Chapter 1. The Foundations: Logic and Proofs

1.13 Rules of inference with quantifiers
Logic and bit operations
Specification consistency

1.13 Rules of inference with quantifiers

universal instantiation

$$\frac{\forall x\, P(x)}{P(c)}$$

where *c* is a particular member of the domain

existential instantiation

$$\frac{\exists x\, P(x)}{P(c)\ \mathit{for\ some\ element}\ c}$$

we cannot select arbitrary *c*, it must be a *c* from the domain such that P(c) is true

universal generalization

$$\frac{P(c)\ \mathit{for\ any\ arbitrary}\ c}{\forall x\, P(x)}$$

where *c* is from the domain

existential generalization

$$\frac{P(c)\ \mathit{for\ some\ element}\ c}{\exists x\, P(x)}$$

1.13 Rules of inference with quantifiers

$$\frac{\begin{array}{l}\forall x\left(P(x) \rightarrow Q(x)\right) \\ P(a)\, \textit{where a is a particular element of the domain}\end{array}}{Q(a)}$$

universal
Modus
Ponens

$$\frac{\begin{array}{l}\forall x\left(P(x) \rightarrow Q(x)\right) \\ \neg Q(a)\, \textit{where a is a particular element of the domain}\end{array}}{\neg P(a)}$$

universal
Modus
Tollens

1.13 Rules of inference with quantifiers

Example 1: (invalid argument)
Let H(x) be “x is happy”. Given the premise $\exists x\, H(x)$ we conclude H(Lola). Therefore Lola is happy. What is wrong with the argument?

1.13 Rules of inference with quantifiers

Example 1: (invalid argument)
Let H(x) be “x is happy”. Given the premise $\exists x\, H(x)$ we conclude H(Lola). Therefore Lola is happy. What is wrong with the argument?

Solution: we have

$$\frac{\exists x\, H(x)}{H(Lola)}$$

1.13 Rules of inference with quantifiers

Example 1: (invalid argument)
Let H(x) be “x is happy”. Given the premise $\exists x\, H(x)$ we conclude H(Lola). Therefore Lola is happy. What is wrong with the argument?

Solution: we have

$$\frac{\exists x\, H(x)}{H(Lola)}$$

mistake: we selected arbitrary x (Lola)

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

a) All students in this class understand logic. Xavier is a student in this class. Therefore, he understands logic.

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

a) All students in this class understand logic. Xavier is a student in this class. Therefore, he understands logic.

$$\frac{\forall x\, U(x)}{U(Xavier)}$$

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

a) All students in this class understand logic. Xavier is a student in this class. Therefore, he understands logic.

$$\begin{array}{ll} 1.\ \forall x\, U(x) & \text{premise} \\ \hline 2.\ U(Xavier) & \text{universal instantiation from 1} \end{array}$$

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

b) Every Computer Science major takes discrete math. Maria is taking discrete math. Therefore, she is a Computer Science major.

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

C(x) *D(x)*

b) Every Computer Science major takes discrete math. Maria is taking discrete math. Therefore, she is a Computer Science major.

$$\begin{array}{l} \forall x\left(C\left(x\right)\rightarrow D\left(x\right)\right) \\ D\left(Maria\right) \\ \hline C\left(Maria\right) \end{array}$$

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

b) Every Computer Science major takes discrete math. Maria is taking discrete math. Therefore, she is a Computer Science major.

$$\begin{array}{l} \forall x\left(C\left(x\right)\rightarrow D\left(x\right)\right) \\ \underline{D\left(Maria\right)\qquad\qquad} \\ C\left(Maria\right) \end{array}$$

argument is incorrect
(fallacy of affirming the conclusion)

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

c) All parrots like fruit. My pet bird is not a parrot. Therefore, he doesn't like fruit.

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

P(x) F(x)

c) All parrots like fruit. My pet bird is not a parrot. Therefore, he doesn't like fruit.

