*Int. J. Engng Ed.* Vol. 13, No. 2, p. 90–97, 1997
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 1997 TEMPUS Publications.

# Comprehension Exercises: A New Type of Examination Question

MIKE HOLCOMBE
*Department of Computer Science, University of Sheffield, 211 Portobello Street, Sheffield, S1 4DP, UK*

*For many subjects in the arts area, particularly languages, comprehension exercises are a fundamental aspect of the teaching of the subject. Students are given a passage of prose or a poem which they have probably not seen before and then asked a number of questions about it to test their understanding of it, to ask them to critically analyse the passage and perhaps to develop some of the ideas and themes further. Since much of computer science is also language based, particularly in the realms of formal methods and the theory of computation, the issue of comprehension, criticism and development are also vital issues. We have been introducing comprehension exercises into some of our examinations in order to develop and assess these skills explicitly. In fact we have been doing this since 1988 and the conclusions from this, together with some examples of examination papers, are given.*

## 1. INTRODUCTION

COMPUTER science and software engineering are highly dependent on the representation of ideas and concepts using a variety of—often highly abstract—languages. These range from the well known programming languages to formal specification languages, formal logics and ultimately, mathematics. In all language-dependent activities the comprehension of the language is a vital component and one that is directly addressed by educators in the foreign language and English language community. With their centuries of experience they have evolved approaches to teaching and assessing which can provide us with many interesting ideas.

Within the realm of teaching programming languages we have often asked students to explain the behaviour of a piece of code, perhaps to criticise it and possibly correct and/or extend it. However, in computing, language does not just stop at programming languages and the emphasis of this paper is on using the same ideas in the context of theoretical computer science and formal methods. It turns out to be an excellent way to find out what students can do when faced with unfamiliar situations and to encourage a deeper appreciation of the subject and the development of a powerful set of skills for coping with potential new ideas and terminology of the future. This last aspect is vital in a subject that is changing so rapidly and the needs of students in their future careers to be able to come to grips with new languages, concepts and discoveries rapidly and in a meaningful way, will be of paramount importance.

* Accepted 30 November 1996.

## 2. BACKGROUND

All undergraduate students in the Department of Computer Science must take a second-year course in Theory of Computing as well as one in Formal Methods. These courses follow introductory modules on these topics in the first year. All third-year Computer Science students take a course on Advanced Theory of Computing; for Software Engineers it is optional. The comprehension exercises are part of the second and third year examinations in Theory of Computing and Advanced Theory of Computing.

The papers are divided into two parts, the first part being a compulsory comprehension exercise and the second part being of a more traditional form. In the first part an article is presented which might range from 3–5 pages long on a subject related to the material that has been discussed in the course but which presents an aspect which is new to the class. The source of the article might be a research paper or a section in a book. Due attention needs to be given to the copyright situation and if permission to reproduce the article is not obtained then it does involve the examiner in writing the article themselves. This is not as onerous as it sounds and one can take the opportunity to define concepts and terms and to provide examples and motivation to an appropriate level for the class.

Following the article are a number of questions that try to:

- establish how much the student has understood;
- determine whether the student can develop some of the ideas—perhaps by completing a proof or explaining how the material applies to a specific example;
- require the student to evaluate the material, perhaps contrast it with other approaches and to critically assess it.

The structure of the papers has to allow for the extra reading time that is needed for these comprehension questions or the articles that form the subject of the comprehension have to be circulated a few days before the examination. We have tried both arrangements and both seem to work. It really depends on the length and complexity of the article whether it needs to be circulated beforehand. If the question is not distributed beforehand then the marks allocated for the first section need to be quite heavily weighted to prevent an unfair situation. Thus, for example, in a 3-hour examination the marks available for the comprehension paper might add up to 40% and the rest of the paper, consisting of a choice of 3 questions from 7 each worth 20%. This was the case for the example in Section 3.2.2 where the paper was circulated prior to the examination. In the example in Section 3.1.1, the comprehension question was not circulated before the examination; it was worth 34 marks and the rest of the paper consisted of 7 standard style questions, worth 17 marks each, of which 4 had to be attempted. (The final percentage mark for the course was obtained by combining the examination mark with the practical and coursework marks from the rest of the course.)

The examples in Section 3 illustrate the sort of articles and questions that we have used in recent years.

The preparation of the class for the papers takes a number of forms. First they are used to obtaining papers and other sources from the library (and more recently the 'Web') summarising their contents and presenting their views to their classmates and tutors. This provides them with some of the skills. In the course of the lectures we also hold a number of tutorial sessions that directly address the process. In the course of these we circulate short articles for them to spend 30 minutes or so reading and then have a few questions for discussion of the type that are found in the examination. This provides them with some further experience in the sort of approaches that might be successful in the tackling of comprehension questions.

## 3. EXAMPLES OF EXAMINATION PAPERS

### 3.1. Second-year comprehension questions

3.1.1. Example 1. Read the following passage carefully and then answer the questions which follow. This passage is closely based on the first few pages of Chapter 1 of J. H. Conway's book *Regular Automata and Finite Machines* (Chapman & Hall, 1971).

**Moore machines**

A *Moore machine* or *Moore automaton* is a special form of finite state transducer where the output from any state s is the same for every input. Formally, a Moore machine M consists of:

(a) an input alphabet, I
(b) an output alphabet, O
(c) a finite set S of states
(d) a transition function, $t: S \times I \to S$
(e) an output function, $o: S \to O$
(f) a particular initial state i.

