EXCELLENCE EXEMPLAR 2022

3

NEW ZEALAND QUALIFICATIONS AUTHORITY
MANA TOHU MĀTAURANGA O AOTEAROA

QUALIFY FOR THE FUTURE WORLD
KIA NOHO TAKATŪ KI TŌ ĀMUA AO!

COMMON ASSESSMENT TASK

Level 3 Digital Technologies and Hangarau Matihiko 2022

91908 Analyse an area of computer science

Credits: Three

Achievement Criteria		
Achievement	**Achievement with Merit**	**Achievement with Excellence**
Analyse an area of computer science.	Analyse, in depth, an area of computer science.	Critically analyse an area of computer science.

Type your School Code and 9-digit National Student Number (NSN) into the space below. (If your NSN has 10 digits, omit the leading zero.) It should look like "`123-123456789-91908`".

[redacted]-91908

There are three questions in this document. **Choose ONE question to answer.**

Make sure you have the PDF of Resource Booklet 91908R. This contains resources for Questions Two and Three.

You should aim to write **800–1500 words** in total.

Your answers should be presented in 12pt Times New Roman font, within the expanding text boxes, and may include only information you produce during this assessment session. Internet access is not permitted.

Save your finished work as a PDF file with the file name used in the header at the top of this page ("`SchoolCode-YourNSN-91908.pdf`").

By saving your work at the end of the examination, you are declaring that this work is your own. NZQA may sample your work to ensure this is the case.

INSTRUCTIONS

There are three questions in this assessment, on the topics of:

- Formal languages (page 3)
- Computer graphics (page 13)
- Computer vision (page 19).

Choose ONE question to answer.

Questions Two and Three require you to refer to the separate resource booklet.

Read all parts of your chosen question before you begin.

EITHER: **QUESTION ONE: Formal languages**

(a) Deterministic finite automata

A deterministic finite automaton (DFA) can be described by a five-element tuple (Q, Σ, δ, q_0, F) where:

- Q is a finite set of states
- Σ (sigma) is a finite, non-empty input alphabet
- δ (delta) is a series of transition functions
- q_0 is the starting state
- F is the set of accepting states.

Figure 1 shows a deterministic finite automaton.

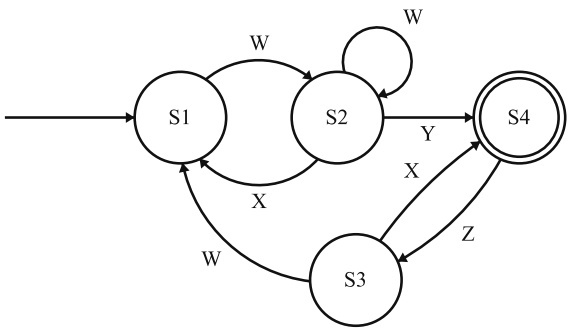


Figure 1.

(i) Complete the following for the DFA in Figure 1.

Q =	S1, S2, S3, S4
Σ =	W, X, Y, Z

δ =

Current state	Input symbol	Next state
S1	W	S2
S2	W	S2
S2	X	S1
S2	Y	S4
S3	W	S1
S3	X	S4
S4	Z	S3

q_0 =	S1
F =	S4

(ii) Which of the following strings are not accepted by the DFA in Figure 1?

(1) WXWYZWWY
(2) WXWYZX
(3) WWWWWY
(4) WWWWYZ

4

(iii) Explain how you came to this conclusion.

(1) travels from S1 > S2 > S1 > S2 > **S4** > S3 > S1 > S2 > **S4** (2) travels from S1 > S2 > S1 > S2 > **S4** > S3 > **S4** (3) travels from S1 > S2 > S2 > S2 > S2 > S2 > **S4** (4) travels from S1 > S2 > S2 > S2 > S2 > **S4** > S3 Of these, (4) does not end on an accepting state.

(b) The following table shows syntax that is sometimes used for regular expressions:

Expression	Description
`[a-z]`	Any single character in the range **A**–**Z**
`[0-9]`	Any single number in the range **0**–**9**
`+`	One or more repetitions of the preceding element
`*`	Zero or more repetitions of the preceding element
`?`	Zero or one of the preceding element
`.`	Any character
`\d`	Any digit
`[abc]`	Only **a**, **b**, or **c**
`[^abc]`	Not **a**, **b**, or **c**
`\w`	Any alphanumeric character
`\W`	Any non-alphanumeric character

Consider the regular expression **`[CFR]an`**

(i) Which words from the following list does the expression describe?

