

Master of Technology in Knowledge Engineering

Annex 1: Algorithm Analysis

GU Zhan (Sam)

© 2014 NUS. The contents contained in this document may not be reproduced in any form or by any means,
without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Data Structures & Algorithms

- A **data structure** is a systematic way of organizing and accessing data
- An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem
 - » Is a step-by-step procedure for performing some task in a finite amount of time
 - » Is appropriate to the data structure used to describe the problem to be solved

Program = Data structure + Algorithm

Data Structures & Algorithms (cont.)

- The goals in designing high-level descriptions of data structures and algorithms:
 - » **Correctness** — to work correctly for **all** possible inputs that one might encounter within a certain domain of interest
 - » **Efficiency** — should be fast and not use any more of the computer's resources, such as memory space than is necessary
- (*) The implementation goals for data structure and algorithm (not discussed in this course)
 - » Robustness, Adaptability, Reusability

Design of Algorithms: problem analysis

- **Input**
 - » **Acceptable data type**
 - » **Possible data range**
 - » **Possible error input**
- **Output**
 - » **Expected result**
 - » **Possible error**
- **Relation from input to output**
 - » **Possible mapping**
 - ♦ **Handling different cases**
 - » **Necessary processing involved**
 - ♦ **Processing steps**

Design of Algorithms: problem modeling

- **Selection of data structure**
 - » **Easy for handling**
 - » **Convenient for description of problem relation**
- **Problem break down**
 - » **Top level tasks**
 - » **Data / information connection between tasks**
 - ♦ **Signals and states to indicate different cases during processing**

Design of Algorithm: working example

- **Example 1:**

- » **Task**

- ♦ find the location of largest integer from the elements of an array that stores positive integers (not necessarily distinct)

- » **Input**

- ♦ A list of integers

- » **Output**

- ♦ The location of the largest integer in the list
 - When there are more than one occurrence of the same integer, the first position should be the result

Design of Algorithms: working example (cont.)

Example 1 (cont.)

- **Data structure**

- » **Array $A[0], \dots, A[n-1]$**
 - ♦ to store the original list
- » **Integer $currentMax$**
 - ♦ the largest integer seen
- » **Integer L**
 - ♦ the position of the integer kept in $currentMax$
- » **Integer i**
 - ♦ the current position of comparison

- **Main task**

- » **Compare all the integers in the list, to find the largest one**
-

Design of Algorithms: working example (cont.)

Example 1 (cont.)

- **Final situation**

- » **When all the integers are checked**

- ◆ Upon that point, *currentMax* should keep the largest integer and *L* should keep the position of the integer stored in *currentMax*

- **Range of comparison**

- » **$A[0] \sim A[n-1]$**

- **Initial setting**

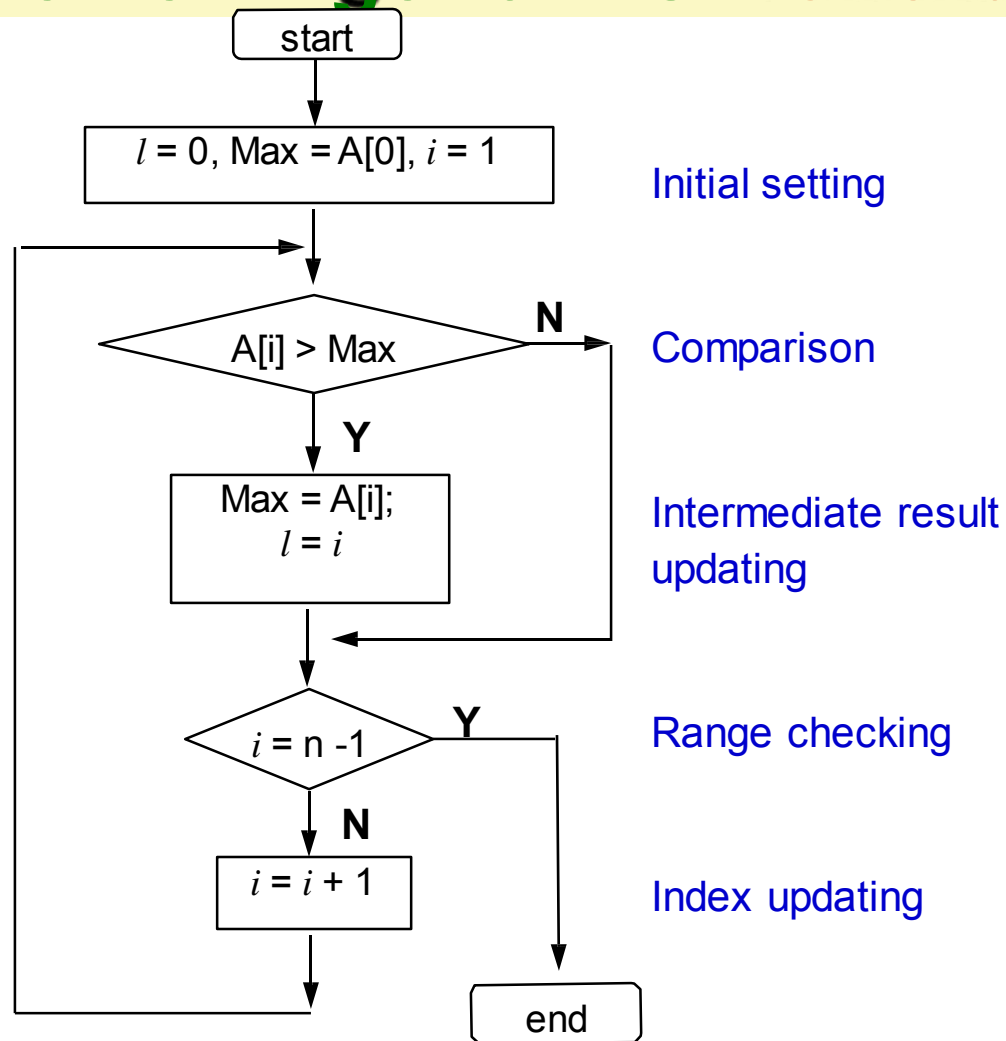
- » **$L = ?$**

- » **$currentMax = ?$**

- » **$i = ?$**

Description of Algorithms: flowchart

Example 1 (cont.)



Description of Algorithms: pseudo-code

- Computer scientists are often asked to describe algorithms in a way that
 - » is intended for human eyes only.
 - » is not a computer program, but the description generated can be easily converted to computer program.
- These “high-level” descriptions called *pseudo-code*
 - » combine natural language and familiar structures from a programming language, in a way that is both clear and informative.
 - » also facilitate the high-level analysis of a data structure or algorithm.

Design of Algorithm: pseudo-code

Example 1 (cont.)

» Pseudo-code

Algorithm **arrayMax(*A*, *n*)**

Input: An array ***A*** storing *n* positive integers

Output: The location of maximum element in the
array ***A***

currentMax := *A*[0]

L := 0

for *i* := 1 **to** *n*-1 **do**

if *currentMax* < *A*[*i*]

then

currentMax := *A*[*i*]

L := *i*

return *L*

Searching Algorithms

- **Searching** problems occur in many applications
 - » Locate an element x in a list of distinct elements a_1, a_2, \dots, a_n , or determine that it is not in the list.
 - » The solution to this search problem is
 - ♦ the location of the term in the list that equals x , or
 - ♦ 0 if x is not in the list
- Typical searching algorithms to be discussed
 - » Linear search (sequential search)
 - » Binary search

Searching Algorithms: linear search

- *Linear search* (sequential search)
 - » Begins by comparing x and a_1, a_2, \dots
 - ♦ When $x = a_1$
 - the solution is the location of a_1 , namely, 1
 - ♦ When $x \neq a_1$ compare x with a_2
 - When $x = a_2$, the solution is the location of a_2 , namely, 2
 - ♦
 - » Continue the process, comparing x with each term of the list until a match is found, or confirmed no match with the entire list

Searching Algorithms: linear search (cont.)

- Linear search (cont.)

Algorithm

linearSearch(integer x , distinct integers a_1, a_2, \dots, a_n)

$i := 1$

while ($i \leq n$ and $x \neq a_i$)

$i := i + 1$

if $i \leq n$ **then** $location := i$ (*if not found, $i > n$*)

else $location := 0$

- ♦ $location$ is the subscript of the term that equals x , or is 0 if x is not found

Searching Algorithms : binary search

- *Binary search*

- » Can be used when the list has terms *occurring in order* of increasing (decreasing)
- » It proceeds by comparing the element to be located to the middle term of the list
 - ♦ The list is then split into two smaller sublists of the same size (or with only one term fewer or more)
- » The search continues by restricting the search to the appropriate sublist based on the comparison of the element to be located and the middle term

Searching Algorithms : binary search (cont.)

• Example 2: Search for 18 in the list

1 2 3 4 5 6 7 10 12 13 15 16 18 19 20 22

- » **Split-1:** 1 2 3 4 5 6 7 10 | 12 13 15 16 18 19 20 22
 - ♦ Compare 18 and the largest term in the first list
 - $10 < 18$, the search is then restricted to the 2nd list
- » **Split-2:** 12 13 15 16 | 18 19 20 22
 - ♦ $16 < 18$, the search is then restricted to the 2nd list
- » **Split-3:** 18 19 | 20 22
 - ♦ $19 > 18$, the search is then restricted to the 1st list
- » **Split-4:** 18 | 19
 - ♦ $18 \leq 18$, the search is then restricted to the 1st list with only one term
 - comparison is made

Searching Algorithms : binary search (cont.)

Algorithm

binarySearch(integer x , increasing integers a_1, a_2, \dots, a_n)

$i := 1$ (the left endpoint of search interval)

$j := n$ (the right endpoint of search interval)

while ($i < j$)

 { $m := \lfloor (i+j)/2 \rfloor$ (get the mid-point)

if $x > a_m$ **then** $i := m+1$

else $j := m$ }

if $x = a_i$ **then** $location := i$

else $location := 0$

Sorting Algorithms

- **Sorting** is an important problem in computing.
 - » It is to reorder the sequence so that the elements are in certain order (e.g. nondecreasing order).
- A large percentage of computing resources is developed to sorting one thing or another.
 - » A large number of sorting algorithms have been devised using distinct strategies
 - » Sorting algorithms to be discussed
 - ◆ Bubble sort
 - ◆ Selection sort

Sorting Algorithms: bubble sort

- The *bubble-sort* is one of the simplest sorting algorithms, but not one of the most efficient.
 - » In each pass, the elements are scanned by increasing rank, from rank 0 to the end of the sequence of *unordered* items.
 - » At each position in a pass, an element is compared with its right neighbor, and if these two consecutive elements are found to be in the wrong relative order, then the two elements are exchanged.
 - ♦ The smaller elements “bubble” to the top as they are exchanged with larger elements.
 - ♦ The larger elements “sink” to the bottom.
 - » The sequence with n items is sorted by completing $n-1$ such passes.

