This document was written by Thomas Curran and Conor McArdle Dublin, which provides an additional reference for computing the minimal cover. A detailed example showing the procedure (see pages 11, 12).

Functional Dependency and Algorithmic Decomposition

In this section we introduce some new mathematical concepts relating to functional dependency and, along the way, show their practical use in relational design. Our goal is to have a toolbox to algorithmically generate relations, which meet our criteria for good relational design:

Eliminate/reduce redundancy and the possibilities for update anomalies by decomposing relations without losing any information from the original relations (i.e. all of the original attributes must be present in the set of resulting relations; decomposition should preserve functional dependencies and not create spurious data when relations are joined).

Closure of a set of FDs:

Given a set of FDs we can usually *infer* additional FDs.
For example, consider the following relation and set of FDs holding on the relation:

TEACH = {Module_ID, Lecturer_ID, Lect_First_Name, Lect_Last_Name}
F = { {Module_ID} → {Lecturer_ID},
{Lecturer_ID} → {Lect_First_Name, Lect_Last_Name}}

Given these FDs we can infer that, for example:

{Module_ID} → {Lect_First_Name, Lect_Last_Name}

In addition, given F, it is also true that, for example:

{Lecturer_ID} → {Lect_First_Name}

We say that these FDs are **logically implied** by F.

We can continue to find other FDs that are logically implied by F. The set of all such FDs (and including those in F) is called the **closure** of F (denoted F^+).

Stating these concepts formally:

An FD f is **logically implied** by a set of FDs F if f holds whenever all FDs in F hold.

The closure of F is the set of all FDs that are implied by F.
The closure of F is denoted as F^+.

In order to fully define logical implication, so that we can compute the closure of a set of FDs, we must have a set of rules (axioms) which govern how an FD can be logically implied by some set of FDs.

The Inference Axioms (Armstrong's Axioms)

(1) If $\mathbf{Y} \subseteq \mathbf{X}$, then $\mathbf{X} \rightarrow \mathbf{Y}$ (reflexivity)

(2) If $\mathbf{Z} \subseteq \mathbf{W}$ and $\mathbf{X} \rightarrow \mathbf{Y}$, then $\mathbf{XW} \rightarrow \mathbf{YZ}$ (augmentation)

(3) If $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{Y} \rightarrow \mathbf{Z}$ then $\mathbf{X} \rightarrow \mathbf{Z}$ (transitivity)

Other rules which may be derived from the axioms include:

If $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{YW} \rightarrow \mathbf{Z}$ then $\mathbf{XW} \rightarrow \mathbf{Z}$

If $\mathbf{X} \rightarrow \mathbf{Z}$ and $\mathbf{X} \rightarrow \mathbf{Y}$ then $\mathbf{X} \rightarrow \mathbf{YZ}$

If $\mathbf{X} \rightarrow \mathbf{YZ}$ then $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{X} \rightarrow \mathbf{Z}$

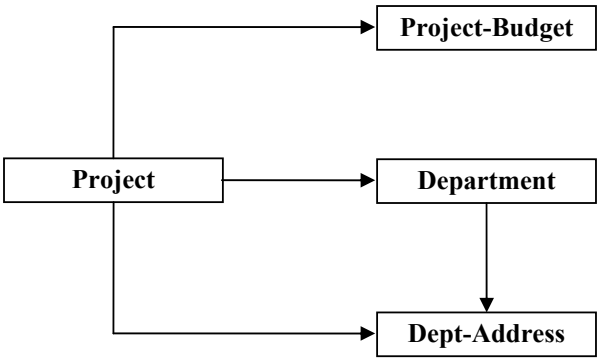
Using these axioms we can find all FDs logically implied by some set of FDs F and hence find its closure F^+.

