M269 TMA03 Topics Revue

M269 Revue Tutorial

Phil Molyneux

10 September 2025

M269 End of Module Tutorial
Agenda

- Welcome & Introductions
- Topics from TMA03
- Abstract Data Types — Bags
- Abstract Data Types — Graphs
- Complexity
- Computability

M269 Tutorial

Introductions — Me

- *Name* Phil Molyneux
- *Background* Physics and Maths, Operational Research, Computer Science
  - Undergraduate: Physics and Maths (Sussex)
  - Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
  - Haskell — pure functional programming language
  - Text editors TextMate, Sublime Text — previously Emacs
  - Word processing and presentation slides in LaTeX
  - Mac OS X
- *Learning style* — I read the manual before using the software (really)

M269 Tutorial

Introductions — You

- *Name* ?
- *Position in M269* ? Which part of which Units and/or Reader have you read ?
- *Particular topics you want to look at* ?
- *Learning Syle* ?

Commentary 1

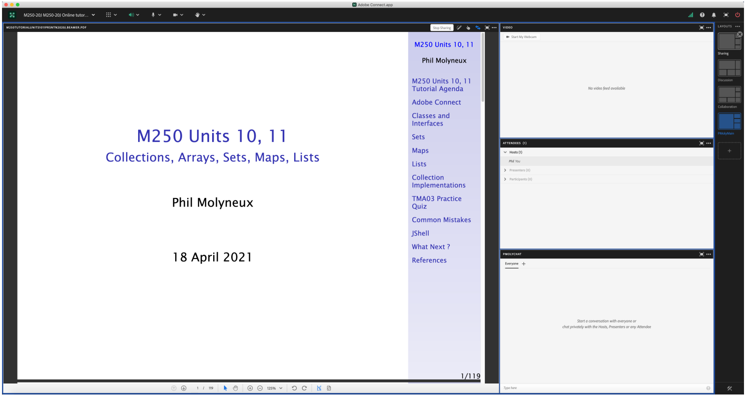
Agenda, Aims and Topics

1 Agenda, Aims and Topics

- Overview of aims of tutorial
- Note selection of topics
- Points about my own background and preferences
- Adobe Connect slides for reference
- Note that the Computability notes are here mainly for reference since the Complexity notes refer to them
- This session is mainly on the Complexity topics

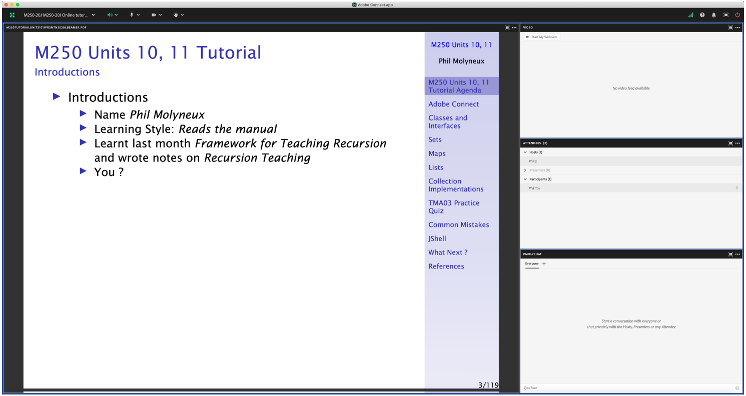
Adobe Connect

Interface — Host View



Adobe Connect

Interface — Participant View



Adobe Connect

Settings

- **Everybody** *Menu bar* › *Meeting* › Speaker & Microphone Setup
- *Menu bar* › *Microphone* › Allow Participants to Use Microphone ✔
- Check Participants see the entire slide **Workaround**
  - *Disable Draw* *Share pod* › *Menu bar* › *Draw icon*
  - *Fit Width* *Share pod* › *Bottom bar* › *Fit Width icon* ✔
- *Meeting* › Preferences › General › **Host Cursor** › Show to all attendees
- *Menu bar* › *Video* › Enable **Webcam** for Participants ✔
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** Meeting › Record Session ✔
- **Documents** Upload PDF with drag and drop to share pod
- Delete *Meeting* › Manage Meeting Information › Uploaded Content and *check filename* › *click on delete*

Adobe Connect

Access

- **Tutor Access**

  TutorHome › M269 Website › Tutorials

  Cluster Tutorials › M269 Online tutorial room

  Tutor Groups › M269 Online tutor group room

  Module-wide Tutorials › M269 Online module-wide room

- **Attendance**

  TutorHome › Students › View your tutorial timetables

- **Beamer Slide Scaling** 440% (422 x 563 mm)

- **Clear Everyone's Status**

  Attendee Pod › Menu › Clear Everyone's Status

- **Grant Access** and send link via email

  Meeting › Manage Access & Entry › Invite Participants. . .

- **Presenter Only Area**

  Meeting › Enable/Disable Presenter Only Area

Adobe Connect

Keystroke Shortcuts

- Keyboard shortcuts in Adobe Connect
- **Toggle Mic** ⌘ + M (Mac), Ctrl + M (Win) (On/Disconnect)
- **Toggle Raise-Hand status** ⌘ + E
- **Close dialog box** ⎋ (Mac), Esc (Win)
- **End meeting** ⌘ + \

Adobe Connect Interface

Sharing Screen & Applications

- Share My Screen ❯ Application tab ❯ Terminal for Terminal
- Share menu ❯ Change View ❯ Zoom in for mismatch of screen size/resolution (Participants)
- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application on the original display
- Beware blued hatched rectangles — from other (hidden) windows or contextual menus
- Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- First time: System Preferences ❯ Security & Privacy ❯ Privacy ❯ Accessibility

Adobe Connect

Ending a Meeting

- *Notes for the tutor only*
- **Student:** Meeting › Exit Adobe Connect
- **Tutor:**
- **Recording** Meeting › Stop Recording ✔
- **Remove Participants** Meeting › End Meeting. . . ✔
  - Dialog box allows for message with default message:
  - *The host has ended this meeting. Thank you for attending.*
- **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording* — edit description and name
- **Meeting Information** Meeting › Manage Meeting Information — can access a range of information in Web page.
- **Delete File Upload** Meeting › Manage Meeting Information › Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

Adobe Connect

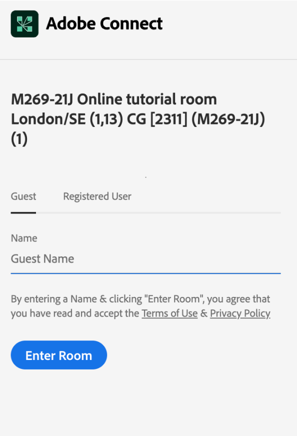
Invite Attendees

- **Provide Meeting URL** Menu › Meeting › Manage Access & Entry › Invite Participants. . .
- **Allow Access without Dialog** Menu › Meeting › Manage Meeting Information provides new browser window with *Meeting Information* Tab bar › Edit Information
- Check *Anyone who has the URL for the meeting can enter the room*
- Default *Only registered users and accepted guests may enter the room*
- **Reverts to default next session but URL is fixed**
- Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- See Start, attend, and manage Adobe Connect meetings and sessions

Adobe Connect

Entering a Room as a Guest (1)

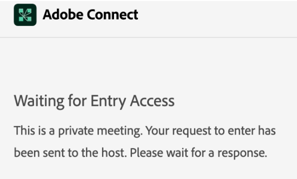
- Click on the link sent in email from the Host
- Get the following on a Web page
- As *Guest* enter your name and click on Enter Room



Adobe Connect

Entering a Room as a Guest (2)

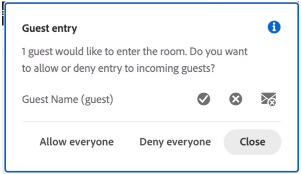
- See the *Waiting for Entry Access* for *Host* to give permission



Adobe Connect

Entering a Room as a Guest (3)

- *Host* sees the following dialog in *Adobe Connect* and grants access



Adobe Connect

Layouts

- **Creating new layouts** example *Sharing* layout
- Menu › Layouts › Create New Layout. . . | Create a New Layout dialog › Create a new blank layout and name it *PMolyMain*
- New layout has no Pods but does have Layouts Bar open (see Layouts menu)
- **Pods**
- Menu › Pods › Share › Add New Share and resize/position — initial name is *Share* n — rename *PMolyShare*
- **Rename Pod** Menu › Pods › Manage Pods. . . | Manage Pods › Select › Rename or Double-click & rename
- Add Video pod and resize/reposition
- Add Attendance pod and resize/reposition
- Add Chat pod — rename it *PMolyChat* — and resize/reposition

Adobe Connect

Layouts

- Dimensions of **Sharing** layout (on 27-inch iMac)
  - Width of Video, Attendees, Chat column 14 cm
  - Height of Video pod 9 cm
  - Height of Attendees pod 12 cm
  - Height of Chat pod 8 cm
- **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)
- **Auxiliary Layouts** name *PMolyAux0n*
  - Create new Share pod
  - Use existing Chat pod
  - Use same Video and Attendance pods

Adobe Connect

Chat Pods

- **Format Chat text**
- Chat Pod > menu icon > My Chat Color
- Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- Note: Color reverts to Black if you switch layouts
- Chat Pod > menu icon > Show Timestamps

Graphics Conversion

PDF to PNG/JPG

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- File › Convert & Modify › Conversion › Convert
- Select files to convert and destination folder
- Click on Start selected Function or ⌘ + ↩

Adobe Connect Recordings

Exporting Recordings

- *Menu bar* 〉 *Meeting* 〉 Preferences 〉 Video
- Aspect ratio 〉 Standard (4:3) (not Wide screen (16:9) default)
- Video quality 〉 Full HD (1080p not High default 480p)
- **Recording** *Menu bar* 〉 *Meeting* 〉 Record Session ✔
- **Export Recording**
- *Menu bar* 〉 *Meeting* 〉 Manage Meeting Information
- *New window* 〉 Recordings 〉 *check Tutorial* 〉 Access Type button
- check Public 〉 check Allow viewers to download
- **Download Recording**
- *New window* 〉 Recordings 〉 *check Tutorial* 〉 Actions 〉 Download File

Commentary 2

Computability

2 Computability

- Description of Turing Machine
- Turing Machine examples
- Computability, Decidability and Algorithms
- Non-computability — Halting Problem
- Reductions and non-computability
- Lambda Calculus (optional)
- Note that the Computability notes are here mainly for reference since the Complexity notes refer to them
- This session is mainly on the Complexity topics

Computability

Ideas of Computation

- The idea of an algorithm and what is effectively computable
- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine. (Unit 7 Section 4)
- See Phil Wadler on computability theory performed as part of the Bright Club at The Strand in Edinburgh, Tuesday 28 April 2015

Computability

Models of Computation

- In automata theory, a *problem* is the question of deciding whether a given string is a member of some particular language
- If Σ is an alphabet, and L is a language over Σ, that is $L \subseteq \Sigma^*$, where Σ^* is the set of all strings over the alphabet Σ then we have a more formal definition of *decision problem*
- Given a string $w \in \Sigma^*$, decide whether $w \in L$
- Example: Testing for a prime number — can be expressed as the language L_p consisting of all binary strings whose value as a binary number is a prime number (only divisible by 1 or itself)
- See Hopcroft (2007, section 1.5.4)

Automate Theory

Alphabets, Strings

- An **Alphabet**, Σ, is a finite, non-empty set of symbols.
- Binary alphabet $\Sigma = \{0, 1\}$
- Lower case letters $\Sigma = \{a, b, \ldots, z\}$
- A **String** is a finite sequence of symbols from some alphabet
- 01101 is a string from the Binary alphabet $\Sigma = \{0, 1\}$
- The **Empty string**, ϵ, contains no symbols
- **Powers**: Σ^k is the set of strings of length k with symbols from Σ
- The set of all strings over an alphabet Σ is denoted Σ^*
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- **Question** Does $\Sigma^0 = \emptyset$? ($\emptyset$ is the empty set)

Automata Theory

Languages

- An **Language**, *L*, is a subset of Σ^*
- The set of binary numerals whose value is a prime
  $\{10, 11, 101, 111, 1011, \ldots\}$
- The set of binary numerals whose value is a square
  $\{100, 1001, 10000, 11001, \ldots\}$

Computability

Church-Turing Thesis & Quantum Computing

- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine.
- **physical Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) by a Universal Turing Machine.
- **strong Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) with polynomial slowdown by a Universal Turing Machine.
- Shor's algorithm (1994) — quantum algorithm for factoring integers — an NP problem that is not known to be P — also not known to be NP-complete and we have no proof that it is not in P
- Reference: Section 4 of Unit 6 & 7 Reader

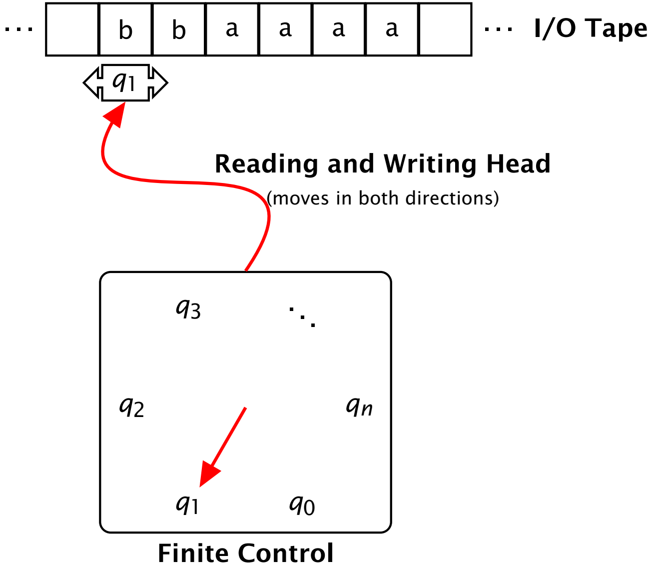
Computability

Turing Machine

- **Finite control** which can be in any of a finite number of *states*
- **Tape** divided into cells, each of which can hold one of a finite number of symbols
- Initially, the **input**, which is a finite-length string of symbols in the *input alphabet*, is placed on the tape
- All other tape cells (extending unbounded left and right) hold a special symbol called *blank*
- A **tape head** which initially is over the leftmost input symbol
- A **move** of the Turing Machine depends on the state and the tape symbol scanned
- A move can change state, write a symbol in the current cell, move left, right or stay
- References: Hopcroft (2007, page 326), Unit 6 & 7 Reader (section 5.3)

Turing Machine Diagram

Turing Machine Diagram



Computability

Turing Machine notation

- Q finite set of states of the finite control
- Σ finite set of *input symbols* (M269 S)
- Γ complete set of *tape symbols* $\Sigma \subset \Gamma$
- δ Transition function (M269 instructions, I)
  $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
  $\delta(q, X) \mapsto (p, Y, D)$
- $\delta(q, X)$ takes a state, q and a tape symbol, X and returns (p, Y, D) where p is a state, Y is a tape symbol to overwrite the current cell, D is a direction, Left, Right or Stay
- q_0 *start state* $q_0 \in Q$
- B *blank symbol* $B \in \Gamma$ and $B \notin \Sigma$
- F set of *final or accepting states* $F \subseteq Q$

Turing Machine Examples

Turing Machine Simulators

- Morphett's Turing machine simulator — the examples below are adapted from here
- Ugarte's Turing machine simulator
- XKCD A Bunch of Rocks — XKCD Explanation

  Image below (will need expanding to be readable)
- The term *state* is used in two different ways:

  The value of the *Finite Control*

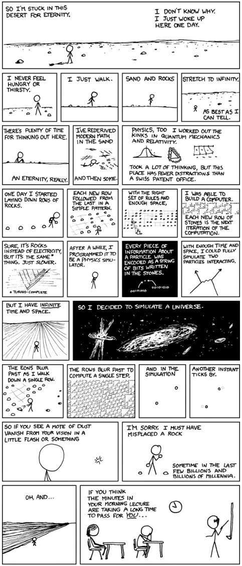
  The overall configuration of *Finite Control* and current contents of the tape

  See Turing Machine: State

  will lead to some confusion

Turing Machine Examples

XKCD A Bunch of Rocks



Turing Machine Examples

Meta-Exercise

- For each of the Turing Machine Examples below, identify

  $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Turing Machine Examples

The Successor Function

- **Input** binary representation of numeral n
- **Output** binary representation of $n+1$
- Example $1010 \mapsto 1011$ and $1011 \mapsto 1100$
- Initial cell: leftmost symbol of n
- **Strategy**
- **Stage A** make the rightmost cell the current cell
- **Stage B** Add 1 to the current cell.
- If the current cell is 0 then replace it with 1 and go to stage C
- If the current cell is 1 replace it with 0 and go to stage B and move Left
- If the current cell is blank, replace it by 1 and go to stage C
- **Stage C** Finish up by making the leftmost cell current

Turing Machine Examples

The Successor Function (2)

- Represent the Turing Machine program as a list of quintuples (q, X, p, Y, D)
- **Stage A**

  $(q_0, 0, q_0, 0, R)$

  $(q_0, 1, q_0, 1, R)$

  (q_0, B, q_1, B, L)
- **Stage B**

  $(q_1, 0, q_2, 1, S)$

  $(q_1, 1, q_1, 0, L)$

  $(q_1, B, q_2, 1, S)$
- **Stage C**

  $(q_2, 0, q_2, 0, L)$

  $(q_2, 1, q_2, 1, L)$

  (q_2, B, q_h, B, R)

Turing Machine Examples

The Successor Function (2a)

- **Exercise** Translate the quintuples (q, X, p, Y, D) into English and check they are the same as the specification
- **Stage A** make the rightmost cell the current cell

  $(q_0, 0, q_0, 0, R)$

  If state q_0 and read symbol 0 then stay in state q_0 write 0, move R

  $(q_0, 1, q_0, 1, R)$

  If state q_0 and read symbol 1 then stay in state q_0 write 1, move R

  (q_0, B, q_1, B, L)

  If state q_0 and read symbol B then state q_1 write B, move L

Turing Machine Examples

The Successor Function (2b)

- **Exercise** Translate the quintuples (q, X, p, Y, D) into English
- **Stage B** Add 1 to the current cell.