Sorting Algorithms: bubble sort (cont.)

Example 3:

pass	swaps	list
		(5, 7, 3, 6, 9, 2)
1 st	7↔3, 7↔6, 9↔2	(5, 3, 6, 7, 2, 9)
2 nd	5↔3, 7↔2	(3, 5, 6, 2, 7, 9)
3 rd	6↔2	(3, 5, 2, 6, 7, 9)
4 th	5↔2	(3, 2, 5, 6, 7, 9)
5 th	3↔2	(2, 3, 5, 6, 7, 9)

Sorting Algorithms: bubble sort (cont.)

- **Properties of bubble-sort algorithm**
 - » In the first pass, once the largest element is reached, it will keep on being swapped until it gets to the last position of the sequence
 - » In the i -th pass, once the i -th largest element is reached, it will keep on being swapped until it gets to the final position in the sequence

Thus, at the end of i -th pass, the rightmost i element of the sequence (i.e. those with ranks from n down to $n-i+1$) are in their final position. It allows the i -th pass to be limited to the first $n-i+1$ elements of the sequence.

Sorting Algorithms: bubble sort (cont.)

- A possible implementation of bubble-sort algorithm on an integer sequence (*array-based*)

Algorithm bubbleSort(integerSequence S):

$n := \text{size-of-}S$

for $i := 0$ to $n-2$ do

for $j := 0$ to $n-2-i$ do


if $S[j] > S[j+1]$ then

swap($S[j]$, $S[j+1]$)

Why “n-2” ?
(for discussion)

- (*) there are also other versions of implementation of bubble-sort algorithm

A Simple Analysis of Bubble-Sort Algorithm

- We consider the part of operation in  as one basic operation step to estimate the running time
 - » For the i -th pass, we need $n-i$ such basic operation steps
 - The overall running time (in the *worst* case) will be
$$(n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2$$
- (*) Bubble-sort is very simple to implement but not a good sorting method, because it takes quadratic time. There are more efficient algorithms for sorting.

Algorithm Analysis

- **Why algorithm analysis**
 - » Once an algorithm is given for a problem and determined to be correct, the next important task is to determine the amount of resources, such as time and space, that the algorithm will require.
 - » This step is called *algorithm analysis*. An algorithm is not useful, if it requires a huge amount of time that is beyond the capability of most current machines, even it is completely correct.

Algorithm Analysis (cont.)

- **How algorithm analysis**
 - » *Big-O notation* is a technique to categorize algorithms with respect to the length of running time and the amount of storage for their execution with data sets of different sizes.
 - » With the power of modern hardware, people usually pay more attention on the running time of an algorithm.
- **What we are going to discuss**
 - » Big-O, big-Omega, big-Theta

Running Time

- The amount of time that any algorithm takes to run almost always depends on the amount of input that it must process. The running time of an algorithm is *a function of the input size*.
- Four common functions in algorithm analysis
 - » *Linear*:
 - ♦ directly proportional to the amount of input N
 - » *Quadratic*:
 - ♦ the dominant term is some constant times N^2
 - » *Cubic*:
 - ♦ the dominant term is some constant times N^3
 - » *$N \log N$* :
 - ♦ the dominant term is n times the logarithm of n ($N \log N$)

Running Time (cont.)

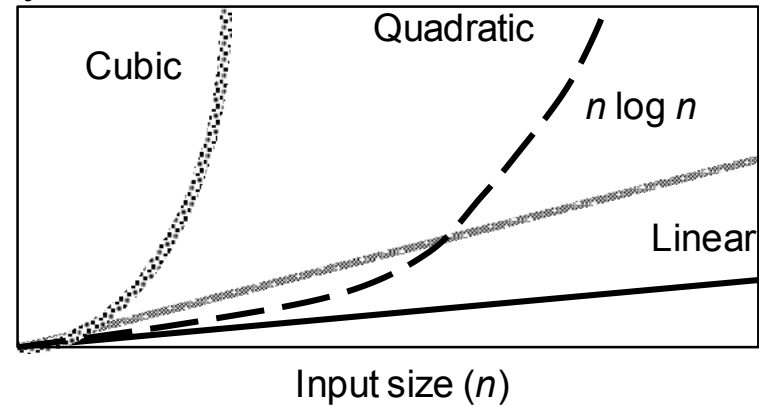
- **Linear**

» E.g.: Downloading a file from the Internet. The download proceeds at 1.6K/sec. Then if the file is N kilobytes, the time to download will be described by $T(N) = N/1.6$

- **Cubic**

» E.g.: $10n^3 + n^2 + 40n + 80$

Running time



Example: running time functions for a given program on a given computer

The Growth of Functions

- *Dominant term*

- » For a cubic function

- ♦ e.g. $10n^3 + n^2 + 40n + 80$

- when n is 1000, the value of the cubic function is 10,001,040,080, almost determined by the cubic term.

- ♦ If we were to use only the cubic term to estimate the entire function, only a very small error of about 0.01% would result.
- ♦ The exact value of the leading constant of the dominant term is not really meaningful for different machines.

The Growth of Functions (cont.)

- Small values of n generally are not important as the running time is not that critical.
 - » We pay more attention on sufficiently large values of n , and when n is getting larger, the leading constants become less significant.
- We use *Big-O notation* to capture the most dominant term in a function and represent the growth rate.

How & Why Big-O

- **Big-O is used to estimate the number of operations an algorithm uses as its input grows.**
 - » **Can estimate the growth of a function without worrying**
 - ♦ about constant multipliers or small order terms
 - ♦ about the hardware and software used to implement an algorithm
 - » **Can assume that the different operations used in an algorithm take the same time, and therefore simplify the analysis.**

How & Why Big-O (cont.)

- **Using Big-O**
 - » **Can determine whether it is practical to use a particular algorithm to solve a problem as the size of the input increases**
 - » **Can compare two algorithms to determine which is more efficient as the size of input grows**

Big-O Notation

- Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers.
 - » We say that $f(x)$ is $O(g(x))$ (“ $f(x)$ is big-oh of $g(x)$ ”)
 - ♦ If there are constants C and k such that
 - $|f(x)| \leq C|g(x)|$
 - Whenever $x > k$
 - » The constants C and k are called witnesses to the relationship $f(x)$ is $O(g(x))$
 - ! When there is one pair of witnesses, there are infinite many pairs of witnesses

Big-O Notation (cont.)

- **Example 4:**

$f(x) = x^2 + 2x + 1$ is $O(x^2)$ (i.e. $g(x) = x^2$)

» When $x > 1$, we have

♦ $0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$

• $C = 4$ and $k = 1$ are the witnesses

» Estimate the size of $f(x)$ when $x > 2$, we have

♦ $0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$

• $C = 3$ and $k = 2$ are also witnesses

Big-O Notation (cont.)

- When $f(x)$ is $O(g(x))$, we say
 - » $g(x)$ and $f(x)$ are of the **same order**
 - When big-O notation is used, the function g in the relationship $f(x)$ is $O(g(x))$ is chosen to be as small as possible.
- Important big-O result
 - » $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots a_1 x + a_0$ is $O(x^n)$
 where $a_n, a_{n-1}, \dots, a_1, a_0$ are real numbers
- Example 5:

$$f(x) = 7x^2 + 10x + 100 \quad \text{is } O(x^2)$$

Average & Worst Case Analysis

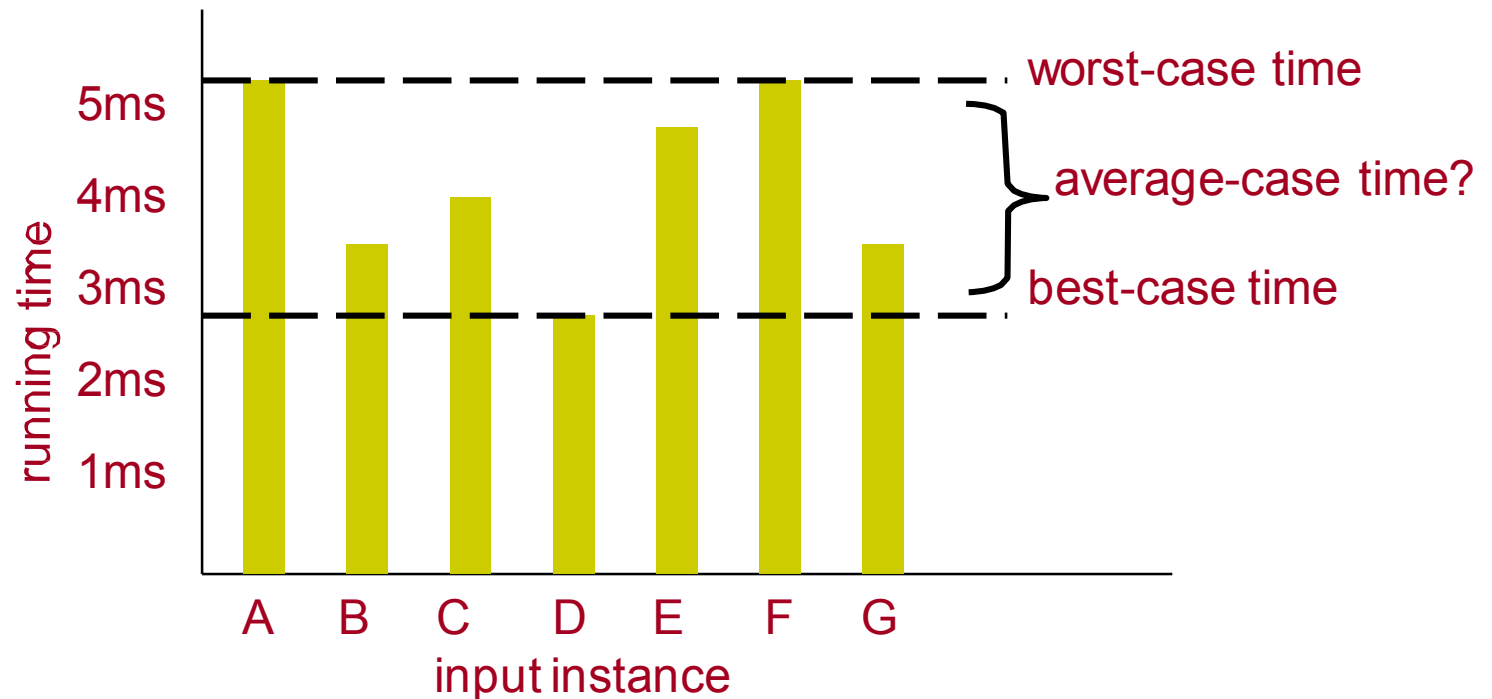
- An algorithm may run faster on some inputs than it does on others. In such cases we may wish to express the running time of such an algorithm as an average taken over all possible inputs.

! It requires us to define a probability distribution on the set of inputs, which is typically a difficult task.

✓ Depending on the input distribution the running time of an algorithm can be anywhere between the worst-case time and the best-case time. So the **worst-case** time gives the **maximum** running time needed.

