Computation of FDs

- Goal: Compute for a given set $F$ of FDs all **logically implied** FDs.

- Let $F^+$ be the set of all FDs that can be logically implied from the FDs in $F$. $F^+$ is called the **closure** of $F$.

- Let $R$ be a relation schema, $F$ a set of FDs and $A, B, C \subseteq R$.

  The following **inference rules** are used to compute $F^+$ (**Armstrong’s axioms**):
  - **reflexivity rule**: Let $B \subseteq A$. Then always $A \rightarrow B$ (special case: $A \rightarrow A$) holds.
  - **augmentation rule**: If $A \rightarrow B$ holds, then also $A \cup C \rightarrow B \cup C$ holds.
  - **transitivity rule**: If $A \rightarrow B$ and $B \rightarrow C$ holds, then also $A \rightarrow C$ holds.

- It can be formally shown that these rules are **sound** and **complete**.
  - **soundness**: Inferred FDs hold for all relations of this schema.
  - **completeness**: All valid FDs in $F^+$ can be logically implied with these rules.
Although Armstrong’s axioms are complete, it is comfortable to add three further inference rules:

- **union rule**: If \( A \rightarrow B \) and \( A \rightarrow C \) holds, then also \( A \rightarrow B \cup C \) holds.
- **decomposition rule**: If \( A \rightarrow B \cup C \) holds, then also \( A \rightarrow B \) and \( A \rightarrow C \) holds.
- **pseudotransitivity rule**: If \( A \rightarrow B \) and \( B \cup C \rightarrow D \) holds, then also \( A \cup C \rightarrow D \) holds.

**example:**

- *supplier* relation with the schema *supplier*(sname, saddr, product, price)
- Valid FDs: \( \{\text{sname}\} \rightarrow \{\text{saddr}\} \), \( \{\text{sname}, \text{product}\} \rightarrow \{\text{price}\} \), \( \{\text{sname}\} \rightarrow \{\text{sname}\} \), \( \{\text{sname}, \text{product}\} \rightarrow \{\text{product}\} \)
- It is to be shown: \( \{\text{sname}, \text{product}\} \rightarrow \{\text{saddr}\} \) is also satisfied.
  
  We have: \( \{\text{sname}\} \rightarrow \{\text{saddr}\} \).
  
  Due to the augmentation rule we obtain: \( \{\text{sname}, \text{product}\} \rightarrow \{\text{saddr}, \text{product}\} \).
  
  Due to the decomposition rule we hence obtain: \( \{\text{sname}, \text{product}\} \rightarrow \{\text{saddr}\} \).
computing the closure $F^+$

$F^+ = F$

repeat

   for each functional dependency $f$ in $F^+$ do
       apply reflexivity and augmentation rules to $F^+$
       add the resulting functional dependencies to $F^+$
   od;

   for each pair of functional dependencies $f_1$ and $f_2$ in $F^+$ do
       if $f_1$ and $f_2$ can be combined using transitivity then
           add the resulting functional dependency to $F^+$
       fi
   od;

until $F^+$ does not change any further
Containment of a FD in a closure $F^+$

- question: Let $F$ be a set of FDs and $A \rightarrow B$ a FD. Does $A \rightarrow B \in F^+$ hold?
- problem: explicit calculation of $F^+$ is too expensive
- instead: calculation of the **closure** $A^+$ of the attribute set $A$ regarding the set $F$
  - $A^+$ consists of all attributes that are functionally determined by $A$.
  - If $B \subseteq A^+$ holds, then also $A \rightarrow B \in F^+$ holds.
- algorithm for inferring $A^+$
  
  **algorithm** AttrClosure($F$, $A$)
  
  // input: a set $F$ of FDs and a set $A$ of attributes
  // output: the complete set $A^+$ of attributes for which holds: $A \rightarrow A^+$
  
  $A^+ := A$;
  repeat
      Old$A^+ = A^+$;
      foreach FD $B \rightarrow C \in F$ do
          if $B \subseteq A^+$ then $A^+ := A^+ \cup C$;
      until $A^+ = OldA^+$;
  return $A^+$
Example for Computing the Attribute Closure

Notation: For the set \{A, C, E\} we permit to write ACE (juxtaposition) to be able to omit braces. In particular, \{D\} is written as D.

Example: Let \( R(A, B, C, G, H, I) \) be a relation schema, and let \( F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CI \rightarrow G\} \) be a set of FDs.

Task: Compute \( AG^+ \).

Solution: We use the algorithm AttrClosure and initialize \( AG^+ \) with \( AG \).

In the loop we set \( Old\_AG^+ := AG \) and check all FDs whether they can contribute to \( AG^+ \).

First we take \( A \rightarrow B \) and check whether \( A \subseteq AG^+ \) holds. This is the case. Therefore, we set \( AG^+ := AG^+ \cup B = ABG \) (due to transitivity).

Next, we take \( A \rightarrow C \). Using the same argument as before, we obtain \( AG^+ := AG^+ \cup C = ABCG \).

Next, we take \( CG \rightarrow H \). We find that \( CG \subseteq AG^+ \) holds. We get \( AG^+ := AG^+ \cup H = ABCGH \).

Next we take \( CI \rightarrow G \). We find that \( CI \not\subseteq AG^+ \) holds.

Since \( Old\_AG^+ \neq AG^+ \) holds, we perform a second loop. We set \( Old\_AG^+ \) to \( AG^+ \), that is, \( Old\_AG^+ := ABCGH \). We see soon that no FD from \( F \) can increase \( AG^+ \). This means that \( Old\_AG^+ = AG^+ \) holds, the algorithm terminates, and we get \( AG^+ := ABCGH \).