We assume that t is a *total* function, so that every state has a transition on every input. The pair (I, O) is called the *console* of M, i.e., the console specifies which symbols are accepted and which are printed by M. We may draw Moore machines in much the same way as finite state machines can be drawn, indicating the initial state with an arrow, and marking each state with its associated output, so that, for example, the Moore machine $M_1$ specified by:

$$I = \{a, b\},\ O = \{0, 1\},\ S = \{i, j\},\ t(i, a) = j,\ t(i, b) = i,$$
$$t(j, a) = j,\ t(j, b) = i,\ o(i) = 0 \text{ and } o(j) = 1,$$

is represented by the diagram (Fig. 1).

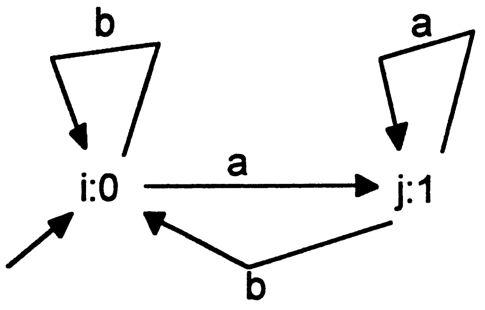


Fig. 1.

The *state* $t^*(s, w)$ *reached from a given state* s *on an input string* w is completely determined by s, w and the transition function t, and may be defined by $t^*(s, \lambda) = s$, $t^*(s, wa) = t(t^*(s, w), a)$. Thus $t^*(i, baa) = j$ for machine $M_1$.

The *path* $p(s, w)$ *taken from state* $s \in S$ *on input string* $w = a_1a_2 \ldots a_n$ is the string of states visited on the way from s to $t^*(s, w)$, and is defined by:

$$p(s, a_1a_2 \ldots a_n) = st^*(s, a_1)t^*(s, a_1a_2) \ldots t^*(s, a_1a_2 \ldots a_n)$$

Thus

$$p(i, baa) = iijj.$$

The *output string* $o^*(p)$ *of any path* p in $S^*$ is the string of outputs of the individual states in p, so $o^*(\lambda) = \lambda$ and $o^*(ps) = o^*(p)o(s)$. Thus $o^*(iijj) = 0011$.

The *observation* $obs(s, w)$ *of a state* s *on an input string* w consists of the output string of the path taken from s on w, i.e., $obs(s, w) = o^*(p(s, w))$. Thus $obs(i, baa) = 0011$.

**Experimenting on Moore machines**

The theory of experiments is developed under the assumption that the only immediate information a user can extract from a Moore machine in a given state s is the output o(s) of that state, so that the user has no direct knowledge of the state of his or her machine. However, by choosing different input strings and observing the outputs that result, a user can infer information about the states of her machine, and in particular may be able to distinguish between the different states.

Formally, we may specify an experiment as a function: $e: O^* \to I \cup A$ where A is an answer space (e.g., $O^*$) disjoint from I. To perform e on a Moore machine $M = (I, O, S, t, o, i)$ (which will be in some given state s) execute the following routine:

(0) At the start, observe the output $x = o(s)$ immediately available.
(1) Suppose the string input so far is w and the string so far observed is z (so that $z = obs(s, w)$). Evaluate e(z).
(2) (a) if $e(z) = a \in I$ then extend w to wa and extend z to zx by applying a to M and observing the new output $x = o(t^*(s, wa))$. Then go to (1).
    (b) if $e(z) \notin I$ then stop, taking e(z) as the outcome of the experiment.

Thus the *choice of a new input depends on the outputs so far observed.* Note that we may perform e on any Moore machine with the same console as M. Note also that it is possible for the performance of an experiment to fail to terminate, and this happens if every application of e to a generated output string yields another member of I.

If every performance of e on M in any state does terminate, we say that e is *finite*, and we may define the *length* of e as the maximum length of an *input* string generated by any performance of e on M.

For example, consider the experiment $e: O^* \to I \cup O^*$ of length 2 which uses the console of $M_1$, and is defined by $e(0) = e(1) = b$; $e(00) = e(01) = e(10) = e(11) = a$; $e(z) = z$ if $|z| > 2$. When e is performed on $M_1$ in state i the result is:

(Step 0) Output is 0 = observed string
(Step 1) Input so far is $\lambda$, observed string $= 0 = \text{obs}(i, \lambda)$. $e(0) = b$
(Step 2) Since $b \in I$, input string becomes $\lambda b = b$; $t(i, b) = i$ and $o(i) = 0$, so observed string becomes $00 = \text{obs}(i, b)$.
(Step 1) $e(00) = a$
(Step 2) $a \in I$, so input string becomes ba; $t(i, a) = j$ and $o(j) = 1$, so the observed string becomes $001 = \text{obs}(i, ba)$.
(Step 1) $e(001) = 001$
(Step 2) $001 \in O^*$ (and is not in I), so take 001 as the outcome of the experiment.

**Questions**

Please give answers to the following:

(a) Draw a diagram of the machine $M_2$ which has the same console as $M_1$, which has state set $S = \{i, j, k\}$ and whose transition and output functions are specified by

$$t(i, a) = j,\ t(i, b) = i;\ t(j, a) = j,\ t(j, b) = k;$$

$$t(k, a) = t(k, b) = k;\ o(i) = o(k) = 0;\ o(j) = 1.$$

[5 marks]

(b) For *each* state $s \in \{i, j\} \subset S$ determine
  (i) $t^*(s, bbaa)$
  (ii) $p(s, bbaa)$
  (iii) $\text{obs}(s, bbaa)$

[5 marks]

(c) A *reduced* experiment would be to simply observe the output from a Moore machine, M, on a given word, w, i.e., to note the value of obs(s, w) where s is the state of M when the experiment begins. Show that a reduced experiment is a special case of a finite experiment e for which the value e(z) depends only on the length of z and not on its contents.

