

Copyright

© 2016 PG Online Limited

The contents of this unit are protected by copyright.

The units and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it are supplied to you by PG Online Limited under licence and may be used and copied by you only in accordance with the terms of the licence agreement between you and PG Online Limited. Except as expressly permitted by the licence, no part of the materials distributed with this unit may be used, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise, without the prior written permission of PG Online Limited.

Licence agreement

This is a legal agreement between you, the teaching institution, and PG Online Limited. PG Online Limited grants to you a non-exclusive, non-transferable, revocable licence to use this unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it in the course of teaching by your teachers and/or employees.

The materials distributed with this unit may be copied and used by your teachers and/or employees on a single site only in the course of their teaching. You warrant that you shall not, and shall procure that each of your teachers and/or employees shall not, share in any way any of the materials or part of the materials with any third party, including users on another site or individuals who are teachers and/or employees of a separate institution. You acknowledge and agree that the materials must remain with you, the teaching institution, and no part of the materials may be transferred to another institution. You also warrant that you shall not, and shall procure that each of your teachers and/or employees shall not, procure, authorise, encourage, facilitate or enable any third party to reproduce these materials in whole or in part without the prior permission of PG Online Limited.

In consideration of the licence granted to you, you shall indemnify PG Online Limited against all liabilities, costs, expenses, damages and losses (including but not limited to any direct, indirect or consequential losses, loss of profit, loss of reputation and all interest, penalties and legal costs and all other professional costs and expenses) suffered or incurred by PG Online Limited arising out of or in connection with the exercise by you of your rights granted under this licence.

Teacher’s Guide

Introduction

This teacher’s guide contains a detailed lesson plan to accompany the set of PowerPoint slides and worksheets for each lesson.

The lessons are designed to form a basis for ideas for the teacher and should be adapted to suit the teaching style and preferences of the individual teacher, and the resources and nature of the individual school or Computing / ICT department.

The material supplied for this unit includes:

* 6 PowerPoint presentations, each designed to cover one lesson
* 6 worksheets
* 6 homework sheets
* An end-of-unit test for assessment purposes

Summary

The unit is subdivided into six topics (plus a test) in order to fit with most school timetables. It is a theoretical unit covering Sections 4.3.3.1, 4.4.2, 4.4.3 and 4.4.5 of the AQA A Level specification. After covering Mealy machines (Section 4.4.2.1) in the first lesson, sets and regular expressions (Section 4.4.2.2 and 4.4.2.3) are covered in Topics 2 and 3. The structure and use of Turing machines that perform simple computations (Section 4.4.5) are discussed in Topic 4, and Backus-Naur form and syntax diagrams (Section 4.4.3.1) are explained in Topic 5. The last topic covered is Reverse Polish notation (Section 4.3.3.1), with students being given plenty of opportunity to practise skills and techniques throughout each lesson.

Learning Outcomes for the unit

**At the end of this Unit all students should be able to:**

* Interpret finite state machines with and without output
* Define a set by listing its members
* Calculate a subset, membership, union, intersection, and difference of given sets
* Form and use simple regular expressions for string manipulation and matching
* Explain the structure of a simple Turing machine.
* Read BNF production rules and validate input strings.
* Convert simple infix form to Reverse Polish Notation and vice versa

**Most students will be able to:**

* Interpret state transition tables for finite state machines with and without output.
* Give examples of a proper subset, countable set, and cardinality
* Describe the relationship between regular expressions and finite state machines.
* Write the set representing the Cartesian product of two sets
* Represent sets using set comprehension and compact representation
* Interpret and represent transition rules for a Turing machine using a transition function or a state transition diagram
* Trace the behaviour of a simple Turing machine
* Interpret and formulate simple BNF production rules for context-free languages
* Interpret and draw syntax diagrams equivalent to a given BNF expression.
* Explain why and where Reverse Polish Notation (RPN) is used

**Some students will be able to:**

* Translate between representations of finite state machines with and without output
* Write a regular expression to recognise the same language as a given finite state machine and vice versa
* Explain the importance of Turing machines, including the Universal Turing machine, to the subject of computation
* Explain why BNF can represent some languages that cannot be represented using Regular Expressions
* Convert complex infix to RPN and vice versa, using multiple methods

Previous Learning

Students would benefit from having studied relevant material from the new KS3 National Curriculum and more specifically a Computer Science related GCSE. However, the material presented in this unit will not assume that students have studied these topics prior to this course.

Students should have studied simple finite state machines, stack and tree data structures and traversals and recursion in the first year of GCE. A review of that material may be helpful.

Suggested Resources

No specific software is required for this unit beyond a standard office suite of applications for the presentation and printing of provided resources.

The textbook *AQA A Level Computer Science (Year 2)* or *AQA A Level Computer Science* (AS and A Level Year 2 in one volume).



A complete course text that provides a comprehensive understanding of each topic in both years of the new AQA A Level Computer Science specification. It is presented in an accessible and interesting way, with many in-text questions to test students’ understanding of the material and ability to apply it.

The complete book is divided into 12 sections, each containing roughly six chapters. Each chapter covers material that can comfortably be taught in one or two lessons. It will also be a useful reference and revision guide for students throughout the AS and A Level courses.

Two short appendices contain A Level content that could be taught in the first year of the course as an extension to related AS topics.

Each chapter contains exercises, some new and some from past examination papers, which can be set as homework. Answers to all these are available to teachers only in a Teachers Supplement which can be ordered from our website www.pgonline.co.uk.

