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 2021

91908 Analyse an area of computer science

Credits: Three

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 "123-123456789-91886".

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

Make sure you have the PDF of the 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 as instructed by your teacher.

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.

Excellence

TOTAL 07

ASSESSOR'S USE ONLY

INSTRUCTIONS

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

- Formal languages (page 3)
- Network communication protocols (page 9)
- Big data (page 14).

Choose only 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) Figure 1 shows a finite-state machine.

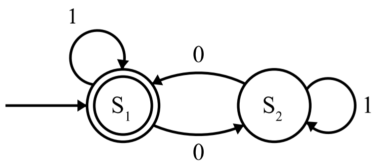


Figure 1.

(i) What is the alphabet used in this machine?

Binary (0 or 1)

(ii) What is the label of the accepting state?

S_1

(iii) Give the shortest string accepted by this machine. Explain your answer.

“ ”, an empty string, the initial state of the FSA is an accepting state

(iv) Can “11011” be considered an accepted string? Explain your answer.

No, the FSA only accepts strings with an even number of “0”, 11011 only has 1 “0” and therefore is not an accepted string

(v) Does this machine accept a string that is the digit “0” repeated 931 times? Explain your reasoning.

No, the FSA will only accept a string that has an even number of “0”s, 931 is not an even number of “0”s.

(b) The following table shows examples of some common regular expressions:

Expression	Description
`a?`	Zero or one of `a`
`a*`	Zero or more of `a`
`a{3}`	Exactly 3 of `a`
`\d`	Any digit
`\D`	Any non-digit
`\w`	Any word character (letter, number, or underscore)
`\W`	Any non-word character
`[A-Z]`	Any single character in the range A to Z

Consider the regular expression `[A-Z]{3}\d{3}`

(i) Give TWO examples of a string this regular expression would accept.

(1)	AAA111
(2)	GSD736

(ii) Explain how you came to this conclusion.

[A-Z] means any single character in the range of A-Z, this includes any capital letter in the alphabet (assuming the standard English Alphabet). {3} indicates exactly 3 of the prior expression, therefore there has to be 3 capital letters at the start of the string.

Following that, there is \d, the immediate follow up of a new expression means that the next expression should be joined onto the end of the last expression, \d means any digit from 0-9 and acts as a shorthand for [0-9]. Finally this is followed by another {3}, meaning that there should be 3 numbers. Putting this all together makes the RE mean 3 capital letters followed by 3 digits

(c) The following are the production rules of a context-free grammar:

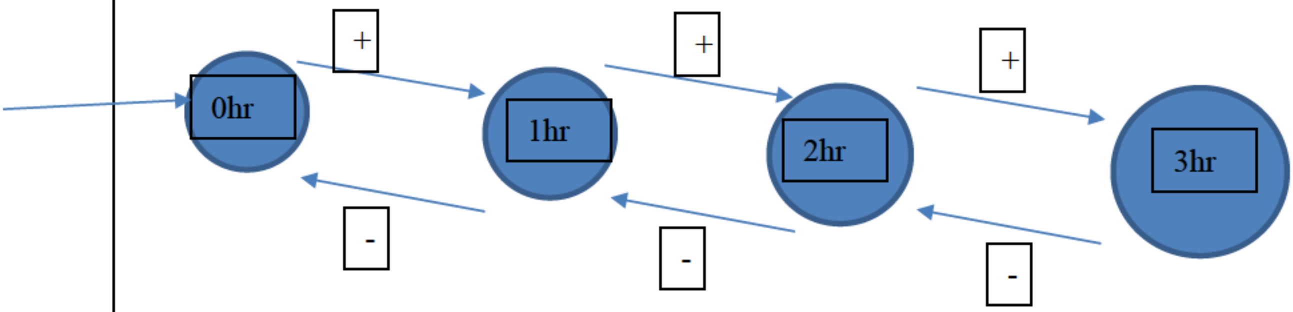
$E \rightarrow N$
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow –E$
$E \rightarrow (E)$
$N \rightarrow 0–9$

Select from these rules and fill in (2) to (4) below to show the sequence you would use to build the expression **(–3)**.