[7 marks]

(d) Consider the experiment $e: O^* \to I \cup O^*$ defined by: $e(0) = b$, $e(1) = c$ and $e(z) = z$ if $|z| > 1$ as performed on the machine $M_3$ below:

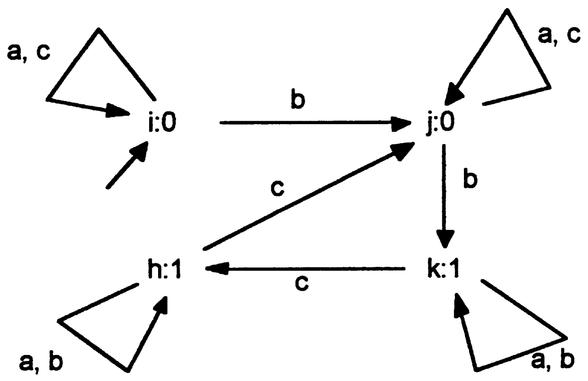


Fig. 2.

  (i) Show that e distinguishes between all states of $M_3$, in the sense that, if s and t are any two distinct states of $M_3$ then the outcome of e when performed on $M_3$ in state s differs from the outcome when performed on $M_3$ in state t.

[7 marks]

  (ii) Show that any *reduced* experiment (single input word) which distinguishes between all states of $M_3$ must have a length of at least 2. Does this show that reduced experiments are less powerful than full experiments? Give reasons.

[7 marks]

  (iii) Find a *reduced* experiment which distinguishes between all states of $M_3$.

[3 marks]

3.1.2. Example 2. Read the following article and attempt the questions at the end of the article.

**Developmental languages**

Most biological organisms that are formed from more than one cell exhibit patterns of growth and development that have intrigued scientists for centuries. If we consider the fact that *all* organisms (except some specialised viruses) start as single cells (in many cases a fertilised egg) and develop by dividing to form two new cells, perhaps with a specific structure and function, over and over again, we see that the modelling of this process could lead to great insights into developmental biology with many opportunities to try and understand why the system sometimes doesn't work and individual organisms fail to develop correctly, sometimes causing life-threatening congenital malformations and malfunctions.

Suppose that we consider each individual cell in an organism as a state machine in a given internal state. Suppose that initially we have a single such cell represented by a circle in a state x say. Let us, further, assume that a beneficial input a is applied to the cell and this causes the cell to split into two cells, one in state x and the other in state y. Further inputs applied to the cell in state x cause further divisions and if the beneficial inputs continue we will see the organism grow as a one dimensional structure as illustrated:

ⓧ $\xrightarrow{\text{input a}}$ ⓧ ⓨ $\xrightarrow{\text{input a}}$ ⓧ ⓨ ⓨ $\xrightarrow{\text{input a}}$ ⓧ ⓨ ⓨ ⓨ $\longrightarrow$

This might be a model of growth of a very simple linear organism; some simple plants behave like this.

We could represent this in a simple abstract algebraic form by regarding the state of the organism at a stage of its development as a string of the form:

$$x,\ xy,\ xyy,\ xyyy, \ldots$$

and the development *rule* as being a substitution rule of the form:

$$x \Rightarrow xy$$

with x as a starting axiom or initial situation. This is very like a grammar and the process of constructing a language. Here we have a simple example of a mechanism that generates words that represent developing organisms as words in a language, a so-called *development language*. The original idea was introduced by A. Lindenmayer and has been extensively studied since then; they are often called L-systems in Lindenmayer's honour. Many complex types of developmental languages have been constructed to simulate the development of a considerable number of biological organisms ranging from seaweeds to flowering plants, to the patterns on the shells of some animals and so on. We now need to formalise some definitions.

**L-schemes and L-systems**

*Definition 1.* Let V be a finite alphabet; consider a finite subset:

$D \subseteq V \times V^*$, we call the pair (V, D) an *L-scheme*.

For each $(a, \alpha) \in D$ we write $a \Rightarrow \alpha$. Each $a \Rightarrow \alpha \in D$ can be regarded as a *development rule* and will form the basis of the method for transforming organisms from one stage of development to another.

If $w_1, w_2, \ldots, w_n \in V^*$ and there exists a sequence of development rules that allow the following:

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_{n+1}$$

we describe this by writing:

$$w_1 \Rightarrow^n w_{n+1}$$

and saying that $w_1$ generates $w_{n+1}$ in n steps.

We write $w \Rightarrow^* w'$ to indicate that there exists a positive integer n and a word $w_{n+1}$ such that $w' = w_{n+1}$ and $w_1 \Rightarrow^n w_{n+1}$. We say that *w generates w′*.

*Example 1.* In the simple situation described above we let:

$$V = \{x, y\} \text{ and}$$

$$D = \{(x, xy)\} \text{ i.e. } \{x \Rightarrow xy\}.$$

Then $x \Rightarrow xy \Rightarrow xyy \Rightarrow xyyy \Rightarrow xyyyy \Rightarrow xyyyy$, i.e. $x \Rightarrow^5 xyyyy$ and $x \Rightarrow^* xyyyy$.

*Definition 2.* An L-system is a triple $G = (V, D, x)$ where $(V, D)$ is an L-scheme and $x \in V^*$. The word x is called the *axiom* of G.

