Algorithms and Data Structures for Data Science

Formal Logic

CS 277
Brad Solomon

August 25, 2021

Department of Computer Science
mp_racing delayed a week

**Key Concepts:**

- Python Control Structures
- Data Parsing
- Logic

12* lab assignments (15 points each)

5 machine problem sets (60 points each)
Reminder: Labs are online

https://courses.grainger.illinois.edu/cs277/fa2021/info/labs/

Two lab sections: 10 AM and 11 AM
Joining class Github

https://edu.cs.illinois.edu/create-ghe-repo/cs277-fa21/

Your CS 277 repository is ready!

You will submit all course assignments through this repository:

- https://github-dev.cs.illinois.edu/cs277-fa21/
- Follow your course-specific instructions on what to do next (you will usually not visit your git URL directly).
Submitting via Gradescope

Must upload directly

Only submit files listed on each assignment
Last time:

Fundamental of Python programming

Selection Control Structures

```python
class SelectionControlStructs:
    def __init__(self, num):
        self.num = num

    def print_selection(self):
        if self.num in [0,1,2,3,4]:
            print("Top 5!")
        elif self.num > 10:
            print("num too large!")
        elif self.num > 15:
            print("will this ever get called?")
        else:
            print(self.num)
```

Functions

Functions organize code with clear I/O

```python
def addTogether(x, y, z=None):
    if z is None:
        return x + y
    else:
        return x + y + z
```

If `__name__ == '__main__':`

```python
def addTogether(x, y, z):
    if z is None:
        return x + y
    else:
        return x + y + z

if __name__ == '__main__':
    print(addTogether(2,3))
    print(addTogether(2,3,5))
```

Object-Oriented Programming

An **object** is a conceptual grouping of variables and functions that make use of those variables

Variables:

Functions:
Object-Oriented Programming

An **object** is a conceptual grouping of variables and functions that make use of those variables

**Variables:**

**Functions:**
from random import randint

class gacha:
    def __init__(self):
        pass

    # Input: Nothing!
    # Output: The random prize that was won
    # Should also modify cash reserve and prizeList
    def pullPrize(self):
        pass

if __name__ == '__main__':
    pass
Object-Oriented Programming

Variables:

Functions:
Object-Oriented Programming

Classes can redefine Python built-in functions

class prize:
    def __init__(self, uid, name, rarity):
        self.rarity = rarity
        self.name = name
        self.uid = uid

    # Representation of object
    def __repr__(self):
        return "({}, {}, {})".format(self.uid, self.name, self.rarity)

    # String Representation of object
    def __str__(self):
        return "({}, {}, {})".format(self.uid, self.name, self.rarity)
Storing a CSV

<table>
<thead>
<tr>
<th>ID</th>
<th>Area</th>
<th>Beds</th>
<th>Baths</th>
<th>Zip</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2222</td>
<td>3</td>
<td>3.5</td>
<td>32312</td>
<td>1981</td>
<td>250000</td>
</tr>
<tr>
<td>2</td>
<td>1628</td>
<td>3</td>
<td>2</td>
<td>32308</td>
<td>2009</td>
<td>185000</td>
</tr>
<tr>
<td>3</td>
<td>3824</td>
<td>5</td>
<td>4</td>
<td>32312</td>
<td>1954</td>
<td>399000</td>
</tr>
</tbody>
</table>
Objects can be used purely to store information

class housingData:
    def __init__(self):
        pass

    def getID(self):
        pass

housing.py
Learning Objectives

Formalize logic expressions and operators

Construct and solve conditional expressions

Motivate and explore quantificational logic
Why formal logic?

Fundamentally useful to CS:

Coding control structures

Set abstract data type

Proofs and theory

Many applications in the real world
Radical Racing Robots

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Robot starts at (2,1)
Radical Racing Robots

Robot starts at (2,1)
Radical Racing Robots

1) Practice writing conditionals and loops

2) Process a multi-file data source

3) Use formal logic to code a virtual robot

Robot starts at (2,1)
Propositional Logic

A declarative sentence which is either True or False
(but not both)

This sentence is a proposition.
States a fact ✅ True!

Computer science is boring.
States a fact ✅ False!

This statement is false.
States a fact ✅ True! False! False!
Propositional Logic

Which of the following are propositions?

Data science is the best science.

The cake is a lie.

