2. LANGUAGES

Miss Zero and her
complement went to sea
In a non-empty alphaboat,
They had (money) and*
*(honey)**
Wrapped up in a finite set.
[* = plenty of]

Said One to his Zero, "let's concatenate
And a long string of characters generate".
But Miss Nought thought she ought not go quite so far
So she pointed to the sky, "See that nice Kleene star."

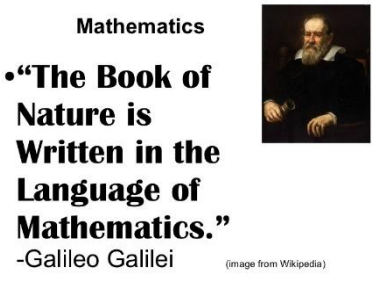
"I'm as pure as that star so my answer is NULL.
I can see from your words that you think I am dull.
My grammar once taught me that strings are fools
Who refuse to obey their production rules."

"So your most irregular language", she said as she wept,
"I could never in a million years condescend to accept!"
They put on their parentheses and sailed back to port
That disappointed Unity and distraught Miss Nought.

§2.1. The Languages of Mathematics

Somebody once said that mathematics is a language. This is very true at many levels. Just as a language is a vehicle for expressing facts rather than being a body of facts in itself, so is mathematics. Moreover many linguists believe that without human language, thought would have remained at a very primitive level. Language is not just a means of communicating thoughts. It's a tool for creating and developing thought. And so it is with mathematics.

In one sense the language of mathematics can be thought of as an extension of our everyday language. The fact that mathematicians have found the need to use specialised symbols often conceals this fact. If you were to pick up a research paper in many fields of advanced mathematics you might find many more words than symbols. Mathematicians use words, symbols or diagrams — whatever works best.



Having said that, mathematics is a language, or is an extension of natural language, let us take a different tack. Mathematics contains many miniature languages for specific purposes. Elementary algebra consists of sentences of the form "expression = expression", "expression < expression", etc. The variables and numbers in the expressions are the nouns, and the symbols

such as “=” and “<” represent verbs. Another very important language is the language of symbolic logic.

§2.2. Languages and Computing Science

Trained computing scientists are familiar with at least one computer language. These are specialised systems for expressing commands, instead of facts. Early languages were Fortran, Cobol and Pascal. Fortan is still used, but Cobol and Pascal are essentially dead languages, like Latin. Modern computing languages include C++ and C#.

"I'M NOT SURE WHICH DEAD LANGUAGE TO TAKE NEXT SEMESTER- LATIN, MIDDLE ENGLISH, OR COBOL."

Since compilers must be able to convert such commands reliably and precisely into machine code, these languages need to be defined in a very tight way.

It is therefore considered to be an important part of the training of any computing scientist to study the theory of languages — how they can be defined and how they are built up. Many of the great names in this area of computing science, names such as Chomsky and Kleene, were actually experts in linguistics who recognised the relevance of their work to computing science and transferred their research into that environment.

§2.3. Strings

If A is any set of symbols (it is usually finite and is called an **alphabet**) then a **string** on A is a finite sequence of symbols juxtaposed without any separators: $a_1a_2 \ldots, a_n$.

Example 1: The following are some strings on the alphabet {1, 2, +, =}:

α is "1+1=2",
β is "21+2122",
γ is "22+12=11",
δ is "=+1+=+"
String α represents a true statement in ordinary arithmetic. String β is a valid arithmetic expression, but is not a statement. String γ is an arithmetic statement but is false. String δ is pure nonsense. At least it doesn't make sense in ordinary arithmetic, though it could be given a meaning. But *all* four are valid strings on the alphabet.

The **length** of the string $\alpha = a_1a_2 \ldots a_n$ is defined to be n and is denoted by $|\alpha|$.

Example 2: For the strings in example 1, $|\alpha| = 5$, $|\beta| = 7$, $|\gamma| = 8$, $|\delta| = 6$.

The **null** string is the unique string of length 0 and is denoted by the symbol λ.

It might seem a bit ridiculous to invent a word, and a symbol, for a string of no symbols. But people once thought that about the number zero. "Nothing", whether it is the number zero, the empty set, or the null string, is a very useful concept. Life would be so much more complicated without it.

The **product** of two strings

$\alpha = a_1 a_2 \ldots a_m$,

$\beta = b_1 b_2 \ldots b_n$.,

is defined to be the string

$\alpha\beta = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$.