Given an L-system we can define the set of all words generated by G to be the set $L(G) = \{w \in V^* | x \Rightarrow^* w\}$

A language $L \subseteq V^*$ is called a *developmental language* if $L = L(G)$ for some L-system G.

*Theorem 1.* Let $(V, D)$ be an L-scheme and consider words $x_1, x_2, y_1, y_2$ in $V^*$, such that $x_1 \Rightarrow^n y_1$ and $x_2 \Rightarrow^m y_2$ for some n, m then $x_1x_2 \Rightarrow^{n+m} y_1y_2$

*Proof.* If $x_1 \Rightarrow^n y_1$ then we can find $w_1, \ldots, w_{n+1} \in V^*$ such that:

$$x_1 = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_{n+1} = y_1.$$

Similarly we can find $z_1, \ldots, z_{m+1} \in V^*$ such that:

$$x_2 = z_1 \Rightarrow z_2 \Rightarrow \cdots \Rightarrow z_{m+1} = y_2.$$

Now $x_1x_2 \Rightarrow w_2x_2 \Rightarrow \cdots \Rightarrow w_{n+1}x_2 = y_1x_2 = y_1z_1 \Rightarrow y_1z_2 \Rightarrow \cdots \Rightarrow y_1z_{m+1} = y_1y_2$ as required.

*Corollary.* If $x_1 \Rightarrow^* y_1$ and $x_2 \Rightarrow^* y_2$ then $x_1x_2 \Rightarrow^* y_1y_2$.

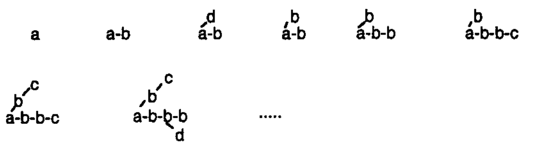
There is a hierarchy of developmental languages determined by the specific types of development rules used in the L-systems defining the language.

*Example 2.* Let $V = \{a, b, c, d, (, )\}$ and $D = \{a \Rightarrow ab, b \Rightarrow (d)b, c \Rightarrow (d)b, d \Rightarrow b, b \Rightarrow bb, b \Rightarrow bc, b \Rightarrow bd\}$. The L-system $G = (V, D, a)$ generates strings including:

$$a \Rightarrow ab \Rightarrow a(d)b \Rightarrow a(b)b \Rightarrow a(b)bb \Rightarrow a(b)bbc \Rightarrow a(bc)bbc$$
$$\Rightarrow a(bc)bb(d)b \Rightarrow \cdots$$

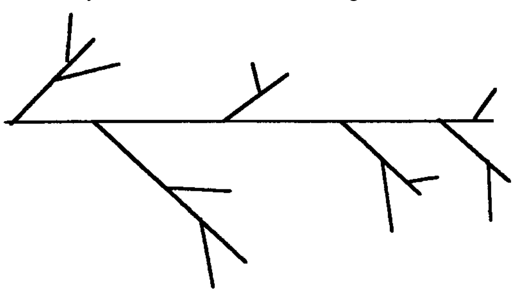
We can interpret the brackets in the following way; a rule such as $b \Rightarrow (d)b$ creates a left branch off the main stem with a cell of state d and the rule $c \Rightarrow (d)b$ is a right branch off the main stem with a cell in state d.

This would result in a sequence depicted in Fig. 3



(3a)

Ultimately structures like the following can be formed:



(3b)

Fig. 3.

which have a passing resemblance to classes of simple aquatic plants that exhibit branching growth structure.

*Example 3.* Consider the following L-system:

$$V = \{S, a, b, c, d, e, f, g, h, i, j, k, l, m, 0, 1, 2\}$$

$$D = \{S \Rightarrow ab, a \Rightarrow dg, b \Rightarrow e0, c \Rightarrow 22, d \Rightarrow 0e, e \Rightarrow cf, f \Rightarrow lc,$$
$$g \Rightarrow hb, h \Rightarrow di, i \Rightarrow jk, j \Rightarrow ml, k \Rightarrow c0, m \Rightarrow 0c,$$
$$0 \Rightarrow 0, 1 \Rightarrow 1, 2 \Rightarrow 2\}.$$

The L-system $(V, D, S)$ can produce the following derivations:

$$S \Rightarrow ab \Rightarrow dge0 \Rightarrow 0ehbcf0 \Rightarrow 0cfdie0221c0$$
$$\Rightarrow 0221c0ejkcf0221220 \Rightarrow 0221220cfm1c0221220$$
$$\Rightarrow 0221220221c0c1220221220221220$$
$$\Rightarrow 0221220221220221220221220221220$$

which can be interpreted as follows:

the system is the stages in the development of a leaf which has mature cell types:

0—cells at notches between lobes,
1—cells at the tips of lobes,
2—non-dividing cells on the margins.

The final string representing the leaf:

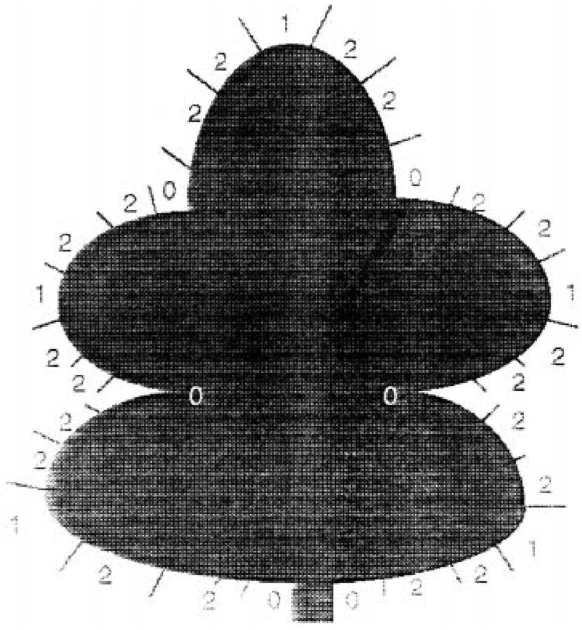


Fig. 4(a).

