Algorithmen für die minimale Überdeckung

Denis Wirries
Dipl.-Math. Mazeyar E. Makoui
Universität Hannover

19. März 2006

Zusammenfassung

Diese Definitionen und Algorithmen dienen zur Berechnung der minimalen Überdeckung von funktionalen Abhängigkeiten. Die Algorithmen werden jeweils in Java implementiert.

1 Definitionen

1.1 Minimale Überdeckung

Definition 1 (Minimale Überdeckung) *Eine Menge von funktionalen Abhängigkeiten (FA) F* ***ist minimal****, wenn gilt:*

1. *Jede rechte Seite von einer FA in F besteht aus einem Attribut.*
2. *Es gibt kein $X \rightarrow A$ in F, so dass die Menge $F - \{X \rightarrow A\}$ äquivalent zu F ist.*
3. *Es gibt kein $X \rightarrow A$ in F und keine echte Untermenge Z von X, so dass $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ äquivalent zu F ist.*

1.2 Äquivalenz von FA-Mengen

Definition 2 (Äquivalenz) *Seien F und G zwei Mengen von FA über A. F und G sind äquivalent, wenn $F^+ = G^+$, d. h. wenn ihre Hüllen gleich sind. F heißt auch* ***Überdeckung*** *von G und umgekehrt.*

1.3 Hülle

Definition 3 (Hülle einer Attributmenge) *X^+ ist die Hülle einer Attributmenge X bzgl. F. Sie umfasst die Menge aller Attribute A_i, für die $X \rightarrow A_i$ in F^+ ist.*

2 Algorithmen

Die Definition der Algorithmen erfolgt in Pseudo-Code. Auf die Definition der einzelnen Datentypen wird aus Übersichtlichkeit verzichtet.

Allgemeine Variablen und ihre Bedeutung[1]:

Variable	Bedeutung
X, Y, Z	Variable für Attribute (können auch zusammengesetzte Attribute sein)
FA	eine funktionale Abhängigkeit mit den Attritbuten X und Y
$X \rightarrow Y$	
F, G	Variablen für die Mengen von funktional Abhängigkeiten
T	Datenstruktur einer Tabelle, wird als Änderungsverfolgung verwendet, siehe Algorithmus 5

[1] wenn nichts anderes definiert ist

Bemerkung 4 *Die Datenstrukturen für die Listen von Attributen und FAs ignorieren mehrfach gleiche Einträge, z. B.* $F = (\{X \to Y\}) \Rightarrow F \leftarrow F \cup \{X \to Y\} \Rightarrow F = (\{X \to Y\})$*, genauso bei Attributlisten.*

2.1 Algorithmus CLOSURE

Ein einfacher Algorithmus zur Bestimmung von X^+(Abschluss von F bzgl. einer Attributmenge X). Der Algorithmus baut iterativ eine Kette der von X erreichbaren Attribute auf. Er liefert eine Liste der Attribute zurück.

Algorithm 1 CLOSURE

Eingabe: Ein Attribut X (auch zusammengesetzte) und eine Menge von FA F
Ausgabe: X^+ (eine Menge von Attributen (auch zusammengesetzte)) bzgl. F

```
 1: OLDDEP ← {}
 2: NEWDEP ← {} ∪ X
 3: while NEWDEP ≠ OLDDEP do
 4:    OLDDEP ← NEWDEP
 5:    for all FA Y → Z ∈ F do
 6:       if Y ⊆ NEWDEP then
 7:          NEWDEP ← NEWDEP ∪ Z
 8:       end if
 9:    end for
10: end while
11: return NEWDEP
```

Bemerkung 5 *Die Variablen OLDDEP und NEWDEP sind in dem Algorithmus 1 Mengen von Attributen.*

2.2 Algorithmus CLOSURE (2)

Sei wieder eine (endliche) Attributmenge U gegeben. Zu einer Menge F von funktionalen Abhängigkeiten und einer Attributmenge $X \subseteq U$ kann man wie folgt den Abschluss

$$(F, X)^* := \{Y \in U | F \Rightarrow X \to Y\}$$

berechnen. Damit kann man für jede Menge F von FAen und jede FA $X \to Y$ entscheiden, ob $F \Rightarrow X \to Y$.