Average & Worst Case Analysis (cont.)

Example 6:



- **Big-O notation describes the growth of functions with the estimation of *upper bound* (worst case)**

Big-Omega

- **Big-Omega notation** $\Omega(g(x))$
 - » Provides a **lower bound** for the size of $f(x)$ for large values of x
 - Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers.
 - » We say that $f(x)$ is $\Omega(g(x))$ (“ $f(x)$ is big-Omega of $g(x)$ ”)
 - ♦ If there are constants C and k such that
 - $|f(x)| \geq C|g(x)|$
- Whenever $x > k$**

Big-Theta Notation

- **Big-Theta notation** $\Theta(g(x))$
 - » Is used to give *both* an upper and lower bound on the size of a function $f(x)$, relative to a reference function $g(x)$
- Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers.
 - » We say that $f(x)$ is $\Theta(g(x))$ (“ $f(x)$ is big-Theta of $g(x)$ ”)
 - ♦ If $f(x)$ is $O(g(x))$ *and* $f(x)$ is $\Omega(g(x))$
 - » If $f(x)$ is $\Theta(g(x))$, we also say that $f(x)$ is of **order** $g(x)$

Big-Theta Notation (cont.)

- Some of the important conclusions

- » When $f(x)$ is $\Theta(g(x))$, it is also the case that $g(x)$ is $\Theta(f(x))$
- » $f(x)$ is $\Theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$

- We can show that $f(x)$ is $\Theta(g(x))$, if we can find *positive real numbers* C_1 and C_2 and positive real number k such that

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)| \quad \text{whenever } x > k$$

- » This means

$$\blacklozenge f(x) \text{ is } O(g(x)) \quad \text{and} \quad f(x) \text{ is } \Omega(g(x))$$

Selection Sort

- **Selection sort:**
 - » **Compare with Bubble Sort**
 - ◆ The strategy of bubble sort is to place the largest array element in the last position of the current array; The *Selection Sort* has a similar plan, but it attempts to avoid the multitude of interchanges of adjacent entries.
- **What Selection Sort does**
 - » On the k -th pass through the *array*, it determines the position of the *smallest* entry among $A[k-1], A[k], \dots, A[n-1]$
 - » Then this smallest entry is swapped with the k -th entry.
 - » k is incremented by 1, and the process is repeated

Selection Sort (cont.)

- Bubble sort

pass

swaps

array

1st

7↔3, 7↔6, 9↔2

(5, 7, 3, 6, 9, 2)

2nd

5↔3, 7↔2

(5, 3, 6, 7, 2, 9)

3rd

6↔2

(3, 5, 6, 2, 7, 9)

4th

5↔2

(3, 5, 2, 6, 7, 9)

5th

3↔2

(3, 2, 5, 6, 7, 9)

(2, 3, 5, 6, 7, 9)

- Selection sort

pass

swaps

array

1st

5↔2

(5, 7, 3, 6, 9, 2)

2nd

7↔3

(2, 7, 3, 6, 9, 5)

3rd

7↔5

(2, 3, 7, 6, 9, 5)

4th

9↔7

(2, 3, 5, 6, 9, 7)

5th

(2, 3, 5, 6, 9, 7)

(2, 3, 5, 6, 7, 9)

Which Sort is Better: a Big-O analysis

- Preparation

- » Using $O[g(n)]$ for two algorithms.

- ♦ When the two algorithms share the same $g(n)$ function, then we can conclude that the differences between them are not approaching an *order of magnitude scale*, so would not appear as dramatic run-time differences

- » The key of doing big-O analysis is to focus our attention on the *loops* in the algorithms, especially *inner loop(s)* if there are any.

Which Sort is Better: a Big-O analysis (cont.)

• Bubble Sort

Algorithm bubbleSort(array A):

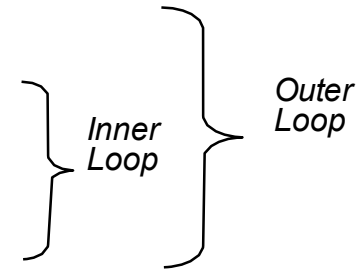
$n := \text{size-of-A}$

for $i := 0$ to $n-2$ do

for $j := 0$ to $n-2-i$ do

if $A[j] > A[j+1]$ then

swap($A[j]$, $A[j+1]$)



• Analysis

- » Number of comparisons at the top of inner loop for each pass: $(n - 1), (n - 2), \dots, 1$
- » Sum = $n(n - 1)/2$ is an $O(n^2)$ algorithm
- » The above implementation of bubble sort is a simple one and it can be improved by considering to exit the loop for the case that the array becomes ordered during an early pass.

Which Sort is Better: a Big-O analysis (cont.)

- Selection Sort

Algorithm selectionSort(array A):

$n := \text{size-of-}A$

for $i := 0$ **to** $n-2$ **do**

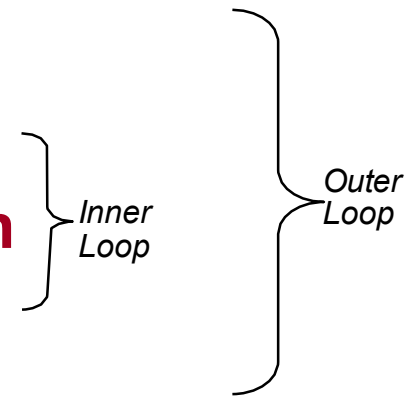
$\text{index} := i$

for $j := i+1$ **to** $n-1$ **do**

if $A[j] < A[\text{index}]$ **then**

$\text{index} := j$

 swap($A[i]$, $A[\text{index}]$)



Which Sort is Better: a Big-O analysis (cont.)

- **Analysis**

- » When using selection sort, the number of comparisons at the top of inner loop is the same as the bubble sort.
- » So it is still an ~~$O(n^2)$ algorithm~~ in terms of number of comparisons.
- ✓ The area in which the selection sort potentially offers better efficiency than bubble sort is that the number of interchanges of data in array location is guaranteed to be $O(n)$ because the swap occurs in the outer loop.
- ! Bubble sort requires $O(n^2)$ comparisons as well as $O(n^2)$ swaps.

Actual Run-Time Efficiency: **Down to the Machine Level**

- **Until now, we discussed the efficiency only at a high level of pseudocode, but did not really go to the machine level.**
 - » **The true run-time efficiency is best measured in fundamental machine operations.**
 - ! One instruction in a high-level language may actually translate into many primitive operations.**

Actual Run-Time Efficiency: **Down to the Machine Level** (cont.)

- Suppose the data being sorted are records, each of which requires 100 units (e.g. bytes) of internal storage. Depending on your computer, one swap in a high-level language could generate a machine language loop with 100 repetitions of such fundamental operations.
 - » If using temporary storage for the swap $A \leftrightarrow B$ (i.e. $\text{Temp} \leftarrow A, A \leftarrow B, B \leftarrow \text{Temp}$), then it actually leads to 300 movements.
 - ? Can we replace this large-scale internal transfer of entire records with the much swifter operation of swapping two integers? — YES

The Time/Space Trade-off:

Sorting Algorithms Implemented with Pointers

- *Physical sorting and logical sorting*
 - » Previous discussion of sorting algorithms assumed that the data are to be *physically sorted*.
 - ♦ The data in the first index of array (list) are the data come first in order according to the key field, and the data in the second index is the second in order, and so on.
 - » If we are only interested in processing the data of a list in order by the key field, it is NOT necessary to get the data physically ordered in computer memory.

The Time/Space Trade-off:

Sorting Algorithms Implemented with Pointers (cont.)

- **Possible strategy: implementation with pointers**
 - » A pointer is a memory location in which we store the location of a data item as opposed to the data item itself. In the case of an array, a pointer is the index position of a data item in the array.
 - » Pointers can keep track of the *logical order* of the data without requiring it to be physically moved.
 - » At the end of sorting, **pointer[0]** tells the location of the data that should come first in the expected order; **pointer[1]** contains the location of the data that should come second; and so on.
 - 👉 It needs extra space for pointers
- (*) This strategy applies to any sorting algorithm

Complexity of Algorithms

- **Linear search (worst-case analysis)**
 - » **The number of comparisons used by the algorithm will be taken as the measure of the time complexity.**
 - ♦ **At each step of the loop in the algorithm, two comparisons are performed**
 - **One to see whether the end of list has been reached and one to compare the element x with a term of the list**
 - ♦ **Finally, one more comparison is made outside the loop**
 - ♦ **If $x = a_i$, then $2i + 1$ comparisons are used**
 - » **Worst-case analysis**
 - ♦ **It requires *at most* $2n + 2$ comparisons ($O(n)$)**

Complexity of Algorithms (cont.)

- **Average-case analysis (linear search)**
 - » The average number of operations used to solve the problem over all inputs of a given size
 - » Usually, is much more complicated than worst-case analysis
 - » But is not difficult for linear search algorithm
 - ♦ The average number of comparisons used (when x is in the list) is
 - $n + 2$ which is $\Theta(n)$

Complexity of Algorithms (cont.)

- **Binary search (worst-case)**

- » **Assume**

- ♦ there are $n = 2^k$ elements in the list a_1, a_2, \dots, a_n
 - k is a nonnegative integer ($k = \log_2 n$)
 - If n is not a power of 2, then $2^k < n < 2^{k+1}$

- » **At each stage,**

- ♦ i and j indicate the left point and right point of the restricted list
 - **Comparison-1:** check if $i < j$ to see whether the restricted list (search range) has more than one term
 - **Comparison-2:** to determine whether x is greater than the middle term of the restricted list

Complexity of Algorithms (cont.)

(cont.)

- » At 1st stage, after 2 comparisons, the search range is reduced to a list with 2^{k-1} elements
- » At 2nd stage, after 2 comparisons, the search range is further reduced to a list with 2^{k-2} elements
- »
- » At (k-1)-th stage, after 2 comparisons, the search range is reduced to $2^1 = 2$ elements
- » At k-th stage, after 2 comparisons, the search range is reduced to $2^0 = 1$ elements
 - ♦ One comparison to confirm that $i = j$
 - ♦ Another comparison to check if the item is x
- » It requires $2\lceil \log_2 n \rceil + 2$ comparisons, is $O(\log n)$

Understanding Complexity of Algorithms

- **Complexity (from lower to higher)**

- » **Constant** $\Theta(1)$
- » **Logarithmic** $\Theta(\log n)$
- » **Linear** $\Theta(n)$
- » **$n \log n$** $\Theta(n \log n)$
- » **Polynomial** $\Theta(n^b)$
- » **Exponential** $\Theta(b^n)$
- » **Factorial** $\Theta(n!)$

Understanding Complexity of Algorithms (cont.)

- *Tractable*

- » A problem is solvable using an algorithm with polynomial worst-case complexity

- *Intractable*

- » A problem cannot be solved using an algorithms with worst-case polynomial time complexity

- *Unsolvable*

- » No algorithm exists for solving the problem

Understanding Complexity of Algorithms (cont.)