This operation is called **concatenation**.

Example 3: Suppose α, β, γ and δ are the strings defined in example 1, and λ is the null string. Then $\alpha\delta$ = "1+1=2=+1+=+" while $\delta\alpha$ = "=+1+=+1+1=2". Neither makes much sense but both are nevertheless valid strings.

Observe that $\alpha\delta \neq \delta\alpha$. In general, strings do not commute. But note that $\alpha\lambda = \lambda\alpha$ since both give just "1+1=2". In fact concatenating the null string onto either end of *any* string leaves that string unchanged.

Properties: For all strings on a given alphabet:

$$(\alpha\beta)\gamma = \alpha(\beta\gamma)$$
$$\alpha\lambda = \lambda\alpha$$
$$|\alpha\beta| = |\alpha| + |\beta|$$
$$|\lambda| = 0$$

The set of all strings on a given alphabet is **closed** under the operation of concatenation, the **associative** law holds and there is an **identity**, namely the null string. However strings don't have **inverses** (apart from the null string which is its own inverse) because concatenation cannot make a string shorter.

An abstract structure in which there's a binary operation that satisfies both the closure law and the associative law, and which has an identity element, is known as a **semigroup**. A **group** is a semigroup in which every element has an inverse. The structure formed by the set of all strings on a given alphabet by the concatenation operation, is therefore a semigroup, but not a group because of the lack of inverses.

§2.4. Languages

Normally we think of languages as special systems where certain strings are given meaning. There are the natural languages with which we communicate with one other in our daily lives such as English, French or Swahili. Then there are the specialised programming languages with which we communicate with computers such as BASIC, COBOL and C#.

But there are many other specialised systems that we do not usually think of as languages. Morse Code is a language that's built on top of a natural language. Chess enthusiasts and knitters have developed specialised languages in which they describe their activities.

There are two aspects to any language – *syntax* and *semantics*. Syntax is concerned with determining whether a given string has a valid form while semantics is concerned with the meanings that are given to acceptable strings. Semantics can only be discussed in the context of a particular language but syntax is more formal and can be discussed abstractly.

The sentence "The silver prejudice accelerated within the complicated sausage" is a syntactically correct English sentence. The verbs, nouns and adjectives are all in correct positions relative to one another. But semantically it is nonsense. It lacks meaning. On the other hand the syntax of the sentence "Is near post-office where?" is incorrectly structured for English, but if a foreign tourist stopped you in the street with this question you might be able to guess that he wanted to be directed to the nearest post-office. So correct syntax is not always necessary for communication. But it certainly makes communication much easier!

Decimal notation for real numbers is a mini-language that uses the ten digits, the minus sign and the decimal point. The semantics of this language involves concepts such as powers of 10 and limits of sequences, but the syntax can be expressed by a few simple rules:

(1) If a minus sign is included it must be at the front.
(2) At most one decimal point may be included and if included it must have a digit on either side.
(3) The first digit cannot be 0 unless it is the only symbol or a decimal point follows.

In order to make our discussion as general as possible we use the term *language* to refer to any (finite or infinite) set of strings.

If A is an alphabet, A^n denotes the set of all strings of length *n* on the alphabet A.

$A^0 = \{\lambda\}$;
A^1 = A (since we identify a string of length 1 with the symbol itself);
A^n denotes the set of all strings of length n, on the alphabet A;
A* denotes the set of all strings, of any length, on *A*;
A^+ denotes the set of all non-null strings on *A*.

Example 4: If A = {a, b},
$A^0 = \{\lambda\}$,
$A^1 = \{a, b\}$,
$A^2 = \{aa, ab, ba, bb\}$,
$A^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$;
$A^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, ...\}$;
$A^+ = \{a, b, aa, ab, ba, bb, aaa, ...\}$

A **language** on an alphabet A is any subset of A*.

Example 5:
The following are a few of the (infinitely) many languages on the alphabet A = {a, b}:

(1) ∅, the empty set;
(2) {b, ab, bab};
(3) {aba, ababa, abababa, ...};
(4) all strings on {a, b} with a prime number of a's;
(5) all palindromes on A, that is, all strings which are the same in both directions;
(6) all strings on A except for abba;
(7) all strings on A (ie A* itself).

