Engineering Part IIA and EIST Part I 1999

Paper E6 (Computing Systems) Solutions

RC 6/99 20 pages

E6 1999

Solutions to E6 computer architecture questions

1. Carry-lookahead and carry-select adders

   (a) [Bookwork] Carry lookahead can be used to determine the carry inputs to each full adder without using ripple carry. For each bit i of the adder, we define two signals, generate gi and propagate pi. Bit i generates a carry if the two bits it's adding are both 1, and propagates a carry if either of the two bits it's adding are 1:

   $$\mathsf{gi = ai.bi} \qquad\qquad \mathsf{pi = ai + bi}$$

   c1, the carry into bit 1, will be 1 if either bit 0 generates a carry or c0 is 1 and bit 0 propagates a carry:

   $$\mathsf{c1 = g0 + p0.c0}$$

   Likewise for c2 and c3:

   $$\mathsf{c2 = g1 + p1.g0 + p1.p0.c0}$$
   $$\mathsf{c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0}$$

   These expressions show how the carry-in signals can be obtained without waiting for them to ripple through a 4-bit adder. Unfortunately, the expressions get more and more complex for larger adders, requiring gates with large fan-ins which are not feasible in practical hardware implementations. For this reason, the design of a carry-lookahead adder is usually hierarchical. At the bottom of the hierarchy, we may choose to use 4-bit adders with carry-lookahead as above. Four of these 4-bit adders can be connected together using a higher level of carry-lookahead, producing a 16-bit adder. Similarly, four of these 16-bit adders can be connected together using an even higher level of carry-lookahead, producing a 64-bit adder.

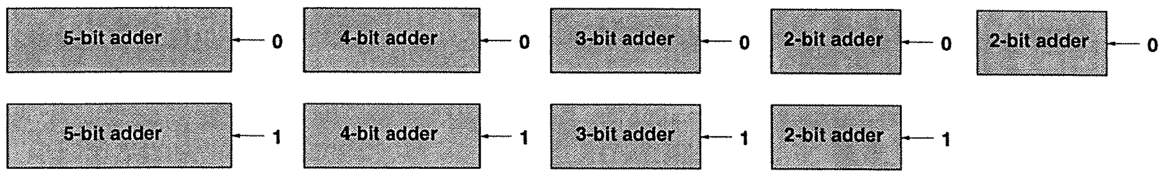
   For an n-bit adder, if we use carry-lookahead to predict m carry-in signals at each level of the hierarchy($m = 4$ in the above example), then we will need $\log_m n$ levels and gates with a fan-in of m. Since each level takes a constant amount of time to operate, the asymptotic time requirement is $O(\log_m n)$. In any reasonably straightforward silicon layout, the asymptotic space requirement is $O(n \log_m n)$.

   The choice of the optimal value for m will depend on the particular implementation technology. The asymptotic complexities suggest that the larger the value of m, the fewer the number of levels and hence the faster the adder. However, this will require higher fan-ins to gates, which may be infeasible in certain technologies and will certainly result in longer propagation delays and hence slower operation.

   (Key points: clear understanding of generate and propagate signals, hierarchy required because of impractical gate fan-ins, asymptotic time and space requirements, optimal number of levels depends on the particular implementation technology.)

(b) (i) For optimal operation, each block should complete its summing just as the carry-out of the previous stage is available. Starting from the right of the carry-select adder, the first 4-bit block will produce its carry-out signal after 4 time units, which is just in time to select between the two sums produced by the next 4-bit block. The carry-in to the next block will be available one time unit later, after a total of 5 time units. This is just when the next (5-bit) block has completed its parallel additions. The carry-in to the next block will be available one time unit later, after a total of 6 time units. This is just when the next (6-bit) block has completed its parallel additions. Continuing this line of argument, it becomes clear that each block should be one bit longer than the next, except for the first two blocks, which should be the same size.