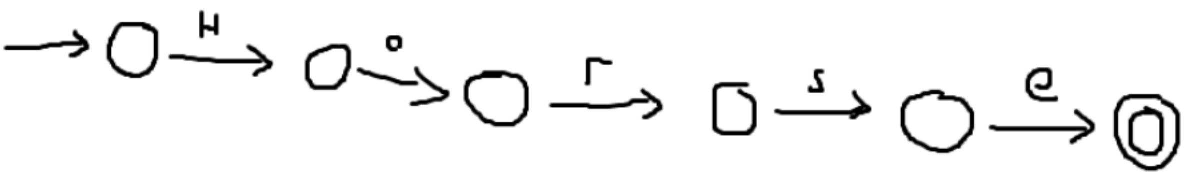
(1)	E
(2)	E -> (E)
(3)	E -> $-E$
(4)	E -> N
(5)	$N \rightarrow 3$

(d) Give a practical example of how formal languages are regularly used by people, and explain how problems can arise when using formal languages in this way.

Formal languages are used everywhere in everyday life, by definition a formal language is a set of strings that follow a set of rules, these strings are built from alphabets. And the alphabets are a finite set of arranged symbols, these symbols can be anything. For example, the numbers “0” and “1” form the binary alphabet, and the characters “G” “C” “A” “T” form the alphabet for genetic code, even the English alphabet is an example of alphabets used by formal languages. These alphabets are used in many formal languages, from using binary buttons to cycle through options on home appliances



The example above shows a simplified version of how a timer system would work with binary input options using the alphabet of “+ -” being able to increase the time by 1 hour by pressing the + button and decreasing the time by 1 hour by pressing the – button. These types of FSA can be found in almost every user interface, another example of how formal languages are used in everyday use would be in the World Wide Web, the ability to enter a search term into a search engine and only receive data that is relevant to you is possible thanks to formal languages that can sort through Yottabytes of data to find only the pieces of data that matched the rule you dictated when you searched the internet, for example, if you search “Horses” on the internet you have, in essence, formed a formal language that will find all instances of your string on the internet and present only the accepted pieces of data, an example of what that would look like as an FSA would be:



Extraneous links have been removed for simplicity as if the FSA diagram showed all the links then the diagram would be unreadable, this is a Deterministic Finite State Automata (DFSA). Meaning all inputs have exactly 1 path and there are no empty paths. The opposite of this is a Non-Deterministic Finite State Automata (NFSA), and as such does not have the limitation of having each input only direct to 1 node, there can be many nodes that 1 input can connect to, the result of this is that there are multiple nodes a string can end on, however the string will still be accepted so long as 1 path led to an accepting state node. There can also be connections without a symbol associated with it, this merely means that the state can change without the need for an input. Examples of this would be in cases where 1 of 2 or more different rules have to be met, such as finding all contacts in a directory, the user could have given their phone number, their address or their email, being able to create an NFSA to search for all 3 at once instead of 3 separate searches is very useful.

The final example is slightly less common, however it is very useful, when researches need to search through genetic code to locate certain important patterns relating to genetic health problems they will need to search through thousands of datapoints and without the use of a formal language to identify patterns and such the task would be almost impossible.

Formal Languages are used in many places and because it is used so broadly, there are bound to be problems that arise with it. Broadly speaking, when a user is operating a home appliance with 1 input as can be common with a cycling option feature, they can run into the issue that they miss their desired option and have to cycle back through the entire FSA, this is especially common and one of the biggest complaints in video games, during dialogue the user will press the button to move to the next piece of dialogue, and if that button happens to be the same input as the button to activate dialogue it will not be uncommon to see the user repeat the same input rapidly and reset the dialogue options. By not including an option to exit/reset a cycle at any point the designer of the FSA creates the option to waste the users time.

(e) Give a brief summary of the similarities and differences between regular languages and context-free languages. In your answer, explain the capabilities of each, and the limits of what they can do.

Regular languages have restrictions on how they can be used, for example they have to follow rules that can be determined by Regular Expressions (RE), and these restrictions mean that more complex rules cannot be used such as checking for Palindromes or more complex grammar rules. A context free language however does not have these restrictions and can check much more complex rules, these context free languages will be found in code validators as they can check strings against more complex rules to check if the code should run or not based on the syntax and grammar of the language.