1. $\forall x(P(x) \rightarrow F(x))$ premise
2. $\neg P(my\ pet)$ premise

$$\neg F(my\ pet)$$

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

P(x) F(x)

c) All parrots like fruit. My pet bird is not a parrot. Therefore, he doesn't like fruit.

1. $\forall x\left(P(x) \rightarrow F(x)\right)$ premise
2. $\neg P(my\ pet)$ premise

---

3. $P(my\ pet) \rightarrow F(my\ pet)$ universal instantiation from 1

Cannot derive $\neg F(my\ pet)$

- fallacy of denying the hypotheses

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

d) Everyone who eats granola every day is healthy. Linda is not healthy. Therefore, Linda doesn't eat granola every day.

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

G(x) *H(x)*

d) Everyone who eats granola every day is healthy. Linda is not healthy. Therefore, Linda doesn't eat granola every day.

$1.\ \forall x\left(G(x) \rightarrow H(x)\right)$ premise

$2.\ \neg H(Linda)$ premise

$\neg G(Linda)$

1.13 Rules of inference with quantifiers

Example 2: for each argument determine whether the argument is correct or incorrect and explain why.

G(x) H(x)

d) Everyone who eats granola every day is healthy. Linda is not healthy. Therefore, Linda doesn't eat granola every day.

$1. \forall x(G(x) \rightarrow H(x))$ premise

$2. \neg H(Linda)$ premise

$3. G(Linda) \rightarrow H(Linda)$ univ. instantiation from 1

$4. \neg G(Linda)$ by Modus tollens from 2 & 3

Logic and Bit Operations

In computer all the information is represented with bits (binary digit). Bit is a symbol with two possible values: 0 (zero), and 1(one)

bit can be used to represent truth values:1 for True, 0 for False

Boolean variable is a variable with two possible values (0,1)

Note: Computer operations correspond to the logical operations.

$\wedge \equiv$ AND
$\vee \equiv$ OR
$\oplus \equiv$ XOR

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0			
0	1			
1	0			
1	1			

Logic and Bit Operations

In computer all the information is represented with bits (binary digit). Bit is a symbol with two possible values: 0 (zero), and 1(one)

bit can be used to represent truth values:1 for True, 0 for False

Boolean variable is a variable with two possible values (0,1)

Note: Computer operations correspond to the logical operations.

$\wedge$ ≡ AND
$\vee$ ≡ OR
$\oplus$ ≡ XOR

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0		
0	1	0		
1	0	0		
1	1	1		

Logic and Bit Operations

In computer all the information is represented with bits (binary digit). Bit is a symbol with two possible values: 0 (zero), and 1(one)

bit can be used to represent truth values:1 for True, 0 for False

Boolean variable is a variable with two possible values (0,1)

Note: Computer operations correspond to the logical operations.

$\wedge \equiv$ AND
$\vee \equiv$ OR
$\oplus \equiv$ XOR

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	
0	1	0	1	
1	0	0	1	
1	1	1	1	

Logic and Bit Operations

In computer all the information is represented with bits (binary digit). Bit is a symbol with two possible values: 0 (zero), and 1(one)

bit can be used to represent truth values:1 for True, 0 for False

Boolean variable is a variable with two possible values (0,1)

Note: Computer operations correspond to the logical operations.

$\wedge \equiv$ AND
$\vee \equiv$ OR
$\oplus \equiv$ XOR

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

In computer all the information is represented with bits (binary digit). Bit is a symbol with two possible values: 0 (zero), and 1(one)

bit can be used to represent truth values:1 for True, 0 for False

Boolean variable is a variable with two possible values (0,1)

Note: Computer operations correspond to the logical operations.