  $(q_1, 0, q_2, 1, S)$

  If state q_1 and read symbol 0 then state q_2 write 1, stay

  $(q_1, 1, q_1, 0, L)$

  If state q_1 and read symbol 1 then state q_1 write 0, move L

  $(q_1, B, q_2, 1, S)$

  If state q_1 and read symbol B then state q_2 write 1, stay

Turing Machine Examples

The Successor Function (2c)

- **Exercise** Translate the quintuples (q, X, p, Y, D) into English
- **Stage C** Finish up by making the leftmost cell current

  $(q_2, 0, q_2, 0, L)$

  If state q_2 and read symbol 0 then state q_2 write 0, move L

  $(q_2, 1, q_2, 1, L)$

  If state q_2 and read symbol 1 then state q_2 write 0, move L

  (q_2, B, q_h, B, R)

  If state q_2 and read symbol B then state q_h write B, move R HALT
- Notice that the Turing Machine feels like a series of `if ... then` or `case` statements inside a `while` loop

Turing Machine Examples

The Successor Function (2d) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Turing Machine Examples

The Successor Function (2e) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{q_0, q_1, q_2, q_h\}$
- q_0 finding the rightmost symbol
- q_1 add 1 to current cell
- q_2 move to leftmost cell
- q_h finish
- $\Sigma = \{0, 1\}$
- $\Gamma = \Sigma \cup \{B\}$
- $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$

  $\delta(q, X) \mapsto (p, Y, D)$

  δ is represented as {(q,X,p,Y,D)}

  equivalent to $\{((q, X), (p, Y, D))\}$ *set of pairs*
- q_0 start with leftmost symbol under head, state moving to rightmost symbol
- B is ␣ a visible space
- $F = \{q_h\}$

Turing Machine Examples

The Successor Function (3)

- **Sample Evaluation** $11 \mapsto 100$
- **Representation** $\cdots B X_1 X_2 \cdots X_{i-1}\, q X_i X_{i+1} \cdots X_n B \cdots$

  $q_0 11$

  $1 q_0 1$

  $11 q_0 B$

  $1 q_1 1$

  $q_1 10$

  $q_1 B00$

  $q_2 100$

  $q_2 B100$

  $q_h 100$
- **Exercise** evaluate $1011 \mapsto 1100$

Turing Machine Examples

Instantaneous Description

- **Representation** $\cdots BX_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n B \cdots$
- q is the state of the TM
- The head is scanning the symbol X_i
- Leading or trailing blanks B are (usually) not shown unless the head is scanning them
- $\vdash_M$ denotes one move of the TM M
- $\vdash_M^*$ denotes zero or more moves
- $\vdash$ will be used if the TM M is understood
- If (q, X_i, p, Y, L) denotes a TM move then
  $X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash_M X_1 \cdots X_{i-2} p X_{i-1} Y \cdots X_n$

Turing Machine Examples

The Binary Palindrome Function

- **Input** binary string *s*
- **Output** YES if palindrome, NO otherwise
- Example 1010 $\mapsto$ *NO* and 1001 $\mapsto$ *YES*
- Initial cell: leftmost symbol of *s*
- **Strategy**
- **Stage A** read the leftmost symbol
- If blank then accept it and go to stage D otherwise erase it
- **Stage B** find the rightmost symbol
- If the current cell matches leftmost recently read then erase it and go to stage C
- Otherwise reject it and go to stage E
- **Stage C** return to the leftmost symbol and stage A
- **Stage D** print YES and halt
- **Stage E** erase the remaining string and print NO

Turing Machine Examples

The Binary Palindrome Function (2)

- Represent the Turing Machine program as a list of quintuples (q, X, p, Y, D)
- **Stage A** read the leftmost symbol

  $(q_0, 0, q_{1_o}, B, R)$

  $(q_0, 1, q_{1_i}, B, R)$

  (q_0, B, q_5, B, S)
- **Stage B** find rightmost symbol

  $(q_{1_o}, B, q_{2_o}, B, L)$

  $(q_{1_o}, *, q_{1_o}, *, R)$ * is a wild card, matches anything

  $(q_{1_i}, B, q_{2_i}, B, L)$

  $(q_{1_i}, *, q_{1_i}, *, R)$

Turing Machine Examples

The Binary Palindrome Function (3)

- **Stage B** check

  $(q_{2_o}, 0, q_3, B, L)$

  (q_{2_o}, B, q_5, B, S)

  $(q_{2_o}, *, q_6, *, S)$

  $(q_{2_i}, 1, q_3, B, L)$

  (q_{2_i}, B, q_5, B, S)

  $(q_{2_i}, *, q_6, *, S)$

- **Stage C** return to the leftmost symbol and stage A

  (q_3, B, q_5, B, S)

  $(q_3, *, q_4, *, L)$

  (q_4, B, q_0, B, R)

  $(q_4, *, q_4, *, L)$

Turing Machine Examples

The Binary Palindrome Function (4)

- **Stage D** accept and print YES

  $(q_5, *, q_{5_a}, Y, R)$

  $(q_{5_a}, *, q_{5_b}, E, R)$

  $(q_{5_b}, *, q_7, S, S)$

- **Stage E** erase the remaining string and print NO

  (q_6, B, q_{6_a}, N, R)

  $(q_6, *, q_6, B, L)$

  $(q_{6_a}, *, q_7, O, S)$

- Finish

  (q_7, B, q_h, B, R)

  $(q_7, *, q_7, *, L)$

Turing Machine Examples

The Binary Palindrome Function (3a) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Turing Machine Examples

The Binary Palindrome Function (3b) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{q_0, q_{1_o}, q_{1_i}, q_{2_o}, q_{2_i}, q_3, q_4, q_5, q_{5_a}, q_{5_b}, q_6, q_{6_a}, q_7, q_h\}$
- q_0 read leftmost symbol
- q_{1_o}, q_{1_i} find rightmost symbol looking for 0 or 1
- q_{2_o}, q_{2_i} check, confirm or reject
- q_3, q_4 check finish or move to start
- q_5, q_6, q_7 print YES or NO and finish
- q_h finish
- $\Sigma = \{0, 1\}$
- $\Gamma = \Sigma \cup \{B, Y, E, S, N, O\}$
- $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
  $\delta(q, X) \mapsto (p, Y, D)$
  δ is represented as {(q,X,p,Y,D)}
  equivalent to $\{((q, X), (p, Y, D))\}$ *set of pairs*
- Start with leftmost symbol under head, state q_0
- B is ␣ a visible space
- $F = \{q_h\}$

Turing Machine Examples

The Binary Palindrome Function (4)

- **Sample Evaluation** 101 $\mapsto$ YES

  $q_0 101 \vdash Bq_{1_i}01 \vdash B0q_{1_i}1 \vdash B01q_{1_i}B$

  $\vdash B0q_{2_i}1$

  $\vdash Bq_3 0B \vdash q_4 B0B$

  $\vdash Bq_0 0B \vdash BBq_{1_o}B$

  $\vdash Bq_{2_o}BB$

  $\vdash Bq_5 BB \vdash Yq_{5_a}B \vdash YEq_{5_b}B \vdash YEq_7 S$

  $\vdash Yq_7 ES \vdash Bq_7 YES \vdash q_7 BYES \vdash q_h YES$

- **Exercise** Evaluate 110 $\mapsto$ NO

Turing Machine Examples

Binary Addition Example

- **Input** two binary numerals separated by a single space $n1 \; n2$
- **Output** binary numeral which is the sum of the inputs
- Example $110110 + 101011 \mapsto 1100001$
- Initial cell: leftmost symbol of $n1 \; n2$
- **Insight** look at the arithmetic algorithm

␣	1	1	0	1	1	0
␣	1	0	1	0	1	1
1	1	0	0	0	0	1

- **Discussion** how can we overwrite the first number with the result and remember how far we have gone ?

Turing Machine Examples

Binary Addition Example — Arithmetic Reinvented

␣	1	1	0	1	1	0
␣	1	0	1	0	1	1
␣	1	1	0	1	1	y
␣	1	0	1	0	1	␣
␣	1	1	1	0	x	y
␣	1	0	1	0	␣	␣
␣	1	1	1	x	x	y
␣	1	0	1	␣	␣	␣
1	0	0	x	x	x	y
␣	1	0	␣	␣	␣	␣
1	0	x	x	x	x	y
␣	1	␣	␣	␣	␣	␣
1	y	x	x	x	x	y
␣	␣	␣	␣	␣	␣	␣
1	1	0	0	0	0	1

Turing Machine Examples

Binary Addition Example (2)

- **Input** two binary numerals separated by a single space
  $n1 \;\; n2$
- **Output** binary numeral which is the sum of the inputs
- Example $110110 + 101011 \mapsto 1100001$
- Initial cell: leftmost symbol of $n1 \;\; n2$
- **Strategy**
- **Stage A** find the rightmost symbol
  If the symbol is 0 erase and go to stage Bx
  If the symbol is 1 erase go to stage By
  If the symbol is blank go to stage F
  dealing with each digit in $n2$
  if no further digits in $n2$ go to final stage
- **Stage Bx** Move left to a blank go to stage Cx
- **Stage By** Move left to a blank go to stage Cy
  moving to $n1$

Turing Machine Examples

Binary Addition Example (3)

- **Stage Cx** Move left to find first 0, 1 or B

  Turn 0 or B to X, turn 1 to Y and go to stage A

  adding 0 to a digit finalises the result (no *carry one*)

- **Stage Cy** Move left to find first 0, 1 or B

  Turn 0 or B to 1 and go to stage D

  Turn 1 to 0, move left and go to stage Cy

  dealing with the *carry one* in school arithmetic

- **Stage D** move right to X, Y or B and go to stage E
- **Stage E** replace 0 by X, 1 by Y, move right and go to Stage A

  finalising the value of a digit resulting from a *carry*

- **Stage F** move left and replace X by 0, Y by 1 and at B halt

Turing Machine Examples

Binary Addition Example (4)

- Represent the Turing Machine program as a list of quintuples (q, X, p, Y, D)
- **Stage A** find the rightmost symbol

  (q_0, B, q_1, B, R)

  $(q_0, *, q_0, *, R)$ * is a wild card, matches anything

  (q_1, B, q_2, B, L)

  $(q_1, *, q_1, *, R)$

  $(q_2, 0, q_{3_x}, B, L)$

  $(q_2, 1, q_{3_y}, B, L)$

  (q_2, B, q_7, B, L)

Turing Machine Examples

Binary Addition Example (5)

- **Stage Bx** move left to blank

  $(q_{3_x}, B, q_{4_x}, B, L)$

  $(q_{3_x}, *, q_{3_x}, *, L)$

- **Stage By** move left to blank

  $(q_{3_y}, B, q_{4_y}, B, L)$

  $(q_{3_y}, *, q_{3_y}, *, L)$

- **Stage Cx** move left to 0, 1, or blank

  $(q_{4_x}, 0, q_0, x, R)$

  $(q_{4_x}, 1, q_0, y, R)$

  (q_{4_x}, B, q_0, x, R)

  $(q_{4_x}, *, q_{4_x}, *, L)$

Turing Machine Examples

Binary Addition Example (6)

- **Stage Cy** move left to 0, 1, or blank

  $(q_{4_y}, 0, q_5, 1, S)$

  $(q_{4_y}, 1, q_{4_y}, 0, L)$

  $(q_{4_y}, B, q_5, 1, S)$

  $(q_{4_y}, *, q_{4_y}, *, L)$

- **Stage D** move right to x, y or B

  (q_5, x, q_6, x, L)

  (q_5, y, q_6, y, L)

  (q_5, B, q_6, B, L)

  $(q_5, *, q_5, *, R)$

- **Stage E** replace 0 by x, 1 by y

  $(q_6, 0, q_0, x, R)$

  $(q_6, 1, q_0, y, R)$

Turing Machine Examples

Binary Addition Example (7)

- **Stage F** replace x by 0, y by 1

  $(q_7, x, q_7, 0, L)$

  $(q_7, y, q_7, 1, L)$

  (q_7, B, q_h, B, R)

  $(q_7, *, q_7, *, L)$

- **Exercise** Evaluate $11 + 10 \mapsto 101$

Turing Machine Examples

The Binary Addition Function (7a) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Turing Machine Examples

The Binary Addition Function (7b) — Meta-Exercise

- Identify $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{q_0, q_1, q_2, q_{3_x}, q_{3_y}, q_{4_x}, q_{4_y}, q_5, q_6, q_7, q_h\}$
- q_0, q_1, q_2 find rightmost symbol of second number
- q_{3_x}, q_{3_y} move left to inter-number blank
- q_{4_x}, q_{4_y} move left to 0, 1 or blank
- q_5 move right to x, y or B
- q_6 replace 0 by x, 1 by y and move right
- q_7 replace x by 0, y by 1 and move left
- q_h finish
- $\Sigma = \{0, 1\}$
- $\Gamma = \Sigma \cup \{B, x, y\}$
- $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
  $\delta(q, X) \mapsto (p, Y, D)$
  δ is represented as $\{(q, X, p, Y, D)\}$
  equivalent to $\{((q, X), (p, Y, D))\}$ *set of pairs*
- Start with leftmost symbol under head, state q_0
- B is ␣ a visible space
- $F = \{q_h\}$

Turing Machine Examples

Binary Addition Example (8a)

- **Exercise** Evaluate $11 + 10 \mapsto 101$
- **Stage A** find the rightmost symbol

  $BBq_0 11B10B$ Note space symbols B at start and end

$\vdash BB1q_0 1B10B$

$\vdash BB11q_0 B10B$

$\vdash BB11Bq_1 10B$

$\vdash BB11B1q_1 0B$

$\vdash BB11B10q_1 B$

$\vdash BB11B1q_2 0B$

$\vdash BB11Bq_{3_x} 1BB$

- **Stage Bx** move left to blank

$\vdash B11q_{3_x} B1BB$

- **Stage Cx** move left to 0, 1, or blank

$\vdash BB1q_{4_x} 1B1BB$

$\vdash BB1Yq_0 B1BB$

Turing Machine Examples

Binary Addition Example (8b)

- **Exercise** Evaluate $11 + 10 \mapsto 101$ (contd)
- **Stage A** find the rightmost symbol

$\vdash$ $BB1BYBq_1 1BB$

$\vdash$ $BB1YB1q_1 BB$

$\vdash$ $BB1YBq_2 1BB$

$\vdash$ $BB1Yq_{3_y} BBBB$

- **Stage Cy** move left to 0, 1, or blank

$\vdash$ $BB1q_{4_y} YBBBB$

$\vdash$ $BBq_{4_y} 1YBBBB$

$\vdash$ $Bq_{4_y} B0YBBBB$

$\vdash$ $Bq_5 10YBBBB$

- **Stage D** move right to x, y or B

$\vdash$ $Bq_5 0YBBBB$

$\vdash$ $B0q_5 YBBBB$

$\vdash$ $Bq_6 0YBBBB$

- **Stage E** replace 0 by x, 1 by y

$\vdash$ $B1Xq_0 YBBBB$

Turing Machine Examples

Binary Addition Example (8c)

- **Exercise** Evaluate $11 + 10 \mapsto 101$ (contd)
- **Stage A** find the rightmost symbol

$\vdash$ $B1XYq_0BBBB$

$\vdash$ $B1XYBq_1BBB$

$\vdash$ $B1XYq_2BBBB$

$\vdash$ $B1Xq_7YBBBB$

- **Stage F** replace x by 0, y by 1

$\vdash$ $B1q_7X1BBBB$

$\vdash$ $Bq_7101BBBB$

$\vdash$ $Bq_7B101BBBB$

$\vdash$ $Bq_h101BBBB$

- This is mimicking what you learnt to do on paper as a child! Real *step-by-step* instructions
- See Morphett's Turing machine simulator for more examples (takes too long by hand!)

Computability

Universal Turing Machine

- Universal Turing Machine, U, is a Turing Machine that can simulate any arbitrary Turing machine, M
- Achieves this by encoding the transition function of M in some standard way
- The input to U is the encoding for M followed by the data for M
- See Turing machine examples

Computability

Decidability

- **Decidable** — there is a TM that will halt with yes/no for a decision problem — that is, given a string *w* over the alphabet of *P* the TM with halt and return yes.no the string is in the language *P* (same as *recursive* in Recursion theory — old use of the word)
- **Semi-decidable** — there is a TM will halt with yes if some string is in *P* but may loop forever on some inputs (same as *recursively enumerable*) — *Halting Problem*
- **Highly-undecidable** — no outcome for any input — *Totality, Equivalence Problems*

Computability

Undecidable Problems

- **Halting problem** — the problem of deciding, given a program and an input, whether the program will eventually halt with that input, or will run forever — term first used by Martin Davis 1952
- **Entscheidungsproblem** — the problem of deciding whether a given statement is provable from the axioms using the rules of logic — shown to be undecidable by Turing (1936) by reduction from the *Halting problem* to it
- **Type inference and type checking** in the second-order lambda calculus (important for functional programmers, Haskell, GHC implementation)
- Undecidable problem — see link to list

Computability

Halting Problem — Sketch Proof (1)

- **Halting problem** — is there a program that can determine if any arbitrary program will halt or continue forever ?
- Assume we have such a program (Turing Machine) `h(f,x)` that takes a program `f` and input `x` and determines if it halts or not

```
h(f,x)
  = if f(x) runs forever
      return True
    else
      return False
```

- We shall prove this cannot exist by contradiction

Computability

Halting Problem — Sketch Proof (2)

- Now invent two further programs:
- q(f) that takes a program f and runs h with the input to f being a copy of f
- r(f) that runs q(f) and halts if q(f) returns True, otherwise it loops

```
q(f)
  = h(f,f)

r(f)
  = if q(f)
      return
    else
      while True: continue
```

- What happens if we run r(r) ?
- If it loops, q(r) returns True and it does not loop — contradiction.
- Scooping theLoop Snooper: A proof that the Halting Problem is undecidable Geoffrey K Pullum (21 May 2024)

Computability

Why undecidable problems must exist

- A *problem* is really membership of a string in some language
- The number of different languages over any alphabet of more than one symbol is uncountable
- Programs are finite strings over a finite alphabet (ASCII or Unicode) and hence countable.
- There must be an infinity (big) of problems more than programs.
- **Computational problem** — defined by a function
- **Computational problem is computable** if there is a Turing machine that will calculate the function.

Computability

Computability and Terminology (1)

- The idea of an *algorithm* dates back 3000 years to Euclid, Babylonians...
- In the 1930s the idea was made more formal: which *functions* are *computable*?
- A *function* is a set of pairs $f = \{(x, f(x)) : x \in X \wedge f(x) \in Y\}$ with the *function property*
- *Function property:* $(a, b) \in f \wedge (a, c) \in f \Rightarrow b == c$
- *Function property:* Same input implies same output
- Note that maths notation is deeply inconsistent here — see Function and History of the function concept
- *What do we mean by computing a function — an algorithm ?*

Functions

Relation and Rule

- The idea of function as a set of pairs (Binary relation) with the function property (each element of the domain has at most one element in the co-domain) is fairly recent — see History of the function concept
- School maths presents us with function as rule to get from the input to the output
- Example: the `square` function: `square` $x = x \times x$
- But lots of rules (or algorithms) can implement the same function
- `square1` x `=` x`^2`
- `square2` x `=` $\overbrace{x + \cdots + x}^{x \texttt{ times}}$ if x is integer

Computability

Computability and Terminology (2)

- In the 1930s three definitions:
- λ-Calculus, simple semantics for computation — Alonzo Church
- General recursive functions — Kurt Gödel
- Universal (Turing) machine — Alan Turing
- Terminology:
  - Recursive, recursively enumerable — Church, Kleene
  - Computable, computably enumerable — Gödel, Turing
  - Decidable, semi-decidable, highly undecidable
  - In the 1930s, *computers* were *human*
  - Unfortunate choice of terminology
- Turing and Church showed that the above three were equivalent
- Church-Turing thesis — function is intuitively computable if and only if Turing machine computable

Computability

Reducing one problem to another

- To reduce problem P_1 to P_2, invent a construction that converts instances of P_1 to P_2 that have the same answer. That is:
  - any string in the language P_1 is converted to some string in the language P_2
  - any string over the alphabet of P_1 that is not in the language of P_1 is converted to a string that is not in the language P_2
- With this construction we can solve P_1
  - Given an instance of P_1, that is, given a string w that may be in the language P_1, apply the construction algorithm to produce a string x
  - Test whether x is in P_2 and give the same answer for w in P_1

Computability

Problem Reduction

- **Problem Reduction — Ordinary Example**
- Want to phone Alice but don't have her number
- You know that Bill has her number
- So *reduce* the problem of finding Alice's number to the problem of getting hold of Bill

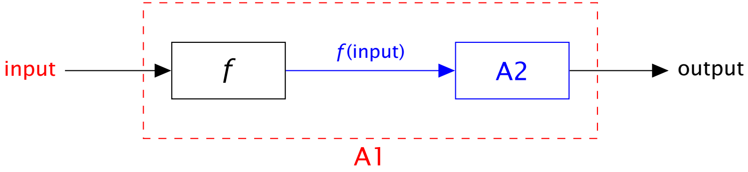
Computability

Direction of Reduction

- The direction of reduction is important
- If we can reduce P_1 to P_2 then (in some sense) P_2 is at least as hard as P_1 (since a solution to P_2 will give us a solution to P_1)
- So, if P_2 is decidable then P_1 is decidable
- To show a problem is undecidable we have to reduce from an known undecidable problem to it
- $\forall x(\mathrm{dp}_{P_1}(x) = \mathrm{dp}_{P_2}(\mathrm{reduce}(x)))$
- Since, if P_1 is undecidable then P_2 is undecidable

Reductions & Non-Computable

Reductions

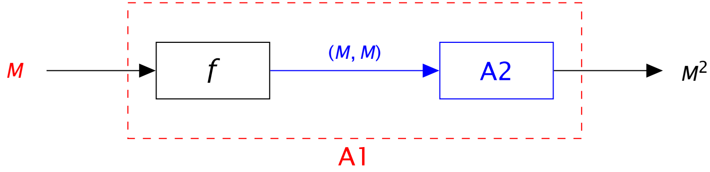


- A *reduction* of problem P_1 to problem P_2
  - transforms inputs to P_1 into inputs to P_2
  - runs algorithm $A2$ (which solves P_2) and
  - interprets the outputs from $A2$ as answers to P_1
- More formally: A problem P_1 is *reducible* to a problem P_2 if there is a function f that takes any input x to P_1 and transforms it to an input $f(x)$ of P_2

  such that the solution of P_2 on $f(x)$ is the solution of P_1 on x

Reductions & Non-Computible

Example: Squaring a Matrix



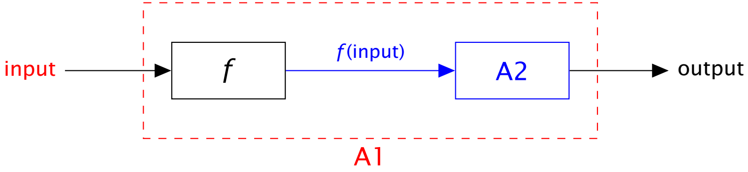
- Given an algorithm ($A2$) for matrix multiplication (P_2)
  - Input: pair of matrices, (M_1, M_2)
  - Output: matrix result of multiplying M_1 and M_2
- P_1 is the problem of squaring a matrix
  - Input: matrix M
  - Output: matrix M^2
- Algorithm $A1$ has