If *L* is a language on an alphabet A:

L^n denotes the set of all strings of the form $w_1 w_2 ... w_n$ where each $w_i \in L$, that is, all strings which can be made up of exactly *n* pieces, each coming from L.

L* denotes the set of all strings which can be made up of any number of pieces, each coming from L. Since the number of pieces could be zero, L* includes the null string, even if the null string doesn't belong to L.

L^+ denotes the set of all strings which can be made up of one or more pieces, each coming from L. In other words

L^+ consists of all the elements of L* except the null string, that is, unless the null string is in L.

L* is called the **Kleene closure** of *L*. The Kleene star of something roughly means "any amount of it".

Example 6: Suppose A = {x, y} and L = {xy, yx, x}. Then A^2 = {xx, xy, yx, yy} and L^2 = {xyxy, xyyx, xyx, yxxy, yxyx, yxx, xxy, xyx, xx}. While there are 9 (= 3^2) combinations for L^2 there are only 8 different elements in L^2 because xyx can be obtained in two different ways.

Example 7: If A = {0, 1} and L is the set of all strings on A which contain an even number of 1's then L^3 consists of all strings on A where the number of 1's is a multiple of 6 and
L* = L.

There are three operations we can define on the set of languages on a given alphabet.

If L, M are languages on an alphabet A then:
L ∩ M is the intersection of the two sets (as usual);
L + M is the union of the two sets, that is, the strings that are in one or both of L and M. In the context of languages it is customary to use the symbol "+" instead of "∪");

LM is {αβ| α ∈ L, β ∈ M}.

Example 8: Let $L = \{0,11\}$ and

$$M = \{\lambda, 11, 1111, 111111, ...\}.$$

Then $L \cap M = \{11\}$;

$L + M = \{\lambda, 0, 11, 1111, 111111, ...\}$;

$LM = \{0, 11, 011, 1111, ...\}$;

$L^* = \{\lambda, 0, 11, 00, 011, 110, 0011, 0110, 1100, 1111, ...\}$

$M^* = M$.

Note that $M = (11)^*$. The fact that $M^* = M$ is a particular case of the general fact that for any language L, we have $L^{**} = L^*$.

Properties: For any languages L, M:

(1) $L^* + M^* \subseteq (L + M)^*$;

(2) $(L \cap M)^* \subseteq L^* \cap M^*$;

(3) $L^* = L\,L^* + \lambda$.

Proof: (1) Suppose $\alpha \in L^* + M^*$.

Then either $\alpha \in L^*$ or $\alpha \in M^*$.

In the first case $\alpha = a_1 a_2 \ldots a_n$ for some $n \geq 0$, where each $a_i \in L$. Since each $a_i \in L + M$ it follows that $\alpha \in (L + M)^*$. Similarly in the second case.

(2) is proved similarly.

(3) The language LL^* consists of strings made up from one or more substrings from L and so is L^+, excluding the null string. Throwing in λ we obtain L^* itself.

Example 9:
Let L = {YES} and M = {NO}. Then the string YESYESNONOYES belongs to (L + M)* but not to L* + M*. This shows that we cannot guarantee equality in (1). In fact the only strings in L* + M* are those which are either repeated YES's or repeated NO's, not mixtures.

Example 10:
Let L = {xyx} and M = {xy, yx, x}. Let α = xyx. Then $\alpha \in L^* \cap M^*$ but $\alpha \notin (L \cap M)^*$ since $L \cap M$ is empty. This shows that we cannot guarantee equality in (2).

§2.5. String Substitution

Every string can be considered as a function by regarding one or more of its symbols as variables. We can then substitute any string in place of each of these variables to produce a new string.

Example 11:
The string xyx could be considered to be a function of x and y, or just of x, or just of y. Suppose we consider it to be a function of both x and y and write $f(x, y) = xyx$.
Then $f(y, x) = yxy$; $f(0, 1) = 010$, $f(★, ☎) = ★☎★$ and $f(☺, ☺) = ☺☺☺$.

We can substitute strings of more than one symbol, instead of just individual symbols. Thus $f(xy, xxx) = xy\ xxx\ xy$. Here we have separated the resulting string to make it easier to see how it was obtained by the substitution

process. But remember that these spaces should not really be there and we must eventually write $f(xy, xxx) = xyxxxxy$.

Another substitution is f(Jack, loves) = JacklovesJack ". We can also build up nests of substitutions, just as we compose functions or construct functions of a function. So, for example, $f(f(0, 1),f(1, 0)) = f(010, 101) = 010101010$.