However, a Regular Language will be much simpler and faster than a context-free language as its rules are much more basic, and as they can both do similar things at a basic level the use of one over the other is on a case by case level depending on the task.

(f) Choose ONE of the following options to answer.

Option (1)

The variation on a finite-state automaton (FSA) shown in Figure 2 has an output on every transition.

For example, the transition labelled “0/1” means that the transition is on a “0” (as in a simple FSA), but it outputs the symbol “1” when it takes the transition.

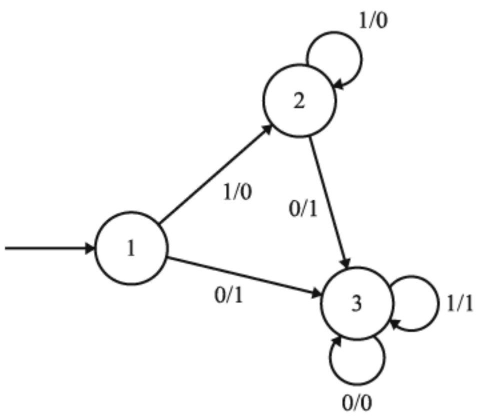


Figure 2. A finite-state automaton

Option (2)

A non-deterministic finite-state automaton (NFA) allows two transitions from a state to have the same label, and accepts a string if taking either choice can lead to an accepting state.

For example, the NFA shown in Figure 3 accepts the strings “b” and “bab”, but not “bbab”.

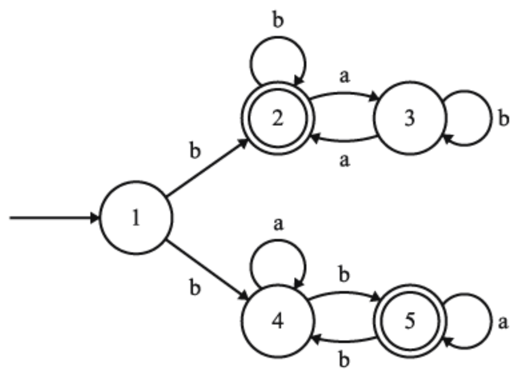


Figure 3. A non-deterministic finite-state automaton

Explain what your chosen machine does, or could do, and give examples of relevant strings to support your answer.

Option (1 or 2)

Option 2

Response

In the example, the NFSA will allow strings that *either* contain an even number (inc. 0) of “a”s or an even number of “b”s. However, the string has to start with a “b”, as determined by the first node only having “b” labeled transitions, as the implication is that if a node does not have a certain symbol transition, then that input leads to a “trap state” where it becomes impossible to get an accepted string and the program should stop its calculations for the string and return “not accepted”. The point of an NFSA is to more simply create an RE that has two or more rules, by having a split in the NFSA the RE can check for either rule being followed and subsequently accept that string. Examples of how this can be used is in an ‘or’ operation in a RE, usually notated by a “|”, accepted strings for the given NFSA would be “bbabb” (even number of “b”s), “baaaaaaaa” (even number of “a”s). and not-accepted strings would be “aabb” (does not start with a “b”) or “bababa” (odd number of both “b”’s and “a”s)

Excellence Exemplar 2021

Subject	Level 3 Digital Technologies	Standard	91908	Total score	07

Q	Grade score	Annotation
1	E7	The candidate has completed all required components of the assessment. In parts (a) to (c), the candidate has interpreted the resources and provided correct answers, with accurate and succinct explanations. In part (d), they have given a good practical example, and demonstrated the ability to explain the sorts of problems that can arise. While the candidate appears to have struggled to respond to part (e), when considered together their response to parts (d) and (e) provides satisfactory coverage of the relevant achievement criteria. The discussion links Regular Languages to Finite State Machines and refers to their limitations. In part (f), the candidate has interpreted the behaviour of the FSM, showing understanding of its inputs and recognising and explaining the specific outputs. The level of understanding demonstrated is not sufficient to justify awarding E8.