Vocabulary

Vocabulary associated with this Unit, such as:

* Finite state machine, Mealy machine, transition, transition condition, state, state transition table
* Set, member, element, ordered, unordered, common sets, set comprehension, compact representation, membership, union, intersection, difference, subset, proper subset, Cartesian product, infinite, finite, countably infinite, cardinality
* ∈ is a member of

∉ is not a member of

∪ union

∩ intersection

\ difference

{ } set

⊆ subset

⊈ not a subset

⊂ proper subset

x Cartesian product

|… | cardinality

∧ AND

ø empty set

* Regular Expressions, precedence, regular language, decompose, | \* + ? ()
* Turing machine, state transition diagram, tape, read-write head, halting, δ, Universal Turing machine, computable
* Backus-Naur Form, terminal, non-terminal, pipe, syntax diagram, parsing, parse tree
* Infix, prefix, postfix, Polish Notation, Reverse Polish Notation, order of precedence

Assessment

Assessment will be by means of regular homework and a test with examination style questions.

Topic plans

|  |  |  |
| --- | --- | --- |
| Topic 1 | Mealy machines |  |
| Learning Objectives:   * Be able to draw and interpret simple state transition diagrams for FSMs with no output and with output * Be able to draw and interpret simple state transition tables for FSMs with no output and with output | | |
| Content | | Resources |
| **Notes**  A Mealy machine is used to translate an input stream to an output stream. Think of a two tape system. The machines described here are quite simple, with 3 or 4 states. Complex systems will have many states.  This YouTube video is a good demonstration of the differences between the FSMs (taught at AS) and the Mealy machine (as taught at A Level). <https://www.youtube.com/watch?v=hJIST1cEf6A>  This YouTube video is another useful demonstration, but it is more complex than the students need to be able to cope with. <https://www.youtube.com/watch?v=S352lyPZP00>  Wikipedia: <https://en.wikipedia.org/wiki/Mealy_machine>  **Starter**  Students should recall the concept of a finite state machine.    Take this opportunity to discuss the ones they’ve done previously at AS Level.  Refresh the vocabulary with the colouring exercise, where students colour the states, transition, and transition conditions on a simple FSM.  Then, students can draw their own FSM of a simple light on/off when pressing a button. They can label the states, transitions, and conditions.  **Give out Worksheet 1.**  **Task 1, questions 1a and 1b** on the worksheet can be used to consolidate prior learning.  **Main**  Remind students that the purpose of FSMs is that they can be implemented by electrical circuits.  Introduce the concept of a Mealy machine by defining it as similar to the FSMs they’ve done before. The important concept is that a single output is determined by the current state and the current input. The output is in addition to a state change. Introduce the special notation (input/output) on the transition arc.  Example 1 is an opportunity to show how to produce a state sequence by tracing the change state and output generated, based on the current input. Students can get lost during this process. The notation on the slide should be read as: Starting in the first column: An FSM in ‘state sequence’ receives an ‘input’. An output is generated. The change of state is registered in the second column.  **Example 1** The idea is to find the path through the machine that will generate an output of 1. A 2-state machine can demonstrate a memory of 2 states, thereby recognising sequences consisting of 2 transitions.  **Example 1** can also be used to show how substrings can be recognised in the state sequence table. The state sequence table is a good illustration of 2 tapes, where the input row is one tape and the output row is the second tape. The patterns of substrings are also discernible in the state sequence table.  Example 2 can be completed by the students. Use this opportunity to make sure that students can perform this task without getting lost in the sequence.  Then, students can do **Task 2** on the worksheet. It’s a partially complete solution. This Mealy deals with repetition, but it is not necessary to discuss this yet. It will be covered in the next exercise. Designing a Mealy machine to deal with repetition requires careful attention to state changes.  Introduce the concept of a state transition table. This is a good way to make sure that you identify all the states and all possible inputs for each state. Ask students to identify the relationship between the FSM and rows in the state transition table. (The number of rows is equivalent to the number of transition arcs). Ask students why this is true. (This is because for every state there are two inputs, 0 or 1). Students can complete the table based on the FSM diagram. Which do students think is easier to read, an FSM diagram or a state transition table? Enforce the idea that both, diagram and table, represent the same Mealy machine.  Example 3 gives students an opportunity to demonstrate the equivalence of diagram and state table, by completing a state sequence for an input string.  Exercise: Students can design a Mealy machine to identify a sequence of an even number of 1s followed by 0. There is a partially complete machine on the next slide for scaffolding, if required. They can draw a Mealy machine diagram. To test their machine, they can construct the state sequence for the input 01011001. Have students identify the part of the state sequence that represents the required sequence. There is a partially completed Mealy machine on the next slide for students who may not be able to design one from scratch.  Worksheet: Students can complete the worksheet problems with Mealy machines. There is an interesting aspect to the solution, where the targeted patterns overlap. The layout of the exercise should lead students to spotting this.  **Plenary**   * Mealy machines have an output dependent upon the current state and the current input. * Mealy machines are a type of Finite State Machine (FSM) * States are drawn as circles. * Transitions are drawn as arcs. * Transition notation is indicated as input/output * Two ways of representing a Mealy machine are a state transition diagram and a state transition table. * The connection between languages and Mealy machines is that outputs can be used to translate from one language to another.   Give out **Homework 1**. | | PowerPoint Guide: Regular languages T1 Mealy machines  Regular languages Worksheet 1 Mealy machines  Regular languages Worksheet 1 Answers  Regular languages Homework 1 Mealy machines  Regular languages Homework 1 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 2 | Sets |  |
| Learning Objectives:   * Be familiar with the concept of a set and the notations used for specifying a set and set comprehension * Be familiar with the compact representation of a set * Be familiar with the concept of finite and infinite sets, countably infinite sets, cardinality of a finite set, Cartesian product of sets * Be familiar with the meaning of the terms subset, proper subset, countable set * Be familiar with set operations: membership, union, intersection, difference | | |
| Content | | Resources |