§2.6. Formal Grammars

Usually not all strings on a given alphabet are given meaning. A language is a subset of the set of all strings on the alphabet. But how can we describe or specify a language. If the language is finite we could simply write out a complete list of all its strings. But most useful languages are infinite. How do we specify an infinite set of strings with a finite description?

The English language contains a very large number of words, but surely only a finite number. That is no doubt true, even though it would be impossible to say exactly how many. But strings of English are not just words – they are sentences, paragraphs and even whole books. While only a finite number of pieces of English have ever been written the number of possible pieces is infinite.

I can emphasise that something is good by using the word 'very'. Something even better would be "very very good". Though the shades of meaning would start to become indistinguishable if we piled one 'very' upon another, syntactically English could cope with "very very very very very good". There is no limit to the number of

times we could use the word ‘very”\’ to describe how exceptionally good something is, so potentially there are infinitely many phrases just using these two words.

In a computer program we allow expressions such as X + X + X + X. While we might prefer the more compact 4*X, syntactically X + X + X + X is permissible. And, since there is no limit to the number of X’s we could include in an expression like this, any computer language that allows such a construction must be infinite. (An actual implementation on a particular computer may impose limits due to memory size, but in the language itself there are no limits.)

So how do we describe an infinite set of strings by a finite expression? One way is to define a **grammar** for the language. This is a set of **production rules** that specify how a valid string could be constructed.

Example 12:
Take the rules:

(1) S $\rightarrow$ The N sat on the N;
(2) N $\rightarrow$ cat;
(3) N $\rightarrow$ mat.

This is a grammar for a tiny fragment of the English language. Here S stands for a sentence. We start with the symbol S. Rule (1) says that we can replace S by the string “The N sat on the N”. Here the symbol N is a place marker which in turn gets replaced using other production rules. Clearly N is meant to stand for a noun, but we don’t need to know that because production rules (2) and (3) define

what N is. The only possible N's in this language are 'cat' and 'mat'. So this grammar can only produce four sentences, of which two are "The cat sat on the mat." and "The mat sat on the cat". What are the other two?

Example 13:
Consider the grammar:

(1) S → S and S
(2) S → The N sat on the N;
(3) N → cat;
(4) N → mat.

Rule (1) is recursive and allows us to build up longer and longer strings. In this language we can say "The cat sat on the mat and the cat sat on the cat and the mat sat on the cat." Clearly even this tiny fragment of the English language is infinite.

Example 14: Consider the grammar:

(1) S → 0;
(2) S → P;
(3) S → –P;
(4) P → N;
(5) P →NT;
(6) T → TT;
(7) T → 0;
(8) T → N;
(9) N → 1;
(10) N → 2;

(11) $N \to 3$;
(12) $N \to 4$;
(13) $N \to 5$;
(14) $N \to 6$;
(15) $N \to 7$;
(16) $N \to 8$;
(17) $N \to 9$.

What language does it generate?

Rules (9) to (17) specify N to be a non-zero digit. Rules (7) and (8) allow T to be any digit. Rule (6) is recursive and allows T to be any finite strings of digits. Rules (4) and (5) specify that P is any string of digits that starts with a non-zero digit, that is, any valid representation of a positive integer. Rules (1), (2) and (3) specify that S is the usual representation of any integer, positive, negative or zero. So this is a grammar for the language of strings that represent integers in decimal notation.

§2.7. Regular Expressions

Another way of giving a finite description of certain languages is to use what are called regular expressions. These expressions themselves form a language and can be described by a formal grammar.

Consider the language $\mathfrak{R}$ on the alphabet $\{0, 1, \varnothing, \lambda, (,), *\}$ defined by the following formal grammar:

(1) $S \to (S)$;

(2) $S \rightarrow SS$;
(3) $S \rightarrow S + S$
(4) $S \rightarrow S^*$;
(5) $S \rightarrow \varnothing$;
(6) $S \rightarrow \lambda$;
(7) $S \rightarrow 0$;
(8) $S \rightarrow 1$.

Any element of this language is called a **regular binary expression**, so called because it will become a way of defining certain languages on the alphabet {0, 1}. With a larger alphabet we would include extra production rules like (7) and (8). Now we will be concentrating on languages of binary strings (0;s and 1's) so we will drop the adjective 'binary' and call the strings **regular expressions**. Any language that can be described by a regular expression will be called a **regular language**.