  $f(M) = (M, M)$

  uses $A2$ to calculate $M \times M = M^2$

Reductions & Non-Computable

Non-Computable Problems



- If P_2 is computable ($A2$ exists) then P_1 is computable (f being simple or polynomial)
- Equivalently If P_1 is non-computable then P_2 is non-computable
- **Exercise:** show $B \to A \equiv \neg A \to \neg B$

Reductions & Non-Computable

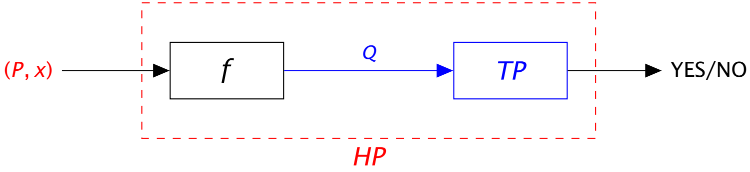
Contrapositive

- Proof by Contrapositive
- $B \rightarrow A \equiv \neg B \vee A$ by truth table or equivalences

  $\equiv \neg(\neg A) \vee \neg B$ commutativity and negation laws

  $\equiv \neg A \rightarrow \neg B$ equivalences
- Common error: switching the order round

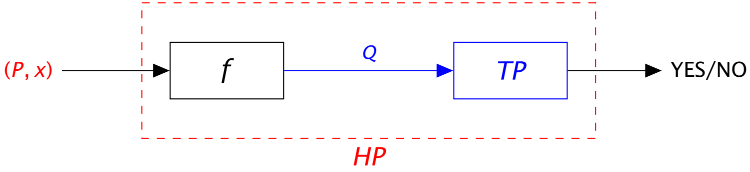
Totality Problem



- **Totality Problem**
  - Input: program Q
  - Output: YES if Q terminates for all inputs else NO
- Assume we have algorithm TP to solve the Totality Problem
- Now reduce the Halting Problem to the Totality Problem

Reductions & Non-Computable

Totality Problem



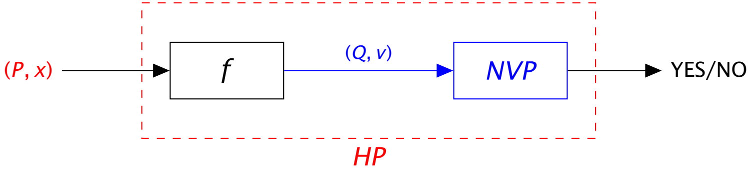
- Define f to transform inputs to HP to TP pseudo-Python

```python
def f(P,x) :
  def Q(y):
    # ignore y
    P(x)
  return Q
```

- Run TP on Q
  - If TP returns YES then P halts on x
  - If TP returns NO then P does not halt on x
- We have *solved* the Halting Problem — contradiction

Reductions & Non-Computable

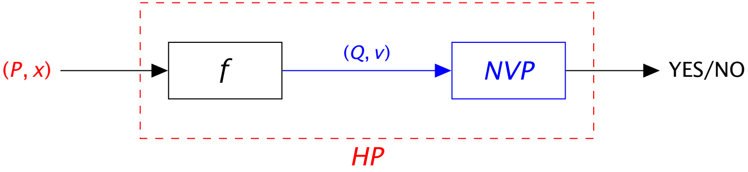
Negative Value Problem



- **Negative Value Problem**
  - Input: program Q which has no input and variable v used in Q
  - Output: YES if v ever gets assigned a negative value else NO
- Assume we have algorithm NVP to solve the Negative Value Problem
- Now reduce the Halting Problem to the Negative Value Problem

Reductions & Non-Computable

Negative Value Problem



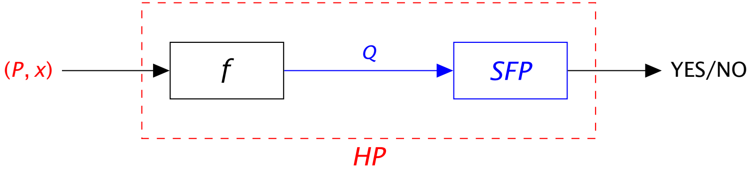
- Define f to transform inputs to HP to NVP pseudo-Python

```
def f(P,x) :
  def Q(y):
    # ignore y
    P(x)
    v = -1
  return (Q,var(v))
```

- Run *NVP* on $(Q, var(v))$ `var(v)` gets the variable name
  - If *NVP* returns YES then P halts on x
  - If *NVP* returns NO then P does not halt on x
- We have *solved* the Halting Problem — contradiction

Reductions & Non-Computable

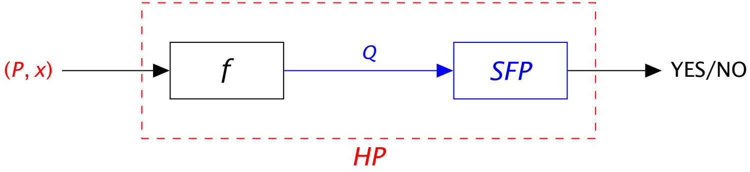
Squaring Function Problem



- **Squaring Function Problem**
  - Input: program Q which takes an integer, y
  - Output: YES if Q always returns the square of y else NO
- Assume we have algorithm *SFP* to solve the Squaring Function Problem
- Now reduce the Halting Problem to the Squaring Function Problem

Reductions & Non-Computable

Squaring Function Problem



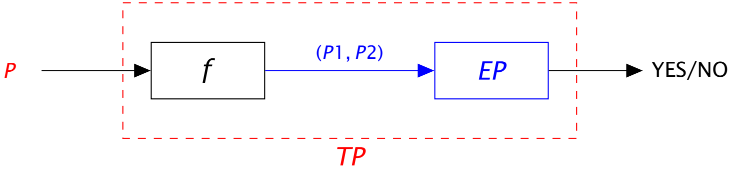
- Define f to transform inputs to HP to SFP pseudo-Python

```
def f(P,x) :
  def Q(y):
    P(x)
    return y * y
  return Q
```

- Run *SFP* on *Q*
  - If *SFP* returns YES then *P* halts on *x*
  - If *SFP* returns NO then *P* does not halt on *x*
- We have *solved* the Halting Problem — contradiction

Reductions & Non-Computable

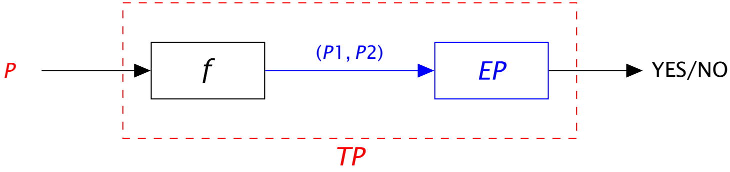
Equivalence Problem



- **Equivalence Problem**
  - Input: two programs $P1$ and $P2$
  - Output: YES if $P1$ and $P2$ solve the same problem (same output for same input) else NO
- Assume we have algorithm EP to solve the Equivalence Problem
- Now reduce the Totality Problem to the Equivalence Problem

Reductions & Non-Computable

Equivalence Problem



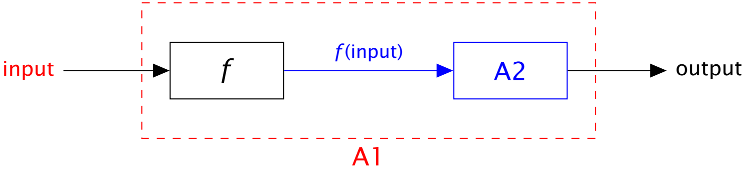
- Define f to transform inputs to TP to EP pseudo-Python

```
def f(P) :
  def P1(x):
    P(x)
    return "Same␣string"
  def P2(x)
    return "Same␣string"
  return (P1,P2)
```

- Run EP on $(P1, P2)$
  - If EP returns YES then P halts on all inputs
  - If EP returns NO then P does not halt on all inputs
- We have *solved* the Totality Problem — contradiction

Reductions & Non-Computable

Rice's Theorem



- Rice's Theorem all non-trivial, semantic properties of programs are undecidable. H G Rice 1951 PhD Thesis
- Equivalently: For any non-trivial property of partial functions, no general and effective method can decide whether an algorithm computes a partial function with that property.
- A property of partial functions is called trivial if it holds for all partial computable functions or for none.

Reductions & Non-Computable

Rice's Theorem

- Rice's Theorem and computability theory
- Let S be a set of languages that is nontrivial, meaning
  - there exists a Turing machine that recognizes a language in S
  - there exists a Turing machine that recognizes a language not in S
- Then, it is undecidable to determine whether the language recognized by an arbitrary Turing machine lies in S.
- This has implications for compilers and virus checkers
- Note that Rice's theorem does not say anything about those properties of machines or programs that are not also properties of functions and languages.
- For example, whether a machine runs for more than 100 steps on some input is a decidable property, even though it is non-trivial.

Lambda Calculus

Motivation

- Lambda Calculus is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution
- Lambda calculus is Turing complete — it can simulate any Turing machine
- Introduced by Alonzo Church in 1930s
- Basis of functional programming languages — Lisp, Scheme, ISWIM, ML, SASL, KRC, Miranda, Haskell, Scala, F#. . .
- **Note** this is not part of M269 but may help understand ideas of computability

Functions

Binding and Substitution

- School maths introduces functions as
  $f(x) = 3x^2 + 4x + 5$
- Substitution: $f(2) = 3 \times 2^2 + 4 \times 2 + 5 = 25$
- Generalise: $f(x) = ax^2 + bx + c$
- What is wrong with the following:
- $f(a) = a \times a^2 + b \times a + c$
- The ideas of free and bound variables and substitution

Expressions

Evaluation Strategies (a)

- In evaluating an expression we have choices about the order in which we evaluate subterms
- Some choices may involve more work than others but the Church-Rosser theorem ensures that if the evaluation terminates then all choices get to the same answer
- The second edition of a famous book on Functional programming — Bird (1998, Ex 1.2.2, page 6) *Introduction to Functional Programming using Haskell* — had the following exercise:
- How many ways can you evaluate $(3 + 7)^2$

  List the evaluations and assumptions
- The first edition — Bird and Wadler (1988, Ex 1.2.1, page 6) *Introduction to Functional Programming* — had the exercise:
- How many ways can you evaluate $\left((3 + 7)^2\right)^2$

Expressions

Evaluation Strategies (b)

- How many ways can you evaluate $(3+7)^2$

  List the evaluations and assumptions
- **Answer** 3 ways
- **Reducible expressions** (redexes)

  $x^2 \rightarrow x \times x$ where x is a term

  $a + b$ where a and b are numbers

  $x \times y$ where x and y are numbers

```
[sqr (3+7),((3+7)*(3+7)),((3+7)*10),(10*10),100]
[sqr (3+7),((3+7)*(3+7)),(10*(3+7)),(10*10),100]
[sqr (3+7),sqr 10,(10*10),100]
```

- The assumed redexes do not include distributive laws

  $(a+b) \times (x+y) \rightarrow a \times x + a \times y + b \times x + b \times y$
- This would increase the number of different evaluations

Expressions

Evaluation Strategies (c)

- How many ways can you evaluate $\left((3+7)^2\right)^2$
- **Answer** 547 ways

```
1 [sqr sqr (3+7),(sqr (3+7)*sqr (3+7)),(sqr (3+7)*((3+7)*(3+7))), (sqr (3+7)*((3+7)*10
2 [sqr sqr (3+7),(sqr (3+7)*sqr (3+7)),(sqr(3+7)*((3+7)*(3+7))), (sqr (3+7)*((3+7)*10)
```

```
546 [sqr sqr (3+7),sqr sqr 10,sqr (10*10), ((10*10)*(10*10)),(100*(10*10)),(100*100),100
547 [sqr sqr (3+7),sqr sqr 10,sqr (10*10),sqr 100,(100*100),10000]
```

- Enumerating all 547 ways may have taken some concentration

Expressions

Evaluation Strategies (d)

- The actual Evaluation strategy used by a particular programming language implementation may have optimisations which make an evaluation which looks costly to be somewhat cheaper
- For example, the Haskell implementation GHC optimises the evaluation of common subexpressions so that (3+7) will be evaluated only once

```
[sqr sqr (3+7),(sqr (3+7)*sqr (3+7)),(sqr (3+7)*((3+7)*(3+7))), (sqr (3+7)*((3+7)*10)
[sqr sqr (3+7),(sqr (3+7)*sqr (3+7)),(sqr(3+7)*((3+7)*(3+7))), (sqr (3+7)*((3+7)*10))
```

Lambda Calculus

Optional Topic

- M269 Unit 6/7 Reader *Logic and the Limits of Computation* alludes to other formalisations with equal power to a Turing Machine (pages 81 and 87)
- The *Reader* mentions Alonzo Church and his 1930s formalism (page 87, but does not give any detail)
- The notes in this section are optional and for comparison with the Turing Machine material
- Turing machine: explicit memory, state and implicit loop and case/if statement
- Lambda Calculus: function definition and application, explicit rules for evaluation (and transformation) of expressions, explicit rules for substitution (for function application)
- Lambda calculus reduction workbench
- Lambda Calculus Calculator

Lambda Calculus

Lambda Terms

- A **variable**, x, is a lambda term
- If M is a lambda term and x is a variable, then $(\lambda x.M)$ is a lambda term — a **lambda abstraction** or function definition
- If M and N are lambda terms, the $(M\ N)$ is lambda term — an **application**
- Nothing else is a lambda term

Lambda Calculus

Lambda Terms — Notational Conveniences

- Outermost parentheses are omitted $(M\ N) \equiv M\ N$
- Application is left associative $((M\ N)\ P) \equiv M\ N\ P$
- The body of an abstraction extends as far right as possible, subject to scope limited by parentheses
- $\lambda x.M\ N \equiv \lambda x.(M\ N)$ and not $(\lambda x.M)\ N$
- $\lambda x.\lambda y.\lambda z.M \equiv \lambda x\, y\, z.M$

Lambda Calculus Semantics

- What do we mean by *evaluating an expression* ?
- To evaluate $(\lambda x.M)N$
- Evaluate M with x replaced by N
- This rule is called β-reduction
- $(\lambda x.M)N \underset{\beta}{\rightarrow} M[x := N]$
- $M[x := N]$ is M with occurrences of x replaced by N
- This operation is called *substitution* — see rules below

Lambda Calculus

β-Reduction Examples

- $(\lambda x.x)z \rightarrow z$
- $(\lambda x.y)z \rightarrow y$
- $(\lambda x.x\,y)z \rightarrow z\,y$

  a function that applies its argument to y
- $(\lambda x.x\,y)(\lambda z.z) \rightarrow (\lambda z.z)y \rightarrow y$
- $(\lambda x.\lambda y.x\,y)z \rightarrow \lambda y.z\,y$

  A *curried* function of two arguments — applies first argument to second
- *currying* replaces $f(x, y)$ with $(f\,x)y$ — nice notational convenience — gives *partial application* for free

Lambda Calculus

Substitution

- To define *substitution* use recursion on the structure of terms
- $x[x := N] \equiv N$
- $y[x := N] \equiv y$
- $(P\,Q)[x := N] \equiv (P[x := N])\,(Q[x := N])$
- $(\lambda x.M)[x := N] = \lambda x.M$

  In $(\lambda x.M)$, the x is a formal parameter and thus a local variable, different to any other
- $(\lambda y.M)[x := N] =$ what?
- Look back at the school maths example above — a subtle point

Lambda Calculus
Substitution (2)

- Renaming *bound variables consistently is allowed*
- $\lambda x.x \equiv \lambda y.y \equiv \lambda z.z$
- $\lambda y.\lambda x.y \equiv \lambda z.\lambda x.z$
- This is called *α-conversion*
- $(\lambda x.\lambda y.x\, y)\, y \rightarrow (\lambda x.\lambda z.x\, z)\, y \rightarrow \lambda z.y\, z$

Lambda Calculus

Substitution (3)

- **Bound and Free Variables**
- $BV(x) = \emptyset$
- $BV(\lambda x.M) = BV(M) \cup \{x\}$
- $BV(M\,N) = BV(M) \cup BV(N)$
- $FV(x) = \{x\}$
- $FV(\lambda x.M) = FV(M) - \{x\}$
- $FV(M\,N) = FV(M) \cup FV(N)$
- The above is a formalisation of school maths
- A Lambda term with no free variables is said to be *closed* — such terms are also called **combinators** — see Combinator and Combinatory logic (Hankin, 2004, page 10)
- **α-conversion**
- $\lambda x.M \underset{\alpha}{\rightarrow} \lambda y.M[x := y]$ if $y \notin FV(M)$

Lambda Calculus

Substitution (4)

- β-reduction final rule
- $(\lambda y.M)[x := N] = \lambda y.M$ if $x \notin FV(M)$
- $(\lambda y.M)[x := N] = \lambda y.M[x := N]$

  if $x \in FV(M)$ and $y \notin FV(N)$
- $(\lambda y.M)[x := N] = \lambda z.M[y := z][x := N]$

  if $x \in FV(M)$ and $y \in FV(N)$

  z is chosen to be first variable $z \notin FV(N\,M)$
- This is why you cannot go $f(a)$ when given
- $f(x) = ax^2 + bx + c$
- School maths — but made formal

Lambda Calculus

Rules Summary — Conversion

- α-**conversion** renaming bound variables
- $\lambda x.M \underset{\alpha}{\rightarrow} \lambda y.M[x := y]$ if $y \notin FV(M)$
- β-**conversion** function application
- $(\lambda x.M)N \underset{\beta}{\rightarrow} M[x := N]$
- η-**conversion** extensionality
- $\lambda x.F\,x \underset{\eta}{\rightarrow} F$ if $x \notin FV(F)$

Lambda Calculus

Rules Summary — Substitution

1. $x[x := N] \equiv N$
2. $y[x := N] \equiv y$
3. $(P\,Q)[x := N] \equiv (P[x := N])\,(Q[x := N])$
4. $(\lambda x.M)[x := N] = \lambda x.M$
5. $(\lambda y.M)[x := N] = \lambda y.M$ if $x \notin FV(M)$
6. $(\lambda y.M)[x := N] = \lambda y.M[x := N]$
   if $x \in FV(M)$ and $y \notin FV(N)$
7. $(\lambda y.M)[x := N] = \lambda z.M[y := z][x := N]$
   if $x \in FV(M)$ and $y \in FV(N)$
   z is chosen to be first variable $z \notin FV(N\,M)$

Lambda Calculus

Lambda Calculus Encodings

- So what does this formalism get us ?
- The Lambda Calculus is Turing complete
- We can encode any computation (if we are clever enough)
- Booleans and propositional logic
- Pairs
- Natural numbers and arithmetic
- Looping and recursion

Lambda Calculus Encodings

Booleans and Propositional Logic

- True $= \lambda x.\lambda y.x$
- False $= \lambda x.\lambda y.y$
- IF a THEN b ELSE $c \equiv a\, b\, c$
- IF True THEN b ELSE $c \rightarrow (\lambda x.\lambda y.x)\, b\, c$
- $\quad \rightarrow (\lambda y.b)\, c \rightarrow b$
- IF False THEN b ELSE $c \rightarrow (\lambda x.\lambda y.y)\, b\, c$
- $\quad \rightarrow (\lambda y.y)\, c \rightarrow c$

Lambda Calculus Encodings

Booleans and Propositional Logic (2)

- Not $= \lambda x.((x\,\text{False})\text{True})$
- Not x = IF x THEN False ELSE True
- Exercise: evaluate Not True
- And $= \lambda x.\lambda y.((x\,y)\ \text{False})$
- And x y = IF x THEN y ELSE False
- Exercise: evaluate And True False
- Or $= \lambda x.\lambda y.((x\ \ \text{True}\ )\ y)$
- Or x y = IF x THEN True ELSE y
- Exercise: evaluate Or False True

Lambda Calculus Encodings

Booleans and Propositional Logic (2) — Exercises

- Exercise: evaluate Not True
- $\rightarrow (\lambda x.((x \ \text{False})\ \text{True}))\ \text{True}$
- $\rightarrow (\text{True}\ \ \text{False})\ \text{True}$
- Could go straight to False from here, but we shall fill in the detail
- $\rightarrow ((\lambda x.\lambda y.x)\,(\lambda x.\lambda y.y))\,(\lambda x.\lambda y.x)$
- $\rightarrow (\lambda y.(\lambda x.\lambda y.y))\,(\lambda x.\lambda y.x)$
- $\rightarrow (\lambda x.\lambda y.y) \equiv$ False
- Exercise: evaluate And True False
- $\rightarrow$(IF x THEN y ELSE False) True False
- $\rightarrow$(IF True THEN False ELSE False) $\rightarrow$False
- Exercise: evaluate Or False True
- $\rightarrow$(IF x THEN True ELSE y) False True
- $\rightarrow$(IF False THEN True ELSE True) $\rightarrow$True