- **Class NP**

- » No algorithm with polynomial worst-case time complexity solves them.
- » But a solution, if known, can be checked in polynomial time

- **NP-complete problems (NP: *nondeterministic polynomial time*)**

- » If any of these problems can be solved by a polynomial worst-case time algorithm, then all problems in the NP class can be solved by polynomial worst-case time algorithms

Master of Technology in Knowledge Engineering

Annex 2: Set Theory

GU Zhan (Sam)

© 2016 NUS. The contents contained in this document may not be reproduced in any form or by any means,
without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Set and Element

- A set is an unordered collection of objects.
 - » $\{a, b, c\} = \{b, c, a\}$
- The objects in a set are also called the **elements**, or **members**, of the set. A set is said to **contain** its elements.
- E.g: **ISS-Lecturer = {..., Liya, ...}**
 - » Liya is a member of ISS-Lecturer
 - » ISS-Lecturer contains Liya as an element

Notation of Sets

- **List notation (brace notation)**

- » **IT-courses = {SE, OS, DB, ...}**

- ♦ ellipse ... are used when a genral pattern of the elements is obvious

- **Set builder notation**

- » **Characterize all the elements in the set by stating the properties they must have in order to be members.**

- ♦ **$A = \{x \mid x > 20\}$**

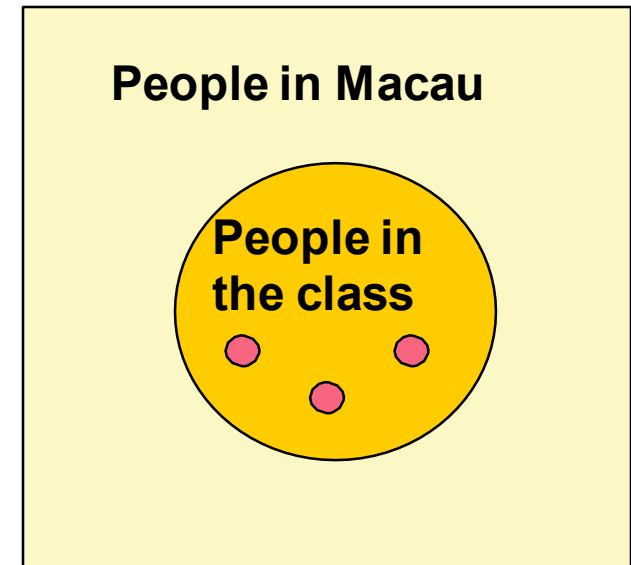
- » **It is often used to describe sets when it is impossible to list all the elements (e.g. infinite number of elements).**

Notation of Sets (cont.)

- **Graphical notation**

- » **Venn diagrams**

- ♦ **Universal set** contains all the objects under consideration: rectangle
 - **People in Macau**
 - ♦ **Sets to be represented: circles or other geometrical figures**
 - **People in this class**
 - ♦ **Particular elements of the set: points**
 - **Girls in the class**

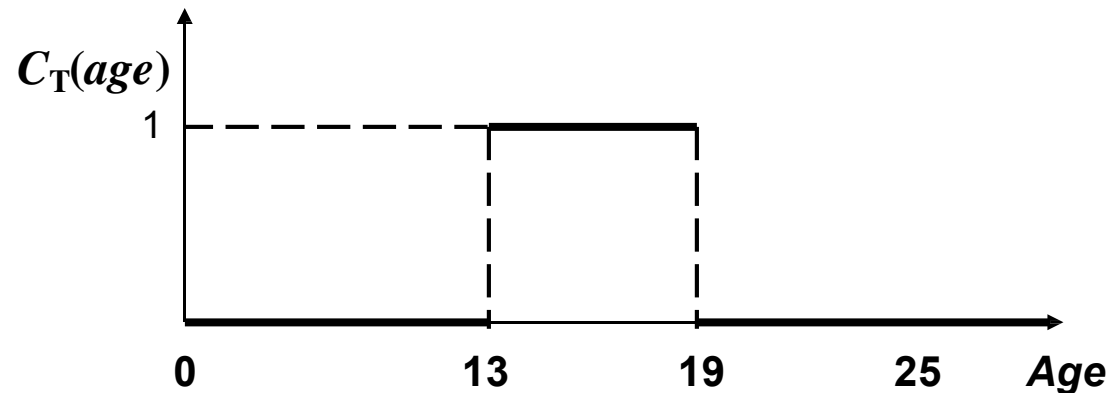


Notation of Sets (cont.)

- **Characteristic function** and graphic representation

» E.g.: Teenage is a set defined on the universal set Age

$$C_T(\text{age}) = \begin{cases} 0, & \text{age} < 13 \\ 1, & 13 \leq \text{age} \leq 19 \\ 0, & \text{age} > 19 \end{cases}$$

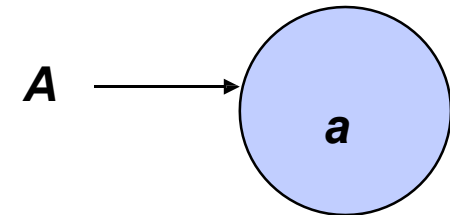


Description of Membership

- Set A and an object a

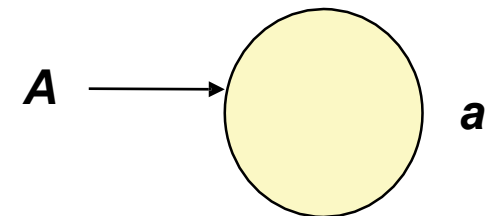
» $a \in A$ means

- ♦ a is an element (a member) of the set A , or
- ♦ a “belongs” to the set A , or
- ♦ a is “in” the set A



» while $a \notin A$ means

- ♦ a is *not* a member of the set A , or
- ♦ a “*not* belong” to the set A , or
- ♦ a is “out” the set A



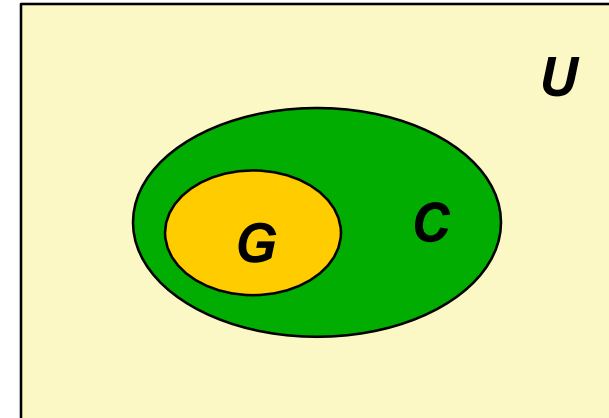
Description of Membership (cont.)

- There are **only** two possible relationships between the element a and a *crisp* set A :
 - » **either** $a \in A$
 - » **or** $a \notin A$
- ☯ Is there any case where the situation cannot be simply described in such a way ?

Universal Set and Subset

- Taking the previous example:

- » U — all people in Macau
- » C — all people in the class
- » G — all girls in the class



- ♦ U is the *universal set*, C and G are sets defined on U .
 - For any a in C , $a \in U$
 - For any g in G , $g \in C, g \in U$
- ♦ G is a *subset* of C : $G \subseteq C$

Relations between Sets

- **Subset**

$A \subseteq B$ A is a subset of B

(Every element of set A is also an element of B)

$A \subset B$ A is a *proper subset* of B

(A is a subset of B , and B contains at least one element that is not a member of A)

$A \not\subseteq B$ A is not a subset of B

(A contains at least one element that is not a member of B)

$A \subseteq A$ a set is always a subset of itself

Relations between Sets (cont.)

- **Equal**

$$A = B$$

A and B are equal sets

($A \subseteq B$ and $B \subseteq A$, A and B have the same elements)

$$A \neq B$$

A and B are not equal

Cardinality of Set

- Let S be a set. The *cardinality* of S is the number of distinct elements in S
 - » $n = |S|$ is a nonnegative integer
 - » The concept of cardinality can only be applied to *finite* sets. A set is said to be *infinite* if it is not finite.
Cardinality cannot be defined for an infinite set.
 - » For finite set A
 - ♦ $|A| \in \mathbb{N}$

Cardinality of Set (cont.)

E.g.:

$$\blacklozenge A = \{a, b, c\} \quad |A| = 3$$

$$\blacklozenge B = \{ \{a, b\}, \{c\} \} \quad |B| = 2$$

$$\blacklozenge C = \{ \{a, b, \{c, d\} \} \} \quad |C| = 1$$

 Cardinality of set is not a set, but a number

Null Set

- **Null set (empty set)**
 - » a special set that has no elements
 - » is denoted by \emptyset or $\{\}$
 - » Null set is a subset of any set (including \emptyset)
 - » $\{\emptyset\} \neq \emptyset$
 - » $|\{\}| = 0$ $|\{\emptyset\}| = 1$

$$\{a\} \neq \{\{a\}\}$$

- ♦ $\{a\}$ is a set containing an element a
- ♦ $\{\{a\}\}$ is a set containing set $\{a\}$ as an element

Power Set

- **Power set**

» Given a set S , the **power set** of S , $P(S)$, is the set of all subsets of the S

» E.g.: $S = \{0, 1, 2\}$

$$P(S) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$$

S contains null set and itself as subsets

- **E.g.**

» $P(\emptyset) = \{\emptyset\}$

» $P(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\}$

👉 for any set S , its power set
 $P(S) \neq \emptyset$

Cartesian Products

- Cartesian product

» *Cartesian product* of set A and set B , denoted by $A \times B$, is the set of all *ordered pairs* (a, b) where $a \in A$ and $b \in B$.

$$A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- Example:

» $A = \{1, 2\}$ and $B = \{h, j, k\}$

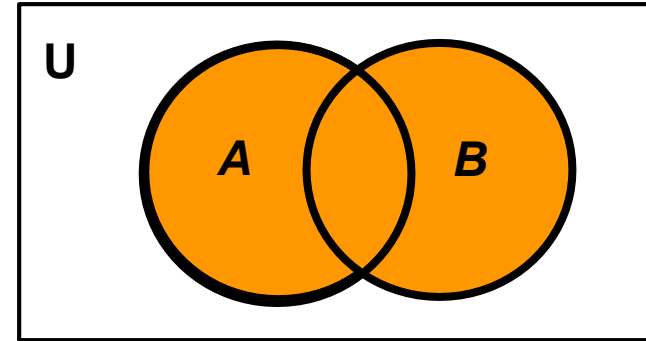
$$A \times B = \{ (1, h), (1, j), (1, k), (2, h), (2, j), (2, k) \}$$

$$B \times A = ? \quad \text{(for your exercise)}$$

Operations on Sets

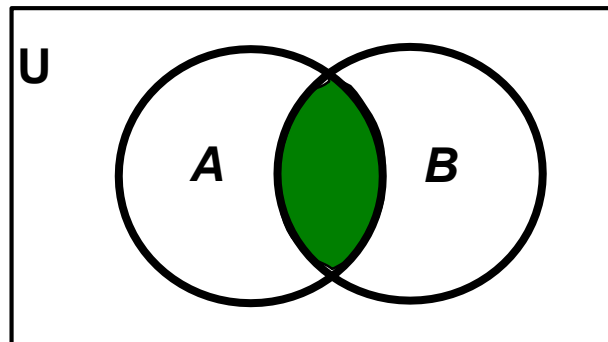
- **Union of A and B**

» $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$



- **Intersection of A and B**

» $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

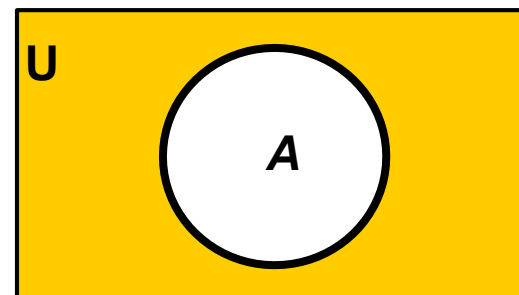


Operations on Sets (cont.)

