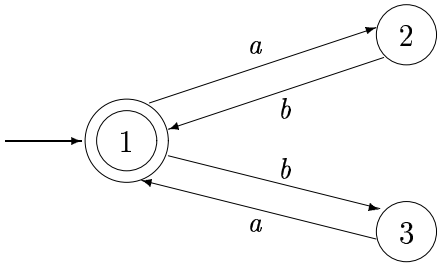
Assessed Assignment 3 - SOLUTIONS

Informatics 1a: Computation and Logic

1. Consider the following finite state acceptor over the alphabet $\{a, b\}$:



(a) You had to test each of the following strings *accepted* or *rejected* by the machine above. The answers are:

STRING	ACCEPT/REJECT
ab	accept
$baab$	accept
aba	reject
$abab$	accept
$ababbb$	reject
$aabb$	reject
$abbabaab$	accept
ϵ	accept

(b) How many strings of length 6 are accepted by this machine? List them here:

Answer: The strings of length 6 accepted by the machine are *ababab*, *ababba*, *abbaab*, *abbaba*, *baabab*, *baabba*, *babaab*, *bababa*.

(c) When the machine is in state 2, it has seen one more a than b's. So, we can say that state 2 *implies* that the machine has seen one more a than b's.
What is implied by state 3, in terms of the number of a's and b's?

Answer: If we are talking *in terms of the number of a's and b's*, then state 3 implies that we have seen one more b than a's.

What is implied by state 1, in terms of the number of a's and b's?

Answer: If we are talking *in terms of the number of a's and b's*, then state 1 implies that we have seen the same number of a's and b's.

(d) What generalisation can you make about the strings that are accepted by this machine? In other words, what property is common to all strings accepted by this machine? Aim to make the strongest statement you can.

Answer: The strongest property that I can make about this machine is:
The FSM accepts the set of strings over $\Sigma = \{a, b\}$ that are made up of some number of copies of ab's and ba's concatenated together.

An *alternative* (correct) answer is:
The FSM accepts a string if and only if:
- it contains the same number of a's as it does b's, and
- it contains no substring of 3 adjacent a's or 3 adjacent b's, and
- the leading character is not the same as the second character, and the last character is not the same as the second last character.

An *alternative* (not a strong property) is:
Every string accepted by FSM has the same number of a's as b's.
(lower score for this answer).

(e) Is every string over alphabet $\{a, b\}$ that has this property accepted by the machine? In other words, take your answer to the last question, call it ϕ and ask yourself if every string over $\{a, b\}$ which has property ϕ is accepted by the machine.

Answer: Partly depends on what the student wrote for the last question.
If the student gave either of the first two answers above (both strong properties) then the answer now is "YES".
As justification (for the first answer), notice that we start at an accept state. Then whenever we read an ab pair, we transition first to state 2, then back to state 1, which is the accept state. Alternatively, if we read a ba pair, we transition first to state 3, then back to state 1, which is the accept state.
So whenever we read either a ab or a ba, we are back to the start state, which is an accept state. So any string consisting of a sequence of ab's and ba's is accepted by our machine.
Justifying the second property is similar but a bit trickier.

If the student gave the third answer above, the answer here is NO. For example, string $aabb$ has the same number of a's as b's but is not accepted by the FSM.

(f) Is the above machine deterministic or non-deterministic? Justify your answer.

Answer: Two possible answers (because of strict/non-strict determinism):
(a) Yes, the machine is deterministic - for every state, every symbol, we have *at most one* transition leaving the state labelled with that symbol. Also there are no ϵ-transitions.
(b) No, the machine is not (strictly) deterministic - there are some (state, symbol) pairs which do not have any transition defined for them. For example, state 3 has no transition for symbol b.

I gave full marks for either answer (with the justification).
If you just said "deterministic" or "non-deterministic" with no justification, I just gave 1 mark.

2. Consider a language over alphabet $\{a, b\}$ with the following property — every string consists of an *even* number of a's and an *even* number of b's.

   We can define a finite state machine which accepts such a language using just *four* states:

   state 1 the machine has seen an *even* number of a's and an *even* number of b's

   state 2 the machine has seen an *even* number of a's and an *odd* number of b's

   state 3 ...

   state 4 ...

   (a) What other properties involving the numbers of a's and b's in the string will be useful in constructing this machine?

   Answer: State 3: The machine has seen an odd number of a's and an even number of b's.

   State 4: The machine has seen an odd number of a's and an odd number of b's.