$\wedge \equiv$ AND
$\vee \equiv$ OR
$\oplus \equiv$ XOR

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a bit string is a sequence of 0's and 1's.
length of a bit string is the number of bits in the string

Example: 00110101110   length=11

bit string

Logic and Bit Operations

Exercise: find the bitwise OR, AND, and XOR of the bit strings 001110101 and 110101100 (*from now on we'll be splitting the bit strings into blocks of four to make them easier to read*).

```
0  0111  0101
1  1010  1100
-------------
               AND
```

```
0  0111  0101
1  1010  1100
-------------
               OR
```

```
0  0111  0101
1  1010  1100
-------------
                XOR
```

x	y	x∧y	x∨ y	x⊕y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: find the bitwise OR, AND, and XOR of the bit strings 001110101 and 110101100 (*from now on we'll be splitting the bit strings into blocks of four to make them easier to read*).

```
0  0111  0101
1  1010  1100
-------------
0  0010  0100  AND
```

```
0  0111  0101
1  1010  1100
-------------
                OR
```

```
0  0111  0101
1  1010  1100
-------------
               XOR
```

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: find the bitwise OR, AND, and XOR of the bit strings 001110101 and 110101100 (*from now on we'll be splitting the bit strings into blocks of four to make them easier to read*).

```
0 0111 0101
1 1010 1100
-----------
0 0010 0100  AND
```

```
0 0111 0101
1 1010 1100
-----------
1 1111 1101  OR
```

```
0 0111 0101
1 1010 1100
-----------
             XOR
```

x	y	x∧y	x∨ y	x⊕y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: find the bitwise OR, AND, and XOR of the bit strings 001110101 and 110101100 (*from now on we'll be splitting the bit strings into blocks of four to make them easier to read*).

```
0  0111  0101
1  1010  1100
-------------
0  0010  0100  AND
```

```
0  0111  0101
1  1010  1100
-------------
1  1111  1101  OR
```

```
0  0111  0101
1  1010  1100
-------------
1  1101  1001  XOR
```

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: Evaluate expression $(0\ 1101 \vee 1001) \oplus (01101 \wedge 00111)$

```
0  1101          0  1101
   1001          0  0111
-------          -------          ------
       OR               AND             ⊕
```

x	y	x∧y	x∨ y	x⊕y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: Evaluate expression (0 1101 ∨ 1001) ⊕ (01101 ∧ 00111)

```
0  1101           0  1101
   1001           0  0111
-------           -------           ------
0  1101 OR        0  0101  AND
                                          ⊕
```

x	y	x∧y	x∨ y	x⊕y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logic and Bit Operations

Exercise: Evaluate expression $(0\ 1101 \vee 1001) \oplus (01101 \wedge 00111)$=0 100

```
0  1101               0  1101               0  1101
   1001               0  0111               0  0101
-------               -------               -------
0  1101 OR            0  0101  AND          0  1000  ⊕
```

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

System Specifications

Having hardware or software specification in natural language we would like to be able to translate them into logical expressions. This will make the specification *precise* and *unambiguous*, and can be used as basis for system development.

System specification is consistent if and only if it doesn't have any *conflicting requirements* that could be used to derive a *contradiction*.

When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

Example 1: Determine whether the system specification is consistent (i.e. not contradictory)

“Whenever the system software is being upgraded, users cannot access the file system. If users can access the file system, then they can save new files. If users cannot save new files, then the system software is not being upgraded”

System Specifications

System specification is consistent if and only if it doesn't have any *conflicting requirements* that could be used to derive a *contradiction*.

When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

Example 1: Determine whether the system specification is consistent (i.e. not contradictory)

“Whenever the system software is being upgraded, users cannot access the file system. If users can access the file system, then they can save new files. If users cannot save new files, then the system software is not being upgraded”

p: “the system software is being upgraded”
q: “users can access the file system”
r: “users can save new files”

System Specifications

System specification is consistent if and only if it doesn't have any *conflicting requirements* that could be used to derive a *contradiction*.