So a regular expression is a string built up from the primitive symbols 0, 1, λ, $\varnothing$ using the operators + and * as well as the left and right parentheses. We interpret parentheses in the usual way, + and * have their usual meanings for languages (union and the Kleene star) and 0 and 1 are the fundamental symbols. Well, actually 0 is the language consisting of just one string, and that string is just 0 (a length 1 string). The symbol 1 represents the language {1} where 1 is the other length 1 string on {0, 1}. Remember that every regular expression represents a language.

Example 15:
(1) 0 + 11 is the language {0, 11}.

(2) (0 + 11)* represents all binary strings in which each run of consecutive 1's has even length.
This is an infinite language and among its elements is 0110001111011. The shortest strings in this language are λ, 0, 00, 11, 000, 011, 110, 0000, 0011, 0110, 1100, 1111.

(3) 010(0 + 11)* represents all strings in (0 + 11)* preceded by 010, such as 0101101111.
(λ + 010)(0 + 11)* represents all strings in (0 + 11)* *possibly* preceded by 010.
Clearly this is the union of the languages could be described as 010(0 + 11)* + (0 + 11)*., such as 0101101111.

(4) (λ + 0)(10*1 + 00)*. In interpreting regular strings we interpret + as the union of two languages and * as the Kleene star. So this is the language of all binary strings which may begin with a 0 and are followed by zero or more pieces, each of which is a string of 0's (possibly none) surrounded by a pair of 1's, or a double 00.

A typical string in this language is 0100000110100111010000000. If we insert some spaces to separate the components we can see how it has been built up:

0 1000001 101 00 11 101 00 00 00

Example 16: Describe the language given by the regular expression (1 + 0)*1101.

Solution: The expression (1 + 0)* represents the language of *all* strings on {0, 1} since it describes strings made up of zero or more pieces, each of which is a 1 or a 0. Hence the language (1 + 0)*1101 describes all strings that end in 1101.

[**WARNING:** You must keep in mind that "+" here represents "or" and not arithmetic addition so you must resist the temptation to write 1 + 0 = 1.]

Example 17: Describe the language given by the regular expression (1 + 000)*.

Solution: The strings in this language are made up of any number of blocks, each of which is 1 or 000. In other words, it consists of all strings where the length of any run of consecutive 0's is a multiple of 3.

Example 18: A regular expression to describe the language consisting of just the three strings 010, 1101, 1111 is just 010 + 1101 + 1111.

Clearly we can do this for any finite language. Hence:

Every finite language is regular

Example 19: Construct a regular expression to describe the language of those binary strings, that is, strings on the alphabet {0, 1}, that contain the substring 1011.

Solution: (0+1)*1011(0+1)*

Example 20: Construct a regular expression to describe the language of all binary strings where every run of consecutive 1's has even length.
Solution: (0 + 11)*

Example 21: Construct a regular expression to describe the language of all binary strings that contain the substring 1011 and where every run of consecutive 1's has even length.
Solution: Wherever it occurs 1011 must be preceded by a further 1 and so the string must contain 11011. Such a string can therefore be broken into three blocks. The first and third blocks can be any string from the language (0 + 11)* while the middle block must be 11011. A suitable regular expression is thus (0+11)*11011(0+11)*
Note that this language is the intersection of the languages in examples 20 and 21. The fact that it too is regular is a particular instance of the following fact.

If L, M are regular languages so is $L \cap M$

The proof must wait until we've encountered Finite State Machines.

This next example is considerably harder than any of the preceding ones.

Example 22: Construct a regular expression to describe the language of all binary strings that do *not* contain the substring 1011.
Solution: One answer is ((0*11*00)*0*11*0(10)*0)*(0* + 11* + 11*0(10)* + 11*0(10)*1), but how on earth do we get such a complicated regular expression? The problem becomes very much simpler after we've learnt about Finite State Acceptors.

Notice that this language is the complement of the one in example 20. The fact that it is regular too is a particular case of the following fact.

The complement of a regular language is regular

All the languages we've considered here have turned out to be regular. Are there languages that are not regular? Indeed there are. One such non-regular language is the language of all regular expressions itself ! But how can we show this? Again, we need to wait until we have encountered Finite State Acceptors.