- **Complement of A**

$$\overline{A} = \{x \mid x \in U, x \notin A\}$$

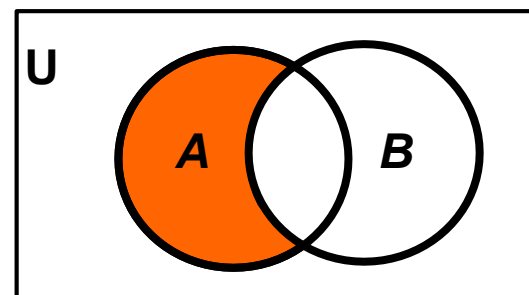
» The complement of a set cannot be discussed without the universal set specified. *(how about other set operations?)*



- **Difference of A and B**

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

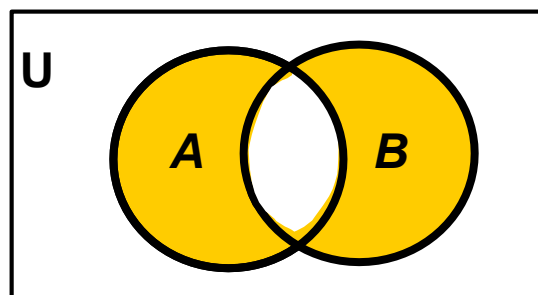
» It is also called the complement of B with respect to A



Operations on Sets (cont.)

- **Symmetric difference of A and B**

$$A \oplus B = \{x \mid (x \in A \text{ and } x \notin B \text{ or } x \notin A \text{ and } x \in B)\}$$



Note:

$$A \oplus B = (A \cup B) - (A \cap B)$$

Basic Operations on Sets (cont.)

- **Example:**

» Let $A = \{1, 2, 3, 4\}$, $B = \{1, 3, 5\}$, $C = \{3, 6, 9\}$

$$A \cup B \cup C = \{1, 2, 3, 4, 5, 6, 9\}$$

$$A \cap B \cap C = \{3\}$$

$$|A| = 4,$$


$$|B| = 3$$

$$A - B = \{2, 4\},$$

$$A - C = \{1, 2, 4\},$$

$$C - A = \{6, 9\}$$

$$A \oplus C = C \oplus A = \{1, 2, 4, 6, 9\}$$

 $A - C \neq C - A$

Master of Technology in Knowledge Engineering

Annex 3: Relations & Functions

GU Zhan (Sam)

© 2016 NUS. The contents contained in this document may not be reproduced in any form or by any means,
without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Why Relations

- **Relations** are of fundamental importance to both the theory and application areas of computer science.
 - » A composite data structure, such as an array, a list, or a tree, is generally used to represent a set of data objects together with a **relation** which holds between members of the set.
 - » Relations which are a part of mathematical model are often **implicitly** represented by relations within a data structure.

What is a Relation

- The order of a relation

- » *is_taller*(x, y) is said to be a *binary relation*, or relation of *order two*, since it requires two objects to make a complete statement.

- » A relation such as *has_two_legs*(x) is a *unary*, or *order one*, relation.

(*) Our discussion will mainly focus on binary relations. Most concepts applicable to binary relations can be generalized to relations of higher order.

Relations and Sets

- Let A and B be sets. A **binary relation** R from A to B is
 - » a subset of $A \times B$ $(R \subseteq A \times B)$
 - » a set of ordered pairs
 - ♦ the first element of each ordered pair comes from A and the second element comes from B
- Use the notation $a R b$ to denote $(a, b) \in R$
- Use the notation $a \cancel{R} b$ to denote $(a, b) \notin R$
- When $(a, b) \in R$, a is said to be **related** to b by R

Modeling Relations

- The concept of relation finds use in areas
 - » *Relational databases*
 - » *Logic programming* (e.g. the “Prolog” language)
- Two common ways of modeling a relation
 - » As a predicate, or
 - » As a set of *ordered tuples* (pairs, if binary relation)

Operations on Relations

- Since a relation is a set of ordered tuples, we can apply all the normal set operations to relations.
 - » Sometimes, the result will also be a relation, but other times it is not.
- Example:
 - » $A = \{a, b, c, d\}, \quad B = \{J, K, Q, R, S\}$
 - » $R = \{(a, J), (a, K), (b, K), (d, S)\}, \quad S = \{(c, Q), (d, S)\}$
 - ♦ $R \cup S = \{(a, J), (a, K), (b, K), (d, S), (c, Q)\}$ ✓
 - ♦ $R \cap S = \{(d, S)\}$ ✓
 - ♦ $R - S = \{(a, J), (a, K), (b, K)\}$ ✓

Operations on Relations (cont.)

Example (cont.)

» **The power set of S**

♦ $P(S) = \{ \{(c, Q), (d, S)\}, \{(c, Q)\}, \{(d, S)\}, \{\} \}$

- is not a relation

👉 because the individual elements of the set are *sets* rather than ordered tuples

Composite Relations

- A single relation can be built up from two or more relations.

- Example: *is_aunt_of*(Lisa, Ben) follows

♦ *is_sister_of*(Lisa, someone), and

♦ *is_parent_of*(someone, Ben)

- Let R_1 be a relation from A to B and R_2 be a relation from B to C .

» The *composite relation* of R_1 and R_2 , from A to C , denoted $R_2 \circ R_1$ is defined as follows:

$$R_2 \circ R_1 = \{ (a, c) \mid$$

$$a \in A \wedge c \in C \wedge \exists b [b \in B \wedge (a, b) \in R_1 \wedge (b, c) \in R_2] \}$$

Composite Relations (cont.)

- **Example: Let**

R_1 : *is_sister_of* = {(Lisa, Mike), (Judy, Kent)}

R_2 : *is_parent_of* = {(Mike, Ben), (Kent, Ray)}

» Thus $R_2 \circ R_1$: *is_aunt_of* = {(Lisa, Ben), (Judy, Ray)}

(*) If R_1 is a relation from A to B and R_2 is a relation from C to D , then $R_2 \circ R_1$ is *not defined* unless $B = C$.

Null Relation

- There are some special types of relations.
 - » The *null relation* is equal to the empty set. $R = \Phi$
- Example: given the following relations:
 - $R_1: \text{is_brother_of} = \{(\text{Dan}, \text{Lisa}), (\text{Ben}, \text{Tina})\} \subseteq P \times P$
 - $R_2: \text{is_parent_of} = \{(\text{Mike}, \text{Ben}), (\text{Mike}, \text{Tina})\} \subseteq P \times P$
 - » where $P = \{\text{Dan}, \text{Lisa}, \text{Ben}, \text{Tina}, \text{Mike}\}$
 - ♦ The composite relation
 - $\text{is_uncle_of} = R_2 \circ R_1 = \{\}$ is a null relation
 - ! *It is defined, but is empty*

Universal Relation

- A relation R defined on $A \times B$ is called the *universal relation* when it is equal to $A \times B$.
- Example: There are 2 classes in a school
Class-A = {Ben, Dan, Esther} Class-B = {Mike, Tina}
» We can have the universal relation R for “a student in class-A knows someone in class-B” (suppose that all people know each other)
 - ♦ $R = A \times B = \{(\text{Ben, Mike}), (\text{Ben, Tina}), (\text{Dan, Mike}), (\text{Dan, Tina}), (\text{Esther, Mike}), (\text{Esther, Tina})\}$

Identity Relation

- The *identity relation* on a set A is the set of ordered pairs of each element of A coupled with itself
 - » $I_A = \{(x, x) \mid x \in A\}$
- **Example:**
 - » We have $\text{Class-A} = \{\text{Ben, Dan, Esther}\}$
 - ♦ We can have the identity relation R for “student x does homework for student x ”

Representing Relations

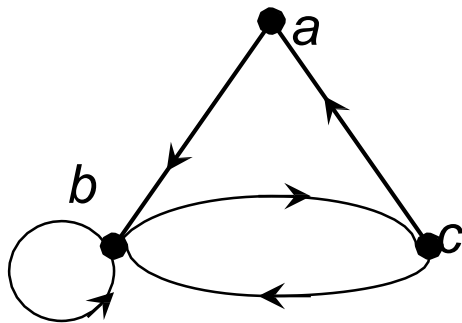
- There are several ways to represent a binary relation between finite sets

» *List of ordered pairs* (discussed previously)

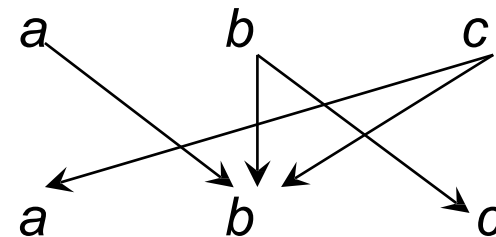
- Example:

$$R = \{(a, b), (b, c), (c, b), (c, a), (b, b)\}$$

» *Directed graph*



Arrow diagram



Representing Relations: Matrices

- Suppose that R is a relation

- » from $A = \{a_1, a_2, \dots, a_m\}$,

- » to $B = \{b_1, b_2, \dots, b_n\}$.

The relation R can be represented by the 0-1 matrix

- » $M_R = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 & \text{if } (a_i, b_j) \in R \\ 0 & \text{if } (a_i, b_j) \notin R \end{cases}$$

Representing Relations: Matrices (cont.)

• Example:

» Let $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2\}$,
and the relation R defined on $A \times B$ be

$$R = \{(a_1, b_2), (a_2, b_2), (a_3, b_1)\}.$$

» The relation can be represented by the
0-1 matrix with its rows for elements of
 A and columns for elements of B .

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

↳ A particular matrix represents a relation R from A to B according to a **particular** order of listing elements of A and B , though the order can be determined arbitrarily.

Operations on Relations: Matrices (cont.)

