First a general remark to explain the terminology 'closed under operation ...'. We say, a collection \( C \) of languages is closed under a binary operation \( \circ \) of languages, if for all \( L_1, L_2 \in C \), it holds that the language \( L_1 \circ L_2 \) is again in \( C \). Similarly, if \( \circ \) is a unary operation, we say \( C \) is closed under \( \circ \), if for all \( L \in C \), we have that \( L^\circ \in C \).

3.14 With the above notation, In this exercise, \( C \) is the collection of all decidable languages and in (a) the operation \( \circ \) is set union, in (b) it is concatenation and in (c) is intersection. These are all binary operations on languages, i.e. they take two arguments. In (c) and (d) we have two examples of \( \circ \) being a unary operation.

(a) Having explained the terminology, all this excercise asks of us is to show the following. For any decidable languages \( L_1 \) and \( L_2 \), show that the language \( L_1 \cup L_2 \) is again decidable. Let us now proceed with this.

Since \( L_1 \) is decidable we have a 1-tape deterministic TM \( M_1 \) such that for all inputs \( x \),

1. if \( x \in L_1 \) then \( M_1(x) \) accepts
2. if \( x \notin L_1 \) then \( M_1(x) \) rejects

Similarly, we have a machine 1-tape deterministic TM \( M_2 \) such that for all inputs \( x \),

1. if \( x \in L_2 \) then \( M_2(x) \) accepts
2. if \( x \notin L_2 \) then \( M_2(x) \) rejects

To show \( L_1 \cup L_2 \) is decidable we must build a TM \( M \) that halts on all inputs, with \( L(M) = L_1 \cup L_2 \). There are at least two possible solutions that work.

**Solution 1:** We take \( M \) to be a two tape TM that does the following:

1. Copy input \( x \) to 2nd tape.
2. Simulate \( M_1(x) \) on the 2nd tape, remember the result in the finite state control
3. Copy input \( x \) to 2nd tape.
4. Simulate \( M_2(x) \) on the 2nd tape, remember the result
5. **ACCEPT** if one of the above simulations ended in accepts, otherwise **REJECT**.

Verify for yourself that for \( x \in L_1 \cup L_2 \), \( M(x) \) accepts, and for \( x \notin L_1 \cup L_2 \), \( M(x) \) rejects.

**Solution 2:** We take \( M \) to be a nondeterministic 1-tape machine that in its start state for each input symbol, and makes two nondeterministic branches, and doesn’t move the head. One branch goes to the start state of machine \( M_1 \), the other goes to the start state of \( M_2 \). Remove the accepts and reject states of \( M_1 \) and \( M_2 \), any transition into these removed states have to be redirected to the accept or reject state of \( M \) itself.

First observe that the machine \( M \) halts on all its inputs, because \( M_1 \) and \( M_2 \) do. Suppose \( x \in L_1 \cup L_2 \), so \( M_i(x) \) accepts for some \( i \in 1, 2 \). Hence there is an accepting path in \( M \) on input \( x \), namely nondeterministically jump to \( M_i \) from the start state and then the computation is identical to \( M_i(x) \), which ends in accept. If \( x \notin L_1 \cup L_2 \), observe that all computation paths of \( M(x) \) will end in reject. Hence by corollary 3.12 of the book, we conclude \( L_1 \cup L_2 \) is decidable.

(b) Again let \( M_i \) be a deterministic 1-tape TM that decides \( L_i \), for \( i = 1, 2 \) respectively. The following is the outline of a deterministic 3-tape machine \( M \) for \( L_1 L_2 \):

**input:** \( x \)

FOR each division \( x = uv \) of the input DO

1. copy \( u \) to the 2nd tape and \( v \) to the 3rd tape
2. Simulate \( M_1(u) \) on the 2nd tape
3. Simulate \( M_2(v) \) on the 3rd tape

1
4. IF both simulations ended in accept THEN HALT & ACCEPT
REJECT

Let's verify the machine does what it is supposed to do. First of all, observe M always halts. If \( x \in L_1L_2 \), then there exist \( u \in L_1 \) and \( v \in L_2 \) such that \( x = uv \). Hence once the for loop of \( M \) reaches this division, in the body both simulations end in accept, so we accept on line 4. If \( x \not\in L_1L_2 \), for any \( u,v \) with \( x = uv \), we have that \( u \not\in L_1 \) or \( v \not\in L_2 \). Hence for each iteration of the for loop, one of the simulations rejects, so we never accept on line 4. Since \( M \) always halts, we must reach the REJECT on the last line.