Lambda Calculus Encodings

Natural Numbers — Church Numerals

- Encoding of natural numbers
- $0 = \lambda f.\lambda y.y$
- $1 = \lambda f.\lambda y.f\ y$
- $2 = \lambda f.\lambda y.f\ (f\ y)$
- $3 = \lambda f.\lambda y.f\ (f\ (f\ y))$
- Successor Succ $= \lambda z.\lambda f.\lambda y.f(z\ f\ y)$
- Succ $0 = (\lambda z.\lambda f.\lambda y.f(z\ f\ y))(\lambda f.\lambda y.y)$
- $\rightarrow \lambda f.\lambda y.f\ ((\lambda f.\lambda y.y)\ f\ y)$
- $\rightarrow \lambda f.\lambda y.f\ ((\lambda y.y)\ y)$
- $\rightarrow \lambda f.\lambda y.f\ y = 1$

Lambda Calculus Encodings

Natural Numbers — Operations

- isZero $= \lambda z.z(\lambda y.\ \text{False}\ )\ \text{True}$
- Exercise: evaluate isZero 0
- If M and N are numerals (as λ expressions)
- Add $M\,N = \lambda x.\lambda y.(M\,x)\,((N\,x)\,y)$
- Mult $M\,N = \lambda x.(M\,(N\,x))$
- Exercise: show 1 + 1 = 2

Lambda Calculus Encodings

Pairs

- Encoding of a pair a, b
- $(a, b) = \lambda x.$ IF x THEN a ELSE b
- FST $= \lambda f.f$ True
- SND $= \lambda f.f$ False
- Exercise: evaluate FST (a, b)
- Exercise: evaluate SND (a, b)

Lambda Calculus Encodings

The Fixpoint Combinator

- $Y = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$
- $Y\,F = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F$
- $\rightarrow (\lambda x.F\,(x\,x))(\lambda x.F\,(x\,x))$
- $F((\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))) = F\,(Y\,F)$
- $(Y\,F)$ is a fixed point of F
- We can use Y to achieve recursion for F

Lambda Calculus Encodings

The Fixpoint Combinator — Recursion

- **Recursion implementation — Factorial**
- Fact $= \lambda f.\lambda n.$ IF $n = 0$ THEN 1 ELSE $n * (f\,(n - 1))$
- $(Y$ Fact$)1 =$ (Fact $(Y$ Fact$))1$
- $\rightarrow$ IF $1 = 0$ THEN 1 ELSE $1 * ((Y$ Fact$)\,0)$
- $\rightarrow 1 * ((Y$ Fact$)\,0)$
- $\rightarrow 1 *$ (Fact $(Y$ Fact$)\,0)$
- $\rightarrow 1 *$ IF $0 = 0$ THEN 1 ELSE $0 * ((Y$ Fact$)\,(0 - 1))$
- $\rightarrow 1 * 1 \rightarrow 1$
- Factorial n = (Y Fact) n
- Recursion implemented with a non-recursive function Y

Computability

Turing Machines, Lambda Calculus and Programming Languages

- Anything computable can be represented as TM or Lambda Calculus
- But programs would be slow, large and hard to read
- In practice use the ideas to create more expressive languages which include built-in primitives
- Also leads to ideas on data types
- Polymorphic data types
- Algebraic data types
- Also leads on to ideas on higher order functions — functions that take functions as arguments or returns functions as results.

Commentary 3

Complexity

3 Complexity

- Complexity Classes **P** and **NP**
- Class **NP**
- NP-completeness
- NP-completeness and Boolean Satisfiability

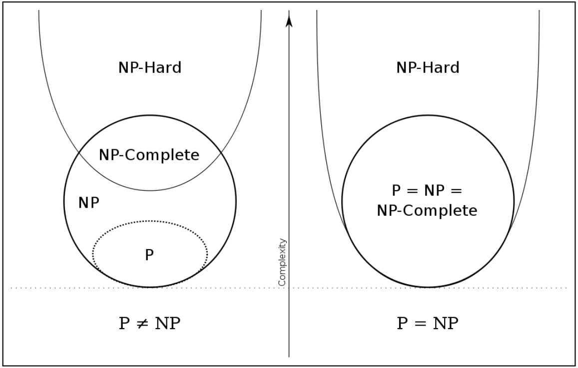
Complexity
P and NP

- P, the set of all decision problems that can be solved in polynomial time on a deterministic Turing machine
- NP, the set of all decision problems whose solutions can be verified (certificate) in polynomial time
- Equivalently, NP, the set of all decision problems that can be solved in polynomial time on a non-deterministic Turing machine
- A *decision problem*, *dp* is *NP-complete* if
  1. *dp* is in NP and
  2. Every problem in NP is reducible to *dp* in polynomial time
- NP-hard — a problem satisfying the second condition, whether or not it satisfies the first condition. Class of problems which are at least as hard as the hardest problems in NP. NP-hard problems do not have to be in NP and may not be decision problems

Complexity

P and NP — Diagram

Euler diagram for P, NP, NP-complete and NP-hard set of problems



Source: Wikipedia NP-complete entry

Class NP

Certificate and Verifier

- To formalise the definition of the class **NP**, we need to formalise the idea of checking a candidate solution
- Define a *certificate* for each problem input that would return Yes
- Describe the *verifier* algorithm
- Demonstrate the *verifier* algorithm has polynomial complexity
- The terms *certificate* and *verifier* have technical definitions in terms of languages and Turing Machines but can be thought of as *candidate solution* and *checker algorithm*

Class **NP**

Example Decision Problems (1)

- **Composite Numbers** Given a number N decide if N is a composite (i.e. non-prime) number
  Certificate factorization of N
- **Connectivity** Given a graph G and two vertices s, t in G, decide if s is connected to t in G.
  Certificate path from s to t
- **Linear Programming** Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots, u_n$ (a linear inequality has the form $a_1 u_1 + a_2 u_2 \cdots + a_n u_n \leqslant b$ for some coefficients $a_1, \ldots, a_n b$), decide if there is an assignment of rational numbers to the variables $u_1, \ldots, u_n$ which satisfies all the inequalities
  Certificate is the assignment

Class **NP**

Example Decision Problems (2)

- The above are in **P**
- *Composite Numbers*, *Connectivity* and *Linear programming* are in **P**
- *Composite Numbers* follows from Integer factorization and the AKS primality test from 2004
- *Connectivity* follows from the breadth-first search algorithm
- *Linear programming* shown to be in **P** by the Ellipsoid method

Class **NP**

Example Decision Problems (3)

- **Integer Programming** some or all variables are restricted to be integers
- **Travelling Salesperson** Given a set of nodes and distances between all pairs of nodes and a number k, decide if there is a closed circuit that visits every node exactly once and has total length at most k
  Certificate sequence of nodesin such a tour
- **Subset sum** Given a list of numbers and a number T, decide if there is a subset that adds up to T
  Certificate list of members of such a subset
- **Independent set (graph theory)** A subgraph of G with of at least k vertices which have no edges between them
  Certificate the list of k vertices
- **Clique problem** Given a graph and a number k, decide if there is a complete subgraph (clique) of size k
  Certificate list pf nodes. For explanation see Prove Clique is NP

Class **NP**

Example Decision Problems (4)

- The above are **NP-complete** — see List of NP-complete problems
- The following two are not known to be **P** nor **NP-complete**
- **Graph Isomorphism** Given two $n \times n$ adjacency matrices M_1,M_2, decide if M_1 and M_2 define the same graph (up to renaming of the vertices)
  Certificate the permutation $\pi : [n] \rightarrow [n]$ such that M_2 is equal to M_1 after reordering the indices of M_1 according to π
- **Integer factorization** Given three numbers N, L, U decide if N has a prime factor p in the interval $[L, U]$
  Certificate is the factorization of N

  Source Arora and Barak (2009, page 49) *Computational Complexity: A Modern Approach* and contained links

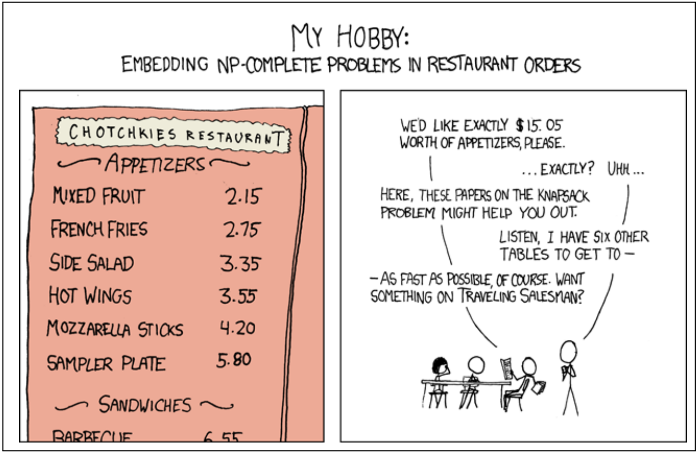
Complexity

NP-complete problems

- Boolean satisfiability (SAT) Cook-Levin theorem
- Conjunctive Normal Form 3SAT
- Hamiltonian path problem
- Travelling salesman problem
- NP-complete — see list of problems

Complexity

Knapsack Problem



Source & Explanation: XKCD 287

NP-Completeness and Boolean Satisfiability

Points on Notes

- The *Boolean satisfiability problem (SAT)* was the first decision problem shown to be *NP-Complete*
- This section gives a sketch of an explanation
- **Health Warning** different texts have different notations and there will be some inconsistency in these notes
- **Health warning** these notes use some formal notation *to make the ideas more precise* — computation requires precise notation and is about manipulating strings according to precise rules.

NP-Completeness and Boolean Satisfiability

Alphabets, Strings and Languages

- Notation:
- Σ is a set of symbols — the alphabet
- Σ^k is the set of all string of length k, which each symbol from Σ
- Example: if $\Sigma = \{0, 1\}$
  - $\Sigma^1 = \{0, 1\}$
  - $\Sigma^2 = \{00, 01, 10, 11\}$
- $\Sigma^0 = \{\epsilon\}$ where ϵ is the empty string
- Σ^* is the set of all possible strings over Σ
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- A *Language*, L, over Σ is a subset of Σ^*
- $L \subseteq \Sigma^*$

NP-Completeness and Boolean Satisfiability

Language Accepted by a Turing Machine

- Language accepted by Turing Machine, M denoted by $L(M)$
- $L(M)$ is the set of strings $w \in \Sigma^*$ accepted by M
- For *Final States* $F = \{Y, N\}$, a string $w \in \Sigma^*$ is accepted by $M \Leftrightarrow$ (if and only if) M starting in q_0 with w on the tape halts in state Y
- Calculating a function (function problem) can be turned into a decision problem by asking whether $f(x) = y$

NP-Completeness and Boolean Satisfiability

The NP-Complete Class

- If we do not know if P $\neq$ NP, what can we say ?
- A language L is *NP-Complete* if:
  - $L \in$ NP and
  - for all other $L' \in$ NP there is a *polynomial time transformation* (Karp reducible, reduction) from L' to L
- Problem P_1 *polynomially reduces* (Karp reduces, transforms) to P_2, written $P_1 \propto P_2$ or $P_1 \leq_p P_2$, iff $\exists f : \mathrm{dp}_{P_1} \rightarrow \mathrm{dp}_{P_2}$ such that
  - $\forall I \in \mathrm{dp}_{P_1} [I \in Y_{P_1} \Leftrightarrow f(I) \in Y_{P_2}]$
  - f can be computed in polynomial time

NP-Completeness and Boolean Satisfiability

The NP-Complete Class (2)

- More formally, $L_1 \subseteq \Sigma_1^*$ polynomially transforms to $L_2 \subseteq \Sigma_2^*$, written $L_1 \propto L_2$ or $L_1 \leq_p L_2$, iff $\exists f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that
  - $\forall x \in \Sigma_1^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$
  - There is a polynomial time TM that computes f
- *Transitivity* If $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$
- If L is NP-Hard and $L \in$ P then P = NP
- If L is NP-Complete, then $L \in$ P if and only if P = NP
- If L_0 is NP-Complete and $L \in$ NP and $L_0 \propto L$ then L is NP-Complete
- Hence if we find one NP-Complete problem, it may become easier to find more
- In 1971/1973 Cook-Levin showed that the Boolean satisfiability problem (SAT) is NP-Complete

NP-Completeness and Boolean Satisfiability

The Boolean Satisfiability Problem

- A propositional logic formula or Boolean expression is built from variables, operators: AND (conjunction, $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$)
- A formula is said to be *satisfiable* if it can be made True by some assignment to its variables.
- *The Boolean Satisfiability Problem* is, given a formula, check if it is satisfiable.
  - *Instance:* a finite set U of Boolean variables and a finite set C of clauses over U
  - *Question:* Is there a satisfying truth assignment for C ?
- A *clause* is is a disjunction of variables or negations of variables
- *Conjunctive normal form (CNF)* is a conjunction of clauses
- Any Boolean expression can be transformed to CNF

NP-Completeness and Boolean Satisfiability

The Boolean Satisfiability Problem (2)

- Given a set of Boolean variable $U = \{u_1, u_2, \ldots, u_n\}$
- A literal from U is either any u_i or the negation of some u_i (written $\overline{u_i}$) usual notation $\neg u_i$
- A clause is denoted as a subset of literals from U — $\{u_2, \overline{u_4}, u_5\}$ usual notation $u_2 \vee \neg u_4 \vee u_5$
- A clause is satisfied by an assignment to the variables if at least one of the literals evaluates to True (just like disjunction of the literals)
- Let C be a set of clauses over U — C is satisfiable iff there is some assignment of truth values to the variables so that every clause is satisfied (just like CNF)
- $C = \{\{u_1, u_2, u_3\}, \{\overline{u_2}, \overline{u_3}\}, \{u_2, \overline{u_3}\}\}$ is satisfiable

  usual notation $(u_1 \vee u_2 \vee u_3) \wedge (\neg u_2 \vee \neg u_3) \wedge (u_2 \vee \neg u_3)$

  assign $(u_1, u_2, u_3) = (T, F, F), (T, T, F), (F, T, F)$
- $C = \{\{u_1, u_2\}, \{u_1, \overline{u_2}\}, \{\overline{u_1}\}\}$ is not satisfiable

  usual notation $(u_1 \vee u_2) \wedge (u_1 \vee \neg u_2) \wedge (\neg u_1)$

NP-Completeness and Boolean Satisfiability

The Boolean Satisfiability Problem (3)

- Proof that SAT is NP-Complete looks at the structure of NDTMs and shows you can transform any NDTM to SAT in polynomial time (in fact logarithmic space suffices)
- SAT is in NP since you can check a solution in polynomial time
- To show that $\forall L \in \mathrm{NP} : L \propto \mathrm{SAT}$ invent a polynomial time algorithm for each polynomial time NDTM, M, which takes as input a string x and produces a Boolean formula E_x which is satisfiable iff M accepts x
- See Cook-Levin theorem

NP-Completeness and Boolean Satisfiability

Coping with NP-Completeness

- What does it mean if a problem is NP-Complete ?
  - There is a P time verification algorithm.
  - There is a P time algorithm to solve it iff P = NP (?)
  - No one has yet found a P time algorithm to solve any NP-Complete problem
  - So what do we do ?
- Improved exhaustive search — Dynamic Programming; Branch and Bound
- Heuristic methods — *acceptable* solutions in *acceptable* time — compromise on optimality
- Average time analysis — look for an algorithm with good average time — compromise on generality (see Big-O Algorithm Complexity Cheatsheet)
- Probabilistic or Randomized algorithms — compromise on correctness

Turing Machine

TMA Question (a)

- The transition function is represented as a Python dictionary mapping stete, symbol to symbol, move, state
- States are represented as strings — we may define Python constants to make life easier (see below)
- What are the states ?
- Tape represented by a list; moves by 1, -1, 0

```
# Moves
RIGHT = 1
LEFT  = -1
STAY  = 0

# States
Start   = "start"
FindA   = "FindA"
Find0   = "Find0"
FindNum = "FindNum"
FinishOK    = "FinishOK"
FinishNotOK = "FinishNotOK"
Stop        = "stop"
```

- Note that the identifiers must be valid Python
- Python has conventions about constantss

Turing Machine

TMA Question (b)

- Describe the actions for each state — possibly using Python dictionary notation (to make shorter work)

```
(Start, "a"):   ("a", RIGHT,FindA),
(Start, "0"):   ("0", RIGHT,Find0),
(Start, "#"):   ("#", RIGHT,FindNum),
(Start, None):  (None,STAY, Stop), # Is empty input allowed ?

(FindA, "a"):   ("a", RIGHT,FinishOK),
(FindA, "0"):   ("0", RIGHT,FindA),
(FindA, "#"):   ("#", RIGHT,FindA),
(FindA, None):  (False,STAY, Stop),

(Find0, "a"):   ("a", RIGHT,Find0),
(Find0, "0"):   ("0", RIGHT,FinishOK),
(Find0, "#"):   ("#", RIGHT,Find0),
(Find0, None):  (False,STAY, Stop),

(FindNum, "a"):   ("a", RIGHT,FindNum),
(FindNum, "0"):   ("0", RIGHT,FindNum),
(FindNum, "#"):   ("#", RIGHT,FinishOK),
(FindNum, None):  (False,STAY, Stop),
```

Turing Machine

TMA Question (c)

- FinishOK and FinishNotOK should tidy up the output and move the read/write head to an approrpiate position

```
(FinishOK, "a"): ("a",RIGHT,FinishOK),
(FinishOK, "0"): ("0",RIGHT,FinishOK),
(FinishOK, "#"): ("#",RIGHT,FinishOK),
(FinishOK, None): (True,STAY,Stop),
```

- What if we wanted to erase everything else and only have False/True as output ?

Program Complexity

Big O Notation

- Measuring program complexity introduced in section 4 of M269 Unit 2
- See also Miller and Ranum chapter 2 Big-O Notation
- See also Wikipedia: Big O notation
- See also Big-O Cheat Sheet

Program Complexity

Big O Notation (2)

- Complexity of algorithm measured by using some surrogate to get rough idea
- In M269 mainly using assignment statements
- For *exact* measure we would have to have cost of each operation, knowledge of the implementation of the programming language and the operating system it runs under.
- But mainly interested in the following questions:
- **(1)** Is algorithm A more efficient than algorithm B for large inputs ?
- **(2)** Is there a lower bound on any possible algorithm for calculating this particular function ?
- **(3)** Is it always possible to find a polynomial time (n^k) algorithm for any function that is computable
- — the later questions are addressed in Unit 7

Program Complexity

Orders of Common Functions

- $O(1)$ **constant** — look-up table
- $O(\log n)$ **logarithmic** — binary search of sorted array, binary search tree, binomial heap operations
- $O(n)$ **linear** — searching an unsorted list
- $O(n \log n)$ **loglinear** — heapsort, quicksort (best and average), merge sort
- $O(n^2)$ **quadratic** — bubble sort (worst case or naive implementation), Shell sort, quicksort (worst case), selection sort, insertion sort
- $O(n^c)$ **polynomial**
- $O(c^n)$ **exponential** — travelling salesman problem via dynamic programming, determining if two logical statements are equivalent by brute force
- $O(n!)$ **factorial** — TSP via brute force.

Program Complexity

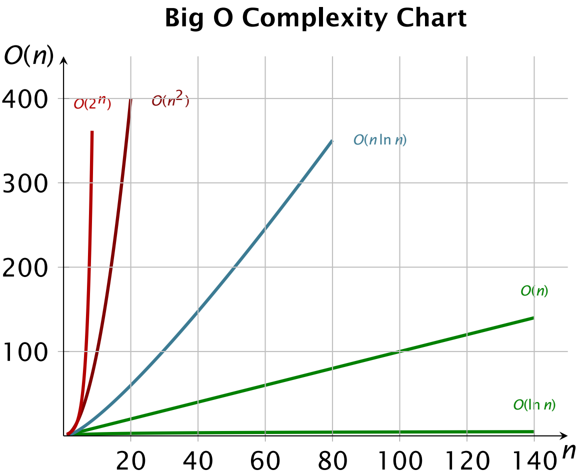
Tyranny of Asymptotics

- Table from Bentley (1984, page 868)
- Cubic algorithm on Cray-1 $3.0n^3$ nanoseconds
- Linear algorithm on TRS-80 $19.5n \times 10^6$ nanoseconds

N	Cray-1	TRS-80
10	3.0 microsecs	200 millisecs
100	3.0 millisecs	2.0 secs
1000	3.0 secs	20 secs
10000	49 mins	3.2 mins
100000	35 days	32 mins
1000000	95 yrs	5.4 hrs

Program Complexity

Big O Complexity Chart



Program Complexity

Big O Notation

- Abuse of notation — we write $f(x) = O(g(x))$
- but $O(g(x))$ is the class of all functions $h(x)$ such that $|h(x)| \leqslant C|g(x)|$ for some constant C
- So we should write $f(x) \in O(g(x))$ (but we don't)
- We ought to use a notation that says that (informally) the function f is bounded both above and below by g asymptotically
- This would mean that for big enough x we have

  $k_1\, g(x) \leqslant f(x) \leqslant k_2\, g(x)$ for some k_1, k_2
- This is Big Theta, $f(x) = \Theta(g(x))$
- But we use Big O to indicate an asymptotically tight bound where Big Theta might be more appropriate
- See Wikipedia: Big O Notation
- This could be Maths phobia generated confusion

Program Complexity

Example

```python
def someFunction(aList) :
  n = len(aList)
  best = 0
  for i in range(n) :
    for j in range(i + 1, n + 1) :
      s = sum(aList[i:j])
      best = max(best, s)
  return best
```

- Example from M269 Unit 2 page 46
- Code in file M269TutorialProgPythonADT.py
- What does the code do ?
- (It was a *famous* problem from the late 1970s/early 1980s)
- Can we construct a more efficient algorithm for the same computational problem ?

