CAB203: Discrete Structures Definition

Author not responsible for consequences caused by any innacuracies.

Markdown Setup

Open in Visual Studio Code and install Markdown Preview Enhanced:
https://marketplace.visualstudio.com/items?itemName=shd101wyy.markdown-preview-enhanced

Typesetting Powered by $\KaTeX$

$\KaTeX$ documentation: https://katex.org/
which uses $\LaTeX$-*like* syntax to display mathematical syxbols

Equivalence

$$\text{Equals:} \quad = \tag{1}$$

$$\text{Greater than:} \quad > \tag{2}$$

$$\text{Greater than or equal to:} \quad \geq \tag{3}$$

$$\text{Less than:} \quad < \tag{4}$$

$$\text{Less than or equal to:} \quad \leq \tag{5}$$

$$\text{Not equal to:} \quad \neq \tag{6}$$

$$\text{Modular equivalence:} \quad \equiv \tag{7}$$

Operators

$$\text{Multiply:} \quad \cdot \tag{8}$$

$$\text{Mod:} \quad \mathrm{mod} \tag{9}$$

$$\text{Implication:} \quad \models \tag{10}$$

$$NOT: \quad \neg \tag{11}$$

$$AND, \quad \wedge \tag{12}$$

$$OR, \quad \vee \tag{13}$$

$$XOR, \quad \oplus \tag{14}$$

$$IF..THEN: \quad \rightarrow \tag{15}$$

$$IF\ AND\ ONLY\ IF: \quad \leftrightarrow \tag{16}$$

Formula Classification

$$\textbf{Tautologies:} \quad alwaysT \tag{17}$$

$$\textbf{Contradictions:} \quad alwaysF \tag{18}$$

$$\textbf{Contingent formulas:} \quad T\ or\ F \quad \text{depending on variables} \tag{19}$$

$$\textbf{Satisfiable formulas:} \quad tautologies\ \ or\ \ contingent\ formulas \tag{20}$$

Set Theory Abstractions

$$\text{Set:} \quad S \tag{21}$$

$$\text{Universe:} \quad U \tag{22}$$

$$\text{Member in set:} \quad \in \tag{23}$$

$$\text{Not member in set:} \quad \notin \tag{24}$$

$$\textbf{Real, } \textit{any number:} \quad \mathbb{R} \tag{25}$$

$$\textbf{Integer, } \textit{only whole:} \quad \mathbb{Z} \tag{26}$$

$$\textbf{Natural}\,_0\textbf{, } \textit{whole non-negatives:} \quad \mathbb{N}_0 \tag{27}$$

$$\textbf{Natural}\,_1\textbf{, } \textit{whole positives:} \quad \mathbb{N}_1 \tag{28}$$

$$\text{Subset:} \quad A \subseteq B \tag{29}$$

$$\text{Proper subset:} \quad A \subset B \tag{30}$$

$$\text{Equality through subsets:} \quad S \subseteq T \,\&\, T \subseteq S \tag{31}$$

Set Constructors

Set listing:

$$SMALLPRIMES = \{1, 2, 3, 5, 7\} \tag{32}$$

Implied pattern: *(bad practice)*

$$EVENS = \{2, 4, 6, 8, \ldots\} \tag{33}$$

Setbuilder notation: “set comprehension”

Subset of elements, that match a condition

$$\{x \in S : \phi(x)\} \tag{34}$$

$$SQUARES = \{x \in \mathbb{Z} : x = y^2 \text{ for some } y \in \mathbb{Z}\} \tag{35}$$

Setbuilder notation: “replacement”

Apply a theory to each member and collect results

$$\{f(x) : x \in S\} \tag{36}$$

$$SQUARES = \{x^2 : x \in \mathbb{Z}\} \tag{37}$$

Set Operators

Union: everything in either

Must define $A \cup B$ to be the smallest set, such that $A \subseteq S$ & $B \subseteq S$

$$\mathrm{A} \cup B = \{x : x \in A \vee x \in B\} \tag{38}$$

Intersection: everything in *both* sides

$$A \cap B = \{x \in A : x \in B\} \tag{39}$$

Difference: remove items in one set from the other

$$A \setminus B = \{x \in A : x \notin B\} \tag{40}$$

Predicates

$$\text{Existential quantification, } \textit{there exists:} \quad \exists \tag{41}$$