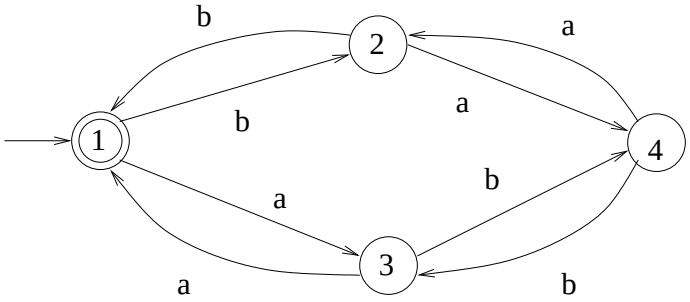
   (I don't care which order you give these answers in (obviously)).

   (b) Draw out a finite state acceptor with these four states which accepts a language whose strings consist of an even number of a's and an even number of b's.

   Note 1: consider zero to be an even number

   Note 2: your machine should accept *at least* the following strings: *aabbbb*, *abbabb*, *babababa*.

   Answer: Here is an FSM with the four states described, to accept strings containing an even number of a's and also an even number of b's.



   (c) Does your machine accept *only* strings which contain an even number of a's? Justify your answer.

   Answer: YES. There is no way a string with an odd number of as could be accepted. That is because when we see our first a (before this stage we are guaranteed to be either in state-1 or state-2) we will enter one of the states state-3 or state-4, both non-accepting. The only way we can exit, either to state-1 or state-2, is to see another a. Again, we move between the set of states {state-1, state-2} and {state-3, state-4} depending on whether we saw an even number or an odd number of a's. Our only chance of acceptance is to end up in

state-1. Therefore we certainly need to see an even number of a's (if we saw an odd number we would end up either in state-3 or state-4).

(d) Does your machine accept *all* strings which contain an even number of a's? Justify your answer.

Answer: NO. One example of a string which has an even number of a's is aab. However, by starting at state 1, and reading this string, we transition to state-3 (on reading the first a), then to state-1 (on reading the second a), and then to state-2 (on reading the b). Then we stop at state-2, which is not an accept state. So the string is rejected (even though it has an even number of a's).

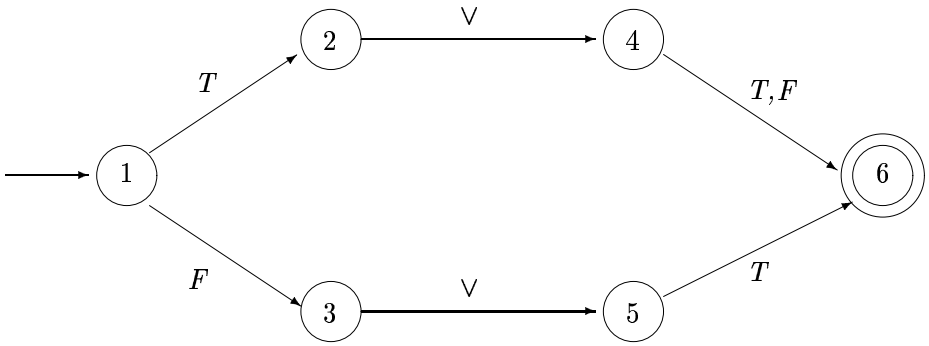
(e) Does your machine accept *only* strings which contain an even number of b's? Justify your answer.

Answer: YES. The justification is similar to the justification of 2(c).

(f) Does your machine accept *all* strings which contain an even number of b's? Justify your answer.

Answer: NO. One example of a string with an even number of b's which is not accepted is abb. Similar argument to 2(d) to check this.

3. Consider the following finite state machine over the alphabet $\{T, F, \vee\}$:



Note that when two transitions go from state s_1 to state s_2 carrying symbols x and y, we can draw this as a single arc labelled x, y.

(a) List all the strings which are accepted by this machine?

Answer: $T \vee T$, $T \vee F$, $F \vee T$.

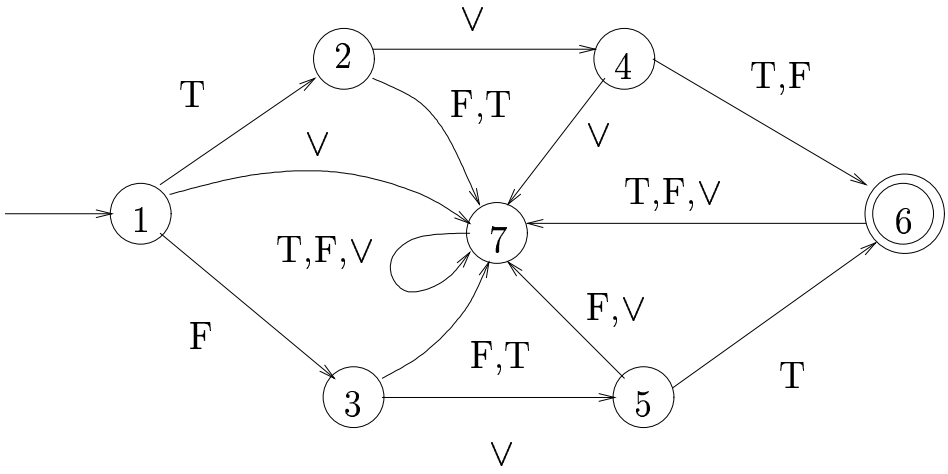
(b) Is this language familiar? Clue: think about what you learned in logic.