Note that the final string represents an element of the language which cannot be further processed. Since we are primarily interested in biological models this naturally leads onto the following idea.

*Definition 3.* Let $(V, D)$ be an L-scheme. We define the *adult language* of the scheme to be the set $\mathbf{A} = \{x \in V^* | x \Rightarrow^* x\}$, thus an adult language is one that is ‘stable’ under the influence of the development rules. For an L-system $G = (V, D, x)$ we define the *adult language* of G to be the adult language of the underlying L-scheme and write $\mathbf{A}(G) = \mathbf{A}$.

An organism which has fully developed may suffer damage due to environmental causes but it may be able to repair or ‘regenerate’ part of itself. Thus our leaf might get damaged during its development but the development rules may still allow it to reach a mature form. Thus, for example, if at the stage ‘0ehbcf0’ it loses the embryonic cell ‘h’ at this stage it can still develop into a deformed, adult shape ‘022122221220221220’.

Now suppose that we allow the deletion of any (finite) substring of symbols at any time during the generation of a string in an L-system. Then, after a period of growth and damage, the damage ceases and the string is allowed to develop normally.

What sort of things can emerge under this type of situation? The set of strings that can be generated after damage of this type is called the *regenerative adult language* of the L system. Let $\Delta$ represent the operator that takes any string in $V^*$ and constructs all possible substrings of this string, thus $\Delta: V^* \to \mathscr{P}(V^*)$, where $\mathscr{P}$ is the power set operator. Now consider the process of deleting substrings from a string, so that if $\alpha \in V^*$ then the set of strings obtained from $\alpha$ by deleting substrings is written:

$$\delta(\alpha) = \{y \in V^* | y = wz \text{ and } \alpha = wvz \text{ and } v \in \Delta(\alpha) \text{ and } w, z \in V^*\}$$

Using these operators it is possible to define, formally, the regenerative adult language of an L-system.

### Questions

1. (a) Contrast the difference between the generation of a language using a phrase structure grammar (as considered in the lecture course) and the generation of a developmental language.
[5 marks]
(b) Investigate a possible converse to Theorem 1. (Prove it if you believe it to be true, exhibit a counter-example otherwise.)
[6 marks]
(c) Consider the L-system defining the leaf structure in Example 3. Look at other possible damage scenarios that might occur during development and the resultant adult form, if any, that can arise. Explore this issue.
[7 marks]
(d) Formulate a mathematical definition of the regenerative adult language of an L-system.
[3 marks]
(e) If $G = (V, D, x)$ is an L-system with adult language $\mathbf{A}(G)$ show that we can effectively replace $G$ by an L-system $G' = (V', D', x')$ such that:

$$\mathbf{A}(G) = \mathbf{A}(G')$$

and

$$x' \text{ is a string of length } 1.$$

[5 marks]
(f) Can you identify any possible applications of developmental languages within Computer Science? Discuss any ideas you have and how useful you think they may be.
[6 marks]

## *3.2. Final (third) year comprehension questions*

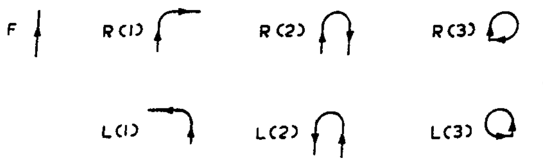
3.2.1. [Only slightly based on: Siromoney G. *et al.*, Kambi kolam and cycle grammars, in *A perspective in theoretical computer science*, R. Narasimhan (ed), World Scientific (1989)]

### Kambi kolam, formal languages, the Universe, and all that!

*Kolam* is a traditional art practiced extensively throughout the southern part of India, where it is used for decorating courtyards. The patterns involved can be very complex, and so an experiment was conducted to find out how such complicated patterns could be stored in, and retrieved from, human memory so easily. The researchers discovered that kolam practitioners describe and draw the designs in terms of 'moves' such as 'go forward', 'take a right turn', and so on, which are highly reminiscent of the commands used in languages like LOGO for the control of a 'turtle'. By treating each kind of move as a terminal symbol, the authors were able to show the relationship between this traditional art form, and formal language theory.

A turtle is basically a drawing device; think of it as something which moves across a piece of paper, following the commands you give it. It carries a pen, which can either be raised, or else in contact with the paper.

A *kambi* is a closed curve, with or without loops, constructed using the turtle instructions given in Fig. 4, as listed.



(4b)

| | |
|---|---|
| F | forward one unit |
| R1 | turn right 90 while drawing |
| R2 | turn right 180 while drawing |
| R3 | turn right 360 while drawing |
| L1 | turn left 90 while drawing |
| L2 | turn left 180 while drawing |
| L3 | turn left 360 while drawing |

Fig. 4(b).

Pictorially, the various moves produce designs as in Fig. 5.