| **Starter**  Discuss with students where they may have encountered sets before.   * Where have you encountered sets before? * What is your definition of a set? * Can you name any sets? * Do you know about any relationships between sets? * Have you seen set relationships drawn as diagrams?   Students may know about sets from Mathematics, although not all specifications cover it. If many have not encountered them before, then spend a few minutes exploring what their intuitive understanding is. Some examples could be a set of playing cards, a set of board markers, the set of prime numbers, and the numbers between 0 and 1.  Students will want to know why they’re doing mathematics in computer science class. At this early stage, it should be enough to introduce the idea of relationships between sets, logic, Boolean algebra, and syntactic analysis of programming languages.  **Main**  **Terminology**  Introduce a more formal definition of a set (unordered, each element only once) and notation for set names and members.  **Define sets by listing each member**  Introduce how to define a set by listing every member. The notation has to be exact and must include the opening and closing curly braces.   * All odd numbers between 0 and 10 inclusive A = {1, 3, 5, 7, 9} * All numbers between 1 and 100 inclusive, which are exactly divisible by 10  B = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}   **Common Sets**  Students will have a concept of these common sets. They could discuss in groups how they refer to each. It’s important that we know the names and representation for the common sets.  The answers are provided on the following slide.  There are a couple of interesting concepts that need to be addressed to head off misconceptions.   * It should be pointed out that infinity () DOES NOT belong to the set of real numbers. * Only A-Level Maths students will have encountered imaginary numbers, but it could be discussed just to point out that complex numbers (those with an imaginary part denoting) DO NOT belong to the set of real numbers.   **Questions:**  This set of questions is included to enable the students to actually think about the relationships between different sets.   * Is each member of **Z** also a member of **Q**?  Yes, because the number 12 can be written as a rational number * Describe an empty set.  One with no members * How might the empty set be represented?  { } is the way which follows on from the definition by listing elements. Introduce the special symbol (Greek letter phi – pronounced “fie” to rhyme with “pie”) * How many different empty sets are there?  This is interesting because students may think that there is a corresponding empty set for each existing set. However, this is not the case. There is only a single empty set. If A and B are both empty, then they have exactly the same members (none).   **Set Comprehension**  Introduce the concept of set comprehension by using an example.   * S = { x | x ∈ **N** ᴧ x is even } is read as “S is the set of all x, such that x is a member of the set of natural numbers and x is even” * After seeing the ‘and’ symbol (ᴧ), what do they think the ‘or’ symbol will be? ⋁   **Compact Representation**  Stress that both Set Comprehension and Compact Representation are equivalent and describe the same set.  Students can complete question 1 on the worksheet to further their understanding of the two different notations.  Give out **Worksheet 2** and ask students to complete the questions in **Task 1**.  Go over the answers when everyone has finished.  **Set Operations**  Students will be able to discern the function of the set operations based on the definitions of the set operation names. Ask students to come up with their own definition of the operations.  Then, show the students the symbols associated with each operations (next slide).  Some examples that can be discussed with the students.   * a ∈ {b, a, c} * 0 ∈ {0, 1} * i ∉ {t, e, a, m} * {t, e, a, m} ∪ {h, a, m} = {a, e, h, m, t} * {h, e, a, t} ∩ {m, e, a, t} = {t, e, a} * {h, e, a, t} \ {m, e, a, t} = {h}   **Exercise**  Students can take a few minutes to construct the resultant sets from the images. Did students remember to include the set notation { }?   * There are only 3 sets shown, because A = C.     Students can complete **Task 2** on the worksheet to consolidate their understanding of the set operations and notation.  **Subsets**  The tricky part here is that a subset can also be an equal set. Make sure students understand the difference between the symbols for a **subset** ⊆ and a **proper subset** ⊂. It makes sense because a proper subset removes the possibility of an equivalent set.  Students can complete **Task 3**, question 5 on the worksheet to consolidate their understanding of the subsets and notation.  **Cartesian product**  Highlight the requirement that pairs are ‘ordered’. Given the English definition, have students come up with the set comprehension definition. A x B = {(a, b) | a ∈ A ᴧ b ∈ B}  Students can now do **Task 4**, question 6 on the worksheet.  **Cardinality**  The concept of cardinality is quite simple. However, the notation is confusing. |A| in sets means the cardinality of A, the number of members in set A. |A| in vectors means the magnitude. |A| with numbers means the absolute value of. Ensure that students understand that the domain dictates the interpretation of the notation. You could ask the students if they’ve seen the notation |x| before. They may be able to bring their own interpretation to the need for separate understanding based on domain.  **Set Concepts**  Students will grasp the concept of a finite set. They’ve been working with them up to now.  Students should understand the concept of an infinite set. They’ll understand about the natural numbers growing ever bigger.  The stretch to assigning each member of a set such as **N**, a positional number from N is not too difficult. There is some help in the textbook for this concept.  Students may understand intuitively that the set of **R** is not countable, but may have difficulty in expressing why. Consider what number you would assign to position 1. Is it 0.1, 0.01, 0.001? The issue is that between any two members of **R**, there is an infinite number of numbers.  **Exercise**   * Planets in the solar system (finite, 8) * Number of possible digits after the decimal in a number (infinite, countably infinite) * Suits in a deck of playing cards (finite, 4) * Negative fractions (infinite, not countable) * Grains of sand on the planet (finite, countable)   The question on negative fractions may cause some discussion. One definition of “countable” is as follows:  “A set is countable if and only its elements can be placed in a one-to-one correspondence with some subset of natural numbers.”  The two sets will therefore contain the same number of elements.  e.g. the elements of the set {2, 4, 6, 8…} can be paired off against the set {1,2,3,4…} and is therefore countably infinite.  The set of negative fractions will include  -1/2, -1/3, -1/4, -1/5… but also -2/3, -3/4, -3/5,… -4/5, -5/6… etc.  There is no way of pairing off elements of this set with the set {1, 2, 3, 4….}  There is an extension task on the Worksheet if students wish to try this.  **Plenary**   * What are the 3 different ways of defining a set? Listing, set comprehension, compact representation * What are the 4 operations and their symbols that can be performed on sets? Union ∪, intersection ∩, difference \, membership ∈ ∉ * What is the difference between an infinite set and a countably infinite set? The later can be assigned cardinality. You can count off the members by using the natural numbers (**N**). The former does not possess this property, for example, R * What is the difference between a subset and a proper subset? A proper subset disallows the two sets being equivalent.   Give out **Homework 2**. | | PowerPoint Guide: Regular languages T2 Sets  Regular languages Worksheet 2 Sets  Regular languages Worksheet 2 Answers  Regular languages Homework 2 Sets  Regular languages Homework 2 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 3 | Regular expressions |  |
| Learning Objectives:   * Understand that a regular expression is a way of describing a set * Understand that regular expressions allow particular types of languages to be described in a convenient shorthand notation * Be able to form and use simple regular expressions for string manipulation and matching * Be able to describe the relationship between regular expressions and finite state machines * Be able to write a regular expression to recognise the same language as a given FSM and vice versa | | |
| Content | | Resources |