$1 + 1 = 3$

Beware the Jabberwock

Python is a language and an animal
Propositional Logic

We can make more complex propositions with operators

Python is a language AND

Python is an animal

1 + 1 = 3 OR

Pineapple on pizza is delicious

An apple a day does NOT keep the doctor away
Logic Operators (*Not*)

The **negation** of $p$ is denoted as $\neg p$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Logic Operators (And)

The conjunction of $p$ and $q$ is denoted as $p \land q$.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \land q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Logic Operators *(Or)*

The **disjunction** of $p$ and $q$ is denoted as $p \lor q$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \lor q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td><strong>F</strong></td>
</tr>
</tbody>
</table>
Logic Operators (*xor*)

The **exclusive or** of $p$ and $q$ is denoted as $p \oplus q$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \oplus q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Logic operators modify or join propositions together

<table>
<thead>
<tr>
<th>English</th>
<th>Logic</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not $p$ (Negation)</td>
<td>$\neg p$</td>
<td></td>
</tr>
<tr>
<td>$p$ and $q$ (Conjunction)</td>
<td>$p \land q$</td>
<td></td>
</tr>
<tr>
<td>$p$ or $q$ (Disjunction)</td>
<td>$p \lor q$</td>
<td></td>
</tr>
<tr>
<td>$p$ xor $q$ (Exclusive Or)</td>
<td>$p \oplus q$</td>
<td></td>
</tr>
</tbody>
</table>
Logical Equivalencies

**Commutative**

\[(p \land q) \equiv (q \land p)\]

\[(p \lor q) \equiv (q \lor p)\]
Logical Equivalencies

**Distributive**

\[ p \land (q \lor r) \equiv \]

\[ p \lor (q \land r) \equiv \]
De Morgan’s Laws

How do we negate a complex proposition?

\[ \neg (p \land q) \equiv \]
De Morgan's Laws

How do we negate a complex proposition?

\[ \neg(p \lor q) \equiv \]
Propositional Logic

Complex propositions can sometimes get confusing

\[ a = True \quad b = False \quad c = True \quad d = True \]

\[ a \land b \lor c \land d \]
Order of Logic Operators

Highest Precedence

\[ \neg p \]

\[ p \land q \]

\[ p \lor q \]

Lowest Precedence
Logic Exercise 1

$p$: It is raining outside.  
$q$: It is snowing outside.

$r$: I am wearing a jacket.  
$s$: I am cold.

\[ \neg p \land \neg s \]

\[ r \land (p \lor q) \]

\[ (p \oplus q) \land (r \lor s) \]
Logic Exercise 2

Write a logic statement: Either apples and oranges grow on trees or money grows on trees, but not both.

\[ a: \text{Apples grow on trees} \]
\[ b: \text{Oranges grow on trees} \]
\[ c: \text{Money grow on trees} \]
Logic Exercise 3

“Does everyone want coffee?”

Nancy: ‘I don’t know’
Carl: ‘I don’t know’
Tiffani: ‘No.’

Who wants coffee?

What is the logic expression?
Conditional Statements

A conditional statement is a compound proposition that means “if p then q” and is denoted as \( p \implies q \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( q )</td>
<td>( p \implies q )</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Logical Equivalent to Conditional

Given $p$ and $q$, can we rewrite an equivalent compound proposition to the conditional $p \implies q$?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$q$</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
De Morgans Laws for Conditionals

How do we negate a conditional proposition?

\[ \neg (p \implies q) \equiv \]
Converse of a conditional

The **converse** of a conditional statement is formed by swapping the position of its variables.

What is the converse of $p \implies q$?

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Contrapositive

The **contrapositive** of a conditional statement is formed by swapping the position of its variables and negating both of them.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$q$</td>
<td>$p \implies q$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Logic Exercise 4

$p$: It is freezing outside.

$q$: My car will start

“If it’s freezing outside, my car won’t start”

Logic:

Converse:

Contrapositive:
Biconditional Statements

A biconditional statement is a compound proposition that means “p if and only if q” and is denoted as $p \iff q$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \iff q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Logical Equivalent to Biconditional

Given \( p \) and \( q \), can we rewrite an equivalent compound proposition to the biconditional?

\[ p \iff q \equiv \]
Logic Exercise 5

Write a logic statement: You will get an ‘A’ in this class if and only if you complete all assignments, ask questions in class, and either take all exams or get all extra credit points.
A twist on the two-guard riddle

One guard tells the truth, the other guard lies.