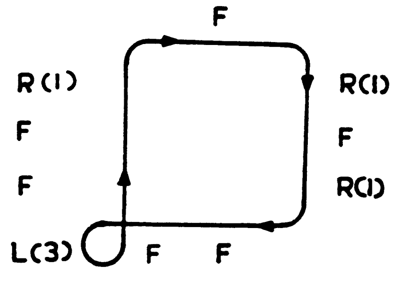


Fig. 5.

In drawing a *kambi*, we insist that the pen is held down on the paper at all times, and that the turtle eventually ends up where it started, and facing in the same direction. A *kolam* design contains one or more kambi's.

We can relate *kambi kolam* designs to formal language theory. For example, consider the three 'anklets of Krishna' given in Fig. 6.

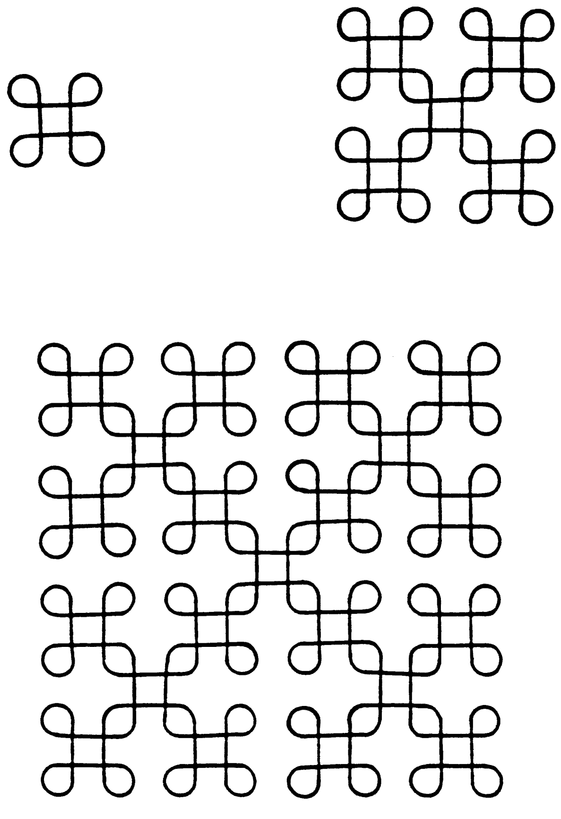


Fig. 6.

These are obviously related; but how? The strings of commands that would have to be issued to the turtle are all generated from the 'grammar':

$$\begin{aligned} F &\Rightarrow F \\ R1 &\Rightarrow R1.F.R3.F.R1 \\ R3 &\Rightarrow R1.F.R3.F.R3.F.R3.F.R1 \end{aligned}$$

where we successively transform the initial 'anklet':
R1.F.R3.F.R3.F.R3.F.R1

**Questions**

(a) To what extent is it reasonable to suggest that *kolam* practitioners actually remember their designs in terms of grammars? What other mechanisms might they be using? [5 marks]

(b) It's also possible to see 'term re-writing' going on in other pictorial settings. For example, suppose that the following commands are available to our turtle:

| | |
|---|---|
| F | forward one unit |
| R | turn right through 60°. |

Start off with the initial figure (an equilateral triangle)

$$(FR^2)^3$$

Consider the production rule (we'll call it '*expand*') given by:

*expand*(*string*) is obtained from *string* by replacing *simultaneously* every occurrence of F in *string* according to the rule

$$F \Rightarrow F.R^5.F.R^2.F.R^5.F$$

Draw the shapes given by:

(i) F
(ii) *expand*(F)
(iii) *expand*(*expand*(F))

[5 marks]

(c) Give an example of a 'grammar' capable of generating all n × n grids, as in Fig. 7.

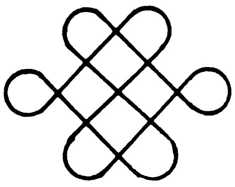


Fig. 7.

Justify your claim that your answer is correct. [8 marks]

(d) It has been observed by many authors that Nature seems to like recursive patterns (e.g. snowflake designs, structures etc). To what extent can this be taken as evidence that (some of) the laws of physics may best be expressed in terms of 'construction grammars' of the kind discussed in this question? [9 marks]

(e) It is important in all disciplines to know the extent to which one's ideas are valid. There are many examples of ideas from one discipline being applied by practitioners of another (e.g. the statement that 'the brain is like a computer', or 'Nature obeys formal language theory'). In your opinion, when is it valid to apply concepts which originate in one discipline, to problems in another, and when is it not valid to do so? How would you identify the 'boundary' between when the application of a concept is, or is not, valid? [13 marks]

3.2.2. Example 2. Re-read the following passage and answer the questions at the end of it.

The following is based on a paper entitled *A Process Algebra with Multiple Clocks* by Henrik Reif Andersen and Michael Mendler of the Department of Computer Science, Technical University of Denmark. The paper is a departmental technical report numbered ID-TR: 1993–122.

**Incorporating time into process calculi**

It is not clear how useful process calculi such as CCS are in specifying real-time systems, because they lack any explicit mention of time. The usual approach to real-time specification involves a fixed, measurable and global notion of time. Here a different approach is used, where real-time constraints are represented using *clocks* which enforce broadcast synchronization between processes. These clocks are taken to be atomic, i.e. without further structure, just as the actions of CCS are atomic.

The resulting concept of time is *abstract*, *qualitative*, and *local*. It is *abstract* because it does not enforce any particular way of implementing clocks; it is *qualitative* because the absolute occurrence time or duration of actions is not constrained, only their sequencing with respect to the clocks; and it is *local* because independent clocks can be used within different subprocesses.