Exercise 11

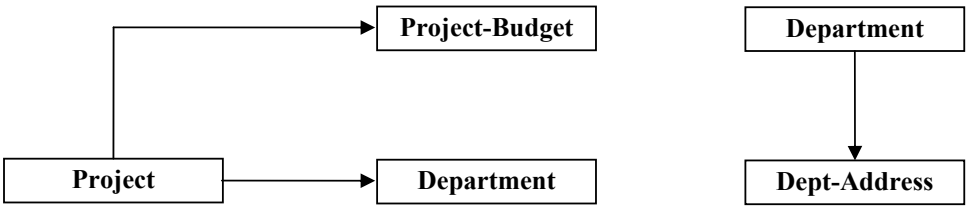
Using the axioms, find the closure F^+ of $F = \{\mathbf{AB} \rightarrow \mathbf{CD}, \mathbf{C} \rightarrow \mathbf{E}\}$.

Application of Closure to Testing Dependency Preservation

Looking again at the PROJECTS schema from our previous 3NF example, we had the following functional dependencies:



We decomposed the PROJECTS relation into two relations on which two corresponding sets of FDs held:



If we look at all FDs in the new schema taken together:

$$\{\text{Project} \rightarrow \text{Project-Budget}, \text{Project} \rightarrow \text{Department}, \text{Department} \rightarrow \text{Dept-Address}\}$$

it appears that we have lost one of the original FDs: $\text{Project} \rightarrow \text{Dept-Address}$.

However, using the axioms, we can infer that $\text{Project} \rightarrow \text{Dept-Address}$ is logically implied by $\text{Project} \rightarrow \text{Department}$ and $\text{Department} \rightarrow \text{Dept-Address}$ so that the FD has not actually been lost.

One of our goals during decomposition is to preserve dependencies. This is important as dependencies represent constraints on the data which must be enforced to maintain the semantics of the relations. Given our definition of closure we can now define precisely how to check if a decomposition is dependency preserving:

A decomposition of a relation **R**, on which the dependencies F hold, into relations **R1** and **R2** is dependency preserving if:

$$(F_{R1} \cup F_{R2})^{+} = F^{+}$$

Where F_{R1} is the set of dependencies that apply between attributes only in R1 and similarly F_{R2} is the set of dependencies that apply between attributes only in R2.

Although the set of dependencies ($F_{R1} \cup F_{R2}$) might be different to the original set of dependencies F, we only require that the two sets are **equivalent** (that is, their closures are the same) for the decomposition to be dependency preserving.

For example, the closure of the PROJECTS relation in the previous example is:

{Project → Project-Budget, Project → Department, Department → Dept-Address, Project → Dept-Address}

The closure of the combined FDs in the decomposed tables is the same, so the decomposition is dependency preserving.

Application of Closure to Testing for Lossless Decomposition

Consider the following example of decomposition:

R1 = ({**A**, **B**, **C**}, {**A** → **B**, **C** → **B**})

RELATION: R1

A	**B**	**C**
a1	b1	c1
a3	b1	c2
a2	b2	c3
a4	b2	c4

If we decompose this into the relations:

R2 = ({**A, B**}, {**A** → **B**})
R3 = ({**B, C**}, {**C** → **B**})

we would have:

RELATION: R2

A	**B**
a1	b1
a3	b1
a2	b2
a4	b2

A → **B**

RELATION: R3

B	**C**
b1	c1
b1	c2
b2	c3
b2	c4

C → **B**

Now, if we join relations **R2** and **R3** over the common attribute **B**, the resultant relation would appear as follows:

A	**B**	**C**
a1	b1	c1
a1	b1	c2
a3	b1	c1
a3	b1	c2
a2	b2	c3
a2	b2	c4
a4	b2	c3
a4	b2	c4

The resultant relation contains more rows (spurious tuples) than existed in the original **R1**. This is called a "lossy decomposition" as it causes constraint information to be lost. Lossy decomposition is extremely undesirable and should be avoided at all costs.

How to test for lossless/lossy decomposition?

Decomposition of **R** into two relations $\mathbf{R_1}$ and $\mathbf{R_2}$ is lossless, with respect to a set of functional dependencies F, if and only if the closure F^+ contains:

$(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow \mathbf{R_1}$ or
$(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow \mathbf{R_2}$

We can interpret this condition as the condition that common attributes must form a superkey in either $\mathbf{R_1}$ or $\mathbf{R_2}$ for the decomposition to be lossless.