Program Complexity

Example (2)

- The code calculates the maximum subsegment of a list
- Described in Bentley (1984), (1988, column 7), (2000, column 7) Also in Gries (1989)
- These are all in a procedural programming style (as in C, Java, Python)
- Problem arose from medical image processing.
- A functional approach using Haskell is in Bird (1998, page 134), (2014, page 127, 133) — a variant on this called the *Not the maximum segment sum* is given in Bird (2010, Page 73) — both of these *derive* a linear time program from the (n^3) initial specification
- See Wikipedia: Maximum subarray problem
- See Rosetta Code: Greatest subsequential sum

Program Complexity

Example (3)

- Here is the same program but modified to allow lists that may only have negative numbers
- The complexity $T(n)$ function will be slightly different
- but the Big O complexity will be the same

```
def maxSubSeg01(xs) :
  n = len(xs)
  maxSoFar = xs[0]
  for i in range(1,n) :
    for j in range(i + 1, n + 1) :
      s = sum(xs[i:j])
      maxSoFar = max(maxSoFar, s)
  return maxSoFar
```

Program Complexity

Example (4)

- Complexity function $T(n)$ for `maxSubSeg01()`
- Two initial assignments
- The outer loop will be executed $(n-1)$ times,
- Hence the inner loop is executed

  $$(n-1)+(n-2)+\ldots+2+1=\frac{(n-1)}{2}\times n$$

- Assume `sum()` takes n assignments
- Hence $T(n)=2+(n+2)\times\left(\frac{(n-1)}{2}\times n\right)$

  $$\begin{aligned}
  &= 2+(n+2)\times\left(\frac{n^2}{2}-\frac{n}{2}\right)\\
  &= 2+\tfrac{1}{2}n^3-\tfrac{1}{2}n^2+n^2-n\\
  &= \tfrac{1}{2}n^3+\tfrac{1}{2}n^2-n+2
  \end{aligned}$$

- Hence $O(n^3)$

Program Complexity

Example (5)

- **Developing a better algorithm**
- Assume we know the solution (`maxSoFar`) for `xs[0..(i - 1)]`
- We extend the solution to `xs[0..i]` as follows:
- The maximum segment will be either `maxSoFar`
- or the sum of a sublist ending at `i` (`maxToHere`) if it is bigger
- This reasoning is similar to divide and conquer in binary search or Dynamic programming (see Unit 5)
- Keep track of both `maxSoFar` and `maxToHere` — the Eureka step

Program Complexity

Example (6)

- **Developing a better algorithm `maxSubSeg02()`**

```python
def maxSubSeg02(xs) :
  maxToHere = xs[0]
  maxSoFar  = xs[0]
  for x in xs[1:] :
    # Invariant: maxToHere, maxSoFar OK for xs[0..(i-1)]
    maxToHere = max(x, maxToHere + x)
    maxSoFar  = max(maxSoFar, maxToHere)
  return maxSoFar
```

- Complexity function $T(n) = 2 + 2n$
- Hence $O(n)$
- What if we want more information ?
- Return the (or a) segment with max sum and position in list

Program Complexity

Example (7)

```python
def maxSubSeg03(xs) :
  maxSoFar = maxToHere = xs[0]
  startIdx, endIdx, startMaxToHere = 0, 0, 0
  for i, x in enumerate(xs) :
    if maxToHere + x < x :
      maxToHere = x
      startMaxToHere = i
    else :
      maxToHere = maxToHere + x

    if maxSoFar < maxToHere :
      maxSoFar = maxToHere
      startIdx, endIdx = startMaxToHere, i

  return (maxSoFar,xs[startIdx:endIdx+1],startIdx,endIdx)
```

- **Developing a better algorithm** maxSubSeg03()
- Complexity function worst case $T(n) = 2 + 3 + (2 + 3)n$
- Hence still $O(n)$
- Note Python assignments, enumerate() and tuple

Program Complexity

Example (8)

- Sample data and output

```python
egList = [-2,1,-3,4,-1,2,1,-5,4]

egList01 = [-1,-1,-1]

egList02 = [1,2,3]

assert maxSubSeg03(egList) == (6, [4, -1, 2, 1], 3, 6)

assert maxSubSeg03(egList01) == (-1, [-1], 0, 0)

assert maxSubSeg03(egList02) == (7, [1, 2, 3], 0, 2)
```

Program Complexity

Python Data Types — Lists

Operation	Notation	Average	Amortized Worst
Get item	`x = xs[i]`	$O(1)$	$O(1)$
Set item	`xs[i] = x`	$O(1)$	$O(1)$
Append	`xs = ys + zs`	$O(1)$	$O(1)$
Copy	`xs = ys[:]`	$O(n)$	$O(n)$
Pop last	`xs.pop()`	$O(1)$	$O(1)$
Pop other	`xs.pop(i)`	$O(k)$	$O(k)$
Insert(i,x)	`xs[i:i] = [x]`	$O(n)$	$O(n)$
Delete item	`del xs[i:i+1]`	$O(n)$	$O(n)$
Get slice	`xs = ys[i:j]`	$O(k)$	$O(k)$
Set slice	`xs[i:j] = ys`	$O(k+n)$	$O(k+n)$
Delete slice	`xs[i:j] = []`	$O(n)$	$O(n)$
Member	`x in xs`	$O(n)$	
Get length	`n = len(xs)`	$O(1)$	$O(1)$
Count(x)	`n = xs.count(x)`	$O(n)$	$O(n)$

- Source https://wiki.python.org/moin/TimeComplexity
- See https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Program Complexity

User Defined Type — Bags

```python
class Bag:

  def __init__(self):
    self.list = []

  def add(self, item):
    self.list.append(item)

  def remove(self, item):
    self.list.remove(item)

  def contains(self, item):
    return item in self.list

  def count(self, item):
    return self.list.count(item)

  def size(self):
    return len(self.list)

  def __str__(self):
    return str(self.list)
```

Using a Data Type

Information Retrieval Functions

- **Term Frequency**, `tf`, takes a string, `term`, and a Bag, `document`

  returns occurrences of `term` divided by total strings in `document`

- **Inverse Document Frequency**, `idf`, takes a string, `term`, and a list of Bags, `documents`

  returns $\log(\texttt{total}/(1 + \texttt{containing}))$ — `total` is total number of Bags, `containing` is the number of Bags containing `term`

- **tf-idf**, `tf_idf`, takes a string, `term`, and a list of Bags, `documents`

  returns a sequence $[r_0, r_1, \ldots, r_{n-1}]$ such that $r_i = \texttt{tf}(\texttt{term}, d_i) \times \texttt{idf}(\texttt{term}, \texttt{documents})$

Complexity

Big-O and Big-Theta Definitions (a)

- We compare the functions implementing algorithms by looking at the asymptotic behaviour of the functions for large inputs.
- If f and g are functions taking taking natural numbers as input (the problem size) and returning nonnegative results (the effort required in the calculations.)
- f is of *order* g and write $f = \Theta(g)$, if there are positive constants k_1 and k_2 and a natural number n_0 such that

  $$k_1\, g(n) \leqslant f(n) \leqslant k_2\, g(n) \text{ for all } n > n_0$$

  This means that some multipliers times $g(n)$ provide upper and lower bounds to $f(n)$
- If we only wanted an upper bound on the values of a function, then you can use Big-O notation.
- We say f is of *order at most* g and write $f = O(g)$, if there is a positive constant k and a natural number n_0 such that

  $$f(n) \leqslant kg(n) \text{ for all } n > n_0$$

Complexity

Big-O and Big-Theta Definitions (b)

- Note that the notation is heavily abused:

  Many authors use Big-O notation when they really mean Big-Θ notation

  We really should define the Θ notation to say that $\Theta(g)$ denotes the set of all functions f with the stated property and write $f \in \Theta(g)$ — however the use of $f = \Theta(g)$ is traditional

- The next section gives some rules for manipulating the notation to calculate overall complexities of functions from their component parts — this also abuses the notation for equality

  Based on Bird and Gibbons (2020, page 25) *Algorithm Design with Haskell* and Graham, Knuth and Patashnik (1994, page 450) *Concrete Mathematics: A Foundation for Computer Science*

Complexity

Big-O and Big-Theta Rules

- $n^p = O(n^q)$ where $p \leqslant q$

  This has some surprising consequences — $n = O(n)$ and $n = O(n^2)$ — remember Big-O just gives upper bounds.
- $O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$
- $\Theta(n^p) + \Theta(n^q) = \Theta(n^q)$ where $p \leqslant q$
- $f(n) = \Theta(f(n))$
- $c \cdot \Theta(f(n)) = \Theta(f(n))$ if c is constant
- $\Theta(\Theta(f(n))) = \Theta(f(n))$
- $\Theta(f(n))\Theta(g(n)) = \Theta(f(n)g(n))$
- $\Theta(f(n)g(n)) = f(n)\Theta(g(n))$

Complexity

Big-Theta Rules — Example

```python
def numVowels(txt : str) -> int ;
  """Find the number of vowels in text

  """

  vowelCount = 0
  vowels = "aeiouAEIOU"

  for ch in txt :
    if ch in vowels :
      vowelCount = vowelCount + 1
  return vowelCount
```

- The rules give

  $\Theta(1) + \Theta(1) + \Theta(n) \times \Theta(|\text{vowels}|) \times \Theta(1)$

  where $n = |\text{txt}|$
- Since $|\text{vowels}| = 10$ the overall complexity is $\Theta(n)$

List Comprehensions

Python

- **List Comprehensions** (tutorial), **List Comprehensions** (reference) provide a concise way of performing calculations over lists (or other iterables)
- Example: Square the even numbers between 0 and 9

```
Python3>>> [x ** 2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
```

  - Example: List all pairs of integers (x, y) such that $x < 4$, $y < 4$ and x is divisible by 2 and y is divisible by 3

```
Python3>>> [(x,y) for x in range(4)
...               for y in range(4)
...               if x % 2 == 0
...                  and y % 3 == 0]
[(0, 0), (0, 3), (2, 0), (2, 3)]
Python3>>>
```

- In general

```
  [expr for target1 in iterable1 if cond1
        for target2 in iterable2 if cond2 ...
        for targetN in iterableN if condN ]
```

- Lots example usage in the algorithms below

List Comprehensions

Haskell

- **List Comprehensions** provide a concise way of performing calculations over lists
- Example: Square the even numbers between 0 and 9

```
GHCi> [x^2 | x <- [0..9], x `mod` 2 == 0]
[0,4,16,36,64]
GHCi>
```

- In general

```
[expr | qual1, qual2,..., qualN]
```

- The qualifiers `qual` can be
  - Generators `pattern <- list`
  - Boolean guards — acting as filters
  - Local declarations with `let decls` for use in `expr` and later generators and boolean guards

List Comprehension Exercises

Activity 1 (a) Stop Words Filter

- Stop words are the most common words that most search engines avoid: `'a','an','the','that',...`
- Using list comprehensions, write a function `filterStopWords` that takes a list of words and filters out the stop words
- Here is the initial code

```
sentence \
 = "the quick brown fox jumps over the lazy dog"

words = sentence.split()

wordsTest \
 = (words == ['the', 'quick', 'brown'
             , 'fox', 'jumps', 'over'
             , 'the', 'lazy', 'dog'])

stopWords \
 = ['a','an','the','that']
```

Go to Answer

List Comprehension Exercises

Activity 1 (a) Stop Words Filter

```
sentence \
 = "the␣quick␣brown␣fox␣jumps␣over␣the␣lazy␣dog"

words = sentence.split()

wordsTest \
 = (words == ['the', 'quick', 'brown'
             , 'fox', 'jumps', 'over'
             , 'the', 'lazy', 'dog'])

stopWords \
 = ['a','an','the','that']
```

- Notice the Python Explicit line joining with (\<nl>) and Python Implicit line joining with ((...))
- The backslash (\) must be followed by an end of line character (<nl>)
- The ('␣') symbol represents a space (see Unicode U+2423 Open Box)

Go to Answer

List Comprehension Exercises

Activity 1 (b) Transpose Matrix

- A matrix can be represented as a list of rows of numbers
- We *transpose* a matrix by swapping columns and rows
- Here is an example

```
matrixA \
 = [[1, 2, 3, 4]
   ,[5, 6, 7 ,8]
   ,[9, 10, 11, 12]]

matATr \
 = [[1, 5, 9]
   ,[2, 6, 10]
   ,[3, 7, 11]
   ,[4, 8, 12]]
```

- Using list comprehensions, write a function `transMat`, to transpose a matrix

Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Write a function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges
- If we do this in the simplest way we get a bias to one argument
- Here is an example of a bias to the second argument

```
yBiasLstTest \
 = (yBiasListing(5,5)
    == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
       , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
       , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
       , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
       , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges
- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument
- Here is an example output

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

▸ Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of lists of all possible pairs in those ranges
- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument — further, the output should be segment by each initial prefix (see example below)
- Here is an example output

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
        , [(0, 1), (1, 0)]
        , [(0, 2), (1, 1), (2, 0)]
        , [(0, 3), (1, 2), (2, 1), (3, 0)]
        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Answer

Compexity

List Comprehensions

- Note that list comprehensions are not in M269
- See Complexity of a List Comprehension

```
[f(e) for e in row for row in mat]
```

- Suppose $f = \Theta(g)$ with n elements in a row and m rows
- Then complexity is
  $\Theta(g(e)) \times \Theta(n) \times \Theta(m) = \Theta(m \times n \times g(e))$

```
[[e**2 for e in row] for row in mat]
```

- $\Theta(e * *2) = \Theta(1)$
- Suppose n is maximum length of a row and m rows
- Then complexity is$\Theta(1) \times \Theta(n) \times \Theta(m) = \Theta(n \times m)$

List Comprehension Exercises

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter
- *Write here:*
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter

```python
def filterStopWords(words) :
  nonStopWords \
   = [word for word in words
           if word not in stopWords]
  return nonStopWords


filterStopWordsTest \
 = filterStopWords(words) \
    == ['quick', 'brown', 'fox'
      , 'jumps', 'over', 'lazy', 'dog']
```

▸ Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix
- *Write here:*
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix

```python
def transMat(mat) :
  rowLen = len(mat[0])
  matTr \
   = [[row[i] for row in mat] for i in range(rowLen)]
  return matTr

transMatTestA \
 = (transMat(matrixA)
     == matATr)
```

- Note that a list comprehension is a valid expression as a target expression in a list comprehension
- The code assumes every row is of the same length
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Note the differences in the list comprehensions below

```
matrixA \
 = [[1, 2, 3, 4]
   ,[5, 6, 7 ,8]
   ,[9, 10, 11, 12]]
```

```
Python3>>> [[row[i] for row in matrixA]
...                  for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
Python3>>> [row[i] for row in matrixA
...                 for i in range(4)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Python3>>> [row[i] for i in range(4)
...                 for row in matrixA]
[1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12]
Python3>>> [[row[i] for i in range(4)]
...                  for row in matrixA]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix
- The Python NumPy package provides functions for N-dimensional array objects
- For transpose see numpy.ndarray.transpose

```
Python3>>> import numpy as np
Python3>>> ar = np.array([[1,2],[3,4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> arT = ar.transpose()
Python3>>> arT
array([[1, 3],
       [2, 4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> ar.shape
(2, 2)
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — first version
- Write here

```
yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
        , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
        , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
        , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
        , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order
- This is the *obvious* but biased version

```
def yBiasListing(xRng,yRng) :
  yBiasLst \
   = [(x,y) for x in range(xRng)
            for y in range(yRng)]
  return yBiasLst

yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
        , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
        , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
        , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
        , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — second version
- Write here

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — second version
- This works by making the sum of the coordinates the same for each prefix

```
def fairListing(xRng,yRng) :
  fairLst \
   = [(x,d-x) for d in range(yRng)
              for x in range(d+1)]
  return fairLst

fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
        , (0, 1), (1, 0)
        , (0, 2), (1, 1), (2, 0)
        , (0, 3), (1, 2), (2, 1), (3, 0)
        , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- Write here

