and produces output signals $s_1, s_2, \ldots, s_n$, also each a bit.
- **Gates** - the basic circuits used to construct more complex logic circuits. The following are the basic gates:

Inverter

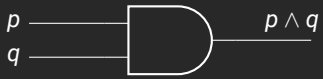
OR Gate

Digital Hardware Design

- Propositional logic can be applied to the design of computer hardware
- **Logic Circuit** - receives input signals $p_1, p_2, \ldots, p_n$, each a bit and produces output signals $s_1, s_2, \ldots, s_n$, also each a bit.
- **Gates** - the basic circuits used to construct more complex logic circuits. The following are the basic gates:

Inverter

OR Gate



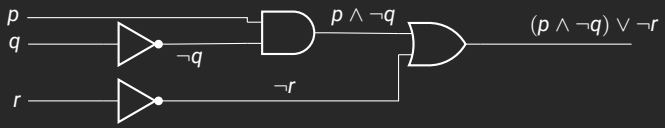
AND Gate

Logic Circuit Example

- Logic Circuit for $(p \wedge \neg q) \vee \neg r$

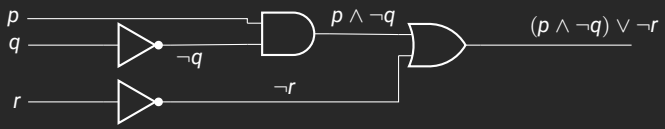
Logic Circuit Example

- Logic Circuit for $(p \wedge \neg q) \vee \neg r$



Logic Circuit Example

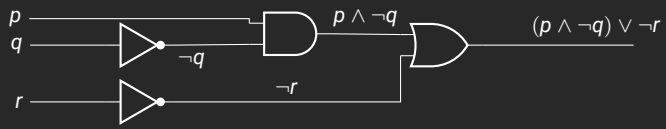
- Logic Circuit for $(p \wedge \neg q) \vee \neg r$



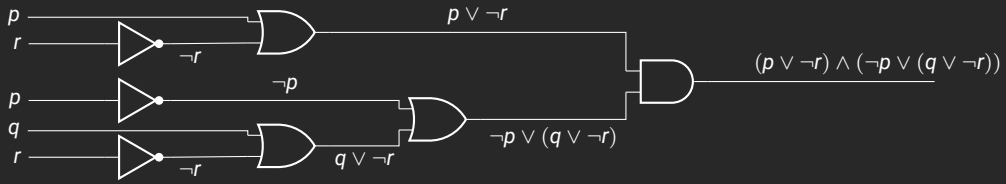
- Logic Circuit for $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$

Logic Circuit Example

- Logic Circuit for $(p \wedge \neg q) \vee \neg r$



- Logic Circuit for $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$



Exercises

Idaho State University | Computer Science

Translate the following statements into propositional logic using the propositions provided:

1. You cannot edit a protected Wikipedia entry unless you are an administrator.
   - *e*: "You can edit a protected Wikipedia entry"
   - *a*: "You are an administrator"

Exercises

Idaho State University | Computer Science

Translate the following statements into propositional logic using the propositions provided:

1. You cannot edit a protected Wikipedia entry unless you are an administrator.
   - e: "You can edit a protected Wikipedia entry"
   - a: "You are an administrator"

   Solution: $\neg a \rightarrow \neg e$

Exercises

Translate the following statements into propositional logic using the propositions provided:

1. You cannot edit a protected Wikipedia entry unless you are an administrator.
   - e: "You can edit a protected Wikipedia entry"
   - a: "You are an administrator"

   Solution: $\neg a \rightarrow \neg e$

2. You can graduate only if you have completed the requirements of your major and you do not owe money to the university and you do not have an overdue library book.
   - g: "You can graduate"
   - m: "You owe money to the university"
   - r: "You have completed the requirements of your major"
   - b: "You have an overdue library book"

Exercises

Translate the following statements into propositional logic using the propositions provided:

1. You cannot edit a protected Wikipedia entry unless you are an administrator.
   - e: "You can edit a protected Wikipedia entry"
   - a: "You are an administrator"

   Solution: $\neg a \rightarrow \neg e$

2. You can graduate only if you have completed the requirements of your major and you do not owe money to the university and you do not have an overdue library book.
   - g: "You can graduate"
   - m: "You owe money to the university"
   - r: "You have completed the requirements of your major"
   - b: "You have an overdue library book"