The optimal design of a 16-bit carry-select adder would look like this:



(ii) For an n-bit adder, the left hand block will need to be of size m, where $m + (m-1) + (m-2) + \ldots + 3 + 2 + 1 + 1 \geq n$. Noting that

$$\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$$

we obtain

$$\frac{m(m+1)}{2} + 1 \geq n \quad \Leftrightarrow \quad m^2 + m - 2n + 2 \geq 0$$

Solving for the equality, we obtain

$$m = \frac{-1 \pm \sqrt{1 + 8(n-1)}}{2}$$

As $n \to \infty$, the positive solution approaches $\sqrt{2}\sqrt{n}$, so $m \geq \sqrt{2}\sqrt{n}$. Since the adder will take m time units to operate, its asymptotic time requirement is therefore $O(\sqrt{n})$. The asymptotic space requirement is clearly $O(n)$.

(c) Knowing the asymptotic behaviour of adders is a useful design tool, but should not be relied on too much, since asymptotic behaviour is only really important when n becomes large. For smaller n, the constant of proportionality might make an $O(n)$ adder faster than an $O(\sqrt{n})$ one. Real adders in real CPU's deal with 64-bit numbers at most, so asymptotic behaviour should not be relied on too much as a design criterion.

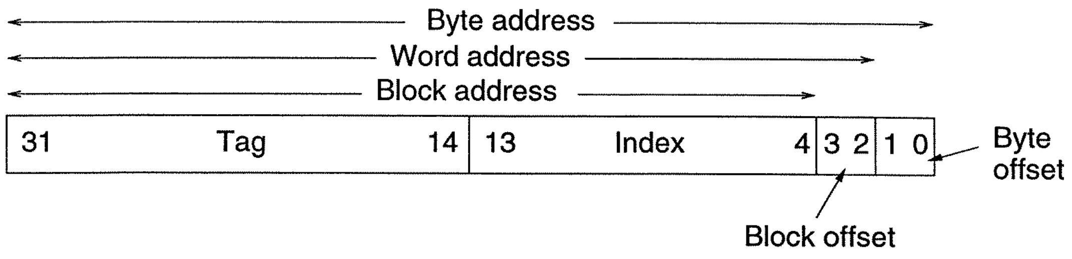
2. Caches and locality of reference

(a) The two properties are:

Temporal locality of reference: if an item is referenced, it tends to be referenced again soon (loops, local variables).

Spatial locality of reference: if an item is referenced, items with nearby addresses will tend to be referenced soon (instructions, data in arrays)

(b) The cache holds 128 KBytes of data, which is 32K words, 8K blocks or 1K sets. Since 1K= 2^{10}, this means that 10 bits of the physical address will act as the cache index. The least significant two bits of the physical address are the byte offset, the next two the block offset. The division of the physical address is therefore:



The index is used to locate a particular set of 8 blocks in the cache. Each of these blocks is checked to see if the strored tag matches the tag field in the physical address. If it does, then the requested word is in the cache, and the block offset is used to extract the appropriate word from the matching block.

(c) The miss rate can be reduced by:

Increasing the block size: this takes better advantage of spatial locality of reference, thus reducing the miss rate. However, there is a corresponding increase in the miss penalty, so the computer will not necessarily go faster.

Increasing the degree of associativity: this allows more flexibility about which block to replace on a miss: sensible block replacement strategies like LRU reduce the miss rate. However, there is a corresponding increase in the hit time and the miss penalty, so the computer will not necessarily go faster.

(d) (i) The cache can store $1024/4 = 256$ real numbers. Since the matrix elements are all `Real` numbers, it follows that the cache can store 256 matrix elements at a time.

(ii) The code segment in Fig. 3 has very poor temporal locality of reference. Even though the elements of `b` and `c` are each referenced 1000 times, successive references to the same element are not close together. The inner loop in `k` references 1000 distinct elements of b and 1000 distinct elements of c before there are any repeat references. These 2000 references will fill and refill the LRU cache approximately 8 times over, with no cache hits. It follows that every reference to a matrix element in Fig. 3 will miss. Since there are 10^9 references to elements of b and c, and 10^6 references to elements of a, there will be approximately 2×10^9 cache misses.