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
         , [(0, 1), (1, 0)]
         , [(0, 2), (1, 1), (2, 0)]
         , [(0, 3), (1, 2), (2, 1), (3, 0)]
         , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- The *inner loop* is placed into its own list comprehension

```python
def fairListingA(xRng,yRng) :
  fairLstA \
   = [[(x,d-x) for x in range(d+1)]
              for d in range(yRng)]
  return fairLstA

fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
         , [(0, 1), (1, 0)]
         , [(0, 2), (1, 1), (2, 0)]
         , [(0, 3), (1, 2), (2, 1), (3, 0)]
         , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

▸ Go to Activity

- **The Divide-and-Conquer Method**

  Many useful algorithms are recursive in structure and often follow a *divide-and-conquer method*

  They break the problem into several subproblems similar to the original problem
- The time analysis is represented by a *recurrence system*
- **References**
- Big *O* notation
- Master theorem
- Cormen et al (2022, chp 4) *Algorithms*
- These notes are partly based on *M261 Mathematics in Computing* and *M263 Building Blocks of Software* and are *not* part of *M269 Algorithms, Data Structures and Computability*

Master Theorem

Recurrence System (a)

- **Recurrence System**

$$
\begin{aligned}
T(1) &= b & (1)\\
T(n) &= bn^{\beta} + cT\left(\frac{n}{d}\right) \qquad \{n = d^{\alpha} > 1\} & (2)
\end{aligned}
$$

- **Typical Expansion**

n	**T(n)**
d^0	b
d^1	$bn^{\beta} + cb$
d^2	$bn^{\beta} + cb\left(\frac{n}{d}\right)^{\beta} + c^2 b$

Master Theorem

Recurrence System (b)

- **General Expansion**

$$
\begin{aligned}
T(n) &= bn^{\beta} + cT\left(\frac{n}{d}\right) \\
&= bn^{\beta} + cb\left(\frac{n}{d}\right)^{\beta} + c^2 T\left(\frac{n}{d^2}\right) \\
&= bn^{\beta}\left(1 + \frac{c}{d^{\beta}} + \left(\frac{c}{d^{\beta}}\right)^2 + \cdots + \left(\frac{c}{d^{\beta}}\right)^{\alpha}\right) \\
T(n) &= bn^{\beta} \sum_{i=0}^{\log_d n} \left(\frac{c}{d^{\beta}}\right)^{i} \qquad (3)
\end{aligned}
$$

Master Theorem

Recurrence System (c)

- **Proof of Closed Form Equation (3)**
- For $n = 1$ equation (3) gives

$$T(1) = b1^{\beta} \sum_{i=0}^{0} \left(\frac{c}{d^{\beta}}\right)^{i} = b \text{ which is correct (same as (1))}$$

- Assume equation (3) holds for $n = d^{\alpha}$. Then for $n = d^{\alpha+1}$

$$\begin{aligned}
T\left(d^{\alpha+1}\right) &= cT\left(d^{\alpha}\right) + bn^{\beta} && \text{by equation (2)} \\
&= cbd^{\alpha\beta} \sum_{i=0}^{\alpha} \left(\frac{c}{d^{\beta}}\right)^{i} + bd^{(\alpha+1)\beta} && \text{by assumption} \\
&= \left(\frac{c}{d^{\beta}}\right) bd^{(\alpha+1)\beta} \sum_{i=0}^{\alpha} \left(\frac{c}{d^{\beta}}\right)^{i} + bd^{(\alpha+1)\beta} \\
&= bd^{(\alpha+1)\beta} \left(\sum_{i=1}^{\alpha+1} \left(\frac{c}{d^{\beta}}\right)^{i} + 1\right) && \text{by rearrangement} \\
&= bd^{(\alpha+1)\beta} \sum_{i=0}^{\alpha+1} \left(\frac{c}{d^{\beta}}\right)^{i} && \text{by rearrangement}
\end{aligned}$$

- Hence equation (3) holds for all $n = d^{\alpha}$ where $\alpha \in \mathbb{N}$

Master Theorem

Cases

1. If $c < d^{\beta}$ then the sum converges and T(n) is $\Theta(n^{\beta})$
2. If $c = d^{\beta}$ then each term in the sum is 1 and T(n) is $\Theta\left(n^{\beta}\log_d n\right)$
3. If $c > d^{\beta}$ then use $\sum_{i=0}^{p} x^i = \dfrac{x^{p+1}-1}{x-1}$

$$T(n) = bn^{\beta}\left[\frac{\left(\frac{c}{d^{\beta}}\right)^{\log_d n+1} - 1}{\left(\frac{c}{d^{\beta}}\right) - 1}\right]$$

$$= \Theta\left(n^{\beta}\left(\tfrac{c}{d^{\beta}}\right)^{\log_d n}\right)$$

$$= \Theta\left(c^{\log_d n}\right)$$

$$= \Theta\left(n^{\log_d c}\right) \text{ since } a^{\log_b x} = x^{\log_b a}$$

Master Theorem Example Usage (1)

Binary Search

- **Algorithm**
- Find mid point and check
  if not equal to target, recurse on half the data
- **Timing equations**

  $T(1) \leqslant 1$

  $T(n) = T\left(\frac{n}{2}\right) + 1$
- Hence $c = 1$, $d = 2$, $\beta = 0 \rightarrow$ case (2)

  $$T(n) = \Theta(\log_2 n)$$

Master Theorem Example Usage (2)

Quicksort

- **Algorithm**
- Best case: splitting on median of data
- Recursively sort each half
- **Timing equations**

  $T(1) \leqslant k$

  $T(n) = 2T\left(\frac{n}{2}\right) + kn$
- Hence $c = 2$, $d = 2$, $\beta = 1 \rightarrow$ case (2)

  $$T(n) = \Theta(n \log_2 n)$$
- See Averages/Median

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (a)

- **Matrix Multiplication**
- Let A, B be two square matrices over a ring, $\mathcal{R}$
- Informally, a *ring* is a set with two binary operations which look similar to addition and multiplication of integers
- The problem is to implement matrix multiplication to find the matrix product $C = AB$
- Without loss of generality, we may assume that A, and B have sizes which are powers of 2 — if A, and B were not of this size, they could be padded with rows or columns of zeroes
- The Strassen algorithm partitions A, B and C into equally sized blocks

  $$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

  with $A_{ij}, B_{ij}, C_{ij} \in \mathrm{Mat}_{2^{n-1} \times 2^{n-1}}(\mathcal{R})$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (b)

- The usual (naive, standard) algorithm gives
$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$
$$= \begin{pmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{pmatrix}$$
- This as 8 multiplications and if we assume multiplication is more expensive than addition then the time complexity is $\Theta(n^3)$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (c)

- The Strassen algorithm rearranges the calculation

  $M_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$

  $M_2 = (A_{21} + A22) \times B_{11}$

  $M_3 = A_{11} \times (B_{12} - B_{22})$

  $M_4 = A_{22} \times (B_{21} - B_{11})$

  $M_5 = (A_{11} + A12) \times B_{22}$

  $M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

  $M_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

- We now express the C_{ij} in terms of the M_k

  $$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$
  $$= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}$$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (d)

- **Strassen Matrix Multiplication Timing Equations**

  $T(n) = 7T\left(\frac{n}{2}\right) + \frac{18}{4}n^2$

  $T(1) \leqslant \frac{18}{4}$
- This is derived from the 7 multiplications and 18 additions or subtractions
- $c = 7$, $d = 2$, $\beta = 2 \rightarrow$ case (3)

  $$T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{2.8}\right)$$

Exponentials and Logarithms

Definitions

- **Exponential function** $y = a^x$ or $f(x) = a^x$
- $a^n = a \times a \times \cdots \times a$ (n a terms)
- **Logarithm** reverses the operation of exponentiation
- $\log_a y = x$ means $a^x = y$
- $\log_a 1 = 0$
- $\log_a a = 1$
- Method of logarithms propounded by John Napier from 1614
- Log Tables from 1617 by Henry Briggs
- Slide Rule from about 1620–1630 by William Oughtred of Cambridge
- *Logarithm* from Greek *logos* ratio, and *arithmos* number *Chambers Dictionary (13th Edition, 2014)*

Exponentiation

Rules of Indices

1. $a^m \times a^n = a^{m+n}$
2. $a^m \div a^n = a^{m-n}$
3. $a^{-m} = \dfrac{1}{a^m}$
4. $a^{\frac{1}{m}} = \sqrt[m]{a}$
5. $(a^m)^n = a^{mn}$
6. $a^{\frac{n}{m}} = \sqrt[m]{a^n}$
7. $a^0 = 1$ where $a \neq 0$

- **Exercise** Justify the above rules
- What should 0^0 evaluate to ?
- See Wikipedia: Exponentiation
- The *justification* above probably only worked for whole or *rational* numbers — see later for exponents with *real* numbers (and the value of *logarithms*, *calculus*. . .)

Logarithms

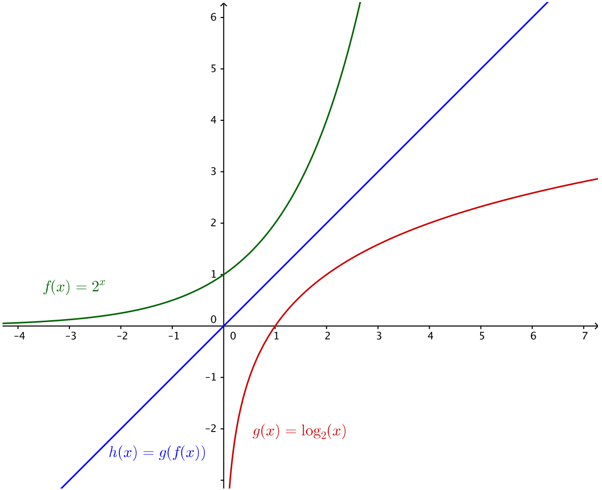
Motivation

- Make arithmetic easier — turns multiplication and division into addition and subtraction (see later)
- Complete the range of elementary functions for differentiation and integration
- An elementary function is a function of one variable which is the composition of a finite number of arithmetic operations ((+), (–), ($\times$), ($\div$)), exponentials, logarithms, constants, and solutions of algebraic equations (a generalization of nth roots).
- The elementary functions include the trigonometric and hyperbolic functions and their inverses, as they are expressible with complex exponentials and logarithms.
- See A Level FP2 for Euler's relation $\mathrm{e}^{i\theta} = \cos\theta + \mathrm{i}\sin\theta$
- In A Level C3, C4 we get $\int \frac{1}{x} = \log_{\mathrm{e}} |x| + C$
- e is Euler's number 2.71828...

Exponentials and Logarithms

Graphs

- See GeoGebra file expLog.ggb



Exponentials and Logarithms

Laws of Logarithms

- **Multiplication law** $\log_a xy = \log_a x + \log_a y$
- **Division law** $\log_a \left(\frac{x}{y}\right) = \log_a x - \log_a y$
- **Power law** $\log_a x^k = k \log_a x$
- **Proof of Multiplication Law**

$$
\begin{aligned}
x &= a^{\log_a x} \\
y &= a^{\log_a y} && \text{by definition of log} \\
xy &= a^{\log_a x} \times a^{\log_a y} \\
&= a^{\log_a x + \log_a y} && \text{by laws of indices} \\
\text{Hence } \log_a xy &= \log_a x + \log_a y && \text{by definition of log}
\end{aligned}
$$

Arithmetic Operations

Inverse Operations

- *Notation helps or maybe not ?*
- **Addition** $\text{add}(b, x) = x + b$
- **Subtraction** $\text{sub}(b, x) = x - b$
- Inverse $\text{sub}(b, \text{add}(b, x)) = (x + b) - b = x$
- **Multiplication** $\text{mul}(b, x) = x \times b$
- **Division** $\text{div}(b, x) = x \div b = \frac{x}{b} = x/b$
- Inverse $\text{div}(b, \text{mul}(b, x)) = (x \times b) \div b = \frac{(x \times b)}{b} = x$
- **Exponentiation** $\exp(b, x) = b^x$
- **Logarithm** $\log(b, x) = \log_b x$
- Inverse $\log(b, \exp(b, x)) = \log_b(b^x) = x$
- *What properties do the operations have that work (or not) with the notation ?*

Arithmetic Operations

Commutativity and Associativity

- **Commutativity** $x \circledast y = y \circledast x$
- **Associativity** $(x \circledast y) \circledast z = x \circledast (y \circledast z)$
- $(+)$ and $(\times)$ are *semantically* commutative and associative — so we can leave the brackets out
- $(-)$ and $(\div)$ are not
- Evaluate $(3 - (2 - 1))$ and $((3 - 2) - 1)$
- Evaluate $(3/(2/2))$ and $((3/2)/2)$
- We have the *syntactic* ideas of left (and right) associativity
- We choose $(-)$ and $(\div)$ to be left associative
- $3 - 2 - 1$ means $((3 - 2) - 1)$
- $3/2/2$ means $((3/2)/2)$
- **Operator precedence** is also a choice (remember BIDMAS or BODMAS ?)
- If in doubt, put the brackets in

Exponentials and Logarithms

Associativity

- What should 2^{3^4} mean ?
- Let $b \wedge x \equiv b^x$
- Evaluate $(2 \wedge 3) \wedge 4$ and $2 \wedge (3 \wedge 4)$
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- Beware spreadsheets Excel and LibreOffice here

Exponentials and Logarithms

Associativity

- $(2^3)^4 = 2^{12}$ and $2^{3^4} = 2^{81}$
- Exponentiation is not semantically associative
- We *choose* the syntactic left or right associativity to make the syntax nicer.
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- $c = \log_b(x \log_b(b^b)) = \log_b(x \cdot (b \log_b b)) = \log_b(x \cdot b \cdot 1)$
- Hence $c = \log_b x + \log_b b = \log_b x + 1$
- Not symmetrical (unless b and x are both 2)
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- $d = \log_b((b \wedge x)(\log_b b)) = \log_b((b \wedge x) \times 1)$
- Hence $d = \log_b(b \wedge x) = x(\log_b b) = x$
- Which is what we want — so exponentiation is *chosen* to be right associative

Exponentials and Logarithms

Change of Base

- **Change of base**

$$\log_a x = \frac{\log_b x}{\log_b a}$$

Proof: Let $y = \log_a x$

$$\begin{aligned} a^y &= x \\ \log_b a^y &= \log_b x \\ y \log_b a &= \log_b x \\ y &= \frac{\log_b x}{\log_b a} \end{aligned}$$

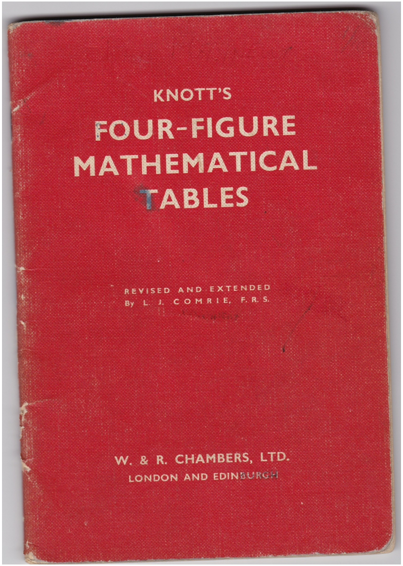
- Given x, $\log_b x$, find the base b
  - $b = x^{\frac{1}{\log_b x}}$
- $\log_a b = \dfrac{1}{\log_b a}$

Before Calculators and Computers

- We had computers before 1950 — they were *humans* with pencil, paper and some further aids:
- **Slide rule** invented by William Oughtred in the 1620s — major calculating tool until pocket calculators in 1970s
- **Log tables** in use from early 1600s — method of logarithms propounded by John Napier
- **Logarithm** from Greek *logos* ratio, and *arithmos* number

Log Tables

Knott's Four-Figure Mathematical Tables



Log Tables

Logarithms of Numbers

2 LOGARITHMS OF NUMBERS

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD								
												1	2	3	4	5	6	7	8	9
10	·0000	0043	0086	0128	0170	0212					42	4	8	13	17	21	25	29	34	38
						0212	0253	0294	0334	0374	40	4	8	12	16	20	24	28	32	36
11	·0414	0453	0492	0531	0569	0607					39	4	8	12	16	19	23	27	31	35
						0607	0645	0682	0719	0755	37	4	7	11	15	18	22	26	30	33
12	·0792	0828	0864	0899	0934	0969					35	4	7	11	14	18	21	25	28	32
						0969	1004	1038	1072	1106	34	3	7	10	14	17	20	24	27	31
13	·1139	1173	1206	1239	1271	1303					33	3	7	10	13	16	20	23	26	30
						1303	1335	1367	1399	1430	32	3	6	10	13	16	19	22	26	29
14	·1461	1492	1523	1553	1584	1614	1644	1673	1703	1732	30	3	6	9	12	15	18	21	24	27
15	·1761	1790	1818	1847	1875	1903	1931	1959	1987	2014	28	3	6	8	11	14	17	20	22	25
16	·2041	2068	2095	2122	2148	2175	2201	2227	2253	2279	26	3	5	8	10	13	16	18	21	23
17	·2304	2330	2355	2380	2405	2430	2455	2480	2504	2529	25	2	5	7	10	12	15	17	20	22
18	·2553	2577	2601	2625	2648	2672	2695	2718	2742	2765	23	2	5	7	9	12	14	16	19	21
19	·2788	2810	2833	2856	2878	2900	2923	2945	2967	2989	22	2	4	7	9	11	13	15	18	20
20	·3010	3032	3054	3075	3096	3118	3139	3160	3181	3201	21	2	4	6	8	11	13	15	17	19
21	·3222	3243	3263	3284	3304	3324	3345	3365	3385	3404	20	2	4	6	8	10	12	14	16	18
22	·3424	3444	3464	3483	3502	3522	3541	3560	3579	3598	19	2	4	6	8	10	11	13	15	17
23	·3617	3636	3655	3674	3692	3711	3729	3747	3766	3784		2	4	6	7	9	11	13	15	17
24	·3802	3820	3838	3856	3874	3892	3909	3927	3945	3962	18	2	4	5	7	9	11	13	14	16
25	·3979	3997	4014	4031	4048	4065	4082	4099	4116	4133	17	2	3	5	7	9	10	12	14	15
26	·4150	4166	4183	4200	4216	4232	4249	4265	4281	4298		2	3	5	6	8	10	11	13	14
27	·4314	4330	4346	4362	4378	4393	4409	4425	4440	4456	16	2	3	5	6	8	9	11	13	14
28	·4472	4487	4502	4518	4533	4548	4564	4579	4594	4609	15	2	3	5	6	8	9	11	12	14
29	·4624	4639	4654	4669	4683	4698	4713	4728	4742	4757		1	3	4	6	7	9	10	12	13
30	·4771	4786	4800	4814	4829	4843	4857	4871	4886	4900	14	1	3	4	6	7	8	10	11	13
31	·4914	4928	4942	4955	4969	4983	4997	5011	5024	5038		1	3	4	5	7	8	10	11	12
32	·5051	5065	5079	5092	5105	5119	5132	5145	5159	5172		1	3	4	5	7	8	9	11	12
33	·5185	5198	5211	5224	5237	5250	5263	5276	5289	5302	13	1	3	4	5	7	8	9	10	12
34	·5315	5328	5340	5353	5366	5378	5391	5403	5416	5428		1	3	4	5	6	8	9	10	11
35	·5441	5453	5465	5478	5490	5502	5514	5527	5539	5551		1	2	4	5	6	7	8	10	11
36	·5563	5575	5587	5599	5611	5623	5635	5647	5658	5670	12	1	2	4	5	6	7	8	10	11
37	·5682	5694	5705	5717	5729	5740	5752	5763	5775	5786		1	2	3	5	6	7	8	9	10
38	·5798	5809	5821	5832	5843	5855	5866	5877	5888	5899		1	2	3	5	6	7	8	9	10
39	·5911	5922	5933	5944	5955	5966	5977	5988	5999	6010	11	1	2	3	4	6	7	8	9	10
40	·6021	6031	6042	6053	6064	6075	6085	6096	6107	6117		1	2	3	4	5	6	7	9	10
41	·6128	6138	6149	6160	6170	6180	6191	6201	6212	6222		1	2	3	4	5	6	7	8	9
42	·6232	6243	6253	6263	6274	6284	6294	6304	6314	6325		1	2	3	4	5	6	7	8	9
43	·6335	6345	6355	6365	6375	6385	6395	6405	6415	6425	10	1	2	3	4	5	6	7	8	9
44	·6435	6444	6454	6464	6474	6484	6493	6503	6513	6522		1	2	3	4	5	6	7	8	9