Exercise: use the definition above to show that decomposition of **R1** = ({**A**, **B**, **C**}, {$\mathbf{A \rightarrow B}$, $\mathbf{C \rightarrow B}$}) into **R2** = ({**A, B**}, {$\mathbf{A \rightarrow B}$}) and **R3** = ({**B, C**}, {$\mathbf{C \rightarrow B}$}) is not lossless.

We will now develop an algorithm that guarantees a lossless, dependency-preserving 3NF decomposition. First we need to consider the notions of *attribute closure* and *minimal cover*.

Attribute Closure

The closure of a set of FDs is of interest, as we have seen, but can be very large and take a long time to compute. There is no efficient method (shortcut) for computing it. For algorithms used in decomposition, typically we only want to check if some given FD $\mathbf{X} \rightarrow \mathbf{Y}$ is in the closure of a set of FDs F. There is a way of doing this without computing the closure F^+. One ad hoc way of answering this is to try to derive $\mathbf{X} \rightarrow \mathbf{Y}$ from F using the inference axioms. Another is by use of the concept of the *closure* of an attribute.

The attribute closure of **X** with respect to the set of FDs F is defined as the set of all attributes **A** such that $\mathbf{X} \rightarrow \mathbf{A}$ is in F^+.

Closure of a set of attributes **X** (denoted $\mathbf{X}^+$) is the set of all the attributes that are functionally dependent on **X**, given some set of functional dependencies F. The attribute closure is much more efficient to compute than F^+. The aim of computing it is to find what attributes depend on a given set of attributes and therefore ought to be together in the same relation. We will use it later to formulate an algorithmic way of decomposing a relation into relations of higher normal form.

Computing the Closure of an Attribute

The following algorithm is used to compute the closure of a set of attributes **X** under the set of FDs F.

```
X+ := X
repeat
  temp_X+ := X+
  for each FD Y→Z in F do
    if Y⊆X+ then X+ := X+∪Z
until (temp_X+ = X+)
```

Explanation of Algorithm:
If the attribute **X** determines some attribute **Y** and in turn **Y** determines **Z**, then **X** also determines **Z** (axiom (3)) and so **Z** is added to $\mathbf{X}^+$. Continuing, if any set of attributes in $\mathbf{X}^+$ determine some other attribute(s) then **X** also determines this attribute(s) and it is added to $\mathbf{X}^+$, and so on.

Attribute Closure Applications

We can use attribute closure to quickly answer questions about specific FDs, and also to find keys of complex relations.

For example, can a given FD $X \rightarrow Y$ be inferred by some set of FDs F?
This is the same as asking: is $X \rightarrow Y$ in F^+? (difficult to find F^+)
Which in turn is the same as asking: is Y in X^+ given F? (easy to find X^+)

Example:
Is $B \rightarrow G$ in F^+ where
R = {A, B, C, D, E} and
$F = \{B \rightarrow CD, E \rightarrow F, D \rightarrow E, B \rightarrow A, AD \rightarrow B, F \rightarrow G\}$

Using the algorithm compute $\mathbf{B}^+$ and then check if it contains **G**:

```
B+ = B
B+ = BCD
B+ = BCDE
B+ = BCDEA
B+ = BCDEAF
B+ = BCDEAFG
stop
```

Yes G is in $\mathbf{B}^+$ so $\mathbf{B} \rightarrow \mathbf{G}$ is in F^+.

Note that B^+ contained all the attributes in the relation **R** and so **B** must be a key to the relation. Finding the closure of attributes can be used as a general method for checking if multiple attributes taken together are relation keys. For example, is **AD** a relation key of **R**? To answer this, find $\mathbf{A}^+$, $\mathbf{D}^+$ and $\mathbf{AD}^+$. If neither $\mathbf{A}^+$ nor $\mathbf{D}^+$ contain all the attributes in **R** but $\mathbf{AD}^+$ does, then **AD** is a relation key.