(iii) The code segment in Fig. 4 has much better temporal locality of reference. The inner loops in j and k reference 100 elements of c and 10 elements each of a and b. All these elements will comfortably fit in the cache. The next iteration of the i loop references the same 100 elements of c and 10 new elements each of a and b. Since the 100 elements of c are already in the cache, and they are all used each time round the i loop, there will be no further cache misses for c until the next iteration of the kk loop. So, for the inner three loops (in i, j and k) there will be a total of $100 + 2 \times 10 \times 1000 = 20100$ cache misses. The three inner loops are enclosed in two nested loops in kk and jj, each of which iterates 100 times. So the total number of cache misses will be $100 \times 100 \times 20100 \approx 2 \times 10^8$, an improvement by a factor of 10 over the code in Fig. 3.

Answer 3

(a) The 2 features that could cause erroneous program behaviour are:

(i) A lack of concurrency management that can lead to a race condition. This can occur as the program is running as a multi-threaded server process. A debit operation on an account can be preempted at any time and at any point in the operation. If this preemption occurs whilst part-way through debiting an account and an operation is started to debit the same account (another account holder is also involved in a transaction) then both operations might have read the same account value into memory which they will then operate on. The operation that finishes last will write back an erroneous value. If the server is run on a parallel machine then true multi-processing is possible, leading to the same problem. An example scenario is:

```
Debit Account A (£50)
Read balance A = £200 ------preempt thread------→ Debit Account A (£30)
                 ←-------------------------------------- Read balance A = £200
Balance = £200-£50
Store balance A (£150)
End
                                                   Balance = £200-£30
                                                   Store balance A (£170)
```

(NB: the preemption here is caused by operating system threads blocking due to I/O operations)

The final value is £170 rather than £130. This is a loss of £50 for the bank. One consequence is thus financial loss. Another scenario is that the account holders can withdraw more money than is actually in the account by two or more concurrent transactions totalling more than the currrent balance experiencing a race condition as described above.

(ii) A year 2000 problem (Y2K). This is due to two factors. Firstly only the trailing two digits of the year are stored and processed by the system (e.g. 98 instead of 1998) – as illustrated in the `card_account` record by field `expiry_year`. Secondly, the statement that checks if the card has expired or not does not take into account the millenium. It uses a straight less than comparison, which can cause erroneous behaviour if the card's expiry year is in the next millenium (00 onwards) and the current year is still in the current millenium. So for example, if the expiry year is 2001 and the current year is 1998 then the comparison (98 < 01) will fail to validate the card. The consequences for the bank are that customers with new debit cards will be refused transactions, leading to their embarrassment and potential loss of custom for the bank.

(b)

(i) The debit operation should be implemented as an atomic action over a particular account, i.e. make it a single unit of concurrency as opposed to a sequence of steps that can be preempted and later resumed at any point. A good way in this case is the use of database locking – so that just the account in question is locked, thus providing maximum concurrency (debit operations on other accounts can proceed). Even more advanced is to propose the use of a transaction processing system, which would most likely use simple locking in this case since only one object/table is operated on. However, full marks will also be awarded here for a solution proposing any working in-memory solution, e.g. semaphores.

```
PROCEDURE debit
.....
WAIT(Semaphore)          LOCK(account)          Start Transaction
Read account value
Date comparison
Balance Comparison
Deduct Amount
Write account value
SIGNAL(Semaphore)        UNLOCK(account)        End Transaction
```

The semaphore is initialised to 1 (a binary semaphore) thus only allowing one thread to be within this zone of mutual exclusion. Other approaches are fine, e.g. machine level instructions to implement atomic actions, monitors, Ada rendezvous etc.

(ii) Use full 4 character year representations. This would solve the problem of comparison since (1998 < 2000) is true, for example.

(c) The debit operation should have been tested as an independent component using test data to provide coverage of a key range of inputs.

Equivalence partitioning techniques should have been used to examine the ranges of input values (in this case the values in the relevant account fields as well as card number and amount to debit). In the case of years, the range over time would be {startyear..99, 00..99} and values of particular interest would be 98, 99, 00, 01.

Dummy accounts could be set up in the database with values in the middle of and at the edges of partitioned ranges. Alternatively, if the database system was not yet developed and tested, test stubs could be written to simulate the database operations and return predetermined test values. A stub to simulate the `get_current_date` procedure could have been used to test date scenarios in the future.