Answer: The language resembles the truth table for disjunction in propositional logic i.e. $X \vee Y$ is accepted by the machine if and only if it is a well-formed (non-embedded) formula which evaluates to true.

(c) The transition relation of the machine is not strictly a *function*, because it is not the case that there is *exactly one* transition from each state for each symbol in the alphabet (for some states, and some symbols, there is no 'next state'). For example, state 1 has no transition labelled $\vee$.

We can turn this machine into a one which has a well-defined transition function, simply by adding one or more 'sink' states i.e. a sink state has no transitions *to* other states, only to itself.

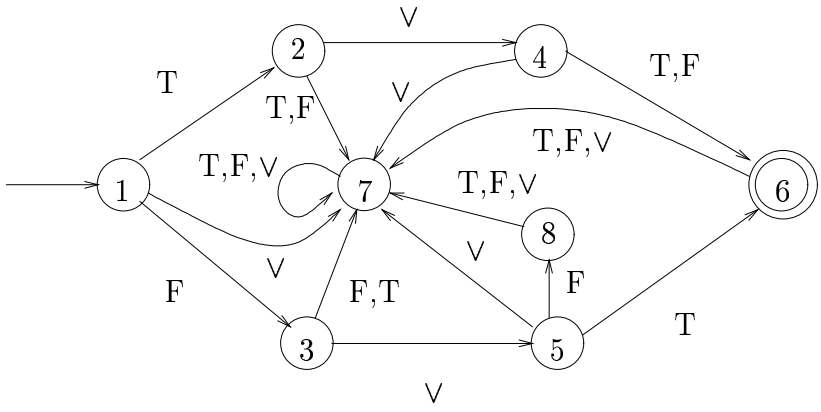
Answer: Here is the new equivalent FSM with the sink state.



(d) Unfortunately, using a single sink state like this does not allow us to distinguish between strings of symbols which are ill-formed propositional logic (e.g. $\vee TF$) and well-formed expressions which evaluate to false (e.g. $F \vee F$).

Convert the above machine into a one with a well-defined transition function, using *two* sink states, one for ungrammatical input, and the other for grammatical input which evaluates to false.

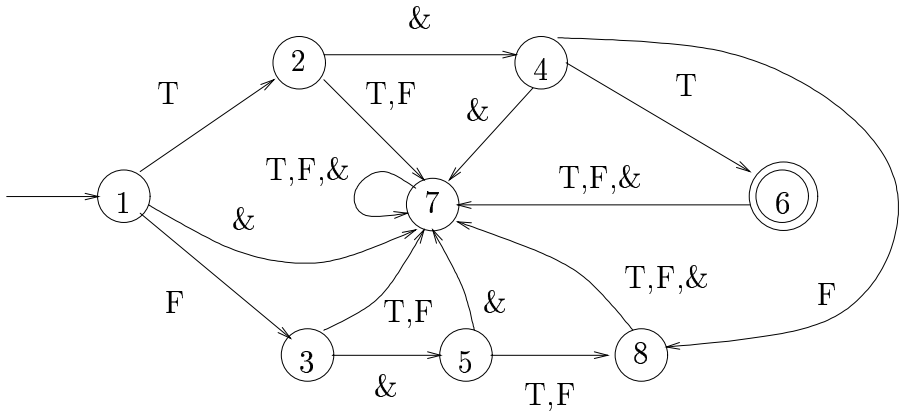
Answer: Here is the answer. State-7 is for ungrammatical input, and state-8 is for grammatical (un-embedded) input which evaluates to False.