Exercise 12

Given the relation:

$\mathbf{R} = (\{V, W, X, Y, Z\}, \{W \rightarrow XY, Y \rightarrow V, Z \rightarrow V, X \rightarrow Z\})$

Compute the X^+, showing each step of the algorithm, and from X^+ determine if
(a) X is a key of R
(b) $X \rightarrow VZ$ holds on R

Using Attribute Closure to Remove Redundant FDs from a Set of FDs

As we've seen, the closure of a set of attributes **X** under a set of FDs *F* is the set of all the attributes that are functionally dependent on **X**. We can use attribute closure to remove redundant FDs from a set. This will be useful in systematically normalising relations (later). As an example, remove all redundant functional dependencies from *F*, using the attribute closure algorithm, where:

$F = \{\mathbf{AD} \rightarrow \mathbf{B}, \mathbf{B} \rightarrow \mathbf{C}, \mathbf{C} \rightarrow \mathbf{D}, \mathbf{A} \rightarrow \mathbf{B}\}$

We can do this by examining each FD in turn to see if it is redundant, i.e. if it can be inferred from the other FDs in *F*, and can thus be removed from *F* to form a smaller equivalent set *G*. We can summarise this algorithm as follows:

```
set G = F
for each FD X→A in G  {
    compute X+ with respect to the set of dependencies (G – (X→A))
    if X+ contains A, then G = (G – (X→A))
  }
```

Applying this algorithm to $F = \{\mathbf{AD} \rightarrow \mathbf{B}, \mathbf{B} \rightarrow \mathbf{C}, \mathbf{C} \rightarrow \mathbf{D}, \mathbf{A} \rightarrow \mathbf{B}\}$

```
G = {AD→B, B→C, C→D, A→B}
For AD→B compute AD+ under (G – (AD→B))
  AD+ = AD
  AD+ = ADB
  AD+ = ADBC
  AD+ = ADBCD
  stop
  G = (G – (AD→B)) = {B→C, C→D, A→B}
For B→C compute B+ under (G – (B→C))
  B+ = B
  stop
  G = {B→C, C→D, A→B}
For C→D compute C+ under (G – (C→D))
  C+ = C
  stop
  G = {B→C, C→D, A→B}
For A→B compute A+ under (G – (A→B))
  A+ = A
  stop
  G = {B→C, C→D, A→B}
end
```

Note that depending on the FDs and their order we can sometimes find a different *G*.

Exercise 13

Remove any redundant FDs from the following sets of FDs, using the attribute closure algorithm:

Set 1	***Set 2***	***Set 3***
$A \rightarrow B$	$XY \rightarrow V$	$PQ \rightarrow R$
$B \rightarrow C$	$ZW \rightarrow V$	$PS \rightarrow Q$
$AD \rightarrow C$	$VX \rightarrow Y$	$QS \rightarrow P$
	$W \rightarrow Y$	$PR \rightarrow Q$
	$Z \rightarrow X$	$S \rightarrow R$

Minimal Cover of a Set of Functional Dependencies

There may be further redundancy in a set of FDs in the form of redundant attributes in the determinants of the FDs. If we also remove such attributes we arrive at what is called a **minimal cover** for the set of FDs. The minimal cover must also have the property of having single attributes on the right hand side of all FDs in the set.

Formally:

The minimal cover for a set of FDs F is the set of FDs G such that:

- $G^+ = F^+$ (that is G and F are equivalent)
- and the right hand side of each FD in G is a single attribute
- and G is minimal, that is, if we remove any attribute from an FD in G or remove any FD from G, then G^+ will no longer equal F^+

We can extend our previous algorithm to obtain a minimal cover algorithm:

```
set G = F
replace each FD X→{A1,A2,...,An} in G by n FDs X→A1,X→A2,...,X→An
for each FD X→A in G  {
    compute X+ with respect to the set of dependencies (G - (X→A))
    if X+ contains A, then G = (G - (X→A))
  }
for each remaining FD X→A in G that has multiple attributes in X
  for each attribute B∈X  {
    compute (X-B)+ with respect to (G-(X→A)) ∪ ((X-B)→A)
    if (X-B)+ contains B, then replace X→A in G with (X-B)→A
  }
```

The first step simply replaces any FDs with multiple attributes on the rhs with equivalent sets of FDs with rhs containing only one attribute.
The second step removes any redundant FDs, using the algorithm we had before.
The third step checks if any attribute on the left hand side of any FD can be removed.
Note: it is sometimes possible to have more than one valid minimal cover for a given set of FDs.