4

Answer

(a) A hard real-time is a system whose operation is incorrect if it does not behave within the parameters of its timing specification. The system must respond to events, e.g. from sensors, within a specific period of time, usually because of some safety-critical requirement, e.g. a nuclear reactor will blow up if not.

Of the listed systems, the nuclear reactor monitor and the fly-by-wire control system are hard real-time. These must respond to hard deadlines, e.g. movement of joystick means the plane must go up by the specified amount within a specified hard deadline. Although the ambulance incident system must be designed for efficiency of operation, it does not have any specific hard deadlines.

(b) It is unlikely that a prototyping approach would be used for requirements capture of most hard real-time systems. Prototyping is used when a software system has to achieve a desired result but it is unclear of the specification of a system to achieve this result. An operational prototype is developed and then experimented with to refine the system requirements. This approach is typically used for interactive systems, for example designing a user interface for air traffic control. In contrast, hard real-time systems are usually well-defined problems. There are a set of events to handle, a set of actions to initiate in response and a set of deadlines that must be met.

Additionally, building an untested prototype of a hard real-time system can be dangerous. Many such systems control safety-critical equipment, e.g. nuclear reactors (in which system errors can be disasterous (a simulation of the system may be used for testing)).

Often real-time systems are tightly coupled to user interfaces, e.g. fly-by-wire aircraft systems. A prototyping approach can be useful for requirements capture to design the user interface, although this is not strictly part of the real-time system itself. (Also many user interfaces are based on the analogue equivalents).

(c) A requirements specification gives a precise description of the software system's functionality and constraints on its operation. It should be unambiguous to avoid misunderstandings between the customer and the developer. A spec is often used as the basis of a contract between these parties.

Many procurers of hard real-time systems (e.g. the UK MoD, US DoD) insist on formal specs because they are mathematical entities, allowing proofs of consistency and possibly ensuring an implementation conforms to its spec (verification). Formal specs also provide an accurate guide for the software tester. Generally, they avoid ambiguity and increase confidence in systems where safety, reliability and security are paramount.

(d) Multi-purpose operating systems are not suitable for hard real-time systems because their scheduling algorithms are not appropriate. Hard real-time systems require events to be handled within specific deadlines. Priority levels can be specified in mainstream OS but hard schedules cannot be guaranteed. Usually real-time systems are statically analysed to determine events and deadlines. An appropriate scheduling algorithm is then devised, e.g. earlist deadline first. A fixed number of processes are deployed, and it is ensured that the system load under all conditions is within safe parameters.

Answers Q5(a) (i) Posterior probabilities using Bayes' formula:

$$P[\omega_j | \underline{x}] = \frac{p(\underline{x}|\omega_j)\, P(\omega_j)}{\sum_i p(\underline{x}|\omega_j)\, P(\omega_i)}$$

- where prior probabilities $P(\omega_j)$ of class ω_j tells us the frequency of occurrence of each class BEFORE we have seen the data. Usually derived from higher-level knowledge about the problem

- $p(\underline{x}|\omega_j)$ is the probability density function of the feature vector $\underline{x}$ corresponding to class ω_j. We can model this in parametric form (eg. Gaussian below) and estimate the parameters from data labelled ω_j.

- $P[\omega_j|\underline{x}]$ is the posterior probability of the data $\underline{x}$ belonging to class ω_j. This is used in Bayes' decision rules

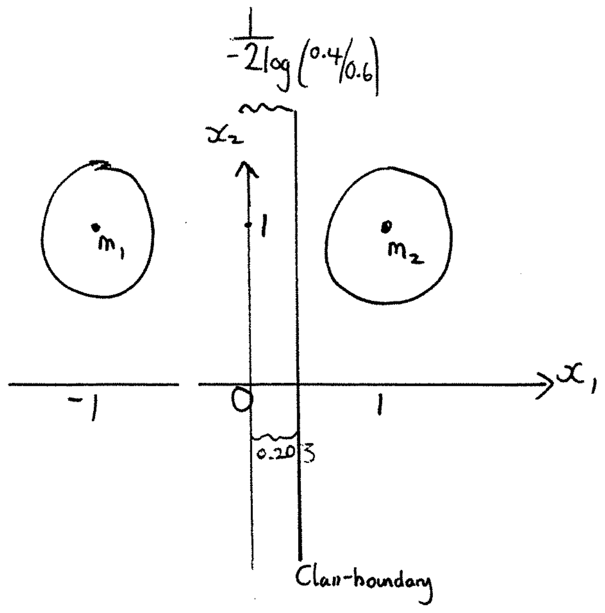
(ii) Calculate C posterior probabilities:

$$P[\omega_1|\underline{x}],\ P[\omega_2|\underline{x}]\ \ldots\ldots,\ P[\omega_c|\underline{x}]$$

and find maximum of these and assign $\underline{x}$ to corresponding class.