45	·6532	6542	6551	6561	6571	6580	6590	6599	6609	6618		1	2	3	4	5	6	7	8	9
46	·6628	6637	6646	6656	6665	6675	6684	6693	6702	6712		1	2	3	4	5	6	7	7	8
47	·6721	6730	6739	6749	6758	6767	6776	6785	6794	6803		1	2	3	4	5	5	6	7	8
48	·6812	6821	6830	6839	6848	6857	6866	6875	6884	6893	9	1	2	3	4	5	5	6	7	8
49	·6902	6911	6920	6928	6937	6946	6955	6964	6972	6981		1	2	3	4	4	5	6	7	8

USEFUL CONSTANTS WITH THEIR LOGARITHMS

	No.	Log.		No.	Log.		No.	Log.
π	3·14159	0·4971	1 radian	57°·296	1·7581	e	2·71828	0·4343
$\frac{1}{\pi}$	0·3183	$\bar{1}$·5029	1 radian	3437′·7	3·5363	M	0·4343	$\bar{1}$·6378
			1 radian	206265″	5·3144	$\frac{1}{M}$	2·3026	0·3622
π^2	9·8696	0·9943	arc 1°	0·017 453 293	$\bar{2}$·2419			
$\sqrt{\pi}$	1·7725	0·2486	arc 1′	0·000 290 888	$\bar{4}$·4637			
$\frac{4}{3}\pi$	4·1888	0·6221	arc 1″	0·000 004 848	$\bar{6}$·6856			

$$\log_e x = \frac{1}{M} . \log_{10} x$$

$$\log_{10} x = M . \log_e x$$

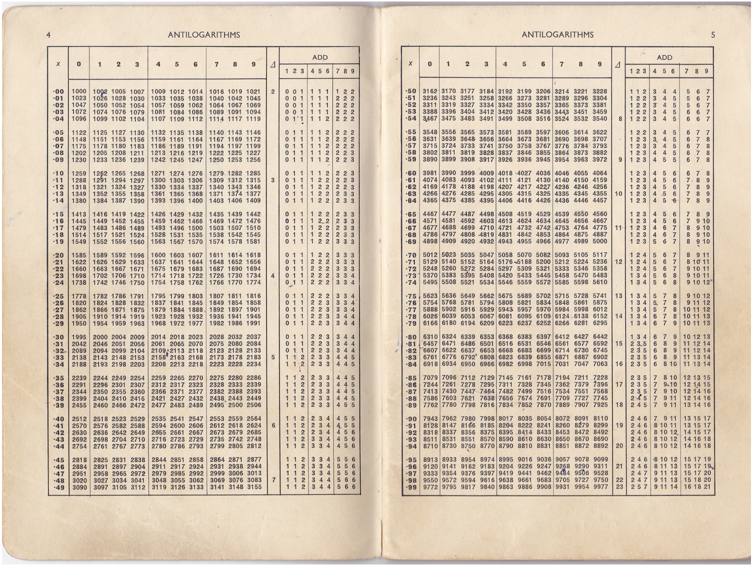
LOGARITHMS OF NUMBERS 3

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD								
												1	2	3	4	5	6	7	8	9
50	·6990	6998	7007	7016	7024	7033	7042	7050	7059	7067		1	2	3	3	4	5	6	7	8
51	·7076	7084	7093	7101	7110	7118	7126	7135	7143	7152		1	2	3	3	4	5	6	7	8
52	·7160	7168	7177	7185	7193	7202	7210	7218	7226	7235		1	2	2	3	4	5	6	7	7
53	·7243	7251	7259	7267	7275	7284	7292	7300	7308	7316		1	2	2	3	4	5	6	6	7
54	·7324	7332	7340	7348	7356	7364	7372	7380	7388	7396	8	1	2	2	3	4	5	6	6	7
55	·7404	7412	7419	7427	7435	7443	7451	7459	7466	7474		1	2	2	3	4	5	5	6	7
56	·7482	7490	7497	7505	7513	7520	7528	7536	7543	7551		1	2	2	3	4	5	5	6	7
57	·7559	7566	7574	7582	7589	7597	7604	7612	7619	7627		1	2	2	3	4	5	5	6	7
58	·7634	7642	7649	7657	7664	7672	7679	7686	7694	7701		1	2	2	3	4	4	5	6	7
59	·7709	7716	7723	7731	7738	7745	7752	7760	7767	7774		1	1	2	3	4	4	5	6	7
60	·7782	7789	7796	7803	7810	7818	7825	7832	7839	7846	7	1	1	2	3	4	4	5	6	6
61	·7853	7860	7868	7875	7882	7889	7896	7903	7910	7917		1	1	2	3	4	4	5	6	6
62	·7924	7931	7938	7945	7952	7959	7966	7973	7980	7987		1	1	2	3	3	4	5	6	6
63	·7993	8000	8007	8014	8021	8028	8035	8041	8048	8055		1	1	2	3	3	4	5	6	6
64	·8062	8069	8075	8082	8089	8096	8102	8109	8116	8122		1	1	2	3	3	4	5	5	6
65	·8129	8136	8142	8149	8156	8162	8169	8176	8182	8189		1	1	2	3	3	4	5	5	6
66	·8195	8202	8209	8215	8222	8228	8235	8241	8248	8254		1	1	2	3	3	4	5	5	6
67	·8261	8267	8274	8280	8287	8293	8299	8306	8312	8319		1	1	2	3	3	4	4	5	6
68	·8325	8331	8338	8344	8351	8357	8363	8370	8376	8382		1	1	2	3	3	4	4	5	6
69	·8388	8395	8401	8407	8414	8420	8426	8432	8439	8445		1	1	2	3	3	4	4	5	6
70	·8451	8457	8463	8470	8476	8482	8488	8494	8500	8506		1	1	2	2	3	4	4	5	6
71	·8513	8519	8525	8531	8537	8543	8549	8555	8561	8567	6	1	1	2	2	3	4	4	5	5
72	·8573	8579	8585	8591	8597	8603	8609	8615	8621	8627		1	1	2	2	3	4	4	5	5
73	·8633	8639	8645	8651	8657	8663	8669	8675	8681	8686		1	1	2	2	3	4	4	5	5
74	·8692	8698	8704	8710	8716	8722	8727	8733	8739	8745		1	1	2	2	3	4	4	5	5
75	·8751	8756	8762	8768	8774	8779	8785	8791	8797	8802		1	1	2	2	3	3	4	5	5
76	·8808	8814	8820	8825	8831	8837	8842	8848	8854	8859		1	1	2	2	3	3	4	5	5
77	·8865	8871	8876	8882	8887	8893	8899	8904	8910	8915		1	1	2	2	3	3	4	4	5
78	·8921	8927	8932	8938	8943	8949	8954	8960	8965	8971		1	1	2	2	3	3	4	4	5
79	·8976	8982	8987	8993	8998	9004	9009	9015	9020	9025		1	1	2	2	3	3	4	4	5
80	·9031	9036	9042	9047	9053	9058	9063	9069	9074	9079		1	1	2	2	3	3	4	4	5
81	·9085	9090	9096	9101	9106	9112	9117	9122	9128	9133		1	1	2	2	3	3	4	4	5
82	·9138	9143	9149	9154	9159	9165	9170	9175	9180	9186		1	1	2	2	3	3	4	4	5
83	·9191	9196	9201	9206	9212	9217	9222	9227	9232	9238		1	1	2	2	3	3	4	4	5
84	·9243	9248	9253	9258	9263	9269	9274	9279	9284	9289		1	1	2	2	3	3	4	4	5
85	·9294	9299	9304	9309	9315	9320	9325	9330	9335	9340		1	1	2	2	3	3	4	4	5
86	·9345	9350	9355	9360	9365	9370	9375	9380	9385	9390	5	1	1	2	2	3	3	4	4	5
87	·9395	9400	9405	9410	9415	9420	9425	9430	9435	9440		0	1	1	2	2	3	3	4	4
88	·9445	9450	9455	9460	9465	9469	9474	9479	9484	9489		0	1	1	2	2	3	3	4	4
89	·9494	9499	9504	9509	9513	9518	9523	9528	9533	9538		0	1	1	2	2	3	3	4	4
90	·9542	9547	9552	9557	9562	9566	9571	9576	9581	9586		0	1	1	2	2	3	3	4	4
91	·9590	9595	9600	9605	9609	9614	9619	9624	9628	9633		0	1	1	2	2	3	3	4	4
92	·9638	9643	9647	9652	9657	9661	9666	9671	9675	9680		0	1	1	2	2	3	3	4	4
93	·9685	9689	9694	9699	9703	9708	9713	9717	9722	9727		0	1	1	2	2	3	3	4	4
94	·9731	9736	9741	9745	9750	9754	9759	9763	9768	9773		0	1	1	2	2	3	3	4	4
95	·9777	9782	9786	9791	9795	9800	9805	9809	9814	9818		0	1	1	2	2	3	3	4	4
96	·9823	9827	9832	9836	9841	9845	9850	9854	9859	9863		0	1	1	2	2	3	3	4	4
97	·9868	9872	9877	9881	9886	9890	9894	9899	9903	9908		0	1	1	2	2	3	3	4	4
98	·9912	9917	9921	9926	9930	9934	9939	9943	9948	9952		0	1	1	2	2	3	3	4	4
99	·9956	9961	9965	9969	9974	9978	9983	9987	9991	9996	4	0	1	1	2	2	2	3	3	4

Only the decimal portion (*mantissa*) of each logarithm is shown in this table. The integral portion (*characteristic*) must be determined independently.

Log Tables

Antilogarithms



4 ANTILOGARITHMS

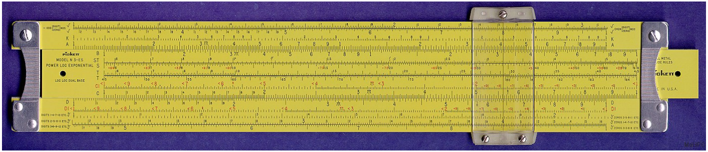
x	0	1	2	3	4	5	6	7	8	9	Δ	ADD								
												1	2	3	4	5	6	7	8	9
·00	1000	1002	1005	1007	1009	1012	1014	1016	1019	1021	2	0	0	1	1	1	1	1	2	2
·01	1023	1026	1028	1030	1033	1035	1038	1040	1042	1045		0	0	1	1	1	1	2	2	2
·02	1047	1050	1052	1054	1057	1059	1062	1064	1067	1069		0	0	1	1	1	1	2	2	2
·03	1072	1074	1076	1079	1081	1084	1086	1089	1091	1094		0	0	1	1	1	1	2	2	2
·04	1096	1099	1102	1104	1107	1109	1112	1114	1117	1119		0	1	1	1	1	2	2	2	2
·05	1122	1125	1127	1130	1132	1135	1138	1140	1143	1146		0	1	1	1	1	2	2	2	2
·06	1148	1151	1153	1156	1159	1161	1164	1167	1169	1172		0	1	1	1	1	2	2	2	2
·07	1175	1178	1180	1183	1186	1189	1191	1194	1197	1199		0	1	1	1	1	2	2	2	2
·08	1202	1205	1208	1211	1213	1216	1219	1222	1225	1227		0	1	1	1	1	2	2	2	3
·09	1230	1233	1236	1239	1242	1245	1247	1250	1253	1256		0	1	1	1	1	2	2	2	3
·10	1259	1262	1265	1268	1271	1274	1276	1279	1282	1285		0	1	1	1	1	2	2	2	3
·11	1288	1291	1294	1297	1300	1303	1306	1309	1312	1315	3	0	1	1	1	2	2	2	2	3
·12	1318	1321	1324	1327	1330	1334	1337	1340	1343	1346		0	1	1	1	2	2	2	2	3
·13	1349	1352	1355	1358	1361	1365	1368	1371	1374	1377		0	1	1	1	2	2	2	3	3
·14	1380	1384	1387	1390	1393	1396	1400	1403	1406	1409		0	1	1	1	2	2	2	3	3
·15	1413	1416	1419	1422	1426	1429	1432	1435	1439	1442		0	1	1	1	2	2	2	3	3
·16	1445	1449	1452	1455	1459	1462	1466	1469	1472	1476		0	1	1	1	2	2	2	3	3
·17	1479	1483	1486	1489	1493	1496	1500	1503	1507	1510		0	1	1	1	2	2	2	3	3
·18	1514	1517	1521	1524	1528	1531	1535	1538	1542	1545		0	1	1	1	2	2	2	3	3
·19	1549	1552	1556	1560	1563	1567	1570	1574	1578	1581		0	1	1	1	2	2	3	3	3
·20	1585	1589	1592	1596	1600	1603	1607	1611	1614	1618		0	1	1	1	2	2	3	3	3
·21	1622	1626	1629	1633	1637	1641	1644	1648	1652	1656		0	1	1	2	2	2	3	3	3
·22	1660	1663	1667	1671	1675	1679	1683	1687	1690	1694		0	1	1	2	2	2	3	3	3
·23	1698	1702	1706	1710	1714	1718	1722	1726	1730	1734	4	0	1	1	2	2	2	3	3	4
·24	1738	1742	1746	1750	1754	1758	1762	1766	1770	1774		0	1	1	2	2	2	3	3	4
·25	1778	1782	1786	1791	1795	1799	1803	1807	1811	1816		0	1	1	2	2	2	3	3	4
·26	1820	1824	1828	1832	1837	1841	1845	1849	1854	1858		0	1	1	2	2	3	3	3	4
·27	1862	1866	1871	1875	1879	1884	1888	1892	1897	1901		0	1	1	2	2	3	3	3	4
·28	1905	1910	1914	1919	1923	1928	1932	1936	1941	1945		0	1	1	2	2	3	3	4	4
·29	1950	1954	1959	1963	1968	1972	1977	1982	1986	1991		0	1	1	2	2	3	3	4	4
·30	1995	2000	2004	2009	2014	2018	2023	2028	2032	2037		0	1	1	2	2	3	3	4	4
·31	2042	2046	2051	2056	2061	2065	2070	2075	2080	2084		0	1	1	2	2	3	3	4	4
·32	2089	2094	2099	2104	2109	2113	2118	2123	2128	2133		0	1	1	2	2	3	3	4	4
·33	2138	2143	2148	2153	2158	2163	2168	2173	2178	2183	5	1	1	2	2	3	3	4	4	5
·34	2188	2193	2198	2203	2208	2213	2218	2223	2228	2234		1	1	2	2	3	3	4	4	5
·35	2239	2244	2249	2254	2259	2265	2270	2275	2280	2286		1	1	2	2	3	3	4	4	5
·36	2291	2296	2301	2307	2312	2317	2323	2328	2333	2339		1	1	2	2	3	3	4	4	5
·37	2344	2350	2355	2360	2366	2371	2377	2382	2388	2393		1	1	2	2	3	3	4	4	5
·38	2399	2404	2410	2415	2421	2427	2432	2438	2443	2449		1	1	2	2	3	3	4	4	5
·39	2455	2460	2466	2472	2477	2483	2489	2495	2500	2506		1	1	2	2	3	3	4	5	5
·40	2512	2518	2523	2529	2535	2541	2547	2553	2559	2564		1	1	2	2	3	4	4	5	5
·41	2570	2576	2582	2588	2594	2600	2606	2612	2618	2624	6	1	1	2	2	3	4	4	5	5
·42	2630	2636	2642	2649	2655	2661	2667	2673	2679	2685		1	1	2	2	3	4	4	5	6
·43	2692	2698	2704	2710	2716	2723	2729	2735	2742	2748		1	1	2	3	3	4	4	5	6
·44	2754	2761	2767	2773	2780	2786	2793	2799	2805	2812		1	1	2	3	3	4	4	5	6
·45	2818	2825	2831	2838	2844	2851	2858	2864	2871	2877		1	1	2	3	3	4	5	5	6
·46	2884	2891	2897	2904	2911	2917	2924	2931	2938	2944		1	1	2	3	3	4	5	5	6
·47	2951	2958	2965	2972	2979	2985	2992	2999	3006	3013		1	1	2	3	3	4	5	5	6
·48	3020	3027	3034	3041	3048	3055	3062	3069	3076	3083	7	1	1	2	3	4	4	5	6	6
·49	3090	3097	3105	3112	3119	3126	3133	3141	3148	3155		1	1	2	3	4	4	5	6	6

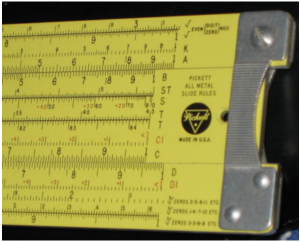
ANTILOGARITHMS 5

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD								
												1	2	3	4	5	6	7	8	9
·50	3162	3170	3177	3184	3192	3199	3206	3214	3221	3228		1	1	2	3	4	4	5	6	7
·51	3236	3243	3251	3258	3266	3273	3281	3289	3296	3304		1	2	2	3	4	5	5	6	7
·52	3311	3319	3327	3334	3342	3350	3357	3365	3373	3381		1	2	2	3	4	5	5	6	7
·53	3388	3396	3404	3412	3420	3428	3436	3443	3451	3459		1	2	2	3	4	5	6	6	7
·54	3467	3475	3483	3491	3499	3508	3516	3524	3532	3540	8	1	2	2	3	4	5	6	6	7
·55	3548	3556	3565	3573	3581	3589	3597	3606	3614	3622		1	2	2	3	4	5	6	7	7
·56	3631	3639	3648	3656	3664	3673	3681	3690	3698	3707		1	2	3	3	4	5	6	7	8
·57	3715	3724	3733	3741	3750	3758	3767	3776	3784	3793		1	2	3	3	4	5	6	7	8
·58	3802	3811	3819	3828	3837	3846	3855	3864	3873	3882		1	2	3	4	4	5	6	7	8
·59	3890	3899	3908	3917	3926	3936	3945	3954	3963	3972	9	1	2	3	4	5	5	6	7	8
·60	3981	3990	3999	4009	4018	4027	4036	4046	4055	4064		1	2	3	4	5	6	6	7	8
·61	4074	4083	4093	4102	4111	4121	4130	4140	4150	4159		1	2	3	4	5	6	7	8	9
·62	4169	4178	4188	4198	4207	4217	4227	4236	4246	4256		1	2	3	4	5	6	7	8	9
·63	4266	4276	4285	4295	4305	4315	4325	4335	4345	4355	10	1	2	3	4	5	6	7	8	9
·64	4365	4375	4385	4395	4406	4416	4426	4436	4446	4457		1	2	3	4	5	6	7	8	9
·65	4467	4477	4487	4498	4508	4519	4529	4539	4550	4560		1	2	3	4	5	6	7	8	9
·66	4571	4581	4592	4603	4613	4624	4634	4645	4656	4667		1	2	3	4	5	6	7	9	10
·67	4677	4688	4699	4710	4721	4732	4742	4753	4764	4775	11	1	2	3	4	6	7	8	9	10
·68	4786	4797	4808	4819	4831	4842	4853	4864	4875	4887		1	2	3	4	6	7	8	9	10
·69	4898	4909	4920	4932	4943	4955	4966	4977	4989	5000		1	2	3	5	6	7	8	9	10
·70	5012	5023	5035	5047	5058	5070	5082	5093	5105	5117		1	2	4	5	6	7	8	9	11
·71	5129	5140	5152	5164	5176	5188	5200	5212	5224	5236	12	1	2	4	5	6	7	8	10	11
·72	5248	5260	5272	5284	5297	5309	5321	5333	5346	5358		1	2	4	5	6	7	9	10	11
·73	5370	5383	5395	5408	5420	5433	5445	5458	5470	5483		1	3	4	5	6	8	9	10	11
·74	5495	5508	5521	5534	5546	5559	5572	5585	5598	5610		1	3	4	5	6	8	9	10	12
·75	5623	5636	5649	5662	5675	5689	5702	5715	5728	5741	13	1	3	4	5	7	8	9	10	12
·76	5754	5768	5781	5794	5808	5821	5834	5848	5861	5875		1	3	4	5	7	8	9	11	12
·77	5888	5902	5916	5929	5943	5957	5970	5984	5998	6012		1	3	4	5	7	8	10	11	12
·78	6026	6039	6053	6067	6081	6095	6109	6124	6138	6152	14	1	3	4	6	7	8	10	11	13
·79	6166	6180	6194	6209	6223	6237	6252	6266	6281	6295		1	3	4	6	7	9	10	11	13
·80	6310	6324	6339	6353	6368	6383	6397	6412	6427	6442		1	3	4	6	7	9	10	12	13
·81	6457	6471	6486	6501	6516	6531	6546	6561	6577	6592	15	2	3	5	6	8	9	11	12	14
·82	6607	6622	6637	6653	6668	6683	6699	6714	6730	6745		2	3	5	6	8	9	11	12	14
·83	6761	6776	6792	6808	6823	6839	6855	6871	6887	6902		2	3	5	6	8	9	11	13	14
·84	6918	6934	6950	6966	6982	6998	7015	7031	7047	7063	16	2	3	5	6	8	10	11	13	14
·85	7079	7096	7112	7129	7145	7161	7178	7194	7211	7228		2	3	5	7	8	10	12	13	15
·86	7244	7261	7278	7295	7311	7328	7345	7362	7379	7396	17	2	3	5	7	9	10	12	14	15
·87	7413	7430	7447	7464	7482	7499	7516	7534	7551	7568		2	3	5	7	9	10	12	14	16
·88	7586	7603	7621	7638	7656	7674	7691	7709	7727	7745		2	4	5	7	9	11	12	14	16
·89	7762	7780	7798	7816	7834	7852	7870	7889	7907	7925	18	2	4	5	7	9	11	13	14	16
·90	7943	7962	7980	7998	8017	8035	8054	8072	8091	8110		2	4	6	7	9	11	13	15	17
·91	8128	8147	8166	8185	8204	8222	8241	8260	8279	8299	19	2	4	6	8	10	11	13	15	17
·92	8318	8337	8356	8375	8395	8414	8433	8453	8472	8492		2	4	6	8	10	12	14	15	17
·93	8511	8531	8551	8570	8590	8610	8630	8650	8670	8690		2	4	6	8	10	12	14	16	18
·94	8710	8730	8750	8770	8790	8810	8831	8851	8872	8892	20	2	4	6	8	10	12	14	16	18
·95	8913	8933	8954	8974	8995	9016	9036	9057	9078	9099		2	4	6	8	10	12	15	17	19
·96	9120	9141	9162	9183	9204	9226	9247	9268	9290	9311	21	2	4	6	8	11	13	15	17	19
·97	9333	9354	9376	9397	9419	9441	9462	9484	9506	9528		2	4	7	9	11	13	15	17	20
·98	9550	9572	9594	9616	9638	9661	9683	9705	9727	9750	22	2	4	7	9	11	13	15	18	20
·99	9772	9795	9817	9840	9863	9886	9908	9931	9954	9977	23	2	5	7	9	11	14	16	18	21

Slide Rules

Pickett N 3-ES from 1967

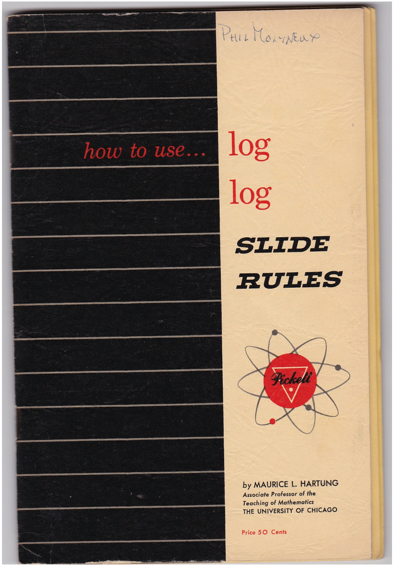




- See Oughtred Society
- UKSRC
- Rod Lovett's Slide Rules
- Slide Rule Museum

Slide Rules

Pickett log log Slide Rules Manual 1953

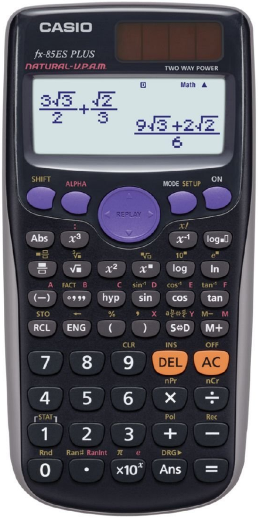


Calculators

HP HP-21 Calculator from 1975 £69

Calculators

Casio fx-85GT PLUS Calculator from 2013 £10



Calculators

Calculator Links

- **HP Calculator Museum** http://www.hpmuseum.org
- **HP Calculator Emulators**
  http://nonpareil.brouhaha.com
- **HP Calculator Emulators for OS X**
  http://www.bartosiak.org/nonpareil/
- **Vintage Calculators Web Museum**
  http://www.vintagecalculators.com

Example Calculation

Log Tables, Slide Rule and Calculator

- Evaluate 89.7×597
- **Knott's Tables**
- $\log_{10} 89.7 = 1.9528$ and $\log_{10} 597 = 2.7760$
- Shows *mantissa* (decimal) & *characteristic* (integral)
- Add 4.7288, take *antilog* to get $5346 + 10 = 5.356 \times 10^4$
- **HP-21 Calculator** — set display to 4 decimal places
- 89.7 $\boxed{\text{log}}$ = 1.9528 and 597 $\boxed{\text{log}}$ = 2.7760
- $\boxed{+}$ displays 4.7288
- 10 $\boxed{\text{ENTER}}$, $\boxed{x \rightleftarrows y}$ and $\boxed{y^x}$ displays 53550.9000
- **Casio fx-85GT PLUS**
- $\boxed{\text{log}}$ 89.7 $\boxed{)}$ = 1.952792443 $\boxed{+}$ $\boxed{\text{log}}$ 597 $\boxed{)}$ = 2.775974331 $\boxed{=}$
- 4.728766774 $\boxed{\text{Ans}}$ + $\boxed{10^x}$ gives 53550.9

Boolean Expressions

Traffic Lights Example (1)

- Consider traffic light at the intersection of roads *AC* and *BD* with the following rules for the *AC* controller
- Vehicles should not wait on red on *BD* for too long.
- If there is a long queue on *AC* then *BD* is only given a green for a short interval.
- If both queues are long the usual flow times are used.
- We use the following propositions:
  - w Vehicles have been waiting on red on *BD* for too long
  - q Queue on *AC* is too long
  - r Queue on *BD* is too long
- Given the following events:
  - `ToBD` Change flow to *BD*
  - `ToBDShort` Change flow to *BD* for short time
  - `NoChange` No Change to lights
- Express above as truth table, outcome tree, boolean expression

Boolean Expressions

Traffic Lights Example (2)

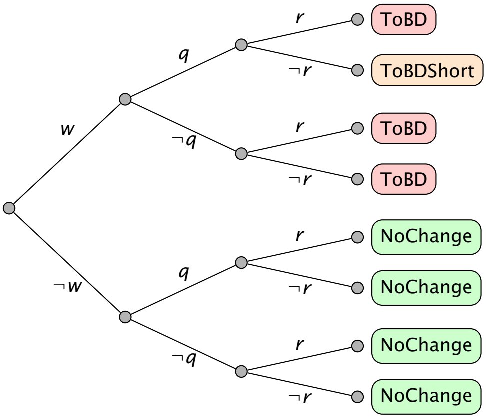
- **Traffic Lights outcome table**

w	q	r	Event
T	T	T	ToBD
T	T	F	ToBDShort
T	F	T	ToBD
T	F	F	ToBD
F	T	T	NoChange
F	T	F	NoChange
F	F	T	NoChange
F	F	F	NoChange

Boolean Expressions

Traffic Lights Example (3)

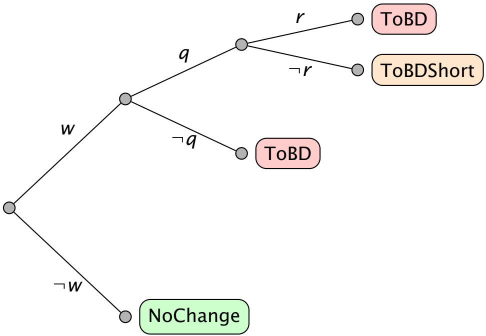
- **Traffic lights outcome tree**



Boolean Expressions

Traffic Lights Example (4)

- **Traffic lights outcome tree** simplified



Boolean Expressions

Traffic Lights Example (5)

- **Traffic Lights code 01**
- See M269TutorialProgPythonADT01.py