Example:

Using the minimal cover algorithm, find a minimal cover of
$F = \{AB \rightarrow D, B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow EF\}$

Step 1. Make right hand sides atomic

$G = \{AB \rightarrow D, B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$

Step 2. Remove any redundant FDs

For $AB \rightarrow D$ compute AB^+ under $(G - (AB \rightarrow D))$
  $AB^+ = ABCDEF$
  D in AB^+ so remove $AB \rightarrow D$ from G
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$
For $B \rightarrow C$ compute B^+ under $(G - (B \rightarrow C))$
  $B^+ = B$
  C not in B^+ so
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$
For $AE \rightarrow B$ compute AE^+ under $(G - (AE \rightarrow B))$
  $AE^+ = AEDF$
  B not in AE^+ so
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$
For $A \rightarrow D$ compute A^+ under $(G - (A \rightarrow D))$
  $A^+ = A$
  D not in A^+ so
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$
For $D \rightarrow E$ compute D^+ under $(G - (D \rightarrow E))$
  $D^+ = DF$
  E not in D^+ so
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$
For $D \rightarrow F$ compute D^+ under $(G - (D \rightarrow F))$
  $D^+ = DE$
  F not in D^+ so
  $G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$

Step 3. Remove any redundant left hand side attributes

$G = \{B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$

For $AE \rightarrow B$
  For A: compute E^+ with respect to $(G-(AE \rightarrow B) \cup (E \rightarrow B))$
    E^+ wrt $\{B \rightarrow C, E \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\} = EBC$
    E^+ doesn't contain A, so A not redundant in $AE \rightarrow B$
  For E: compute A^+ with respect to $(G-(AE \rightarrow B) \cup (A \rightarrow B))$
    A^+ wrt $\{B \rightarrow C, A \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\} = ABDEFC$
    A^+ contains E, so E is redundant in $AE \rightarrow B$

Minimal closure = $\{B \rightarrow C, A \rightarrow B, A \rightarrow D, D \rightarrow E, D \rightarrow F\}$

Exercise 14

Find a minimal cover for the following set of functional dependencies:

$A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C$
$AC \rightarrow D$

Algorithmic 3NF Normalisation

We can use our minimal cover algorithm to systematically decompose relations into a higher normal form, whilst meeting our objectives of preserving dependencies (at least for 3NF) and avoiding loss in the decomposition (for either 3NF or BCNF).

Below is an algorithm for decomposing a relation to 3NF. It will always produce a lossless decomposition which is dependency preserving (proof omitted).

```
given F, find a minimal cover G for F

for each set of FDs in G of the form {X→A1,X→A2,...,X→An}
containing alls FDs in G with the same determinant X
{
  create a relation R = {X,A1,A2,...,An}
}

place any remaining attributes in a single relation
```

We stop here at 3NF but note that the maths can be applied to systematically normalising to higher normal forms. Note however, that we cannot guarantee that there will always exist a BCNF (or higher normal form) which is lossless and dependency preserving. For this reason, 3NF is often a high enough normal form for many practical applications.

Exercise 15

By first finding a minimal cover, normalise the relation R to 3NF.

$R = (\{A, B, C, D, E, F\}, \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, D \rightarrow EF\})$

Summary of Relational Database Design Procedures

Two alternative approaches to relational analysis and design:

1: Modelling plus Decomposition/Verification Approach

- Gather information about application and analyse
- Find functional dependencies between attributes
- Make E-R diagram of schema
- Refine E-R diagram, e.g. convert many-to-many relationships
- Examine resulting tables and normalise if necessary

2: Pure Algorithmic Decomposition Approach

- Gather information about application and analyse
- Find functional dependencies between attributes
- Put all attributes into one large relation and systematically normalise/decompose into final schema.