`Can, Dan, Fan, Man, Pan, Ran`

Can, Fan, Ran

(ii) Explain how you came to this conclusion.

[CFR] matches exactly one of "C", "F", or "R", and *an* matches "an" literally. As the pattern is *[CFR]an* the pattern can only match **C**an **F**an or **R**an

(iii) What regular expression could you write that would find all of the words in the following list?

```
babble, bebble, bibble, bobble, bubble
```

b[aeiou]bble

(iv) Explain how you came to this conclusion.

Every word is of the form b_bble, with _ representing any vowel aeiou. One can use the square brackets to match exactly one of these vowels *[aeiou]*, and the literal characters for the rest of the text.

(c) Figure 2 shows the production rules of a context-free grammar, and Figure 3 shows productions which have been applied to non-terminals from left to right in this order.

$E \rightarrow N$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow -E$

$E \rightarrow (E)$

$N \rightarrow 0–9$

Figure 2.

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow E + E$

$E \rightarrow N$

$E \rightarrow N$

$E \rightarrow N$

$N \rightarrow 7$

$N \rightarrow 3$

$N \rightarrow 2$

Figure 3.

(i) What are the non-terminal symbols used in the production rules in Figure 2?

E, N

(ii) What are the terminal symbols used in the production rules in Figure 2?

+, *, -, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

(iii) Give the expression built by the sequence in Figure 3, showing the result from applying each production.

E, E*E, (E)*E, (E+E)*E, (N+E)*E, (N+N)*E, (N+N)*N, (7+N)*N, (7+3)*N, (7+3)*2

(iv) Explain what it would mean if there was no way to generate a particular expression using these productions. You may wish to illustrate this with an example that cannot be generated by the above grammar.

The expression E-E (for example, 2-1) cannot be generated using these productions.

This is because there is a production rule *E->E+E*, and *E->-E*, but no production *E->E-E*.

Therefore, you can generate the expression 2+-1 as

E->E+E (E+E)

E->N (N+E)

E->-E (N+-E)

E->N (N+-N)

N->2 (2+-N)

N->1 (2+-1)

But as there is no way to get two expressions with *only* a subtraction symbol between them, this expression can be generated.

This means that, if you provided such an expression to a parser implementing these production rules, it would be considered invalid.

Here's an example attempting to parse 2-1:

2-1

2-N (N->0-9)

N-N (N->0-9)

N-E (E->N)

EE (E->-E)

There is no production for EE.

(d) If a language can be recognised by a finite state machine (FSM) it is said to be a regular language, and if there is a finite state machine that recognises a language, then that language must be a regular language.

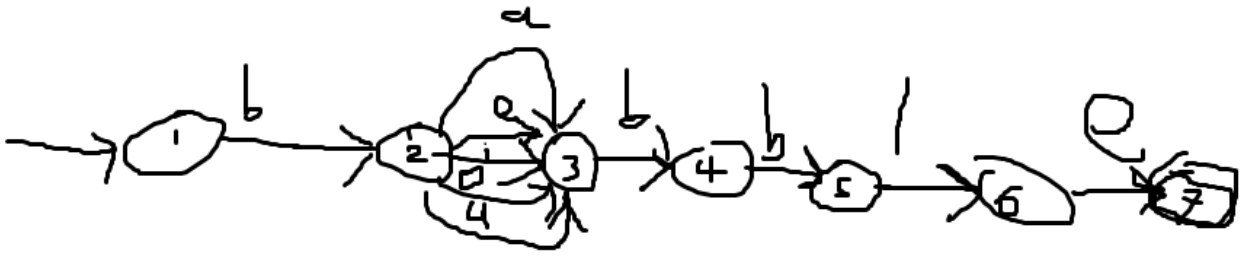
Using examples to support your reasoning, explain what this statement means.

You might consider:

- the relative capabilities of formal languages that you have studied
- the limits of what they can do.