   Solution: $g \rightarrow (r \wedge \neg m \wedge \neg b)$

Exercises

Idaho State University | Computer Science

Express these specifications using the following propositions:

- *p*: “The user enters a valid password”
- *q*: “Access is granted”
- *r*: “The user has paid the subscription fee”

1. “The user has paid the subscription fee, but does not enter a valid password”

Exercises

Idaho State University | Computer Science

Express these specifications using the following propositions:

- p: “The user enters a valid password”
- q: “Access is granted”
- r: “The user has paid the subscription fee”

1. “The user has paid the subscription fee, but does not enter a valid password”

   Solution: $r \wedge \neg p$

Exercises

Idaho State University | Computer Science

Express these specifications using the following propositions:

- p: "The user enters a valid password"
- q: "Access is granted"
- r: "The user has paid the subscription fee"

1. "The user has paid the subscription fee, but does not enter a valid password"

   Solution: $r \wedge \neg p$

2. "If the user has not entered a valid password but has paid the subscription fee, then access is granted"

Exercises

Idaho State University | Computer Science

Express these specifications using the following propositions:

- p: "The user enters a valid password"
- q: "Access is granted"
- r: "The user has paid the subscription fee"

1. "The user has paid the subscription fee, but does not enter a valid password"

   Solution: $r \wedge \neg p$

2. "If the user has not entered a valid password but has paid the subscription fee, then access is granted"

   Solution: $(\neg p \wedge \neg r) \rightarrow g$

Exercise

Construct a combinatorial circuit using inverters, OR gates, and AND gates such that:

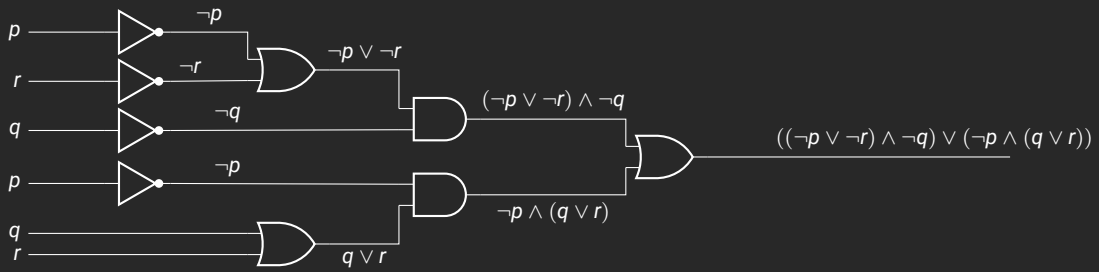
- Produces the output $((\neg p \vee \neg r) \wedge \neg q) \vee (\neg p \wedge (q \vee r))$
- From input bits p, q, and r.

Exercise

Idaho State University | Computer Science

Construct a combinatorial circuit using inverters, OR gates, and AND gates such that:

- Produces the output $((\neg p \vee \neg r) \wedge \neg q) \vee (\neg p \wedge (q \vee r))$
- From input bits p, q, and r.



Satisfiability

CS 1187

Satisfiability

- **Satisfiable** - when a compound proposition has an assignment of truth values to its variables that makes it true (i.e., it is a tautology or contingency)

Satisfiability

- **Satisfiable** - when a compound proposition has an assignment of truth values to its variables that makes it true (i.e., it is a tautology or contingency)

- **Unsatisfiable** - when a compound proposition is false for all assignments of truth values to its variables
  - A proposition is only unsatisfiable if its negation is true for all assignments of truth values to its variables

Satisfiability

- **Satisfiable** - when a compound proposition has an assignment of truth values to its variables that makes it true (i.e., it is a tautology or contingency)
- **Unsatisfiable** - when a compound proposition is false for all assignments of truth values to its variables
  - A proposition is only unsatisfiable if its negation is true for all assignments of truth values to its variables
- **Solution** - to a satisfiability problem is a particular assignment of truth values that makes a compound proposition true

Satisfiability Examples

Idaho State University | Computer Science

Determine whether the following compound propositions are satisfiable

- $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$

Satisfiability Examples

Idaho State University | Computer Science

Determine whether the following compound propositions are satisfiable

- $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$
  - We can reason about truth value, and with that we can see that this is true, as long as p, q, and r all have the same truth value