$$\text{Universal quantification, } \textit{for all:} \quad \forall \tag{42}$$

Exponents

$$\text{Exponent in } a^3 : \quad 3 \tag{43}$$

$$\text{Base in } a^3 : \quad a \tag{44}$$

$$\text{kilo:} \quad 2^{10} \tag{45}$$

$$\text{mega:} \quad 2^{20} = (2^{10})^2 \tag{46}$$

$$\text{giga:} \quad 2^{30} = (2^{10})^3 \tag{47}$$

$$\text{tera:} \quad 2^{40} = (2^{10})^4 \tag{48}$$

$$\text{peta:} \quad 2^{50} = (2^{10})^5 \tag{49}$$

$$\text{exa:} \quad 2^{60} = (2^{10})^6 \tag{50}$$

Laws of Exponents

$$(ab)^n = a^n \cdot b^n \tag{51}$$

$$a^m \cdot a^n = a^{m+n} \tag{52}$$

$$a^{m-n} = \frac{a^m}{a^n} \quad (\text{when } a \neq 0) \tag{53}$$

$$a^{-n} = \frac{1}{a^n} \quad (\text{when } a \neq 0) \tag{54}$$

$$a^0 = 1 \tag{55}$$

$$(a^m)^n = a^{m \cdot n} \tag{56}$$

Laws of Logarithms

$$\log_a 1 = 0 \tag{57}$$

$$\log_a a = 1 \tag{58}$$

$$\log_a(x \cdot y) = \log_a x + \log_a y \tag{59}$$

$$\log_a x^y = y \log_a x \tag{60}$$

$$\log_a \frac{1}{y} = -\log_a y \tag{61}$$

$$\log_a \frac{x}{y} \tag{62}$$

$$\log_b x = (\log_b a) \cdot \log_a x \tag{63}$$

Ceiling and Floor

$$\text{Ceiling, round up:} \quad \lceil a \rceil \tag{64}$$

$$\text{Floor, round down:} \quad \lfloor a \rfloor \tag{65}$$

Bits

$$\text{Bit string:} \quad \overline{x} \tag{66}$$

$$\text{Mod:} \quad \text{mod} \tag{67}$$

$$\texttt{NOT:} \quad \backsim \tag{68}$$

$$\texttt{AND:} \quad \& \tag{69}$$

$$\texttt{OR:} \quad | \tag{70}$$

$$\texttt{XOR:} \quad \hat{} \tag{71}$$

Scientific Notation

$$\text{Sign:} \quad \pm \tag{72}$$

$$\text{Significant digits:} \quad a.bc \tag{73}$$

$$\text{Exponent:} \quad e \tag{74}$$

$$\text{Base:} \quad 10 \tag{75}$$

IEEE Half-Precision

$$\overbrace{0}^{s}\ \overbrace{10101}^{e}\overbrace{1010101010}^{f} \tag{76}$$

$$\begin{aligned} \text{Sign:} \quad & s(1 - bit) && (77) \\ \text{Exponent:} \quad & e(5bits) && (78) \\ \text{Significant digits:} \quad & f(\text{10 bits dropping leading 1}) && (79) \end{aligned}$$

Recursion

Factorial function on $\mathbb{N}$ defined as:

$$n! = \prod_{j=1}^{n} j = 1 \cdot 2 \cdots (n-1) \cdot n \tag{80}$$

Also $n!$ recursively:

$$n! = \begin{cases} 1 & : n = 1 \\ n(n-1)! & : n > 1 \end{cases} \tag{81}$$

Fibonacci sequence:

$$f(n) = \begin{cases} 1 & : n = 1 \\ 1 & : n = 2 \\ f(n-1) + f(n-2) & : n > 2 \end{cases} \tag{82}$$

Python

```
"""
MODULO OPERATOR
"""

print(17 % 4) # Prints 1 to the console
# 1



"""
EXPONENTS AND LOGARITHMS
"""

>>> import math         # Must import math library
>>> math.log2(8)        # log2 means log base 2 and returns a float (decimal)
# 3.0
>>> 2 ** 3              # ** is exponentiation
# 8
>>> math.log2(100) / math.log2(10)  # base transformation
# 2.0



"""
UFT-8
"""

>>> ord('A')    # Convert character to UTF code point
# 65
>>> chr(65)     # Convert UTF code point to character
# 'A'



"""
HEXADECIMAL
"""

>>> hex(65532)                          # Hex string from integer
# '0xfffc'
>>> "The number is {:x}".format(65532)  # Alternative method
# 'The number is fffc'
>>> 0xfffc                              # Hex literal integer
# 65532
```

```
"""
NUMBERS
"""

>>> 9/2     # Regular division always returns float
# 4.5
>>> 9//2    # Floor division returns integer
# 4



"""
FIBONACCI
"""

def F(n):
if n == 1: return 1
elif n == 2: return 1
else: return F(n-1) + F(n-2)



"""
SET THEORY
"""

>>> S = {1,2,3}; T = {1,3,5}     # Braces define sets
>>> S.add(4); print(S)           # Sets are mutable
# {1, 2, 3, 4}
>>> S.remove(4); print(S)
# {1, 2, 3}

>>> 1 in S                       # Testing membership
# True
>>> 4 in S
# False

>>> S.issubset({1,2,3,4,5})      # Testing subset
# True
>>> S <= {1,2,3,4,5}             # Alternative subset test
# True

>>> S.union(T)                   # Using union
# {1, 2, 3, 5}
>>> S|T                          # Alternative union
```

```
# {1, 2, 3, 5}
>>> S.union(T, {8,9}, {10,11})   # Multiple union
# {1, 2, 3, 5, 8, 9, 10, 11}
>>> someSets = [S, T, {8,9}, {10,11}]
>>> set.union(*someSets)         # Splat operator
# {1, 2, 3, 5, 8, 9, 10, 11}

>>> S.intersection(T)            # Splat and multi-sets would also work
# {1, 3}
>>> S & T                        # Alternative intersection
# {1, 3}

>>> S - T                        # Testing difference
# {2}

>>> S.isdisjoint(T)              # is S & T empty?
# False

>>> len(S)                       # Size of S
# 3

# Setbuilder notation combines of comprehension and replacement:
>>> S = {0, 2, 4, 6, 8}
>>> def p(x): return x % 2 == 0
...
>>> def p(x): return x * 5
...
>>> { f(x) for x in S if p(x) }
# {0, 40, 10, 20, 30}
>>> { s * 5 for s in range(0, 10) if s % 2 == 0 }
# {0, 40, 10, 20, 30}



"""
PREDICATES AND QUANTIFIERS
"""

>>> def p(x,y):      # Any function that returns T/F to be used as predicate
   return x >= y
>>> p(2,1)
# True

>>> def p(x): return x >= 0
>>> S = { -1, 0, 1 }; T = { 0, 1, 2 }
>>> Sp = [ p(x) for x in S ]  # List of booleans
```

```
>>> Tp = [ p(x) for x in T ]  # List of booleans
>>> all(Tp)                   # Check for all: ALL True
# True
>>> all(Sp)                   # Check there  exists: at least ONE True
# True
>>> any(p(x) for x in S)      # Generator expression
# True
```