| **Notes**  Terminology is important in this topic to tie the previous two topics together. When talking about the strings that match a regular expression, use the term ‘set of strings’. This ties to the previous topic.  When building Regular Expressions, start with a simple sub-part, check it, and then add them together to make more complex expressions.  If students have used Regular Expressions in programming, then they will be more familiar with the notation that looks something like this: [A-Z][a-z]\*[0-9]{2}. However, this topic is about connecting sets and languages, with the language being defined by the RegExp syntax, so don’t get confused between the two. The syntax of Regular Expressions for languages is limited to (), \*, +, ? |.  **Starter**  Depending on the type of programming experience, students may have encountered Regular Expressions in their programming. If so, then ask students to describe their understanding of Regular Expressions.  Connect the functionality of Regular Expressions used to define a set with the last topic, Sets. The example on the slide defines the set S by listing the members. Then, it provides a set comprehension notation that is reasonably equivalent. S = { s | s begins with ‘a’ ᴧ is followed by any number of ‘b’}    (Read as “S is defined as the set of all members s, such that s begins with ‘a’ and is followed 0 or more b” Then, it shows the Regular Expression. The main thing to point out is that they’re all equivalent definitions of a set.    **Main**  **Notation**  The definition of regular expressions incorporates many more symbols, but the ones given here are the most frequently used. Students may want to make notes from this table to use in the remainder of this lesson. This site has a nice long list: <http://www.regular-expressions.info/refquick.html>  **Translate**  Students should try to translate these simple regular expressions to an English definition, then define the associated set by using the listing method from the previous topic.   * *a+b+*   + One or more a, followed by one or more b   + The set of strings that satisfies this regular expression can be defined as: {ab, aabb, abbbb, aaaabbbbbb, …}   + ‘Why is the string ‘a’ or ‘b’ rejected?’ Because the + symbol means there has to be at least one of each.   + Why is the string ‘bbbaaa’ rejected? Because the adjacency of a to b in the expression means they have to come in that order. * *a\*b*   + Zero or more a, followed by a single b   + The set of strings that satisfies this regular expression can be defined as: {b, ab, aab, aaab, …}   + Why is the string aaa rejected? Because there must be at least the single b in the definition. * (a?b)|(ba+)   + (Zero or one a, followed by b) or (b followed by one or more a)   + The set of strings that satisfies this regular expression can be defined as: {b, ab, ba, baa, baaa, …}   + Why is the string b rejected? Because if the string begins with a b, there has to be at least one a.   + Why is the string bb rejected? Because the | means that the alternatives are mutually exclusive. * *(ab.)+*   + A group of a single a followed by a single b, where the pair is repeated one or more times.   + The set of strings that satisfies this regular expression can be defined as: {ab, abab, ababab, abababab, …}   **Form**  A good way to approach these is to generate a few members of the set, then convert them to a Regular Expression.  An 01 pair repeated any number of times  {01, 0101, 010101, …}  *(01)+*  One or more 0s bounded on either end by a single 1  {101, 1001, 10001, …}  *10\*1*  S = {10, 100, 1000, … 01, 011, 0111, …}  *(10\*)|(01\*)*  S = {10001, 1000, 1011, 10111}  *10 (00)|(11)1?*  **Use**  Note that if students have used Regular Expressions in programming, then they will be more familiar with the notation that looks something like this: [A-Za-z][0-9]. However, this topic is about connecting sets and languages so the RegExp syntax is not exactly suitable.  This is a challenge of finding the ones that fail all the checks.  *a+b* 🡪 ab, a, b, baa, aab  *(bc)\*a* 🡪 a, bcaa, bca, bcbca, bcabca  *LL?D?D where L = {A…Z} D = {0 …9}* 🡪 SO14, gu13, BH12, 1CH87, DT6, W1  *0(0|1)\** 🡪 0, 00, 01, 000, 101 001, 010  **Worksheet Exercises: Task 1 questions 1 and 2**  Students have a go at doing some exercises.  **Regular language**  Introduce the concept of a regular language. It can be simply defined by a Regular Expression or an FSM. Therefore, it must be possible to convert between an FSM and a Regular Expression.  Ensure that students understand that it is perfectly valid for an FSM to have more than a single accepting state.  **FSM building blocks**  The next slide is a very simple presentation of the different building blocks that students will need to translate FSM to Regular Expressions and vice versa. These are incomplete. They do not follow the rules of an output for every possible input, but it’s the structure of the pattern we’re trying to identify. It’s good to introduce the idea of building up the patterns to construct an FSM for a Regular Expression.  It would be worth going over these step-by-step with students, or let one of them explain. Have students transfer these to their notes because they’ll be useful for the rest of this topic.  **Find a Regular Expression from FSM**  Note: This FSM is incomplete. The next slide will show the complete one.  The best way to do this is generate some good and some bad strings to make sure you can read it. Be sure to cover the whole FSM, watching the ending states.  Then, parse the FSM in parts that are recognisable from the building blocks section. Use colours or boxes to dissect the FSM.  Lastly, run all the strings generated previously through the Regular Expression to see if they behave as predicted.  **Complete FSM**  This image shows the completed FSM with all states and a transition for every input in each possible state. Students can double check the strings to see the behaviour.  **Worksheet Exercise Task 2 question 3**  Students have a go at doing some exercises.  **Decompose an FSM**  Note that for clarity, some transitions not shown. Next slide shows complete FSM.  Now, it’s time to go the other way, from a Regular Expression, construct the FSM. An issue here is getting the order of precedence correct.    Break the expression down into pieces, remembering the building blocks. Then, decompose each piece, by breaking down again, if required.  This is not as complicated as it looks. Recall that an FSM has to handle every possible input from every possible state. This FSM has 5 possible inputs, so it has to handle 5 possible transitions from each state. Luckily, many of them go to a rejection state. That is S6 on the new diagram. In S6, the FSM just spins. This is a simple concept, but can result in some complex looking FSM diagrams. Students would probably not have to deal with a 5 input FSM.  Take the simple two-pronged diagram, make a space in the middle for S6, and then put in the rejection transitions.  **Check strings**  Have students verify the operation of this complete FSM for the given strings.  **Exercise: Worksheet 3, Task 3 question 4**  Students have a go at doing the same decomposition. Answer on next slide.  **Plenary**  Name three ways to describe a Regular Language. Set Notation, Regular Expression and Finite State Machine.  Write all three for strings beginning with ‘a’ followed by any number of ‘b’  Set notation: S = {a, ab, abb, abbb, ...} *or alternatively*  S = { s | s begins with ‘a’ ᴧ is followed by zero or more of ‘b’}  Regular Expression: S can be defined by a *ab\**  Finite state machine: (see below)  C:\Users\Rob\AppData\Roaming\PixelMetrics\CaptureWiz\Temp\114.png  Hand out **Homework 3**. | | PowerPoint Guide: Regular languages T3 Regular Expressions  Regular languages Worksheet 3 Regular Expressions  Regular languages Worksheet 3 Answers  Regular languages Homework 3 Regular Expressions  Regular languages Homework 3 Answers |