eg. in 2-class problem decide $\begin{cases} \text{class } \omega_1 & \text{if } P[\omega_1|\underline{x}] > P[\omega_2|\underline{x}] \\ \omega_2 & \text{otherwise.} \end{cases}$

Q5(b)



(i) $P(\omega_1) = 0\cdot6 \quad P(\omega_2) = 0\cdot4$

Optimal decision rule, (assuming equal costs of errors), to minimise the prob. of misclassification error is :

$$\text{if } P[\omega_1 | \underline{x}] > P[\omega_2 | \underline{x}_2] \quad \underline{x} \in \omega_1$$

Bayes' rule.

$$\frac{p(\underline{x}|\omega_1)P(\omega_1)}{p(\underline{x}|\omega_1)P(\omega_1)+p(\underline{x}|\omega_2)P(\omega_2)} > \frac{p(x|\omega_2)P(\omega_2)}{p(\underline{x}|\omega_1)P(\omega_1) + p(\underline{x}|\omega_2)P(\omega_2)}$$

Since the denominator is same in both expressions and taking logs of both sides :

$$\log p(\underline{x}|\omega_1) + \log P(\omega_1) > \log p(\underline{x}|\omega_2) + \log P(\omega_2)$$

$$-\frac{1}{2}(\underline{x}-\underline{m}_1)'(\underline{x}-\underline{m}_1) > -\frac{1}{2}(\underline{x}-\underline{m}_2)'(\underline{x}-\underline{m}_2) + \log\frac{P(\omega_2)}{P(\omega_1)}$$

$$-\|\underline{x}-\underline{m}_1\|^2 > -\|\underline{x}-\underline{m}_2\|^2 + 2\log(0.4/0.6)$$

Rearrange to give $x_1 = \frac{1}{2}\log(1.5)_{10}$

Q5(b)(cont).

Decision rule:

Class 1 if $d_1 < d_2 - \boxed{\frac{1}{2}} \log {0.4}/{0.6}$

Class 2 otherwise

where $d_1^2 = \| \underline{x} - \underline{m}_1 \|^2$, euclidean distance-squared to mean of class

or $x_1 = \frac{1}{2} \log(1.5)$

ii) Class-boundary (see sketch) goes closer to $\underline{m}_2$. Class-boundary is a line (it is a minimum distance classifier)

c) . ROC curve is useful when the cost of different types of errors can change at the time of use of classifier. Often this is the case in medical diagnostics problems, for example, where the user might want to set a threshold to strike a balance between TRUE POSITIVE and FALSE POSITIVES.

For example in question use:

$$d_1 < d_2 - \theta \Rightarrow \text{class } 1$$

as a decision rule and change θ to get an ROC curve.

Q6 (a) The pattern classification problem can be treated as an interpolation problem by finding a function $g(\underline{x})$ that satisfies $f_i = g(\underline{x}_i)$ for the training data where we specify real target ~~values~~ values for each class:

eg. $f_i = +1$ for $\underline{x}_i \in \omega_1$
$f_i = -1$ for $\underline{x}_i \in \omega_2$