• Example:

» $S_1 = \{1, 2, 3\}$, $S_2 = \{a, b, c\}$, $S_3 = \{x, y, z\}$

» R_1 is a relation
from S_1 to S_2

$$\mathbf{M}_{R_1} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{M}_{R_2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

» R_2 is a relation
from S_2 to S_3

» $R_2 \circ R_1$ is a composite relation
from S_1 to S_3

$$\mathbf{M}_{R_2 \circ R_1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Operations on Relations: Matrices (cont.)

- Example:**

» Given relation R , where the matrix representing R is

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

» The matrix for R^2 is

$$\mathbf{M}_{R^2} = \mathbf{M}_R^{[2]} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

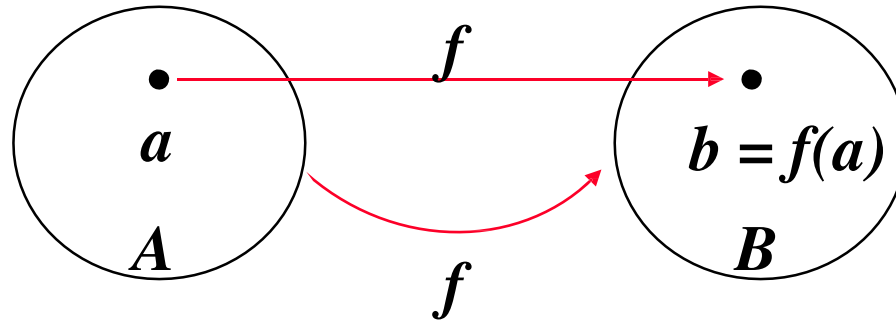
What is a Function

- Let A and B be sets. A **function** (or **map**, or **transformation**) f from A to B , denoted $f: A \rightarrow B$, is a relation from A to B such that for **every** $a \in A$, there exists a **unique** $b \in B$ such that $(a, b) \in f$. If $(a, b) \in f$, then we write $f(a) = b$.
- A function f from A to B is a binary relation from A to B with the special properties:
 - » **Every** element of A occurs as the first component of an ordered pair of f
 - » If $f(a) = b$ and $f(a) = c$, then $b = c$ (the value of $f(a)$ must be unique)

Basic Concepts of Function

- If f is a function from A to B , we say that A is the *domain* of f and B is the *codomain* of f .
- If $f(a) = b$, we say that b is the image of a and a is a pre-image of b .
- The *range* of f is the set of all images of elements of A .
- If f is a function from A to B , we say that f *maps* A to B .

Basic Concepts of Function (cont.)



- **Example:**

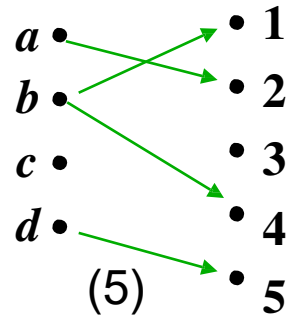
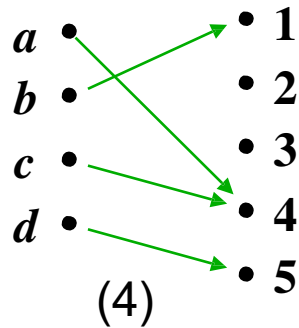
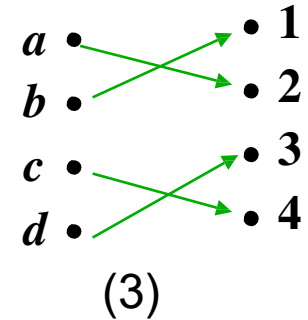
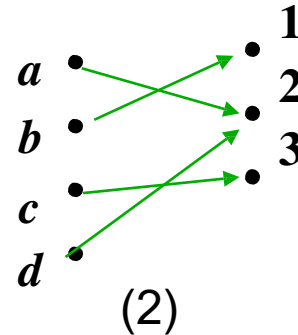
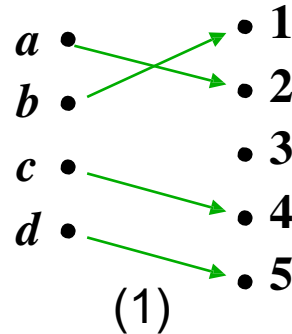
- » $g: \{1, 2, 3\} \rightarrow \{a, b, c\},$
- » $g(1) = a, g(2) = c, g(3) = c$
 - ♦ $\{1, 2, 3\}$ is the domain
 - ♦ $\{a, b, c\}$ is the codomain
 - ♦ the range of g is $\{a, c\}$

One-to-One & Onto

- A function f is said to be *one-to-one*, or *injective*, if and only if $f(x) = f(y)$ implies that $x = y$ for all x and y in the domain of f .
- A function f from A to B is called *onto*, or *surjective*, if and only if for every element $b \in B$ there is an element $a \in A$ with $f(a) = b$.
- A function f is an *one-to-one correspondence*, or a *bijective*, if it is both one-to-one and onto.

One-to-One & Onto (cont.)

- Example:**



(1) One-to-one, not Onto

(2) Onto, not One-to-one

(3) One-to-one correspondence

(4) neither One-to-one, nor Onto

(5) not a function

- Let A be a set. The **identity function** on A is the function $\iota_A: A \rightarrow A$, where $\iota_A(x) = x$

» The identity function assigns each element to itself. It is one-to-one and onto.

Inverse Function

- Let f be an one-to-one correspondence from the set A to the set B .
 - » The *inverse function* of f , denoted f^{-1} , is the function that assigns to an element $b \in B$ the unique element $a \in A$ such that $f(a) = b$.
 - ♦ i.e.: $f(a) = b$, and $f^{-1}(b) = a$
 - » An one-to-one correspondence is called *invertible*.
 - ♦ If a function is not an one-to-one correspondence, it is *not invertible* since we cannot define the inverse of such a function.

Inverse Function (cont.)

- **If f is not one-to-one**
 - » some element b in the codomain is the image of more than one element in the domain.
 - ♦ i.e.: the inverse is not a function
- **If f is not onto**
 - » for some element b in the codomain, no element a in the domain exists for which $f(a) = b$.
 - ♦ i.e.: no corresponding value for b as the inverse, so it is not a function

Inverse Function (cont.)

- **Example:**

» Let f be the function from $\{a, b, c\}$ to $\{1, 2, 3\}$ such that $f(a) = 1$, $f(b) = 2$, and $f(c) = 3$.

♦ The f is invertible:

- $f^{-1}(1) = a$,
- $f^{-1}(2) = b$, and
- $f^{-1}(3) = c$.

Master of Technology in Knowledge Engineering

Unit 1

Intelligent Systems & Techniques for Business Analytics

Exercise & Discussion — Representation & Reasoning

GU Zhan (Sam)

© 2016 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Rules of Inference in Propositional Logic

- Example 4.4 (for exercise)
 - » Hypotheses
 - ◆ If you send me an e-mail, then I will finish writing the program ($e \rightarrow p$)
 - ◆ If you do not send me an e-mail, then I will go to sleep early ($\neg e \rightarrow s$)
 - ◆ If I go to sleep early, then I will wake up feeling refreshed ($s \rightarrow r$)
 - » Propositions used
 - ◆ you send me an e-mail, e
 - ◆ I finish writing the program p
 - ◆ I go to sleep early, s
 - ◆ I wake up feeling refreshed, r

Rules of Inference in Propositional Logic (cont.)

Example 4.4 (for exercise) (cont.)

» Conclusion

- ♦ If I do not finish writing the program, then I will wake up feeling refreshed ($\neg p \rightarrow r$)

» How to establish the argument?

Exercise: First Order (Predicate) Logic

- Express the following descriptions using FOL sentences
 - » Any person who is smart buys insurance
 - » Some people invested in stock lost money
 - » Everyone has only one best friend except him/herself

Exercise: who killed Tuna?

- Given the background information below:
 - » Jack loves all animals.
 - » Anyone who loves all animals does not kill an animal.
 - » Either Jack or Curiosity killed the cat, which is named Tuna.
 - » Did Curiosity kill Tuna?
 - ◆ Use the following predicates:
 - $\text{Animal}(x)$ — x is an animal
 - $\text{Cat}(x)$ — x is a cat
 - $\text{Loves}(x, y)$ — x loves y
 - $\text{Kills}(x, y)$ — x kills y

Exercise: who killed Tuna? (cont.)

- Use proof by refutation and *linear input resolution* to show that “*Curiosity killed Tuna*”.
 - » Express the sentences for related background knowledge, and the goal in first-order logic.
 - » Convert the first-order logic formulas to clausal form.
 - » You are required to show your resolution steps with necessary unification and variable renaming.

Master of Technology in Knowledge Engineering

Unit 1

Intelligent Systems & Techniques for Business Analytics

Exercise & Discussion — Representation & Reasoning

GU Zhan (Sam)

© 2016 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Rules of Inference in Propositional Logic

- Example 4.4 (for exercise)
 - » Hypotheses
 - ◆ If you send me an e-mail, then I will finish writing the program ($e \rightarrow p$)
 - ◆ If you do not send me an e-mail, then I will go to sleep early ($\neg e \rightarrow s$)
 - ◆ If I go to sleep early, then I will wake up feeling refreshed ($s \rightarrow r$)
 - » Propositions used
 - ◆ you send me an e-mail, e
 - ◆ I finish writing the program p
 - ◆ I go to sleep early, s
 - ◆ I wake up feeling refreshed, r

Rules of Inference in Propositional Logic (cont.)

Example 4.4 (for exercise) (cont.)

» Conclusion

- ♦ If I do not finish writing the program, then I will wake up feeling refreshed ($\neg p \rightarrow r$)

» How to establish the argument?

Rules of Inference in Propositional Logic (cont.)

- Argument

1) Step

Reasoning using rule of inference

$$e \rightarrow p$$

original hypothesis

2) $\neg p \rightarrow \neg e$

contrapositive of the step 1

3) $\neg e \rightarrow s$

original hypothesis

4) $\neg p \rightarrow s$

hypothetical syllogism using steps 2 and 3

5) $s \rightarrow r$

original hypothesis

6) $\neg p \rightarrow r$

hypothetical syllogism using steps 4 and 5

Exercise: First Order (Predicate) Logic

- Express the following descriptions using FOL sentences
 - » Any person who is smart buys insurance
 - » Some people invested in stock lost money
 - » Everyone has only one best friend except him/herself

Exercise FOL: sample answer

- Any person who is smart buys insurance

$$\forall x [\text{smart}(x) \Rightarrow \text{buy_insurance}(x)]$$

- Some people invested in stock lost money

$$\exists x \text{ stock}(x) \wedge \text{lost_money}(x)$$

Exercise FOL: sample answer (cont.)

- “Everyone has exactly one best friend”
 - » “ y is the best friend of x ” $B(x, y)$

Exercise FOL: sample answer (cont.)

- “Everyone has exactly one best friend”
 - » “y is the best friend of x” $B(x, y)$
 - » for *every* person x, $\forall x$
 - ♦ *there is* some other person y such that $\exists y \quad y \neq x$
 - y is the best friend of x, *and* $B(x, y)$

Exercise FOL: sample answer (cont.)

