1. Turing recognizable languages:

**Union:** For any two Turing-recognizable languages L1 and L2, let M1 and M2 be the TMs that recognize them. We construct a TM M0 that recognizes the union of L1 and L2: On input w:

1. Run M1 and M2 alternately on w step by step. If either accepts, accept. If both halt and reject, reject.

If either M1 or M2 accepts w, M0 accepts w because the accepting TM arrives to its accepting state after a finite number of steps. Note that if both M1 and M2 reject and either of them does by looping, then M0 will loop as well.

**Concatenation:** Let K and L be two Turing recognizable languages, and let MK and ML indicate the Turing machines that recognize K and L respectively. We construct a non-deterministic Turing machine MKL that recognizes the language KL.

1. Non-deterministically input w into w1 and w2
2. Run MK on w1. If it halts and rejects, reject.
3. Run ML on w2. If it accepts, accept. If it halts and rejects, reject.

**Star**: For a Turing recognizable language L, construct a nondeterministic Turing machine ML∗ that recognizes L ∗.

1. On input w, non-deterministically cut w into parts w1w2.
2. Run ML on w(i) for all i. If ML accepts all of them, accept. If ML halts and rejects for any i, reject.

If there is a way to cut w into strings w1w2, such that each w(i) ∈ L, then there is a computation path in ML∗ that accepts w in a finite number of steps.

**Intersection**: Let K and L be two Turing recognizable languages, and let MK, ML, MK∩L indicate the Turing machines recognizing K, L, K ∩ L respectively. We use MK and ML to construct MK∩L.

1. On input w, run MK on w. If it halts and rejects, reject. If it accepts, go to step 2.
2. Run ML on w. If it halts and rejects, reject. If it accepts, accept.

MK∩L accepts a string w only if both MK and ML accept, thus w belongs to K ∩ L.

1. Turing-decidable languages

**Union:** For any two decidable languages L1 and L2, let M1 and M2 be the TMs that decide them. We construct a TM M0 that decides the union of L1 and L2: On input w:

1. Run M1 on w. If it accepts, accept.
2. Run M2 on w. If it accepts, accept. Otherwise, reject.

M0 accepts w if either M1 or M2 accepts it. If both reject, M0 rejects.

**Concatenation**: Let K, L be decidable languages. The concatenation of languages K and L is the language KL = {xy|x ∈ K and y ∈ L}. Since K and L are decidable languages, it follows that there exist Turing machines MK and ML that decide the languages K and L. In order to prove that KL is decidable, we can construct a Turing machine that decides KL. 4 This machine, MKL can use the machines MK and ML to decide if a string is in KL or not. Consider an input string w. We need to decide if w is of the form xy for x ∈ K and y ∈ L. If so, there must be a position at which we can partition w into x and y. Since there are only finitely many ways to partition the string, we can try all possibilities and accept if there is such a partition and reject otherwise.

1. On input w, non-deterministically partition w into strings xy.
2. Input x to MK and y to y on ML.
3. Accept if both MK and ML accept, else reject. If there is an accepting computation path, then we have found a successful split and the string is in KL. If all computation paths reject, then the string is not in KL. In either case, it is easy to see that the machine MKL halts.

Thus, KL is decidable.

**Star**: For a language L, L ∗ = {x ∈ L∪LL∪LLL…} all strings obtained by concatenating L with itself, and so on. To show that L ∗ is decidable, the idea is similar to the previous solution. We want to find cuts of the input string w, such that each of them is accepted by the TM ML that decides L. Let ML∗ be the machine that that decides L ∗.

1. On input w : For each way to cut w into parts w1w2…wn
2. Run ML on w(n) for i = 1…n.
3. If ML accepts each of the strings we accept.
4. If all cuts have been tried without success, reject.

**Intersection**: Let K and L be two Turing decidable languages, and let MK and ML indicate the Turing machines deciding K and L respectively. Let MK∩L indicate the Turing machine deciding K ∩ L.

1. On input w to MK∩L,
2. Input w to MK.
3. If MK rejects, reject.
4. Else Input w to ML.
5. If ML accepts, accept. Else reject.
6. Proof: Assume LA is decidable. Suppose H is a decider for LA. On input <M>, where M is a Turing machine that accepts if given the input 001, H halts and accepts if M accepts the string it is given. H halts and rejects if M does not accept the input string. Then, a new Turing machine must be constructed that has H as a subroutine. This new Turing machine will be referred to as D. D calls H to determine if M has been accepted. If H accepts, the output is shown as reject, and if H rejects, the output is shown as accept. Then, D is run with its own description, accepting if rejected and rejecting if accepted. Since this is a contradiction, the language is undecidable. This has been proven in ATM, as a language that takes a Turing machine and it's accept state will be undecidable.
7. Proof: Show that the language can be deconstructed to be ATM, and the proof is done. However, in this case, the extra step of having a reject could be difficult. In the first language, it easily reduces to ATM. However, this has more information. Since the rejected string is included in the first language anyway, as only one string is accepted, this language also reduces to ATM. Therefore, it is undecidable.