For linear interpolation, $g(\underline{x}) = \omega_0 + \sum_{j=1}^{d} \omega_j x_j$

$$g(\underline{x}) = \omega_0 + \underline{w}^T \underline{x}$$

(b) We minimize the interpolation error:

$$E = \sum_{i=1}^{N} \left(f_i - g(\underline{x}_i)\right)^2 = \sum_{i=1}^{N} \left(f_i - w_0 - \underline{w}^T \underline{x}_i\right)^2$$

with respect to the unknowns $\underline{w_0}$ and $\underline{w}$. We can treat $\underline{w_0}$ and $\underline{w}$ uniformly + with same notation by defining:

$$\underline{y} = \begin{bmatrix} x \\ 1 \end{bmatrix} \quad \text{and} \quad \underline{a} = \begin{bmatrix} \underline{w} \\ w_0 \end{bmatrix}$$ (d+1 dimensional vectors)

$$\therefore E = \sum_{i=1}^{N} \| (\underline{a}^T y_i - f_i) \|$$ ← quadratic function of unknowns.

$$= \| Y\underline{a} - \underline{b} \|$$ where Y: $N \times (d+1)$ matrix with rows y_i^T

$\underline{b}$ = vector of targets f_i

6(b) cont

The solution (in a least-squares sense). By differentiation and looking at minima:

$$\frac{\partial E}{\partial \underline{a}_i} = 0 \quad \text{or} \quad 0 = 2Y^T(Y\underline{a} - \underline{b})$$

$$\underline{\underline{\hat{a}} = [Y^TY]^{-1}Y^T\underline{b}} \quad (\text{ie pseudo inverse of } Y)$$

6(c). An alternative is to calculate ∇J, the derivative of E with respect to elements of $\underline{a}$ and starting from a random guess of $\underline{a}$, iterate toward the solution:

$$\underline{a}^{(k+1)} = \underline{a}^{(k)} - \rho \nabla J(\underline{a}^k) \quad \text{where } \rho \text{ is a small gain.}$$

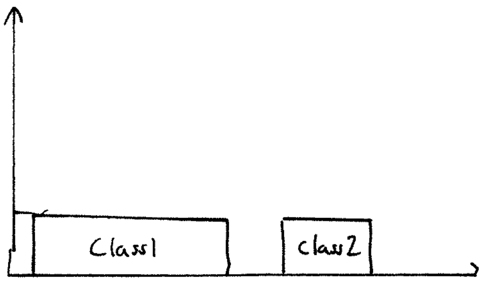
For the linear model E is quadratic with a single global minima so gradient search will converge to the correct solution.

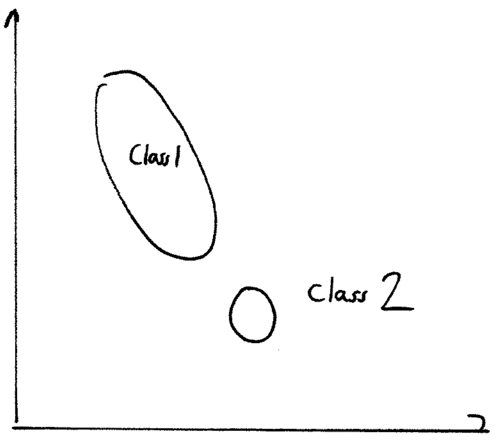
ρ is an arbitrary constant; ρ too small mean convergence is too slow; ρ too large leads to oscillations.

Curvature (second derivative) can be used to descend faster. Convergence rate is determined by the eigenvalues of $[Y^TY]$.

(d) Perceptron algorithm doesn't minimize the interpolation error. Instead, at any stage in the iterative estimation, only examples that are misclassified contribute to the correction.
(note: error function for misclassified example is linear.

6(d) Examples where interpolation error fails and the perceptron succeeds might be:





Q7 (a) The solution of many problems can be described by finding a sequence of actions that lead to a desirable goal.

In a well-defined problem:

Initial state S

Goal state G

Operator or successor functions
- for each state x, $S(x)$ return the set of states reachable from x with one action

In state-space search we systematically examine all states reachable from initial state S ~~by~~ any sequence of actions to find a path.

state-space

path

path cost

goal test.

Informed and uninformed methods.

Q7(b) A^* search algorithm:

1. Form a queue, Q, of partial paths.
   Let Q initially be zero-cost, zero-step path from root to nowhere.

2) Until is empty or goal has been reached

   - remove first path from Q
   - form new paths, extending by one step
   - add new paths to Q
   - sort by sum of costs + lower bound estimate of cost remaining
   - if 2 or more paths contain a common node, delete all but the path that reaches the common node with lowest cost.

depth-first — extend deepest node first
$O(b^d)$ in time, $O(bd)$ in memory, complete but not optimal. No discrimination + suffers from unbalanced trees.

breadth-first
— expand shallowest node first
— $O(b^d)$ time and memory
— complete but not optimal

A^* — Optimal and complete. Memory + time depends on efficiency of evaluation function (how close to true cost is estimate of remaining cost).