- “Everyone has exactly one best friend”
 - » “y is the best friend of x” $B(x, y)$
 - » for *every* person x, $\forall x$
 - ♦ *there is* some other person y such that $\exists y \quad y \neq x$
 - y is the best friend of x, *and* $B(x, y)$
 - ♦ for *all* z, if z is a person *other than* y, $\forall z \quad z \neq y$
 - then z is *not* the best friend of x $\neg B(x, z)$

Exercise FOL: sample answer (cont.)

- “Everyone has exactly one best friend”

- » “y is the best friend of x”

 $B(x, y)$

- » for *every* person x,

 $\forall x$

- ♦ *there is* some other person y such that

 $\exists y \quad y \neq x$

- y is the best friend of x, *and*

 $B(x, y)$

- ♦ for *all* z, if z is a person *other than* y,

 $\forall z \quad z \neq y$

- then z is *not* the best friend of x

 $\neg B(x, z)$

$$\forall x \exists y \forall z ((y \neq x) \wedge B(x, y) \wedge ((z \neq y) \Rightarrow \neg B(x, z))) \text{ or}$$

$$\forall x \exists y \forall z ((y = \neg x) \wedge B(x, y) \wedge ((z = \neg y) \Rightarrow \neg B(x, z)))$$

- ♦ Alternative expression: $\forall x \exists!$ y ((y ≠ x) ∧ B(x, y))

uniqueness

Exercise FOL: sample answer (cont.)

- “Everyone has exactly one best friend”
 - » “y is the best friend of x” $B(x, y)$
 - » for *every* person x, $\forall x$
 - ♦ *there is some other person* y such that $\exists y \quad y \neq x$
 - y is the best friend of x, *and* $B(x, y)$
 - ♦ *Everyone (z) is not* y, *and* $\forall z \quad z \neq y$
 - that person z is *not* the best friend of x $\neg B(x, z)$
- $$\forall x \exists y \forall z ((y \neq x) \wedge B(x, y) \wedge ((z \neq y) \wedge \neg B(x, z)))$$

Exercise: who killed Tuna?

- Given the background information below:
 - » Jack loves all animals.
 - » Anyone who loves all animals does not kill an animal.
 - » Either Jack or Curiosity killed the cat, which is named Tuna.
 - » Did Curiosity kill Tuna?
 - ◆ Use the following predicates:
 - $\text{Animal}(x)$ — x is an animal
 - $\text{Cat}(x)$ — x is a cat
 - $\text{Loves}(x, y)$ — x loves y
 - $\text{Kills}(x, y)$ — x kills y

Exercise: who killed Tuna? (cont.)

- Use proof by refutation and *linear input resolution* to show that “*Curiosity killed Tuna*”.
 - » Express the sentences for related background knowledge, and the goal in first-order logic.
 - » Convert the first-order logic formulas to clausal form.
 - » You are required to show your resolution steps with necessary unification and variable renaming.

Exercise: who killed Tuna? (solution)

Knowledge Base:

$\text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y)$

$\text{Loves}(x, y) \wedge \text{Animal}(y) \rightarrow \neg \text{Kills}(x, y)$

$\text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

CNF conversion:

$\text{Loves}(x, y) \wedge \text{Animal}(y) \rightarrow \neg \text{Kills}(x, y) \equiv \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y)$

$\alpha = \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\neg \alpha = \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\text{KB} \wedge \neg \alpha \equiv \text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y) \vee \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee$

$\text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna})) \vee \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\equiv \text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y) \vee \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna}))$

$x = \text{Jack}$

$y = \text{Cat}(\text{Tuna})$

$\equiv \text{Loves}(\text{Jack}, \text{Cat}(\text{Tuna})) \wedge \text{Animal}(\text{Cat}(\text{Tuna})) \vee \neg [\text{Loves}(\text{Jack}, \text{Cat}(\text{Tuna})) \wedge \text{Animal}(\text{Cat}(\text{Tuna}))] \vee \neg \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna}))$

$\equiv \{\}$

Thus we reject: $\neg \alpha = \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

But accept: $\alpha = \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

Exercise: financial advisor

- Background

- » A simple financial advisor is used to help users decide whether to invest in stocks or savings account.
- » Some investors may want to split their money between the two.
- » An investment strategy will be recommended depending on their income and the current amount they have saved according to certain criteria

Exercise: financial advisor (cont.)

- Criteria

- » Individuals with inadequate savings should always make increasing the amount saved their 1st priority, regardless of their income.
- » Individuals with adequate savings and adequate income should consider a riskier but potentially more profitable investment in the stock market.
- » Individuals with a lower income who already have adequate savings may want to consider splitting their surplus income between savings and stocks, to increase the cushion in savings while attempting to increase their income through stocks.

Exercise: financial advisor (cont.)

- Adequacy setting
 - » The adequacy of both savings and income is determined by the number of dependents an individual must support.
 - ◆ Our rule is to have at least \$5,000 in the bank for each dependent.
 - ◆ An adequate income must be a steady income and supply at least \$15,000 per year plus an additional \$4,000 for each dependent.

Exercise: financial advisor (cont.)

Your task

1. Represent the above knowledge using predicate calculus statements.
2. Perform a consultation (using *unification* and *Modus Ponens*) with the “system” using the following profile
 - » – an individual with 3 dependents, \$22,000 in savings, and a steady income of \$25,000.
 - » Find out what is the investment strategy recommended by the system.
3. To start you off, the following predicates can be used:

<i>savings_account(adequate)</i>	<i>investment(stocks)</i>
<i>savings_account(inadequate)</i>	<i>investment(savings)</i>
<i>income(adequate)</i>	<i>investment(combination)</i>
<i>income(inadequate)</i>	

Exercise: financial advisor (cont.)

Hint:

- Step 1: Create predicate logic statements for the knowledge given.
» You need NOT convert to Clausal Normal Form for this exercise!
- Step 2: Create predicate logic statements (you should have 3 chunks- pls note that the 3 chunks are conjunction knowledge) for the “information” given and add it to the above knowledge base.
- Step 3: Using the newly added knowledge and scanning (from top to bottom) the KB, try to find patterns where *unification* can take place.
- Step 4: once you find unification, make the substitution to yield a new knowledge that will be added to the KB.
- Step 5: Using the newly added knowledge, scan the KB to find if Modus Ponens can be applied. If not, repeat steps 3-5
- STOP when you have reached a conclusion – ie. The predicate *investment(????)* is left behind after applying Modus Ponens.

Who killed Tuna?

Knowledge Base:

$\text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y)$

$\text{Loves}(x, y) \wedge \text{Animal}(y) \rightarrow \neg \text{Kills}(x, y)$

$\text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\text{Cat}(\text{Tuna})$

CNF conversion:

$\text{Loves}(x, y) \wedge \text{Animal}(y) \rightarrow \neg \text{Kills}(x, y) \equiv \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y)$

$\alpha = \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\neg \alpha = \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\text{KB} \wedge \neg \alpha \equiv$

$\text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y) \vee \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee$
 $\text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna})) \vee \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

$\text{Loves}(\text{Jack}, y) \wedge \text{Animal}(y) \vee \neg [\text{Loves}(x, y) \wedge \text{Animal}(y)] \vee \neg \text{Kills}(x, y) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna}))$

$x = \text{Jack}$

$y = \text{Cat}(\text{Tuna})$

$\text{Loves}(\text{Jack}, \text{Cat}(\text{Tuna})) \wedge \text{Animal}(\text{Cat}(\text{Tuna})) \vee \neg [\text{Loves}(\text{Jack}, \text{Cat}(\text{Tuna})) \wedge \text{Animal}(\text{Cat}(\text{Tuna}))] \vee$
 $\neg \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna})) \vee \text{Kills}(\text{Jack}, \text{Cat}(\text{Tuna}))$

$= \{\}$

Thus we reject: $\neg \alpha = \neg \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

But accept: $\alpha = \text{Kills}(\text{Curiosity}, \text{Cat}(\text{Tuna}))$

Financial Advisor

KB:

1. $\text{savings_account}(\text{inadequate}) \rightarrow \text{investment}(\text{savings})$
2. $\text{savings_account}(\text{adequate}) \wedge \text{income}(\text{adequate}) \rightarrow \text{investment}(\text{stocks})$
3. $\text{savings_account}(\text{adequate}) \wedge \text{income}(\text{inadequate}) \rightarrow \text{investment}(\text{combination})$
4. $\text{RequiredSavings}(\text{person}) = \$5,000 \times \text{NumberOfDependents}(\text{person})$
5. $\text{CurrentSavings}(\text{person}) \geq \text{RequiredSavings}(\text{person}) \rightarrow \text{Savings}(\text{adequate})$
6. $\text{CurrentSavings}(\text{person}) < \text{RequiredSavings}(\text{person}) \rightarrow \text{Savings}(\text{inadequate})$
7. $\text{RequiredIncome}(\text{person}) = \$1,500 + \$4,000 \times \text{NumberOfDependents}(\text{person})$
8. $\text{CurrentIncome}(\text{person}) \geq \text{RequiredIncome}(\text{person}) \rightarrow \text{income}(\text{adequate})$
9. $\text{CurrentIncome}(\text{person}) < \text{RequiredIncome}(\text{person}) \rightarrow \text{income}(\text{inadequate})$

New KB / Scenario:

1. $\text{CurrentSavings}(\text{person}) = \$22,000$
2. $\text{NumberOfDependents}(\text{person}) = 3$
3. $\text{CurrentIncome}(\text{person}) = \$25,000$

Inference:

- ❖ $\text{RequiredSavings}(\text{person}) = \$5,000 \times \text{NumberOfDependents}(\text{person}) = \$15,000$
- ❖ $\text{CurrentSavings}(\text{person}) \geq \text{RequiredSavings}(\text{person}) \equiv \$22,000 > \$15,000 \rightarrow \text{savings_account}(\text{adequate})$
- ❖ $\text{RequiredIncome}(\text{person}) = \$1,500 + \$4,000 \times \text{NumberOfDependents}(\text{person}) = \$27,000$
- ❖ $\text{CurrentIncome}(\text{person}) < \text{RequiredIncome}(\text{person}) \equiv \$25,000 < \$27,000 \rightarrow \text{income}(\text{inadequate})$
- ❖ $\text{savings_account}(\text{adequate}) \wedge \text{income}(\text{inadequate}) \rightarrow \text{investment}(\text{combination})$

Master of Technology in Knowledge Engineering

Unit 1

Intelligent Systems & Techniques for Business Analytics

Exercise & Discussion — Bayesian Nets

GU Zhan (Sam)

© 2017 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Exercise

- An admission committee of a college is trying to determine the probability that "an admitted applicant is really qualified". [Roventa & Spircu, *Management of Knowledge Imperfection in Building Intelligent Systems*, 2009]
- Your task (given the background information)
 - » Construct a Bayesian network
 - » For each node, determine the conditional probability table
 - » Determine the probability that an admitted applicant is really qualified
 - » Make your assumption if there is any conditions currently missing in the background information given