|  |  |  |  |
| --- | --- | --- | --- |
| Topic 4 | The Turing Machine | |  |
| Preparation   * In the folder you will find a document called **kinaesthetic activity.docx**. You will need one sheet between two students. The sheet contains of three blank “tapes” and squares containing 0s, 1s, space symbols, states and arrows to indicate current state. (Space symbols are not really needed, the square on the tape can just be left blank.) * The symbols need to be cut into individual squares. (Don’t cut the strips representing the “tape” into squares.)   Learning Objectives:   * Understand the structure and use of Turing machines that perform simple computations * Understand that a Turing machine can be viewed as a computer with a single fixed program * Represent transition rules using a transition function or state transition diagram * Hand-trace a simple Turing machine * Explain the importance of Turing machines and the Universal Turing machine to the subject of computation | | | |
| Content | | | Resources |
| **Notes**  The last task on the worksheet is an extension task. It can be made as challenging as required. For example, instead of providing a partially complete state transition diagram, ask students to draw one themselves. Instead of completing the trace table, provide only the starting state row and ask students to trace the entire computation.  **Additional Materials**  Turing’s Google Doodle   * <http://www.google.com/doodles/alan-turings-100th-birthday> * <https://www.youtube.com/watch?v=84pbZSt_a9k> * <http://www.i-programmer.info/news/82-heritage/4403-google-doodle-a-turing-machine-puzzle.html>   Visualisation Tool   * <http://turingmaschine.klickagent.ch/>   **Starter**  **Background**  A bit of history about the inventor of the Turing machine.  **The Turing machine**  Students need to familiarise themselves with the representation of a Turing machine that will be used in the remainder of this unit. The sample assessment materials and the exam papers seem to just leave blank cells without a character at all.  Questions for students:   * What set of symbols can be seen in this image? 0, 1, SQUARE. This is the alphabet for this machine. This is a 3-symbol Turing Machine because it can process only 3 symbols. * Is the tape finite or infinite? Infinite * What represents the read-write head? The up arrow * What state is the machine in at this moment? State S1   **The controller**  In these Turing machines, the read-write head moves left and right. The read-write head must move on every state transition.  Note that the erase is implemented as an overwrite operation, there is no physical erase functionality in a Turing machine. The equivalent is “write blank”.  **Main**  **How does it work?**  Questions for students:   * Identify the states. S0,S1, S2 * Identify the starting state. S0 * Identify the stopping/halting state. S2 * Identify the state transitions. The arcs.   Given the transition 0,0,R, which is the input? Output? Movement? The format of all transitions is input, output, head move.  **A kinaesthetic activity**  Students can simulate the transitions using the additional resources (Kinaesthetic Activity).  **Worksheet Exercise**  Students can now complete **Task 1** on the worksheet.  **Transition function (defined and practice)**  Ensure that the difference in ordering parameters on the transition arcs and the δ function is highlighted so that students don’t become confused.  Students can have a go at translating the example on the slide.  δ (S7, 1) = (S3, 1, R) “If the machine is currently in state S7 and the symbol ‘1’ is read from the tape, then write ‘1’ to the tape, transition to state S3, and move head right.”  **Exercise**  During this exercise, ensure that students don’t get the order of the parameters mixed up. The order of the items in the labels on the transition arcs is “Input, Output, Head move”. Answers on next slide.  **Worksheet**  Students can complete **Task 2** on the worksheet.  **Importance of TM**  The Turing Machine provides a mathematical definition for what is computable.  **How many TMs?**  This provides justification for extending the concept of a Turing machine.  **Universal Turing Machine**  Ensure that students recognise that the UTM is just another TM, where the input tape contains the description of another TM and its data.  **Stored program computer**  The von Neumann model is not required in this section, but it places the learning of Turing machines in context.  **Worksheet – Extension task**  Students can complete **Task 3** on the worksheet. By the time students have completed this one, Turing machines should hold no mysteries for them! It may take some time to complete and could be finished for discussion next lesson.  **Plenary**  State the **four** components that are used to express a Turing machine.  It has:   * A finite set of states * A finite alphabet of symbols * An infinite tape, marked off in squares * A read-write head that can move left and right one square at a time   Describe a state transition diagram.   * It is a visual representation of the behaviour of a Turing machine. It contains symbols for states, state transitions, input, and output.   Describe the relationship between a transition function and a state transition diagram.   * The transition function can be used to produce the same rules for transitions as illustrated on a state transition diagram. This provides for a more compact definition which can be manipulated mathematically.   Explain the impact of the Turing machine on the subject of computation.   * A TM can compute anything that is actually computable.   Why might a UTM be considered an interpreter for a TM?   * Because a UTM simulates the behaviour of a TM. A UTM interprets the definition of the TM (its transition functions) and applies them to the TM’s input.   Hand out **Homework 4**. | | PowerPoint Guide: Regular languages T4 The Turing Machine  Suggestions for starter, but not required.  <https://www.youtube.com/watch?v=E3keLeMwfHY>  <https://www.youtube.com/watch?v=FTSAiF9AHN4>  Regular languages Worksheet 4 The Turing Machine  Regular languages Worksheet 4 Answers  Kinesthetic Activity.docx  Suggestions for plenary, but not required.  <https://www.youtube.com/watch?v=cYw2ewoO6c4>  Regular languages Homework 4 The Turing Machine  Regular languages Homework 4 Answers | |