The calculus presented is called PMC, and is an extension of CCS by multiple clocks, with associated timeout and clock ignore operators. The semantics of PMC is based upon labelled transition systems with separate action and clock transitions.

*Syntax.* The *processes* of PMC are generated by the following grammar:

$$P ::= 0 | A | \alpha.P | P_1 + P | P_1 | P_2 | P \backslash L | [P_1]\sigma(P_2) | P \uparrow \sigma$$

where $A$ is an agent constant, $\alpha$ is an action (a name, co-name or the silent action $\tau$), $L$ is a set of labels (names or co-names) and $\sigma$ is a member of the finite set $\Sigma$ of clocks.

*Informal semantics.* The meanings of the nil process 0, agent constants $A$, the prefixed process $\alpha.P$, the sum $P_1 + P_2$, the parallel composition $P_1 | P_2$ and the restriction $P \backslash L$ are the same as in CCS. $[P_1]\sigma(P_2)$ represents a *timeout*, in that it behaves like $P_1$ with respect to actions or clocks distinct from $\sigma$, but is transformed into $P_2$ if a clock tick $\sigma$ occurs. $P \uparrow \sigma$ behaves as the process $P$, but always permits a tick of the clock $\sigma$ without changing state, and thus we call $\uparrow$ the *ignore* operator.

*Operational semantics.* Formally, the semantics of PMC is given by two sets of transition rules, the *action* rules and the *clock progress* rules. Action rules:

$$\alpha.P \xrightarrow{\alpha} P \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P | Q \xrightarrow{\tau} P' | Q'} \quad \frac{P \xrightarrow{\alpha} P'}{P | Q \xrightarrow{\alpha} P' | Q} \quad \frac{Q \xrightarrow{\alpha} Q'}{P | Q \xrightarrow{\alpha} P | Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \backslash L \xrightarrow{\alpha} P' \backslash L} (\alpha, \bar{\alpha} \notin L) \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} (A \overset{def}{=} P)$$

$$\frac{P \xrightarrow{\alpha} P'}{[P]\sigma(Q) \xrightarrow{\alpha} P'} \quad \frac{P \xrightarrow{\alpha} P'}{P \uparrow \sigma \xrightarrow{\alpha} P' \uparrow \sigma}$$

Clock progress rules:

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P + Q \xrightarrow{\sigma} P' + Q'} \quad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P | Q \xrightarrow{\sigma} P' | Q'}$$

$$\frac{P \xrightarrow{\sigma} P'}{P \backslash L \xrightarrow{\sigma} P' \backslash L} \quad \frac{P \xrightarrow{\sigma} P'}{A \xrightarrow{\sigma} P'} (A \overset{def}{=} P)$$

$$[P]\sigma(Q) \xrightarrow{\sigma} Q \quad \frac{P \xrightarrow{\sigma'} P'}{[P]\sigma(Q) \xrightarrow{\sigma'} P'} (\sigma \neq \sigma)$$

$$P \uparrow \sigma \xrightarrow{\sigma} P \uparrow \sigma \quad \frac{P \xrightarrow{\sigma'} P'}{P \uparrow \sigma \xrightarrow{\sigma'} P' \uparrow \sigma} (\sigma' \neq \sigma)$$

Note that clocks and actions are distinct, the action rules say nothing about clock transitions and the clock rules say nothing about action transitions.

One important special case of a timeout is the *wait* process $\sigma.P$ which is defined by $\sigma.P \overset{def}{=} [0]\sigma(P)$ and which waits for the clock $\sigma$ to tick before proceeding, stopping all other clocks in the meantime.

*Modelling synchronous systems in PMC.* Let us consider a restricted version of PMC in which there is a single clock $\sigma$. A single-clock synchronous system is built up from a number of functional blocks (such as $\vee$- and $\wedge$-gates, etc.) using a number of registers, all triggered by the same global clock signal. We may model single-clock synchronous systems in the restricted calculus, as the following circuit example shows.

The functional operation of the circuit is easy to specify: the initial output $y$ is the maximum of the value initially stored in the register $\Delta$ and the input $x$, and stays at this value unless and until the input $x$ goes high, whereupon $y$ goes high and remains high. Operationally, this is achieved thus: at the $t$-th clock tick the register $\Delta$ reads its input $v$ and writes its value to its output $w$. This value, together with the new value input on $x$, is processed through the $\vee$-gate and the resulting value branches onto the output $y$ and also onto $v$ where it is ready to be read by the register on the next clock tick (Fig. 8).

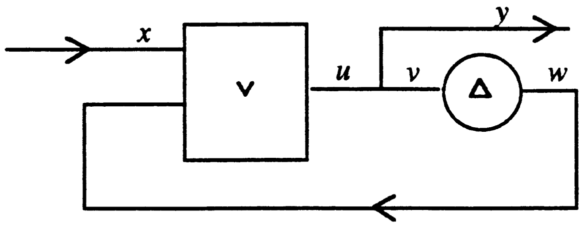


Fig. 8.

If $\sigma$ models the global clock driving this system, we may specify the functional block (the $\vee$-gate) and the register by the PMC processes:

$$Or \overset{def}{=} v(b).x(c).\bar{u}(b \vee c).\sigma.Or$$

$$Register \overset{def}{=} v(b).\sigma.\bar{w}(b).Register$$

We may encode the value-passing as in CCS by defining $x(b).P$ to be $x_0.P(0) + x_1.P(1)$ and $\bar{x}(b).P$ to be $\bar{x}_b.P$, as we are dealing with boolean signals.