EXERCISES FOR CHAPTER 2

EXERCISES 2A (Languages)

Ex 2A1: If *A* is the alphabet {+,–,*} write out the elements of A^2. How big is A^3 ?

Ex 2A2: Let L be the language {ab, ba, aba}. Write out the elements of L^2. How big is L^3 ?

Ex 2A3: Let L = {alas, men, sink, two} and let
M = {a, last, ink, womens}.
Show that $L^* \cap M^*$ contains a non-null string.
[**NOTE:** These strings contain no spaces to separate words.]

Ex 2A4: Is $1000110010101 \in (10^*10^*)^3$?

Ex 2A5: Let L = (01 + 11 + 011)* and let M = 011*01.
(a) Write down all strings in L whose length is less than 6.
(b) Prove that L properly contains M (ie $M \subseteq L$ but $M \neq L$).
(c) Let A = {00, 11, 10} and B = {1, 00, 10}.
Find a string of length 2 which is in $A^* \cap B^*$ but not in $(A \cap B)^*$.

Ex 2A6: Prove that if L, M are languages then:

$$L^*M = M + L^*LM.$$

EXERCISES 2B (Regular Expressions)

Ex 2B1: Construct a regular expression that describes the language of all binary strings that have even parity (ie have an even number of 1's).

Ex 2B2: A valid date is a string of the form d/m/y where d is the day of the month, m is the month (1 to 12) and y is the final two digits of the year. We don't allow leading zeros so that 06/07/2009 should be written 6/7/09.
Write a regular expression, as short as possible, to represent the set of all valid dates in October 2009.

Ex 2B3: Find a regular expression that represents all strings on {0, 1} which start with 1011 and have an even number of 1's and exactly two 0's.

Ex 2B4: Write down three strings of length 4 in the language described by the regular expression:

$$1(11 + 0)^* + 1011.$$

Ex 2B5: (a) Find a regular expression for the set of all binary strings which contain the substrings:

000 and 11.

(b) Find a regular expression for the set of all binary strings which contain the substrings:

001 and 11.

Ex 2B6: Find a regular expression for all binary strings that do not contain the substring 111.

Ex 2B7: (a) Find three strings in the regular language:

$$(\lambda + 11)^*1^*0 + 1^*(10 + \lambda),$$

including one ending in 0 and one ending in 1.
(b) Find a simpler regular expression for this language.

Ex 2B8:
(a) Prove that $(11 + 111)^* = \lambda + 11\ 1^*$.
(b) Let L = $(111111 + 1111111111)^*$ and
M = $1111111111111111(11)^*$.
Prove that every string in M is also in L.
Find all the strings in L which are not in M.

Ex 2B9: Find a regular expression for all binary strings which begin with 111 and do not contain 000.

Ex 2B10: (a) Find a regular expression for all binary strings which start with 110 and which do not contain the string 1111.
(b) Let α^n denote the string of *n* successive 1's.
Find all strings which are in α^{2*} but not in $(\alpha^4 + \alpha^{10})^*$.
(c) Find a grammar for the language of all valid parenthesis strings. These are those strings on the alphabet {(,)} where the parentheses are nested validly.

Ex 2B11: Show that:

$$(\lambda + 0)(10^*1 + 00)^* = 0^*(10^*1(00)^*)^*.$$

SOLUTIONS FOR CHAPTER 2

Ex 2A1: A^2 = {**++**, **+–**, **+***, **–+**, **––**, **–***, ***+**, ***–**, ****}, A^3 has 27 elements.

Ex 2A2: A^2 = {abab, abba, ababa, baab, baba, baaba, abaab, abaaba}. NOTE ababa occurs twice so there are only 8 distinct elements in A^2.
A^3 has only 21 elements:
ababab, ababba, abababa, abbaab, abbaba, abbaaba, ababaab, ababaaba, baabab, baabba, baababa, babaab, bababa, babaaba, baabaab, baabaaba, abaabab, abaabba, abaababa, abaabaab, abaabaaba

Ex 2A3: L* ∩ M* contains "alastwomensink"

Ex 2A4: Yes. In fact $(10^*10^*)^3$ consists of all binary strings that start with 1 and have exactly 6 1's altogether.