Remark: Again there is an alternative solution, using nondeterminism. Instead of deterministically checking all divisions \( x = uv \), 'nondeterministically guess' a division and then simulate \( M_1(u) \) and \( M_2(v) \), and accepts if both accept otherwise reject. To 'nondeterministically guess' a division one does the following: Have transition \( \delta(q,a) = (q,a,R) \) for \( a \in \Sigma \), and a transition from \( q \) reading symbols from \( \Sigma \) to a routine where we copy everything up to where the head is now to the 2nd tape, and everthing after it to the 3rd tape. So we will have a computation path for each division \( x = uv \).

(c) Let \( M \) decide \( L \). The way to do this one is similar to (b), we simply check for all division \( x = z_1...z_t \) of the input whether all \( z_i \) are in \( L \) using simulation of \( M(z_i) \). If there is such a division, we accept otherwise after checking all finitely many possible divisions, we reject. The resulting procedure will always halt, and accepts if and only if \( x \in L^* \).

(d) Let \( M \) be a deterministic TM deciding \( L \). Let \( M' \) be the TM obtained from \( M \) by swapping the accept and reject states. First observe, \( M' \) always halts. Secondly, \( M'(x) \) accepts \( \iff M(x) \) rejects \( \iff x \not\in L \) \( \iff x \not\in \overline{L} \).

(e) Again let \( M_i \) be a deterministic 1-tape TM that decides \( L_i \), for \( i = 1,2 \) respectively. The following is the outline of a deterministic 2-tape machine \( M \) for \( L_1 \cap L_2 \):

1. Copy input \( x \) to 2nd tape.
2. Simulate \( M_1(x) \) on the 2nd tape, remember the result in the finite state control
3. Copy input \( x \) to 2nd tape.
4. Simulate \( M_2(x) \) on the 2nd tape, remember the result
5. ACCEPT if both of the above simulations ended in accepts, otherwise REJECT.

Verify for yourself that for \( x \in L_1 \cap L_2 \), \( M(x) \) accepts, and for \( x \not\in L_1 \cap L_2 \), \( M(x) \) rejects.

3.15 In this exercise, in reference to my initial remark on page 1, \( C \) will be the collection of all Turing acceptable languages.

(a) Observe that we can't just use solution 1 of 3.14(a), because for an input \( x \not\in L_1 \), but \( x \in L_2 \), the simulation of \( M_1(x) \) on line 2, might never halt, so the algorithm never gets to the point of finding that \( M_2(x) \) accepts.

The second solution however, using nondeterminism, does work. Let's verify this. Let \( M_i \) be a 1-tape deterministic TM with \( L(M_i) = L_i \), for \( i = 1,2 \). Let \( M \) be the constructed from \( M_1 \) and \( M_2 \) in the same way as before. Suppose \( x \in L_1 \cup L_2 \). So \( x \in L_i \) for some \( i \in \{1,2\} \). On input \( x \), if \( M \) guesses to go into the \( M_i \) subroutine, eventually we reach an accept, because \( M_i \) accepts every word in \( L_i \). Conversely, for \( x \not\in L_1 \cup L_2 \), neither \( M_1(x) \) and \( M_2(x) \) accept. Hence \( M \) will never accept. By corollary 3.11 from the book, we conclude \( L_1 \cup L_2 \) is Turing acceptable.

As a final remark, we can modify solution 1 as follows to build a 3-tape TM \( M \) that accepts \( L_1 \cup L_2 \):

1. copy input \( x \) to both the 2nd and 3rd tape
2. Simultaneously simulate \( M_1(x) \) on the second tape, and \( M_2(x) \) on the 3rd tape.
3. Wait until one of the simulations halts and accepts
4. ACCEPT
Verify for yourself that this machine will accept an input $x$ if and only if $x \in L_1 \cup L_2$.

(b) Again, the deterministic solution of 3.14(a) suffers from the same kind of defect as we pointed to in 3.15(a). The nondeterministic solution of guessing a division as in 3.14(b) does work however, and convince yourself, that indeed that if $M_1$ and $M_2$ accept $L_1$ and $L_2$, respectively, that the nondeterministic machine as constructed in 3.14(b), indeed accepts $L_1L_2$. However, let me use this exercise to showcase another technique, namely repeatedly simulating machines for some fixed number of steps $s$, where we take the number of steps large each iteration.