|  |  |  |
| --- | --- | --- |
| Topic 5 | Backus-Naur Form |  |
| Learning Objectives:   * Explain why BNF can represent some languages that cannot be represented using Regular Expressions * Use Backus-Naur Form (BNF) to represent language syntax and formulate simple production rules * Draw a syntax diagram to represent an equivalent BNF expression | | |
| Content | | Resources |
| **Starter**  Students may have a discussion about whether the non-sensible sentences are really sentences. This is a good time to remind them of the difference between semantics (meaning) and syntax (form). If they’re working with different definitions of a sentence, then there is no way to answer the question.  The next slide more precisely defines what a sentence is. This should convince them that all of the sentences are syntactically correct, but may be semantically incorrect. Introduce the idea of the need to precisely define languages.  Students should identify that more tools are needed to define some languages. Backus-Naur form provides greater definition.  **Main**  Some history about Backus-Naur Form is provided. Students may be surprised that as far back as 1959 there were programming languages. In this case it was Algol 58 and Algol 60.  **Additional Materials**  There is an interesting report dated 1958 by Backus and others, edited by Naur, on the language Algol and its definition, which brings the history of the first high-level language to life.  It contains some original BNF from the man himself describing the syntax of Algol 60. <http://www.masswerk.at/algol60/report.htm#5_2>  The section on why we need BNF brings together the previous sections about sets and regular expressions. Specifying context-free languages as a superset of regular expressions is helpful visualisation to remind students that some languages cannot be defined by regular expressions.  **BNF meta-symbols**  Meta-symbols are introduced using a colour-coded system. Discuss with students the different symbols, identifying them in the example below the table.  Some postcodes to consider are: MP75 Ab34 ANR12 PO18 22FC D34G  (Ab34, ANR12, 22FC, D34G are all invalid) In each case, have students identify which rule is broken.  Use of the OR operator (pipe) can confuse students. Enforce the idea that it separates alternatives and is not equivalent to a Boolean operator between two operands. The brackets on the second example should clear up any confusion.  Then, students can have a go at verifying their own understanding on the final set of examples: A38, R8a4, B73, X3a9, a2, 7a2, k4, the invalid codes are: R8A4, X3a9, 7a2.  Finally, remind students that computers cannot deal with ambiguity. Every instruction must be interpreted in a single way.  **Exercise –production rules**  The exercise to write production rules is aimed to have students write the basic rules that they’ll need for most of the remaining lesson. For now, numerical values can be treat as a series of single digits.  Refer back if necessary to earlier slide, “BNF Meta-symbols”. Answers on next slide  Later, we will address how to create multi-digit numbers as single non-terminals. Typical responses are found on the next slide.  Ensure students can follow the set of rules forming the simple 2-part postcode.  Complete **Task 1** on **Worksheet 5**. There may be alternative solutions to the task. This is a good opportunity for students to work in groups to check their work.  The section on recursion may present challenges, if students haven’t yet covered recursion. If this is the case, then the simplest analogy is to define it as reusing itself on the right side of an assignment. BNF can be understood without full appreciation for functional recursion. The definition of the slide shows how to break each step of the recursion down into a simpler step.  **Exercise – variable name**  The exercise involving recursion is about 3 slides long, but includes an answer. Introduce the exercise and ensure that students understand the requirements of the variable name. You may choose to highlight that ‘\_ABC’ is not a valid variable name, but you may want to hold that for after they’ve attempted the rules. Four of the BNF rules have been given. These are the basic rules they’ve already written. It is possible to complete the exercise with only 3 more rules. However, students may not produce an optimal number of non-terminals, but that’s not an issue. On the slide with the answer, student could be asked to check for ‘\_’ and ‘\_Ab12’ as a variable name. They could check each other’s rules to see if they are correct. *Note, that it is the last rule that prevents the underscore beginning a variable name. It requires <upper> as first character in any right hand format.*  Complete **Task 2** on the worksheet. There may be alternative solutions to the task. This is a good opportunity for students to work in groups to check their work.  **Production rules**  Students need to be able to check syntax by looking at the BNF. An example is given here. 7+3 (which needs brackets around it), 45, -2, are invalid.  7 – (6+8) is invalid because it needs brackets around it. (Answers on next slide  **Parsing**  Parsing is done by programming language translators to determine if the input string is formed in a valid way and can continue on to be processed. The example here is a simple addition or subtraction expression with full brackets. Ensure students are convinced that these BNF are valid rules for the type of simple expression being discussed in this section.  **Parse tree**  The parse tree is given on the next slide. Students should trace the expression to convince themselves that the tree is a true reflection of the BNF rules being applied to the input string.  **Syntax diagrams**  Introduce syntax diagrams as an alternative to BNF, which also defines context-free languages. Just review the possible symbols. The use of circles or ellipses is not important and neither is the position of the looping arrow (top or bottom).  The syntax diagram example goes back to the basic BNF rules. Students can then draw syntax diagrams for them. Answers on next slide. Note that the <sign> rule does not allow for a null value. Omission of a <sign> in the <integer> rule is done by the left side of the pipe. This is shown in the illustration for <integer>. <integer> must have at least a single digit. The next slides shows the syntax diagram for <integer> broken down into its constituent parts. These definition sets are equivalent.  Complete Task 3 on the worksheet. There may be alternative solutions to the task. This is a good opportunity for students to work in groups to check their work.  **Plenary**   * What method may be used to represent regular languages? *Regular expressions and Finite State Machines (See Spec Section 4.4.2.3)* * What methods is used to represent context-free languages? *Backus-Naur form or syntax diagrams* * Give a description of Backus-Naur Form (BNF). *A meta-language which defines the language of a context-free language; a set of symbols which defines a language* * State another method used to represent a context-free language. *Syntax diagrams* * Why do we need these representations? *Because computers do not allow for ambiguity in statements.*   Give out **Homework 5.** | | PowerPoint Guide: Regular languages T5 Backus-Naur Form  Regular languages Worksheet 5 Backus-Naur Form  Regular languages Worksheet 5 Answers  Regular languages Homework 5 Backus-Naur Form  Regular languages Homework 5 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 6 | Reverse Polish notation |  |
| Learning Objectives:   * Convert simple expressions in infix form to Reverse Polish Notation (RPN) and vice versa * Be aware of why and where RPN is used | | |
| Content | | Resources |