We still need to implement the forking of the signal $u$; we do this by specifying an independent synchronous component *Fork* which has one input $u$ and copies it once to its two outputs $v$ and $y$ before every clock tick. The order in which the outputs are delivered is unspecified, so we define:

$$Fork \overset{def}{=} u(b).(\bar{v}(b).\bar{y}(b).\sigma.Fork + \bar{y}(b).\bar{v}(b).\sigma.Fork)$$

The whole circuit is obtained by composing the three processes together and restricting the resultant process on the set of ports $L \overset{def}{=} \{u, v, w\}$ which represent internal wires:

$$Circuit(b) \overset{def}{=} (Or \,|\, \bar{w}(b).Register \,|\, Fork)\backslash L.$$

The only complication is the need to load an initial value $\bar{w}(b)$ into the register.

**Questions**

(a) Use the operational rules of PMC to give derivation trees for the two transitions below:

$$[0]\sigma(P)|[a.[0]\sigma(Q)]\sigma(R) \xrightarrow{a} [0]\sigma(P)|[0]\sigma(Q) \xrightarrow{\sigma} P|Q$$

[4 marks]

(b) Show that the only possible transition of $\sigma.P$ is $\sigma.P \xrightarrow{\sigma} P$ and hence that the only possible *clock* transition of $(\sigma.P)|Q$ is $(\sigma.P)|Q \xrightarrow{\sigma} P|Q'$ where $Q \xrightarrow{\sigma} Q'$, and the only possible *clock* transition of $(\sigma.P) + Q$ is $(\sigma.P) + Q \xrightarrow{\sigma} P + Q'$ where $Q \xrightarrow{\sigma} Q'$.

[6 marks]

(c) (i) Suppose the first input to the synchronous circuit and the initial value stored in the register are both 0. Give a sequence of transitions which take the circuit from its initial state to the state where it is again about to write a value on the $w$ port, i.e. give a non-empty sequence $t$ of actions for which

$$Circuit(0) \xrightarrow{t} Circuit(0).$$

You should give all the intermediate derivatives of *Circuit*(0). [Hint: there are six transitions; the first five correspond to the passing of values along wires and the last is a clock tick.]

[8 marks]

(ii) Is there an alternative sequence of actions to the sequence you have given in part (i) above? If so, give one. You need not specify the intermediate derivatives.

[2 marks]

(iii) Suppose you have designed a process $P$ to consume the outputs of the circuit on the $\bar{y}$ port. How might you use the ignore operator $\uparrow$ to ensure that your process could always synchronise with the clock ticks of *Circuit*?

[4 marks]

(d) Write a few paragraphs discussing the advantages and disadvantages of using PMC to model concurrent systems, both synchronous and asynchronous. You may care to address the following questions:
- What aspects of such systems, if any, can be better modelled in PMC than in CCS?
- What tools would a designer need to use the calculus effectively in the design of a large system?
- What sorts of properties of a system specified in PMC would you expect to be able to prove, and what sort of theory might be needed to enable such proofs?
- How faithfully does PMC model time?

[12 marks]

(e) Which two transition rules of PMC are redundant if there is only one clock?

[4 marks]

## 4. CONCLUSIONS

It might seem that this sort of question would be unpopular with students, this is far from the case, in our experience. One point that is often made is that it is difficult to revise for such a questions apart from having a general familiarity with the subject area. Even when articles are distributed before the examination it doesn't seem to cause any problems. The performance of the students has been rewarding so far and we are regularly surprised at how sophisticated their answers are. The participants also understand the benefits in the sense that it is helping them to develop skills which might only be developed in an *ad hoc* manner and which are vital for a successful and rewarding professional life in the industry. Some of the skills needed for updating and retraining are being developed in this process.

The original idea for this activity arose while I was involved in the development of a novel school mathematics syllabus (Further mathematics 'A' Level under the auspices of the Northern Ireland Examinations Board), see [1–11]. Subsequently the idea has been introduced into the SMP syllabus [12]. The application of comprehension to computing was a short step.

I would like to thank my colleagues Matt Fairtlough and Mike Stannett who collaborated with me in this work.

## REFERENCES

1. M. Holcombe, *Bulletin IMA* **18** (1982) pp. 12–17.
2. M. Fitzpatrick and S. K. Houston, Mathematical modelling in Further Mathematics, in J. S. Berry *et al.* (eds), *Mathematical Modelling Courses*, Ellis Horwood, London (1987).
3. S. K. Houston, *Teaching Mathematics and its Applications*, **8** (1989) pp. 115–122.
4. B. Greer and R. J. McCartney, in *New Directions in Mathematical Education*, in B. Greer and G. Mulhern (eds). Routledge, London (1989).
5. A. R. Nicholoson, *Teaching Mathematics and its Applications*, **8** (1989) pp. 184–188.
6. R. J. McCartney, *Teaching Mathematics and its Applications*, **9** (1990) pp. 6–14.
7. M. Fitzpatrick and G. Greer, *Teaching Mathematics and its Applications*, **9** (1990) pp. 150–158.
8. S. K. Houston, in *Teaching Mathematical Modelling and its Applications*, M. Niss *et al.* (eds), Ellis Horwood, Chichester (1991).
9. A. R. Nicholson, *Int. Jour. Math. Educ. Sci. Technol.*, **22** (1992) pp. 45–49.
10. S. K. Houston, *Teaching Mathematics and its Applications*, **12** (1993) pp. 60–73.
11. S. K. Houston, *Teaching Mathematics and its Applications*, **12** (1993) pp. 113–120.
12. S. Dolan, *Teaching Mathematics and its Applications*, **7** (1988) pp. 1–10.