```python
def trafficLights01(w,q,r) :
  """
  Input 3 Booleans
  Return Event string
  """
  if w :
    if q :
      if r :
        evnt = "ToBD"
      else :
        evnt = "ToBDShort"
    else :
      evnt = "ToBD"
  else :
    evnt = "NoChange"
  return evnt
```

Boolean Expressions

Traffic Lights Example (6)

- **Traffic Lights test code 01**

```
trafficLights01Evnts = [((w,q,r), trafficLights01(w,q,r))
                        for w in [True,False]
                        for q in [True,False]
                        for r in [True,False]]

assert trafficLights01Evnts \
  == [((True, True, True), 'ToBD')
      ,((True, True, False), 'ToBDShort')
      ,((True, False, True), 'ToBD')
      ,((True, False, False), 'ToBD')
      ,((False, True, True), 'NoChange')
      ,((False, True, False), 'NoChange')
      ,((False, False, True), 'NoChange')
      ,((False, False, False), 'NoChange')]
```

Boolean Expressions

Traffic Lights Example (7)

- **Traffic Lights code 02** compound Boolean conditions

```python
def trafficLights02(w,q,r) :
  """
  Input 3 Booleans
  Return Event string
  """
  if ((w and q and r) or (w and not q)) :
    evnt = "ToBD"
  elif (w and q and not r) :
    evnt = "ToBDShort"
  else :
    evnt = "NoChange"
  return evnt
```

- What objectives do we have for our code ?

Boolean Expressions

Traffic Lights Example (8)

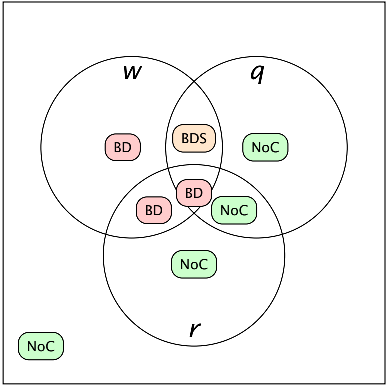
- **Traffic Lights test code 02**

```
trafficLights02Evnts = [((w,q,r), trafficLights02(w,q,r))
                                 for w in [True,False]
                                 for q in [True,False]
                                 for r in [True,False]]

assert trafficLights02Evnts == trafficLights01Evnts
```

Boolean Expressions

Traffic Lights Example (9)



- **Traffic Lights Venn diagram**
- OK using a fill colour would look better but didn't have the time to hack the package

Boolean Expressions

Validity

- **Validity** of Boolean expressions
- **Complete** every outcome returns an event (or error message, raises an exception)
- **Consistent** — we do not want two nested `if` statements or expressions resulting in different events
- We check this by ensuring that the events form a disjoint partition of the set of outcomes — see the Venn diagram
- We would quite like the programming language processor to warn us otherwise — not always possible

Booleans Expressions

Rail Ticket Exercise (1)

- Rail ticket discounts for:
  - *c* Rail card
  - *q* Off-peak time
  - *s* Special offer
- 4 fares: Standard, Reduced, Special, Super Special
- Rules:
  1. Reduced fare if rail card or at off-peak time
  2. Without rail card no reduction for both special offer and off-peak.
  3. Rail card always has reduced fare but cannot get off-peak discount as well.
  4. Rail card gets super special discount for journey with special offer
- Draw up truth table, outcome tree, Venn diagram and conditional statement (or expression) for this

Booleans Expressions

Rail Ticket Exercise (2)

- Rail ticket outcome table

c	q	s	Event
T	T	T	Super Special
T	T	F	Reduced
T	F	T	Super Special
T	F	F	Reduced
F	T	T	Special
F	T	F	Reduced
F	F	T	Special
F	F	F	Standard

Booleans Expressions

Rail Ticket Exercise (3)

- Rail ticket outcome table
- Note that it may be more convenient to change columns

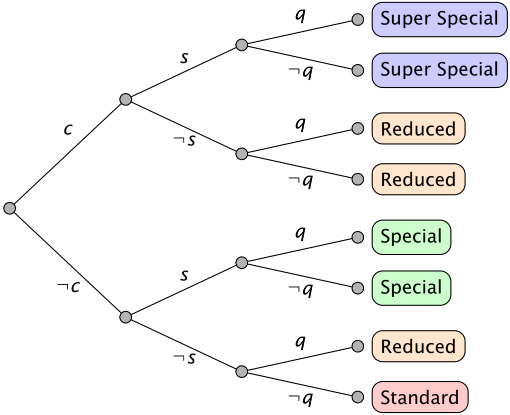
c	s	q	Event
T	T	T	Super Special
T	T	F	Super Special
T	F	T	Reduced
T	F	F	Reduced
F	T	T	Special
F	T	F	Special
F	F	T	Reduced
F	F	F	Standard

- Real fares are a little more complex — see brfares.com

Boolean Expressions

Rail Ticket Exercise (4)

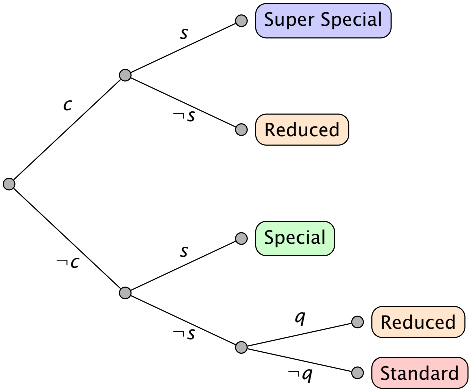
- **Rail Ticket outcome tree**



Boolean Expressions

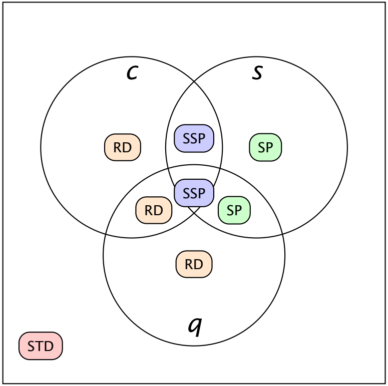
Rail Ticket Exercise (5)

- **Rail Ticket outcome tree** simplified



Boolean Expressions

Rail Ticket Example (6)



- **Rail Ticket Venn diagram**

Boolean Expressions

Rail Ticket Example (7)

- **Rail Ticket code 01**

```python
def railTicket01(c,s,q) :
  """
  Input 3 Booleans
  Return Event string
  """
  if c :
    if s :
      evnt = "SSP"
    else :
      evnt = "RD"
  else :
    if s :
      evnt = "SP"
    else :
      if q :
        evnt = "RD"
      else :
        evnt = "STD"
  return evnt
```

Boolean Expressions

Rail Ticket Example (8)

- **Rail Ticket test code 01**

```
railTicket01Evnts = [((c,s,q), railTicket01(c,s,q))
                              for c in [True,False]
                              for s in [True,False]
                              for q in [True,False]]

assert railTicket01Evnts \
  == [((True, True, True), 'SSP')
       ,((True, True, False), 'SSP')
       ,((True, False, True), 'RD')
       ,((True, False, False), 'RD')
       ,((False, True, True), 'SP')
       ,((False, True, False), 'SP')
       ,((False, False, True), 'RD')
       ,((False, False, False), 'STD')]
```

Boolean Expressions

Rail Ticket Example (9)

- **Rail Ticket code 02** compound Boolean expressions

```python
def railTicket02(c,s,q) :
  """
  Input 3 Booleans
  Return Event string
  """
  if (c and s) :
    evnt = "SSP"
  elif ((c and not s) or (not c and not s and q)) :
    evnt = "RD"
  elif (not c and s) :
    evnt = "SP"
  else :
    evnt = "STD"
  return evnt
```

Boolean Expressions

Rail Ticket Example (10)

- **Rail Ticket test code 02**

```python
railTicket02Evnts = [((c,s,q), railTicket02(c,s,q))
                              for c in [True,False]
                              for s in [True,False]
                              for q in [True,False]]

assert railTicket02Evnts == railTicket01Evnts
```

Propositional Calculus

Introduction

- Unit 2 section 3.2 *A taste of formal logic* introduces Propositional calculus
- A language for calculating about Booleans — *truth values*
- Gives operators (connectives) conjunction ($\wedge$) AND, disjunction ($\vee$) OR, negation ($\neg$) NOT, implication ($\Rightarrow$) IF
- There are 16 possible functions $(\mathbb{B}, \mathbb{B}) \rightarrow \mathbb{B}$ — see below — defined by their truth tables
- **Discussion** Did you find the truth table for implication weird or surprising ?

Propositional Calculus

Implication

- **Implication** has a **negative** definition — we accept its truth unless we have contrary evidence
- $T \Rightarrow T == T$ and $T \Rightarrow F == F$
- Hence 4 possibilities for truth table

p	q	$p \Rightarrow q$	q	$p \Leftrightarrow q$	$p \wedge q$
T	T	T	T	T	T
T	F	F	F	F	F
F	T	T	T	F	F
F	F	T	F	T	F

- $(\Rightarrow)$ must have the entry shown — the others are taken
- Do not think of *p causing q*

Propositional Calculus

Functional Completeness, Boolean Programming

- **Functionally complete** set of connectives is one which can be used to express all possible connectives
- $p \Rightarrow q \equiv \neg p \vee q$ so we could just use $\{\neg, \wedge, \vee\}$
- **Boolean programming** — we have to have a functionally complete set but choose more to make the programming easier
- *Expressiveness* is an issue in programming language design

Propositional Calculus

NAND, NOR

- **NAND** $\boldsymbol{p \overline{\wedge} q}$, $\boldsymbol{p \uparrow q}$, Sheffer stroke
- **NOR** $\boldsymbol{p \overline{\vee} q}$, $\boldsymbol{p \downarrow q}$, Pierce's arrow
- See truth tables below — both $\{\uparrow\}, \{\downarrow\}$ are functionally complete
- **Exercise** verify
  - $\neg p \equiv p \uparrow p$
  - $p \wedge q \equiv \neg(p \uparrow q) = (p \uparrow q) \uparrow (p \uparrow q)$
  - $p \vee q \equiv (p \uparrow p) \uparrow (q \uparrow q)$
  - $\neg p \equiv p \downarrow p$
  - $p \wedge q \equiv (p \downarrow p) \downarrow (q \downarrow q)$
  - $p \vee q \equiv \neg(p \downarrow q) = (p \downarrow q) \downarrow (p \downarrow q)$
- Not a novelty — the Apollo Guidance Computer was implemented in NOR gates alone.

Truth Function

Truth Function References

- The following appendix notes illustrate the 16 binary functions of two Boolean variables
- See Truth function
- See Functional completeness
- See Sheffer stroke
- See Logical NOR

Truth Function

Table of Binary Truth Functions

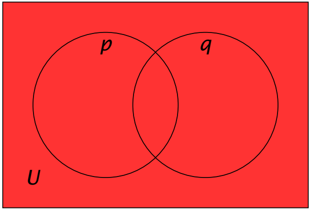
p	q	$\top$	$p \vee q$	$p \Leftarrow q$	p	$p \Rightarrow q$	q	$p \Leftrightarrow q$	$p \wedge q$
T	T	T	T	T	T	T	T	T	T
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

p	q	$\bot$	$p \overline{\vee} q$	$p \nLeftarrow q$	$\neg p$	$p \nRightarrow q$	$\neg q$	$p \nLeftrightarrow q$	$p \overline{\wedge} q$
T	T	F	F	F	F	F	F	F	F
T	F	F	F	F	F	T	T	T	T
F	T	F	F	T	T	F	F	T	T
F	F	F	T	F	T	F	T	F	T

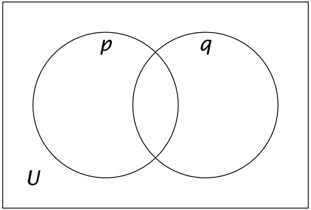
Truth Function

Tautology/Contradiction

- **Tautology** True, $\top$, *Top*



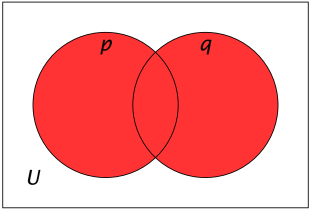
- **Contradiction** False, $\perp$, *Bottom*



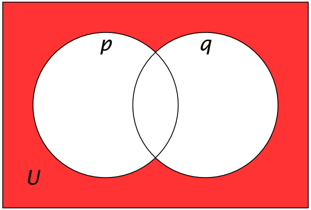
Truth Function

Disjunction/Joint Denial

- **Disjunction** OR, $p \vee q$



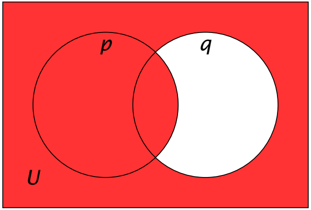
- **Joint Denial** NOR, $p \overline{\vee} q$, $p \downarrow q$, *Pierce's arrow*



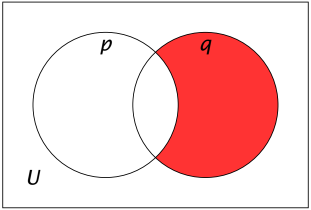
Truth Function

Converse Implication/Converse Nonimplication

- **Converse Implication** $p \Leftarrow q$



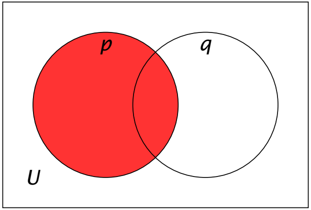
- **Converse Nonimplication** $p \nLeftarrow q$



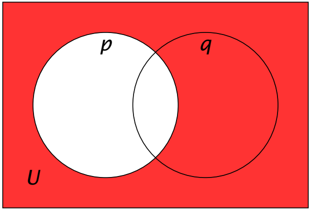
Truth Function

Proposition p/Negation of p

- **Proposition** p



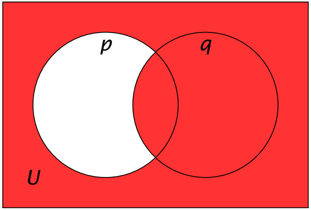
- **Negation of** p



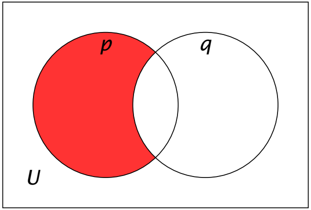
Truth Function

Material Implication/Material Nonimplication

- **Material Implication** $p \Rightarrow q$



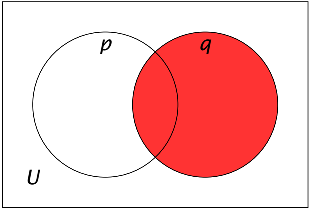
- **Material Nonimplication** $p \not\Rightarrow q$



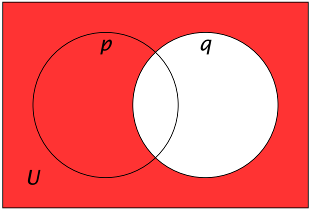
Truth Function

Proposition q/Negation of q

- **Proposition q q**



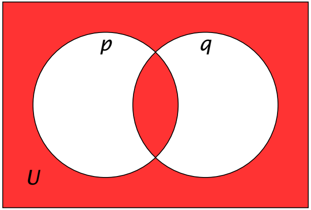
- **Negation of q $\neg q$**



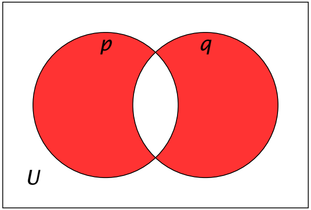
Truth Function

Biconditional/Exclusive disjunction

- **Biconditional** If and only if, IFF, $p \Leftrightarrow q$



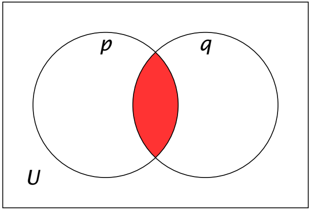
- **Exclusive disjunction** XOR, $p \not\Leftrightarrow q$



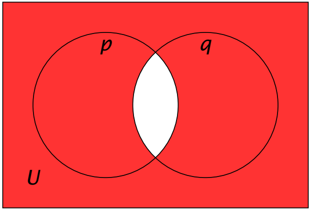
Truth Function

Conjunction/Alternative denial

- **Conjunction** AND, $p \wedge q$



- **Alternative denial** NAND, $p \barwedge q$, $p \uparrow q$, *Sheffer stroke*



Web Sites

Computability

- **Logic**
  - WFF, WFF'N Proof online
- **Computability**
  - Computability
  - Computable function
  - Decidability (logic)
  - Turing Machines
  - Universal Turing Machine
  - Turing machine simulator
  - Lambda Calculus
  - Von Neumann Architecture
  - Turing Machine XKCD 205 Candy Button Paper
  - Turing Machine XKCD 505 A Bunch of Rocks
  - RIP John Conway Why can Conway's Game of Life be classified as a universal machine?
  - Phil Wadler Bright Club on Computability
  - Bridges: Theory of Computation: Halting Problem
  - Bridges: Theory of Computation: Other Non-computable Problems

Web Sites
Complexity

- **Complexity**
  - Complexity class
  - NP complexity
  - NP complete
  - Reduction (complexity)
  - P versus NP problem
  - Graph of NP-Complete Problems