| **Notes**  Revision of order or precedence, binary trees, inorder traversal, preorder traversal, and postorder traversal may be helpful.   * <http://www.mathblog.dk/tools/infix-postfix-converter/> * <http://www.cs.man.ac.uk/~pjj/cs212/fix.html> * <http://scriptasylum.com/tutorials/infix_postfix/algorithms/infix-postfix/> * <http://scriptasylum.com/tutorials/infix_postfix/infix_postfix.html>   **Starter**  Students may well be able to make a connection between the format of the strings on the right and the names on the left. If required, they could have a clue about the importance of the part in front of the word ‘fix’. The answer is on the next slide.  Each type of representation is described, with the operators highlighted. A bit of history is provided to set the context for why so many different notations have evolved.  **Main**  **RPN – Why use it**  Justification for having RPN is given as “Eliminates the need for brackets in sub-expressions as the order of evaluation is unambiguous”, with a further point also given. Students often write a very superficial answer “No brackets needed” in response to an exam question which will get them no marks.  **Order of precedence**  Order of precedence rules need to be established so that expressions without brackets are interpreted correctly. Students may well be familiar with BODMAS or something similar. Be sure to go over this with students before moving on. There are some examples for the students to put brackets on so they can have a little refresher.  **Conversion**  The notes on conversion are just an introduction to show students the agenda for the upcoming sections. The same example has been used to introduce each translation method. It is about the same complexity as previous exam questions. However, some of the exercises are more challenging.  *Note that for the exam, the students will not need to know both the bracketing and the numbering methods of translation. Both methods are given so that the students can decide which one they think simpler and revise that one.*  **Translation by numbering**  Translation by numbering is explained using the example. The example on the right has been presented one step at a time. The important aspect to remember is that some reading ahead may be required. After working through the example, students can try an exercise form themselves. The answer is given on the third slide in each method sequence.  **Translation by bracketing**  Translation by bracketing is explained using the example. Coloured pens/markers may be useful in this exercise. Otherwise, remind students to take care to keep track of paired brackets, perhaps by drawing a bar above sub-expressions.  Where operators are of equal precedence, brackets are added to enforce left to right processing. For example, the blue brackets are added to force the \* to happen before the / because it is to the left of it, not because it has higher precedence.  The last step in this process requires that all brackets be removed. Students should see that in this case, the same expression is generated without the use of brackets, thereby using a less complex representation. After working through the example, students can try an exercise for themselves. The answer is given on the final slide in each method sequence.  The section on ‘checking’ is provided so that students can verify their own work. It is a reversal of the bracketing translation method. Remind students that excessive brackets can be removed unless the brackets override precedence rules.  Students can now complete **Task 1 on Worksheet 6**. It covers numbering, bracketing, and checking of answers.  **Translation by binary tree**  Before covering the translation by binary tree, students may need a bit of a refresher on binary trees. Remind students that each node can have no more than 2 child nodes. Also, remind students that trees don’t necessarily need to be balanced. A review of pre-, in-, and post-order traversals would be helpful at this point.  Translation by binary tree is introduced next. The key to picking an operator for the root is to try to choose one of high priority in the expression, rather than one buried inside a series of brackets. Adding extra brackets to split the expression into two halves may be helpful – e.g. (a\*(b + c))/d.  The choice is not critical, and would be given in any exercise which involved constructing a binary expression tree.  The same example is used in the tree translation. After working through the example, students can try an exercise for themselves.  **Binary tree - Exercise**  The minus operator (-) has been chosen here as the root. You could ask students whether they could choose ^, + or / instead. They could choose + but it is probably more logical to view the expression as two bracketed halves, (w ^ x + z) – (x / w) and choose the operator not inside a bracket.  Students can now complete **Task 2** on the worksheet. It covers translation by binary tree.  **Conversion of RPN to infix by scanning**  Conversion of RPN to infix by scanning is demonstrated using the same example as in previous illustrations. After working through the example, students can try an exercise form themselves. The answer is given on the third slide in each method sequence.  **Conversion of RPN to infix by bracketing**  Conversion of RPN to infix by bracketing is demonstrated using the same example as in previous illustrations. This is similar to the ‘checking’ algorithm shown previously. After working through the example, students can try an exercise form themselves. The answer is given on the third slide in each method sequence.  Students can now complete Task 3 on the worksheet. It covers RPN to infix conversion.  **The evaluation of RPN using a stack**  The evaluation of RPN using a stack is covered in the last section. At this point, it might be useful to remind students of the operations on a stack, especially the push and pop.  *Note that evaluate RPN using a stack is not on the specification, and students will not be required to do this in an exam. This is included here to give them a practical example of the use of a stack, which ties in with 4.2.1.4 and 4.2.3.1*  In the instructions of the evaluation, remind students that the order of popping operands is very important. If the order is wrong, the divisions will not work properly. After working through the example, students can try an exercise for themselves. The answer is given on the third slide in each method sequence.  Students can now complete Task 4 on the worksheet. It covers evaluation of RPN using a stack.  **Plenary**  The plenary is a quick run through of each of the different techniques covered in this unit.   * Convert RPN to infix   + b f ^ 3 \* e +   + *b ^ f \* 3 + e* * Convert infix to RPN   + 7 \* ~4 + 3   + *7 4 ~ \* 3 +* * Evaluate RPN, showing stack   + 3 4 \* 12 6 / +  |  |  |  | | --- | --- | --- | | ***Stack*** | ***Pop, execute, push*** | ***3 4 \* 12 6 / +*** | | *4*  *3* |  | *\** | | *12* | *12 = 3 \* 4* |  | | *6*  *12*  *12* | *2 = 12 / 6* | */* | | *2*  *12* |  | *+* | | *14* | *14 = 12 + 2* |  |  * Build a binary tree   + 4 \* 3 + 7 \* 2   Choose the root by bracketing the two halves of the expression and selecting the operator in the middle:  C:\Users\Gavin Parsons\AppData\Roaming\PixelMetrics\CaptureWiz\Temp\61.png   * + Show prefix (Polish Notation) equivalent   + *+ \* 4 3 \* 7 2*   + Show postfix (Reverse Polish Notation) equivalent   + *4 3 \* 7 2 \* +*   Homework has been provided for this lesson, and the students will also need to revise for the assessment test. | | PowerPoint Guide: Regular languages T6 Reverse Polish Notation  Regular languages Worksheet 6 Reverse Polish Notation  Regular languages Worksheet 6 Answers  Regular languages Homework 6 Reverse Polish Notation  Regular languages Homework 6 Answers |