This statement describes regular languages and finite state machines as equivalent, that is, that you can represent a regular language as a finite state machine and vice-versa.

If you represent a regular expression to match a pattern, for example, *b[aeiou]bble*, as an FSM, the input alphabet would be defined as b,a,e,i,o,u,l, and the FSM would be as follows:



(artist's depiction of a FSM of the RegEx *b[aeiou]bble*)

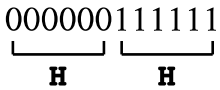
Generally speaking, you can encode a literal letter as a transition from one state to the next, you can encode one of multiple letters (square bracket []) as multiple transitions to the same state, you can encode a fixed number of a pattern as a chain of transitions through a fixed number of consecutive states, a variable number (* and +) as a transition directed into the same state, or for more complex patterns, a loop of states, and so on.

Because of FSM and regular languages' equivalence, some restrictions can be inferred. A regular language, as an FSM, has no memory, no data other than its current state within the FSM. This means you cannot keep track of recursive instances of patterns, nor can you match nested brackets. Most forms of RegEx do not allow for lookahead or lookbehind, where you can match a pattern in text ahead of or behind the current point in the match, as that extension does not map cleanly to FSM. This restriction can also constrain the type of patterns you can make.

Essentially, regular languages (such as regular expressions and FSM) are restricted in terms of complexity, due to their design, which can make them inappropriate for certain tasks. Context-free grammars (CFGs) are commonly used in these cases, as they have far less restrictions. There is always a trade off however, as CFGs typically run slower and use far more memory.

(e) It is possible to write a simple program that can recognise any string in the language **HH**, where a string is made up of a number of 0s followed by the same number of 1s.

Here is an example string, where six 0s are followed by six 1s:



Is it possible to create a finite state automata (FSA) for this language? Explain your answer by identifying the key problem related to regular languages (languages that use regular expressions and finite state automata). How do we get around this problem?

Regular languages, if you consider them as an FSA, clearly do not have any concept of data, and so have no way to keep track of the number of 0's encountered. This means it's nearly impossible to ensure the number of 0's and 1's is equal, as the only way to keep track of the number of 0's is by having each state account for a different number of 0's counted. This workaround requires you to choose an upper limit on the number of 0/1s to accept, and generate predetermined expressions for them. Without an upper limit, you'd have an infinitely long RegEx or an FSA with infinite states. In RegEx:

(0{1}1{1})|(0{2}1{2})|(0{3}1{3})|…

FSA:

This is a very bad solution, and would be far inferior to just writing code to count the number, but it does work, at least with this restriction applied.

Creating a game

Your friend is creating a simple computer game, but they need some help. They want to program the enemy sprites to chase the player's sprite, but aren't sure how to do so.

You have just finished a computer science topic on formal languages and suggest that an FSM could be a good starting point to solve their problem. You sketch the diagram in Figure 4 to get your friend started.