Guard 1: “The exit is behind this door and death is behind the other door”

Guard 2: “The exit is behind a door and death is behind the other door”
A twist on the two-guard riddle

One guard tells the truth, the other guard lies.

Guard 1: “The exit is behind this door and death is behind the other door”
Guard 2: “The exit is behind a door and death is behind the other door”
A twist on the two-guard riddle

One guard tells the truth, the other guard lies.

Guard 1: “The exit is behind this door and death is behind the other door”

Guard 2: “The exit is behind a door and death is behind the other door”

________________________  ⇒  __________________________
Propositions vs Predicates

A **predicate** is a sentence with variables which can be turned into a proposition when assigned values.

```python
x = 5
y = "Bees"
z = [1, 2, 5]

# Propositions
print(x == 3)
print("b" in "Bees")
print(len(z)> 1)
```

`prop_vs_pred.py`
Predicate Logic

A *predicate* is a sentence with variables which can be turned into a proposition when assigned values

\[ P(x): \text{“} x \text{ goes to UIUC”} \]

\[ Q(x): x^2 + 5 \geq 10 \]
Predicate Logic

A *predicate* is a sentence with variables which can be turned into a proposition when assigned values

\[ P(x): \text{“} x \text{ is odd.} \text{“} \]
Domain of Discourse \( \left( P(x), x \in \{ D \} \right) \)

*The set of values we allow a variable to take*

1) **Constrain a predicate to a set of values**

\[ P(x) = "x < 5" \] (True for an infinite number of values)
Domain of Discourse \( (P(x), x \in \{D\}) \)

The set of values we allow a variable to take

1) Constrain a predicate to a set of values

\( P(x) = "x < 5" \) (True for an infinite number of values)

\( P(x) = "x < 5", x \in \mathbb{N} \)
Domain of Discourse \( (P(x), x \in \{D\}) \)

*The set of values we allow a variable to take*

1) **Constrain a predicate to a set of values**

\( P(x) = "x < 5" \) (True for an infinite number of values)

\( P(x) = "x < 5", x \in \mathbb{N} \) (True for \{0,1,2,3,4\}, a finite set!)

'\( x \) is in the domain of natural numbers (0, 1, 2, …)'
Domain of Discourse \((P(x), x \in \{D\})\)

*The set of values we allow a variable to take*

1) **Constrain a predicate to a set of values**

2) **Prevent nonsense statements** (Neither True nor False)

\[ P(x) = "10 / x > 1" \]

\[ x = \text{Any value that isn't a number} \]

\[ \text{E.g. } '10 / \text{Zebra} > 1' \]

\[ x = 0 \]

\[ \text{E.g. } '10 / 0 > 1' \]
Domain of Discourse \( \left( P(x), x \in \{ D \} \right) \)

*The set of values we allow a variable to take*

1) *Constrain a predicate to a set of values*

2) *Prevent nonsense statements*  \( \text{ (Neither True nor False) } \)

\[ P(x) = \text{“}10 / x > 1\text{“}, x \in \mathbb{R}_{\neq 0} \quad (-e, \pi, 0.5, 12/5, \ldots) \]

\( x = \) Any value that isn't a number

E.g. ‘10 / Zebra > 1’

\( x = 0 \)

E.g. ‘10 / 0 > 1’
Predicate Programming Logic

myList = [3, 6, 7, 2, 1]

“Return all values which are even in myList”

Domain:

Variables:

Predicates:
Predicate Programming Logic

“Return the first row in the matrix with at least two 1s in it.”

Domain:

Variables:

Predicates:
Radical Racing Robots

“RacingBot should move one square in a cardinal direction away from its current position.”
Radical Racing Robots

“RacingBot should move one square in a cardinal direction away from its current position but cannot move backwards.”
Radical Racing Robots

“If multiple moves are available, move in a clockwise order but don’t move backwards from our current position.”
Radical Racing Robots

“If there are no valid moves, racingBot should stay in place.”
Radical Racing Robots

“If racingBot’s current position has a value greater than 1, it’s movement is based on the value of it’s cargo.”

“If there exists three unique cargo objects $x$, $y$, $z$ such that $x = y + z$, racingBot should move north”