(e) Draw a deterministic finite state machine with a well-defined transition function, over the alphabet $\{T, F, \&\}$, which accepts only those strings which are

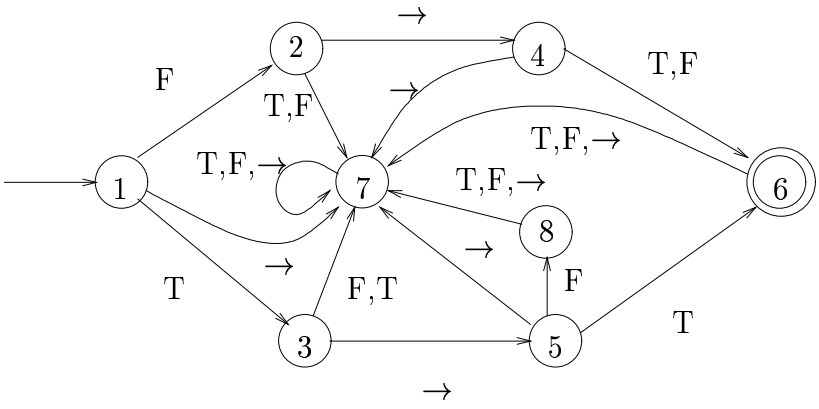
grammatical expressions of propositional logic, containing a single instance of the conjunction connective &, and which evaluate to true. Use *two* sink states to distinguish ungrammatical input from grammatical input evaluating to false:

Answer: Here is the answer. State-7 is for ungrammatical input, and state-8 is for grammatical (un-embedded) input which evaluates to False.



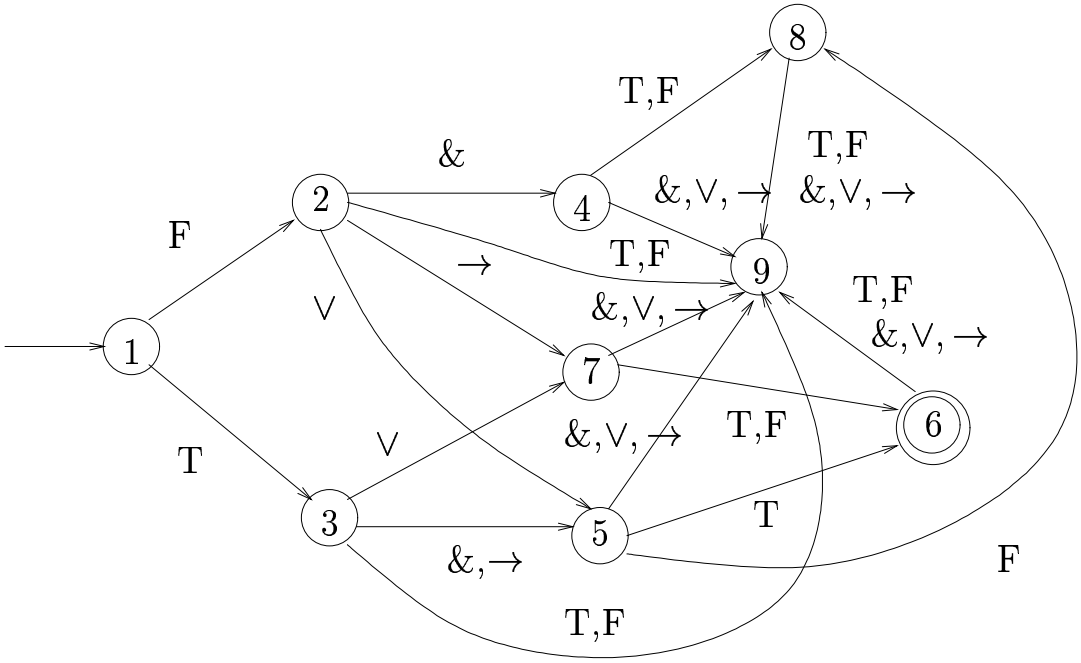
(f) Draw a deterministic finite state machine over alphabet $\{T, F, \rightarrow\}$ which accepts only those strings which are grammatical expressions of propositional logic, containing a single instance of the implication connective $\rightarrow$, and which evaluate to true. Use *two* sink states to distinguish ungrammatical input from grammatical input evaluating to false:

Answer: Here is the answer. State-7 is for ungrammatical input, and state-8 is for grammatical (un-embedded) input which evaluates to False.



(g) Take the three deterministic machines you have already designed (for $\vee$, & and $\rightarrow$), and combine them into a single deterministic finite state acceptor, with a well-defined transition function, over the alphabet $\{T, F, \vee, \&, \rightarrow\}$, which accepts only those strings which are grammatical expressions of propositional logic, containing exactly one connective, and which evaluate to true. Include two sink states, as before, to distinguish ungrammatical input from false input. Note that you do not need to worry about embedded expressions such as $T \rightarrow (T \& F)$.

Answer: Here is the answer. Notice that state-8 is the state for grammatical (unembedded) input which is False, and state-9 is the state for ungrammatical input:



Assessed assignment 3 was written by Mark McConville and Mary Cryan. If you have any questions about it, post them to the `inf1a` *newsgroup.*