|  |  |
| --- | --- |
| Unit assessment | |
| Learning Outcomes:  Students will   * apply their knowledge in answers to a range of questions * be able to highlight areas of strength and any gaps in their understanding of computers | |
| Content | Resources |
| Students should complete the **Assessment Test**.  If students have not covered all of the material in the unit, the test has been broken down into sections that can easily be removed or skipped as required.  These tests have been designed to be printed and answered by hand. Alternatively they could be uploaded and incorporated into an automated test as part of many modern VLEs. | Regular languages Final assessment  Regular languages Final assessment Answers |

Acknowledgements

The authors and publisher would like to thank all contributors for their kind permission to reproduce their photographs or images, screenshots of their websites or other copyright material in the PowerPoint Guides.

Alan Turing photograph by kind permission of the Provost and Fellows, King’s College, Cambridge’. Reference number information: ‘Archives Centre, King’s College, Cambridge. AMT/K/7/12.

Images otherwise stated © Shutterstock

Every effort has been made to trace the copyright holders and we apologise in advance for any unintentional omissions. We would be pleased to insert the appropriate acknowledgement in any subsequent edition of this publication.

This material contains links to relevant websites featured in the teacher’s guide. Every effort has been made to ensure that at the time of distribution, the links remain unbroken, the material remains up-to-date and that links are not inadvertently linked to sites that could be considered offensive. PG Online cannot be held responsible for the content of any website mentioned in this material. It is sometimes possible to find relocated sites by typing the original URL into a browser. Any errors should be reported directly to support@pgonline.co.uk and changes will be made in any subsequent editions of the material.

Artwork



Yellow Vessel © 2015 Andrew Bird

Acrylic on canvas, 61x61cm

www.abirdart.co.uk