```
1.
By reflexivity and A \subseteq AB, AB -> A.
Apply Augmentation on AB -> A with B, AB -> AB.
Apply Augmentation on AB -> C with A, AB -> AC.
Apply Transitivity on AB -> AB and AB -> AC, AB -> AC.
Apply Augmentation on A -> D, AC -> CD.
Apply Transitivity on AB -> AC and AC -> CD, AB -> CD.
Apply Transitivity on AB -> CD and CD -> EF, AB -> EF.
Apply Reflection on F \subseteq EF, EF -> F.
Apply transitivity on AB -> EF and EF -> F, AB -> F.

2.
Premise: X -> Y and YW -> Z.
Assume r_1[XW] = r_2[XW]
      By def. and r_1[XW] = r_2[XW], r_1[X] = r_2[X].
      Using X -> Y, we conclude r_1[Y] = r_2[Y].
      By def., r_1[YW] = r_2[YW].
      Using YW -> Z, we have r_1[Z] = r_2[Z].
      By def, XW -> Z.
Hence, XW -> Z.

3.
Assume R[X] \subseteq S[Y] and S[Y] \subseteq T[Z]
      By def, \pi_x R \subseteq \pi_Y S and \pi_Y S \subseteq \pi_Z T.
      By transitivity of \subseteq, \pi_x R \subseteq \pi_Z T.
      By def, R[X] \subseteq T[Z].
Hence, R[X] \subseteq T[Z].

4.
Premise: X->>Y and XY -> Z.
Assume r_1[X] = r_2[X]
      By X->>Y and r_1[X] = r_2[X],
            there exists r_3 such that r_1[XY] = r_3[XY] and r_2[XZ] =
r_3[XZ].
      As XY -> Z, we have,
            there exists r_3 such that r_1[Z] = r_3[Z] and r_2[XZ] =
r_3[XZ].
      By def of r_1[Z] = r_3[Z], we have r_1[Z \ (X U Y)] = r_3[Z \ (X U
Y)].
      By def of r_2[XZ] = r_3[XZ], we have r_2[Z] = r_3[Z], then r_2[Z \
(X U Y)] = r_3[Z \ (X U Y)].
      Combine these two expression, we have r_1[Z \ (X U Y)] = r_2[Z \ (X
U Y)].
Hence, we proved r_1[X] = r_2[X] => r_1[Z \ (X U Y)] = r_2[Z \ (X U Y)],
which is, by def., X -> Z\(X U Y).

5.
By constradiction, assume XW -> Y and XY -> Z
      Construct a table like below:
            X W Y Z
            1 0 0 0
            1 1 1 1
      Abviously, XW -> Y and XY -> Z holds.
      However, X -> Z doesn't hold.
```

Hence, we prove XW -> Y and XY -> Z => X -> Z is not true.

6.
By G |= X -> Y and the fact that the closure algorithm is complete, we have y \in closure(G, X), for all member y \in Y. We want to prove X -> y from G using only R1-R3.
We want to prove that any sound derivation made by the Closure algorithm can also be derived using the inference rules R1,ÄìR3. The sound derivation can be two cases:

Case closure := X.
In Closure algorithm:
Previous: closure = \emptyset.
After: closure = X.
Only using R1-R3:
We want to prove \emptyset -> X only using R1-R3.

By reflexivity with \emptyset \subseteq X, we have \emptyset -> X.

Case closure := closure U B.
In Closure algorithm:
Previous:
closure = closure.
X -> y for every y in closure.
there exits (A -> B) \in G/.
A \subseteq closure.
B \not\subseteq closure.
After:
closure = closure U B.
Only using R1-R3:
we want to prove closure -> closure U B only using R1-R3.

Apply augmentation on A -> B with closure and with the fact that A \subseqeq closure and B \not\subseteq closure, we have closure -> closure U B.

Therefore, we proved any sound derivation made by the Closure algorithm can also be derived using the inference rules R1,ÄìR3. That is if G |= X -> Y holds for some set of functional dependencies G, then we can derive X -> Y from G using only the inference rules R1, R2, and R3.

7.