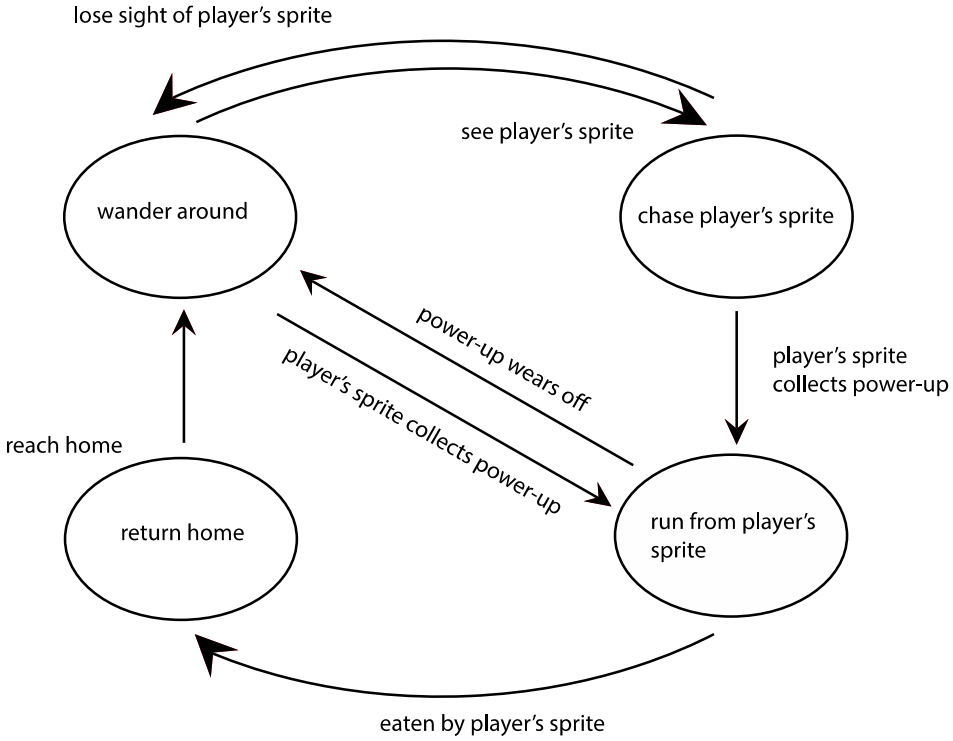


Figure 4. Behaviour of enemy sprite

(f) Explain how your understanding of FSMs and formal languages could be used to help get your friend started. In your answer you may also discuss where else formal languages are used and how they affect people.

As finite state machine encodes a finite set of states, and transitions between states, it's perfect for modelling a system with predetermined functionality and well-defined changes between functionalities. Because of this, FSM is commonly used in video game AI to model non-playable character behavior, as it can usually be clearly defined into modes of operation (such as different tactics, or different means of movement) and transitions between each mode (typically triggered by actions of the player, or a change in the game's internal state).

FSMs are successfully used to model such behavior in games, for example, in Half-Life.

The power of FSMs is in its simplicity, as it can easily be understood and edited visually, allowing for powerful changes to be applied, even by artists and designers with less programming experience.

FSMs can also be applied in user interfaces, for example. Clicking on a button and being taken to another page is an example of a transition, and each page of the UI is its own state. While most

UI take a more complex approach, to enable tracking the history of pages visited in order to support history/back functionality, a FSM is a perfectly good model for simple UI. Hierarchical FSM is an extension that better facilitates the organization and structuring of large and complex FSM for these purposes.

Regular expressions are another example of a valuable use of formal languages. They are commonly used for pattern-matching in programming. For example, on an automated system for an online chat community, I wrote RegEx patterns to detect specific phrases and combinations of keywords in discussions of illegal activity, and report them to the appropriate administrators. RegEx can also be applied to find many kinds of patterns in text, such as email addresses, mailing addresses, and IP addresses. Essentially, it's the swiss-army-knife of pattern-matching in text. Using capturing groups, you can even extract specific data from matches, and substitute matches for other things, useful for refactoring variable names in large codebases. This is how the refactoring feature is implemented in most IDEs (integrated development environments).

CFGs (context-free grammars) are used for more complex forms of grammar that regular languages cannot encode. For example, they can handle recursion and matching of nested brackets/open/close statements where regular languages cannot. This makes them well suited for languages with control-flow syntax, such as almost every major programming language, which need to match scopes being opened and closed, and recursive use of grammar.

It's worth noting that not all programming languages require a CFG. For example, assembly (or machine code) does not contain any control flow, it doesn't have any recursive syntax, or nested brackets, so most flavors of assembly can be parsed by a regular expression.

Formal languages form the foundation of computer science, as CFGs underpin the grammar of almost all programming languages. FSMs also fulfil an important role in modelling many systems that involve states and transitions, which has a surprising amount of roles in programming. Regular Expressions are a powerful tool found everywhere in programming, almost always the first thing a programmer will reach to when searching for patterns in text.

Essentially, formal languages have had a great impact on the world.

(g) Refer to the FSM in Figure 5.

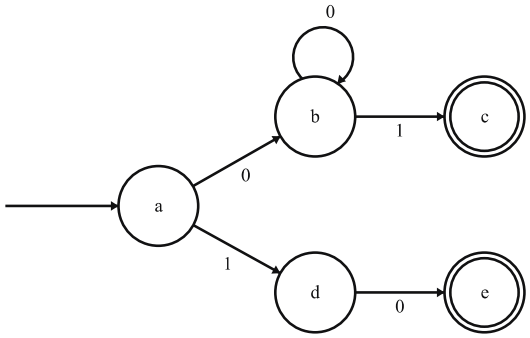


Figure 5.

