Decomposition into BCNF

Example 1

Let's take R=ABCDE, F = {A -> BC, C -> DE)

First, let's compute the attribute closure:
A+ = ABCDE
B+ = B
C+ = CDE
D+ = D
E+ = E

Inspecting this attribute closure tells us the candidate keys for R are A.

Now that we know the candidate keys, we can begin to decompose it.

Looking at the 1st FD in F, A -> BC. Is A a key for R? The answer is yes, so this FD does not violate BCNF.

Looking at the 2nd FD in F, C -> DE. Is C a key for R? The answer is no. We decompose R into (CDE) (ABC) – i.e. creating two sub schemas, one with the attributes of the violating FD, the other with the original attributes minus the right hand side of the violating FD.

Ok, now let's inspect those. In (ABC), A is still the key, so the first FD is still not in violation. In (CDE) C is the key, so C -> DE is also not in violation. This decomposition is in BCNF.

Example 2

R = (ABCD)
F = {AB -> C, B -> D; C -> A}

First, let's compute the attribute closure:
A+ = A
B+ = BD
C+ = AC
D+ = D
AB+ = ABCD
BC+ = ABCD

So our candidate keys are AB and BC.
Now we start the algorithm:

AB -> C
Does the first violate BCNF, i.e. is AB a key for R? The answer is yes, so no violation.

B -> D
Does this violate BCNF? Yes, because B is not a key. So we create 2 relations:
(BD) (ABC)

For (BD) the candidate key is B. The only FD that applies here is B -> D, so it is in BCNF.
For (ABC), the candidate keys are AB and BC. The first FD applies, AB -> C, and AB is a key so it is in BCNF. The second FD doesn't apply (there is no D in it), and the third FD does apply (C -> A). Is C a key – no, so we need to decompose by creating a new relation schema made up of the FD, and pulling out the right side of the FD from the original.
(BD)(CA)(BC)

BD is still in BCNF as before
CA has C as the candidate key, and the only FD that applies is C-> A. It is in BCNF.
BC has BC as the candidate key, and no FDs apply, so it is in BCNF.

Our final decomposition is:
(BD)(CA)(BC)

Note: This is a perfect example of a BCNF decomposition where we did not preserve dependencies. We have lost the ability to check AB -> C without doing a join. In this case a 3NF decomposition would be better served, which was back at:

(BD)(ABC).

In this situation, BD was in BCNF, so it is in 3NF by definition.
(ABC) still has 2 candidate keys, AB and BC. The first FD applies, AB -> C, and AB is a key so it is in BCNF (and by definition 3NF). The second FD doesn't apply (there is no D in it), and the third FD does apply (C -> A). While this one violated BCNF, it doesn't violate 3NF because A is a part of a key of (ABC). This decomposition is then 3NF and dependency preserving.