When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

Example 1: Determine whether the system specification is consistent (i.e. not contradictory)

“Whenever the system software is being upgraded, users cannot access the file system. If users can access the file system, then they can save new files. If users cannot save new files, then the system software is not being upgraded”

p: “the system software is being upgraded”
q: “users can access the file system”
r: “users can save new files”

$$\begin{cases} p \to \neg q \\ q \to r \\ \neg r \to \neg p \end{cases}$$

System Specifications

System specification is consistent if and only if it doesn't have any *conflicting requirements* that could be used to derive a *contradiction*.

When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

Example 1: Determine whether the system specification is consistent (i.e. not contradictory)

“Whenever the system software is being upgraded, users cannot access the file system. If users can access the file system, then they can save new files. If users cannot save new files, then the system software is not being upgraded”

p: “the system software is being upgraded”
q: “users can access the file system”
r: “users can save new files”

$$\left\{\begin{array}{l} p \rightarrow \neg q \\ q \rightarrow r \\ \neg r \rightarrow \neg p \end{array}\right.$$

let's check if there exists an assignment of truth values (to p, q, and r) that makes all of these compound propositions true.

System Specifications

Example 1: Determine whether the system specification is consistent (i.e. not contradictory)

$$\begin{cases} p \to \neg q \\ q \to r \\ \neg r \to \neg p \end{cases}$$

let's check if there exists an assignment of truth values (to p, q, and r) that makes all of these compound propositions true.

If p is True, then $\neg q$ should be also True (otherwise the implication $p \to \neg q$ will be False). Therefore q is False. From the second implication since q is False, r may have any truth value, since it anyway leads to $q \to r$ being True. So let's see the last implication: $\neg r \to \neg p$.: we assumed that p is True, therefore $\neg p$ is False, and $\neg r$ has to have truth value False, in which case r is True. So, if p is True, q is False and r is True, each of our compound propositions are True.

Therefore, the system specification is consistent.

P.S. we could also do it with a truth table – do it for practice.

System Specifications

Example 2: Determine whether the system specification is consistent.

"The system is in multiuser state if and only if it is operating normally. If the system is operating normally the kernel is functioning.
The kernel is not functioning or the system is in interrupt mode. If the system is not in multiuser state, then it is in interrupt mode. The system is not in interrupt mode."

System Specifications

Example 2: Determine whether the system specification is consistent.

“The system is in multiuser state if and only if it is operating normally. If the system is operating normally the kernel is functioning.
The kernel is not functioning or the system is in interrupt mode. If the system is not in multiuser state, then it is in interrupt mode. The system is not in interrupt mode.”

p: “the system is in multiuser state”
q: “it is operating normally”
r: “the kernel is functioning”
s: “the system is in interrupt mode”

$$\left\{ \begin{array}{l} p \leftrightarrow q \\ q \rightarrow r \\ \neg r \vee s \\ \neg p \rightarrow s \\ \neg s \end{array} \right.$$

same technique:
let's check if there exists an assignment of truth values (to p, q, r, and s) that makes all of these compound propositions true.

System Specifications

Example 2: Determine whether the system specification is consistent.

$$\left\{\begin{array}{l} p \leftrightarrow q \\ q \rightarrow r \\ \neg r \vee s \\ \neg p \rightarrow s \\ \neg s \end{array}\right.$$

same technique:
let's check if there exists an assignment of truth values (to p, q, r, and s) that makes all of these compound propositions true.

1) $\neg s$ must be True, therefore s is False
2) if s is False, then in order for implication $\neg p \rightarrow s$ to be True, $\neg p$ must be also False. Therefore, p is True.
3) if p is True, then in order for $p \leftrightarrow q$ to be True, q must be True.
4) if q is True then in order for $q \rightarrow r$ to be True, r also must be True.
5) if r is True, and s is False, then $\neg r \vee s$ is False ∨ False, which gives us False.

So, there is no assignment of truth values that makes all of the above propositions true. Hence, the system specification is inconsistent.