(i) What does this machine accept?

It accepts two types of strings: “10”, and one or more “0”s followed by a “1”. e.g 10 01 001 0001 etc In terms of a regular expression, *10* or *0+1*

(ii) Explain what happens if the strings `111` or `1010` are applied.

If you consider the string “111”, it must first take the bottom transition from the starting state, as it begins with a “1”. From there, the only valid transition is a “0”, so this string cannot be accepted, as it does not end with exactly one “0”. For the string “1010”, starting with a “1”, again it must take the bottom transition from the starting state, and from there it can take the transition to the accepting state due to the next character “0”. However, there are still the characters “10” unaccounted for. Although the FSM is in the accepting state *c*, this string would not be accepted, because it has remaining characters, and no outward transitions to account for them.

This page has been deliberately left blank.

OR: **QUESTION TWO: Computer graphics**

This question includes references to **Resources A**, **B**, **and C** on pages 2 and 3 of the resource booklet.

(a) (i) What are matrix transformations used for in computer graphics?

(ii) Why are matrix transformations used in computer graphics?

(b) Translation, scaling, and rotation can all be performed on a single shape.

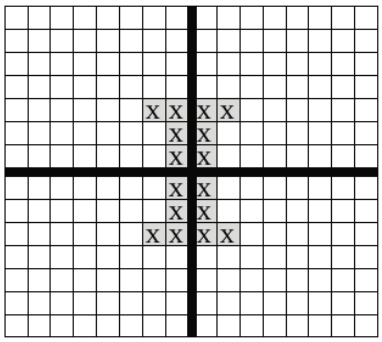
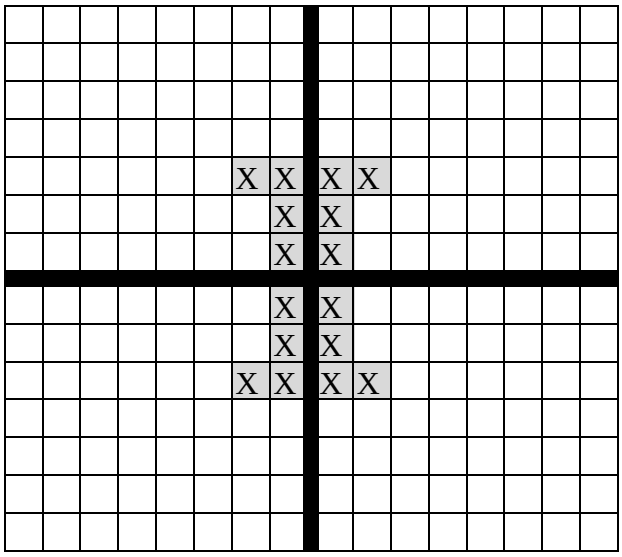


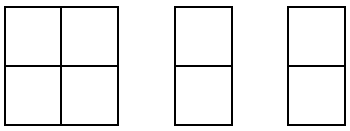
Figure 6.

(i) Using the multiplication matrix $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$, complete a 2D scaling effect on the shape in Figure 6 by filling in the new scaled shape with crosses (X) on the graph below.



(ii) Using the matrix, explain how you knew where to position the point at (2,3) after scaling had been applied.

You may use the boxes below to support your answer.



(c) (i) The mathematical formula for calculating the slope of a line is:

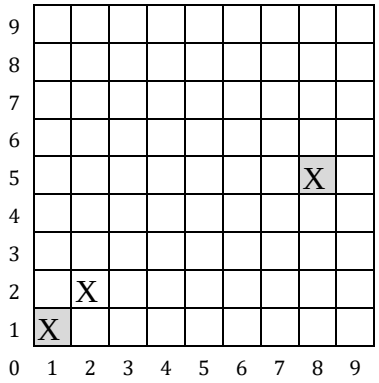
$y = mx + b$, where m is the slope and b is the y intercept.