C+
C+ = C.
C+ = AC (because of C -> A).
C+ = ACE (because of AC -> E).
C+ = ABCE (becuase of E -> B).
C+ = ABCDE (because of BC -> D).

```
Then, no dependency A -> B in G can meet the requirement A \subseteq C+,
B \not \subseteq C+.
Hence, the closure of C is ABCDE.
--------------------------------------------------------------------------
--------------
(EA)+
(EA)+ = AE.
(EA)+ = ABE (because of E -> B).
(EA)+ = ABDE (because of AB -> D).
(EA)+ = ABDE (because of AB -> D).
Then, no dependency A -> B in G can meet the requirement A \subseteq
(EA)+, B \not \subseteq (EA)+.
Hence, the closure of (EA)+ is ABDE.
--------------------------------------------------------------------------
--------------

8.
Included: consider dependency whose LHS in the G.
C+ = AC
      C -> A
      C -> AC

D+ = AD
      D -> A
      D -> AD

E+ = BE
      E -> B
      E -> EB

(AB)+ = (AB)D
      AB -> D
      AB -> ABD

(AC)+ = (AC)E
      AC -> E
      AC -> (AC)E

(BC)+ = (BC)D
      BC -> D
      BC -> (BC)D

Others: consider dependency whose LHS NOT in the G.
where Y \subseteq X \subseteq {A, B, C, D, E}
X -> Y

RESULT:
// ps(A+) --- all power sets of set A+.
A+ = {A}, G+ = {A -> A}
B+ = {B}, G+ = {B -> B}
C+ = {ABCDE}, G+ = {C -> ps(A+)}
D+ = {AD}, G+ = {D -> ps(D+}}
E+ = {BE}, G+ = {E -> ps(E+}}
AB+ = {ABD}, G+ = {AB -> ps(AB+}}
```

```
AC+ = {ABCDE}, G+ = {AC -> ps(AC+)}
AD+ = {AD}, G+ = {AD -> ps(AD+)}
AE+ = {ABE}, G+ = {AE -> ps(AE+)}
BC+ = {ABCDE}, G+ = {BC -> ps(BC+)}
BD+ = {ABD}, G+ = {BD -> ps(BD+)}
BE+ = {BE}, G+ = {BE -> ps(BE+)}
CD+ = {ABCDE}, G+ = {CD -> ps(CD+)}
CE+ = {ABCDE}, G+ = {CE -> ps(CE+)}
DE+ = {ABDE}, G+ = {DE -> ps(DE+)}
ABC+ = {ABCDE}, G+ = {ABC -> ps(ABC+)}
ABD+ = {ABD}, G+ = {ABD -> ps(ABD+)}
ABE+ = {ABE}, G+ = {ABE -> ps(ABE+)}
ACD+ = {ABCDE}, G+ = {ACD -> ps(ACD+)}
ACE+ = {ABCDE}, G+ = {ACE -> ps(ACE+)}
ADE+ = {ABDE}, G+ = {ADE -> ps(ADE+)}
BCD+ = {ABCDE}, G+ = {BCD -> ps(BCD+)}
BCE+ = {ABCDE}, G+ = {BCE -> ps(BCE+)}
BDE+ = {ABDE}, G+ = {BDE -> ps(BDE+)}
CDE+ = {ABCDE}, G+ = {CDE -> ps(CDE+)}
ABCD+ = {ABCDE}, G+ = {ABCD -> ps(ABCD+)}
ABCE+ = {ABCDE}, G+ = {ABCE -> ps(ABCE+)}
ABDE+ = {ABDE}, G+ = {ABDE -> ps(ABDE+)}
ACDE+ = {ABCDE}, G+ = {ACDE -> ps(ACDE+)}
BCDE+ = {ABCDE}, G+ = {BCDE -> ps(BCDE+)}
ABCDE+ = {ABCDE}, G+ = {ABCDE -> ps(ABCDE+)}
```

Note the fact there is NO c in the RHS, so a superkey must contains C. If it constains C, it can implies ABCDE by our closure. Therefore, if and only if C and superset of C should be superkeys, that is:

superkeys:

- C
- AC
- BC
- DC
- EC
- AEC
- ABC
- ADC
- EBC
- EDC
- BDC
- BCDE
- ACDE
- ABCE
- ABCD
- ABCDE

Based on this, the key (the one with minimalist size) is C.

9.

{ AB -> D, AC -> E, BC -> D, C -> A, D -> A, E -> B }

Apply augmentation on C -> A, we have BC -> AB.

Apply transtivity on BC -> AB and AB -> D, we have BC -> D.
So no need to explicitly inlcude BC -> D.

{ AB -> D, AC -> E, C -> A, D -> A, E -> B }
Apply augmentation on C -> A with C, we have C -> AC.
Apply transitivity on C -> AC and AC -> E, we have C -> E.
So add C -> E to this set doesn't change its property.

{ AB -> D, AC -> E, C -> A, D -> A, E -> B, C -> E }
Apply augmentation on C -> E with A, we have AC -> AE.
Apply decomposition on AC -> AE, we have AC -> E.
So no need to explicitly inlcude AC -> E.

{ AB -> D, C -> A, D -> A, E -> B, C -> E }
Similarly as above.

{ B -> D, C -> A, D -> A, E -> B, C -> E }
Apply transitivity on C -> E and E -> B, we have C -> B.
Apply transitivity on C -> B and B -> D, we have C -> D.
Apply transitivity on C -> D and D -> A, we have C -> A.
So no need to explicitly inlcude C -> A.

{ B -> D, D -> A, E -> B, C -> E }