Let $M_1$ and $M_2$ be deterministic 1-tape machines such that $L(M_i) = L_i$ for $i = 1, 2$. We are going to build a 4-tape machine $M$. The 2nd tape is for simulation of $M_1$. The 3rd tape is for simulation of $M_2$. On the 4th tape we keep a counter $s$, which we initialize at 1. $M$ will do the following:

input: $x$

$s = 1$

REPEAT
FOR each division $x = uv$ of the input DO
{
1. copy $u$ to the 2nd tape and $v$ to the 3rd tape
2. Simulate $M_1(u)$ on the 2nd tape for $s$ steps
3. Simulate $M_2(v)$ on the 3rd tape for $s$ steps
4. IF both simulations ended in accept THEN HALT & ACCEPT
}
$s = s + 1$
FOREVER

Convince yourself, that indeed we can do the simulation on line 2 and 3 for a fixed number of steps, in the Turing Machine formalism. (Namely, one would $s$ symbols on the 4th tape and let the 4th head move right for each step of $M_i$, until you hit Blank, after which you go out of the simulation routine).

Let’s verify $L(M) = L_1L_2$. Suppose $x \in L_1L_2$. Hence there are $r \in L_1$ and $t \in L_2$ such that $x = rt$. Since $r \in L_1$, $M_1(r)$ accepts eventually, say after $k$ steps. Likewise $M_2(t)$ accepts after some number of steps $l$. Observe, each iteration of the for loop only takes some finite number of steps. Now, either $M$ will enter the for loop with $s = \max(l, k)$, or it must have accepted before that. In the latter case, we conclude $x$ is accepted. In the former case, observe that, since each iteration of the for loop takes finite amount of time, we will try after some finite number of steps the subdivision $x = rt$. That is we enter the for loop with $u = r$ and $v = t$. Since $s \geq k$, the simulation $M_1(r)$ will reach it accept. Similarly, $M$ will see that $M_2(t)$ accepts. Hence on line 4, $M$ accepts.

Conversely, if $x \notin L_1L_2$, any subdivision $x = uv$ has $u \notin L_1$ or $v \notin L_2$. Hence the criteria on line 4 for accepting will never be reached. Hence $M$ will never accept $x$.

(c) Let Suppose $M$ is a TM with $L(M) = L$. We can use the same technique as in (b) to build a 3-tape TM $M'$ that accepts $L'$. The machine looks like this:

input: $x$

int $s = 1$;

boolean all_accept;

REPEAT
{
FOR each subdivision $x = z_1...z_t$ DO
{
    all_accept = TRUE
    FOR $i = 1$ to $t$ DO
    {
1. copy $z_i$ to the 2nd tape
2. Simulate $M(z_i)$ for $s$ steps
}
3. IF $M(z_i)$ on line 2 did not accept
4. THEN all_accept = FALSE
} IF all_accept == TRUE
THEN ACCEPT
}
s = s + 1
FOREVER

Verify for yourself, similarly as in (b), that if $x \in L^*$ then $M'(x)$ accepts, and if $x \not\in L^*$, then $M'(x)$ does not accept.

(d) Let $L(M_1) = L_1$ and $L(M_2) = L_2$. We have to build a TM $M$ that accepts $L_1 \cap L_2$. Here it is (2 tapes):

input: $x$
1. copy input to 2nd tape
2. Simulate $M_1(x)$ on tape 2, if it rejects LOOP FOREVER
3. copy input to 2nd tape
3. Simulate $M_2(x)$ and if it accepts, ACCEPT, otherwise LOOP FOREVER

Let’s verify the machine $M$ does what it is supposed to do. For $x \in L_1 \cap L_2$, on lines 2 and 3 the simulations will both accept, which leads $M$ to accept. For $x \not\in L_1 \cap L_2$, if $x$ is not in $L_2$ we will never satisfy the criteria of line 3 which makes $M$ accept, so $M$ does not accept. If $x \in L_2$, then it must be that $x \not\in L_1$, so $M$ never get past line 2, and thus doesn’t accept.

3.19 The issue this excercise undoubtedly is trying to point out is the following. If God exists, then $A = \{1\}$, which is decidable. If God does not exists, then $A = \{0\}$, which is decidable. In both cases, $A$ is a decidable language. So eventhough one doesn’t know what $A$ actually is, you know it is decidable. However, this way of reasoning does assume, that you can actually ascribe meaning to the statement ‘God exists’, which in my book requires a definition of what exactly the term ‘God’ refers to. Otherwise, A would be ill-defined.