Explain why this formula works well for drawing lines on paper but does not work well for drawing lines on a computer screen. Refer to **Resources A and B** to support your answer.

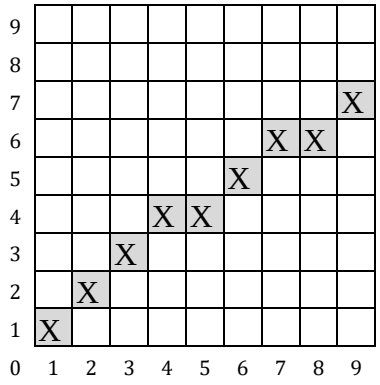
(ii) Using the algorithm in **Resource B**, calculate the points that would be plotted in order to draw a line between (1,1) and (8,5).

Points plotted	P	x coordinate	y coordinate
1		1	1
2	–5	2	2
3			
4			
5			
6			
7			
8	1	8	5

(iii) Fill in the pixels with the points you calculated in (ii). The first pixel has been done for you.



(iv) Below is an example of a line drawn on a screen using the coordinates (1,1) and (9,7).



Explain the concept of anti-aliasing and demonstrate how it could be used to make this line appear smoother by adding crosses (X) in the table above.

3D Computer Graphics

Your friend has just bought a new computer and wants to play the latest high-resolution video games and do intensive 3D modelling. Their computer does not have a dedicated graphics card or GPU and they are complaining that the game lags and looks dull, and that rendering and creating 3D models is taking forever.

(d) (i) How would you explain to your friend what is causing the problem? Refer to specific algorithms and computer science concepts in your answer.

(ii) Explain how this same problem affects people in other scenarios where computer graphics are used.

(iii) Explain how these problems are being solved.

This page has been deliberately left blank.

OR: **QUESTION THREE: Computer vision**

This question includes references to **Resources C to H** on pages 3 to 5 of the resource booklet.

(a) (i) In relation to computer vision, explain the issue of noise. Refer to **Resources C and D** to support your answer.

(ii) How are these issues addressed? Refer to **Resource D** to support your answer.

(b) You may refer to **Resources E to H** to support your answers for part (b).

(i) What is Canny edge detection used for?

(ii) Describe a simple way edge detection can be carried out.

(iii) What are the challenges involved in implementing accurate edge detection?

(iv) How are these challenges addressed?

(c) Explain techniques that can be used to provide depth information in computer vision. What is needed to get good accuracy of depth?

(d) (i) How do facial recognition systems work in order to recognise a face and identify it accurately?

(ii) What is the difference between face detection and face recognition? What are some applications of each of these techniques?

(iii) How can these applications impact on humans both positively and negatively?

Source (adapted): https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

(e) Referring to the statement above, explain how computer vision is changing and will continue to change. How could these changes have both positive and negative impact on humans? You may use an example you have studied this year to further support your answer.

Acknowledgments
Material from the following sources has been adapted for use in this assessment:

"Theory of Computation"; Portland State University: Prof. Harry Porter

https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/previousinformation/artificialintelligence1finitestatemachines/2016%20Tutorial%208%20-%20Finite%20State%20Machines.pdf

https://en.wikipedia.org/wiki/Computer_graphics/

https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

Excellence Exemplar 2022

Subject	Digital Technologies and Hangarau Matihiko Level 3	Standard	91908	Total score	07

Q	Grade score	Annotation
1	E7	At Achieved level the candidate answers all components with no significant errors. The candidate is able to succinctly explain their responses, showing understanding of the key aspects, the algorithms and techniques. The candidate confidently shows understanding of the concepts and can then use their understanding to applying these concepts to examples provided as well as their own examples they have learnt. At Merit level the candidate demonstrates their understanding through interaction with the provided reference material and is able to show both detailed explanations of the Formal Languages, and repeatedly links these back to their everyday uses. The candidate compares and contrasts areas in Formal Languages and is able to back their understanding up with examples. The candidates answers show a clear level of understanding with no contradictory statements or off topic content. At Excellence level the candidate showcases drawing insightful conclusions about the computer science area throughout the paper. The candidate answered all questions well. To get the 08 the candidate could have identified and explained trap states in (g), explained why the FSM didn't show the full diagram, and potentially identified other examples of this used elsewhere