Satisfiability Examples

Idaho State University | Computer Science

Determine whether the following compound propositions are satisfiable

- $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$
  - We can reason about truth value, and with that we can see that this is true, as long as p, q, and r all have the same truth value

- $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$

Satisfiability Examples

Idaho State University | Computer Science

Determine whether the following compound propositions are satisfiable

- $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$
  - We can reason about truth value, and with that we can see that this is true, as long as p, q, and r all have the same truth value

- $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$
  - Again we reason about truth values, and we can see that as long as one variable is true and another false, then this proposition is true

Applications of Satisfiability

Satisfiability problems come into play in several areas

- Constraint satisfaction
  - Planning systems
  - Suduko
  - N-Queens problems
- Complexity Analysis and Theory

Exercises

- Use the conditional-disjunction equivalence to find an equivalent compound proposition that does not involve conditionals

$$p \rightarrow \neg q$$

Exercises

Idaho State University | Computer Science

- Use the conditional-disjunction equivalence to find an equivalent compound proposition that does not involve conditionals

$$p \rightarrow \neg q$$

$p \rightarrow \neg q \quad \equiv \quad \neg p \vee \neg q$ by the conditional-disjunction equivalence

Exercises

Idaho State University | Computer Science

- Use the conditional-disjunction equivalence to find an equivalent compound proposition that does not involve conditionals

$$p \rightarrow \neg q$$

$p \rightarrow \neg q \quad \equiv \quad \neg p \vee \neg q$ by the conditional-disjunction equivalence

$$(p \rightarrow q) \rightarrow r$$

Exercises

- Use the conditional-disjunction equivalence to find an equivalent compound proposition that does not involve conditionals

$$p \to \neg q$$

$$p \to \neg q \quad \equiv \quad \neg p \vee \neg q \quad \text{by the conditional-disjunction equivalence}$$

$$(p \to q) \to r$$

$$\begin{aligned}
(p \to q) \to r &\equiv \neg(p \to q) \vee r && \text{by the conditional-disjunction equivalence} \\
&\equiv \neg(\neg p \vee q) \vee r && \text{by the conditional-disjunction equivalence} \\
&\equiv (\neg\neg p \wedge \neg q) \vee r && \text{by the second De Morgan's Law} \\
&\equiv (p \wedge \neg q) \vee r && \text{by the Double Negation Law}
\end{aligned}$$

Meta-Logic

CS 1187

Meta-Logic

- **Meta-Logic:** concerned with stepping outside of the language of logic
  - Allows us to make statements about the properties of a logical system
  - Allows us to talk about logic rather than to simply use logic

- We looked at three methods of reasoning, each with different styles
  - Truth tables: enable the calculation of values (meanings) of propositions, yielding *semantic reasoning*
  - Inference rules allowed reasoning based on structure (*syntactic reasoning*)
  - Boolean algebra allowed for reasoning to prove equality of two expressions, or to calculate the values of expressions

Meta-Logic

- To use meta-logic, we need a vocabulary

- Towards this, we have two notions of truth: *semantic* and *syntactic*, corresponding to the $\vDash$ and $\vdash$ operators, respectively
  - $P_1, P_2, \ldots, P_n \vdash Q$ means that there is a proof which infers Q from the assumptions $P_1, P_2, \ldots, P_n$ using rules of inference and natural deduction.
  - $P_1, P_2, \ldots, P_n \vDash Q$ means that Q must be True if $P_1, P_2, \ldots, P_n$ are all true, but it says nothing about whether we have proof or if one is possible.

Meta-Logic

- **Consistent:** A formal system is consistent if the following statement is true for all WFFs *a* and *b*:
  If $a \vdash b$ then $a \models b$
  - That is, the system is *consistent* if each proposition is provable using inference rules is *actually* true.

- **Complete:** A formal system is complete if the following statement is true for all WFFs *a* and *b*:
  If $a \models b$ then $a \vdash b$
  - The system is complete if the inference rules are powerful enough to prove every proposition which is true

- **Theorem:** Propositional Logic is both consistent and complete

- **Godel's Theorem:** Any logical system powerful enough to express ordinary arithmetic must be either inconsistent or incomplete. Thus it is impossible to capture all of mathematics in a safe logical system.

For Next Time

- Review DMUC Chapter 6.6 – 6.10
- Review this Lecture
- Read Chapter 7
- Come To Lecture

Are there any questions?