Q7 (c)

(i) Configuration space representation has advantage that optimal path will consist of moves to visible vertices in straight-lines.
Problem is well-defined and has a simple successor function and a finite branching factor.

(ii) depth-first will produce

S – A – B – C – D – E – F – H – G (8 depth.).

breadth-first S – K – J – G (cost ≃ 14.6),

A* search will produce S – E – F – P – G (cost ≃ 13.9)
(use geometric distance for evaluation function)

(iii) if rotation is allowed the configuration space will be 3D.
Rectangle vertices are vertices in 3D space.

Shortest path will be S – F′ – G where F′ is real vertex nearest rectangle.

Q8(a)(i) Logic is a formal language to represent ~~th~~ to reason about knowledge in a computer-tractable form.
It is concise, avoids ambiguity, context insensitive, expressive, and effective for making inferences

Proposition symbols are used to represent facts (either true or false). Each symbol can mean what we want it to be.

Propositions are combined with logical connectives to generate sentences with more complex meanings. Meaning of sentences is rep. by truth-tables

Inferences are made from the sound rules of inference. (proof theory).

(ii) Knowledge database, KB of clauses or axioms.
Prove theorem P by adding $\neg P$ to database + find a contradiction.

$$KB \wedge \neg P \rightarrow FALSE$$

$$\equiv KB \rightarrow P.$$

$A \rightarrow B$

premises

conclusion.

(b).

A	B	C	$\neg A \vee B$	$A \vee C$	$B \vee C$		
F	F	F	T	F	F		$F \rightarrow F$
F	F	T	T	T	T	✓	$T \rightarrow T$
F	T	F	T	F	T		$F \rightarrow T$
F	T	T	T	T	T	✓	$T \rightarrow T$
T	F	F	F	T	F		
T	F	T	F	T	T		prove that it is sound
T	T	F	T	T	T	✓	
T	T	T	T	T	T	✓	

When premises are true, conclusion is also true.

8 (c) (i).

$$\forall x \forall y \forall z \Big[[\text{son}(x,y) \vee \text{daughter}(x,y)] \wedge [\text{son}(z,y) \vee \text{daughter}(z,y)] \longrightarrow \text{sibling}(x,y) \wedge \text{sibling}(y,x) \Big]$$

ii)

$\neg \text{son}(x1, y1) \vee \neg \text{son}(z1, y1) \vee \text{sibling}(x1, y1)$ (1)

$\neg \text{son}(x2, y2) \vee \neg \text{daughter}(z2, y2) \vee \text{sibling}(x2, y2)$ (2)

$\neg \text{daughter}(x3, y3) \vee \neg \text{daughter}(z3, y3) \vee \text{sibling}(x3, z3)$ (3)

$\neg \text{sibling}(x4, y4) \vee \text{sibling}(y4, x4)$ (4)

son (Edward, Henry) (5)

daughter (Elisabeth, Henry) (6)

daughter (Mary, Henry) (7)

daughter (Elisabeth, Anne) (8)

son (Elisabeth, Charles) (9)

iii) Prove $\exists u\, [\text{sibling}(\text{Elisabeth}, u)]$?

Add $\neg \exists u\ \text{sibling}(\text{Elisabeth}, u) \equiv \neg \text{sibling}(\text{Elisabeth}, u5)$ in CNF. (10)

Use resolution inference rule and unification to find a contradiction and hence prove the theory.

Resolve (3) and (10), unifying $x3/\text{Elisabeth}$ and $u5/z3$

$\neg \text{daughter}(\text{Elisabeth}, y3) \vee \neg \text{daughter}(u5, y3)$ (11)

Resolve (11) and (6), unifying $y3/\text{Henry}$

$\neg \text{daughter}(u5, \text{Henry})$ (12)

Contradiction with (7) ∴ $u5 = \text{Mary}$.

19