Ex 2A5: (a) λ, 01, 11, 011, 0111, 0101, 01011, 1101, 1111, 11011, 01101, 01111.
(b) The null string is in L but not in M, so M ≠ L.
A typical string in M has the form 01^m01 where $m \geq 1$.
If m is even, say $m = 2k$, then:

$$01^{2k}01 = (011)(11)^{k-1}(01) \in L.$$

If m is odd, say $m = 2k+1$ then

$$01^{2k+1}01 = (01)(11)^k(01) \in L.$$

Hence every string in M belongs to L and so M ⊆ L.
(c) 11

Ex 2A6: M + L*LM = (λ + L*L)M = L*M since L*L consists of all strings in L* except λ.

Ex 2B1: 0*(10*10*)*. Note also that the first 0* allows for possible leading 0's.

Ex 2B2: October has 31 days. So a suitable regular expression is:
((λ+1+2)(1+2+3+4+5+6+7+8+9)+(1+2+3)0+31)/10/09

Ex 2B3: Such a string will have the form $10111^m\ 01^n$ where $m + n$ is odd. Hence either m is even and n is odd, in which case the string can be expressed as 1011(11)*01(11)*, or m is odd and n is even, in which case the string can be expressed as 10111(11)*0(11)*. A suitable regular expression is thus 1011(11)*01(11)* + 1011(11)*10(11)*, which can be simplified to 1011(11)*(01+10)(11)*.

Ex 2B4: 1110, 1000, 1011.

Ex 2B5: (a) (0 + 1)*000(0 + 1)*11(0 + 1)*
+ (0 + 1)*11(0 + 1)*000(0 + 1)*.
(b) (0 + 1)*001(0 + 1)*11(0 + 1)*
+ (0 + 1)*11(0 + 1)*001(0 + 1)* + (0 + 1)*0011(0 + 1)*.

Ex 2B6: (0 + 10 + 110)*(λ + 1 + 11).

Ex 2B7: (a) eg 10, 110, 11; (b) 1*(λ + 0).

Ex 2B8: (a) LHS = all strings built up from 11 and 111. The result follows from the fact that the non-negative integers which can be expressed in the form $2h + 3k$ consist of all except 1. This is the set described by the RHS.

(b) By induction, if the integer $n \geq 8$ then n can be expressed in the form $3h + 5k$.

Thus $\{m \mid m = 6h + 10k$ for some integers $h, k\} = \{0, 6, 10, 12\} \cup \{2n \mid n \geq 8\}$ and so the strings in M are in L. The strings in L which are not in M are:

λ, 111111, 1111111111, 111111111111.

Ex 2B9: Consider such a string. It begins with 111. Let's divide the rest, if any, into blocks after each subsequent 1. Since we can't have three 0's in a row, these blocks must be just 1, or 01, or 001. And we can have any number of these.

So consider the regular expression

$$111(1 + 01 + 001)^*.$$

But this doesn't allow for a possible 0 or 00 at the end. So we must write $(\lambda + 0 + 00)$ as a last block. A complete regular expression for this language is thus:

$$111(1 + 01 + 001)^* (\lambda + 0 + 00)$$

Ex 2B10: (a) $110(0 + 10 + 110 + 111\ 0)^*$

(b) The strings in α^{2*} are all strings of 1's of even length. Those in $(\alpha^4 + \alpha^{10})^*$ are those whose length is expressible in the form $4h + 10k$ for some non-negative integers h, k. Every even number from 8 on is so expressible but 2 and

6 are not. So the strings in α_2* which are not in $(\alpha^4 + \alpha^1{}_0)$* are precisely 11 and 111111.

[To show that every even number $N \geq 8$ is expressible as $4h + 10k$ we argue thus.
If $N = 4n$, we can write it as $4n + 0k$.
If $N = 4n + 2$ we can write it as $4(n - 2) + 10$. If $N \geq 8$ then $n \geq 2$ so $n - 2$ is indeed a non-negative integer.]

(c) $S \rightarrow (S)$; $S \rightarrow SS$; $S \rightarrow \lambda$.

Ex 2B11: $(10*1 + 00)*$ consists of all strings of the form $0^{\&}10*10^{\&} \ldots 10*10^{\&}$ where each $0^{\&}$ represents an even number of 0's. The factor $\lambda + 0$ changes the first $0^{\&}$ to 0*. Now $0*10*10^{\&} \ldots 10*10^{\&}$ can be broken up as $0*(10*10^{\&}) \ldots (10*10^{\&})$ and so the language is $0*(10*1(00)*)*$.