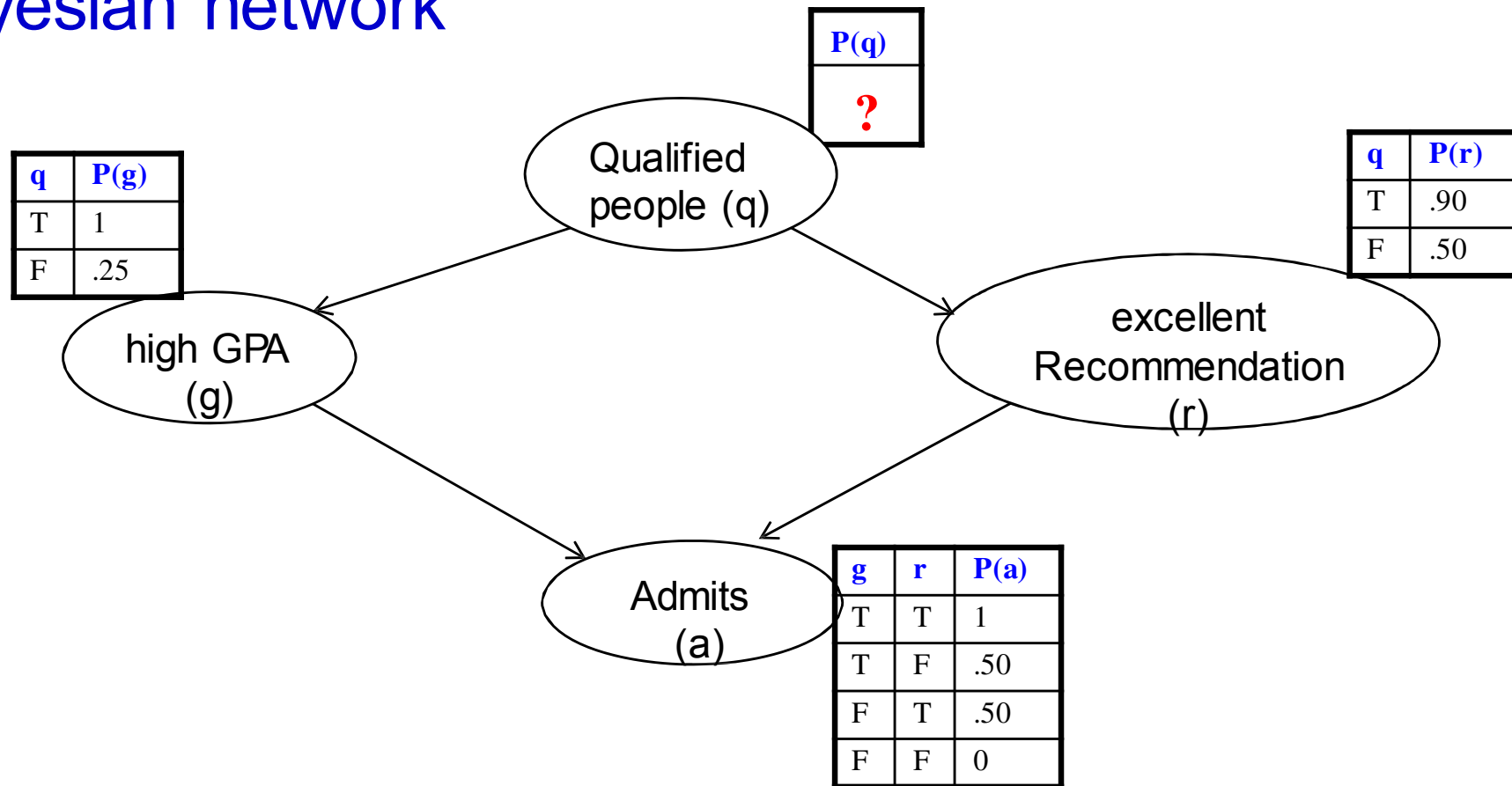
Exercise (cont.)

- Background information

- Qualified people have high GPA, however only about 90% of qualified people are able to obtain excellent recommendations.
- About a half of non-qualified people also possess excellent recommendations and about a quarter of non-qualified people have high GPA.
- The admission committee “admits” all applicants who have high GPA and possess excellent recommendation. All applicants who have not a GPA **and** do not possess excellent recommendation are “rejected”. The committee “admits” half of the rest applicants.

Exercise: Discussion

- Bayesian network



Master of Technology in Knowledge Engineering

Unit 1

Intelligent Systems & Techniques for Business Analytics

Exercise & Discussion

— Bayesian Nets

GU Zhan (Sam)

© 2017 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Exercise

- An admission committee of a college is trying to determine the probability that "an admitted applicant is really qualified". [Roventa & Spircu, *Management of Knowledge Imperfection in Building Intelligent Systems*, 2009]
- Your task (given the background information)
 - » Construct a Bayesian network
 - » For each node, determine the conditional probability table
 - » Determine the probability that an admitted applicant is really qualified
 - » Make your assumption if there is any conditions currently missing in the background information given

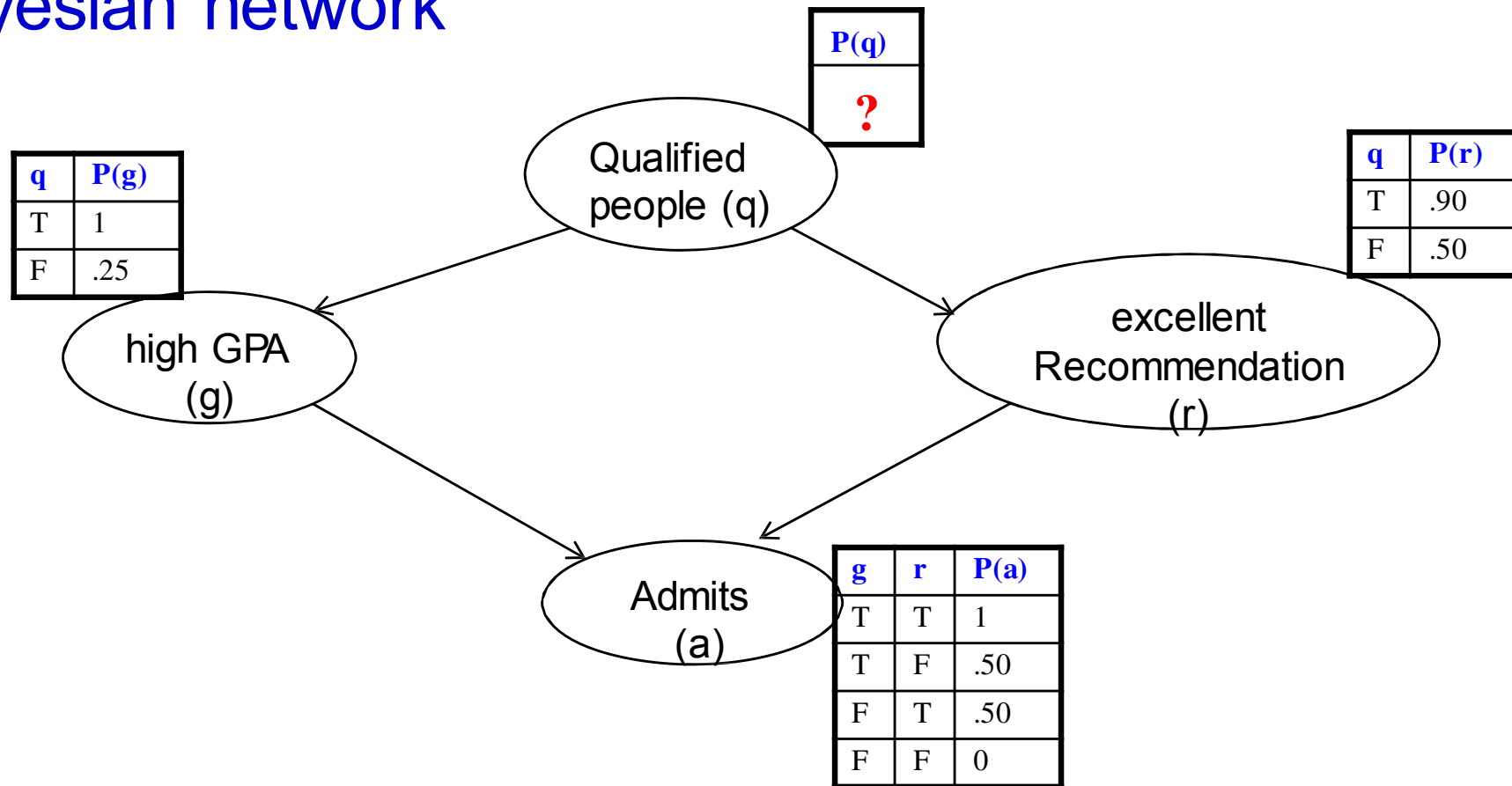
Exercise (cont.)

- Background information

- Qualified people have high GPA, however only about 90% of qualified people are able to obtain excellent recommendations.
- About a half of non-qualified people also possess excellent recommendations and about a quarter of non-qualified people have high GPA.
- The admission committee “admits” all applicants who have high GPA and possess excellent recommendation. All applicants who have not a GPA **and** do not possess excellent recommendation are “rejected”. The committee “admits” half of the rest applicants.

Exercise: Discussion

- Bayesian network



Exercise: Discussion (cont.)

- Inference

- » Assumption of missing condition:

- ♦ $P(q) = 0.3$

- » What we need determine

$$P(q|a) = \frac{P(a|q) \cdot P(q)}{P(a)} \approx 0.52$$

$$P(a|q) = P(a|g \wedge r) \cdot P(g|q) \cdot P(r|q) + P(a|g \wedge \neg r) \cdot P(g|q) \cdot P(\neg r|q) + \\ P(a|\neg g \wedge r) \cdot P(\neg g|q) \cdot P(r|q) + P(a|\neg g \wedge \neg r) \cdot P(\neg g|q) \cdot P(\neg r|q) = 0.95$$

$$P(a) = P(a|g \wedge r) \cdot P(g) \cdot P(r) + P(a|g \wedge \neg r) \cdot P(g) \cdot P(\neg r) + \\ P(a|\neg g \wedge r) \cdot P(\neg g) \cdot P(r) + P(a|\neg g \wedge \neg r) \cdot P(\neg g) \cdot P(\neg r) = 0.5475$$

$$P(g) = P(g|q) \cdot P(q) + P(g|\neg q) \cdot P(\neg q) = 1 \times 0.3 + 0.25 \times 0.7 = 0.475$$

$$P(r) = P(r|q) \cdot P(q) + P(r|\neg q) \cdot P(\neg q) = 0.9 \times 0.3 + 0.5 \times 0.7 = 0.62$$