Algorithm 2 CLOSURE (2)

Eingabe: Ein Attribut X (auch zusammengesetzte) und eine Menge von FA F
Ausgabe: X^+ (eine Menge von Attributen (auch zusammengesetzte)) bzgl. F

```
 1: UNUSED ← F
 2: CLOSURE ← X
 3: repeat
 4:    for all FA Y → Z ∈ UNUSED do
 5:       if Y ⊆ CLOSURE then
 6:          UNUSED ← UNUSED −{Y → Z}
 7:          CLOSURE ← CLOSURE ∪ Z
 8:       end if
 9:    end for
10: until UNUSED, CLOSURE unchanged
11: return CLOSURE
```

Bemerkung 6 *Dieser Algorithmus gleicht dem Prinzip aus dem Algorithmus 1 und wird deshalb nicht implementiert.*

2.3 Algorithmus MEMBER

Testen der Mitgliedschaft einer FA bzgl. F, d. h. es wird geprüft, ob die FA in F ist. Der Algorithmus prüft, ob die rechte einer FA: $X \rightarrow Y$ in dem Abschluss von X bzgl. F liegt.

Algorithm 3 MEMBER

Eingabe: FA $X \rightarrow Y$, F
Ausgabe: `TRUE`, wenn $F \models X \rightarrow Y$, sonst `FALSE`
1: **if** $Y \subseteq$ CLOSURE(X, F) **then**
2: **return**(`TRUE`)
3: **else**
4: **return**(`FALSE`)
5: **end if**

Bemerkung 7 *Der Algorithmus verwendet den Algorithmus 1 (CLOSURE).*

2.4 Algorithmus Minimale Überdeckung (MINCOVER)

Für eine gegebene Menge von funktionalen Abhängigkeiten lässt sich eine äquivalente Menge von Funktionalrelationen mit jeweils einem Attribut auf der rechten Seite (kanonisierte FAs) finden. Dabei wird eine FA vorläufig entfernt und dann geprüft, ob die entfernte FA sich aus den noch vorhandenen FAs rekonstruieren lassen.

Algorithm 4 MINCOVER (Prof. Dr. Udo Lipeck, DBS2a-Vorlesung WS05/06)

Eingabe: eine Menge F von FAs
Ausgabe: eine minimale Überdeckung für F
1: $G \leftarrow F$
2: **for all** FA $\{X \rightarrow Y\} \in F$ **do**
3: **if** MEMBER$(\{X \rightarrow Y\}, G - \{X \rightarrow Y\})$ **then**
4: $G \leftarrow G - \{X \rightarrow Y\}$
5: **end if**
6: **end for**
7: **return** G

Bemerkung 8 *Dieser Algorithmus verwendet den Algorithmus 3 (MEMBER) und implizit auch den Algorithmus 1 (CLOSURE).*

Bemerkung 9 *Zeile 2: Bei MEMBER($\{X \rightarrow Y\}$, $G - \{X \rightarrow Y\}$) soll (und darf!) die Menge G nicht verändert werden.*

2.5 Ein weiterer Algorithmus zur Berechung der Minimalen Überdeckung (MINCOVER(2))

Dieser Algorithmus berechnet auch die minimale Überdeckung einer Menge von FAs. Er geht dabei aber ein wenig anders vor als der Algorithmus 4, er wählt dabei nicht wahllos eine FA aus und entfernt diese, um zu prüfen, ob die entfernte FA sich noch rekonstruieren lässt (Algorithmus 4 Zeile 3), sondern arbeitet sich systematisch durch die Menge (siehe Abbildung 1).

Er wählt in einem Schritt eine FA nach der anderen aus (getrennt von einander), entfernt diese und prüft, ob die entfernte FA noch rekonstruierbar ist (Algorithmus 3). Wenn dies möglich ist, wird mit dieser Menge iterativ weiter gearbeitet, wenn nicht wird keine neue minimale Menge hinzugefügt. Die Entfernungen der FAs werden in einer Tabelle gespeichert, um so alle möglichen Entfernungen nach zu vollziehen.

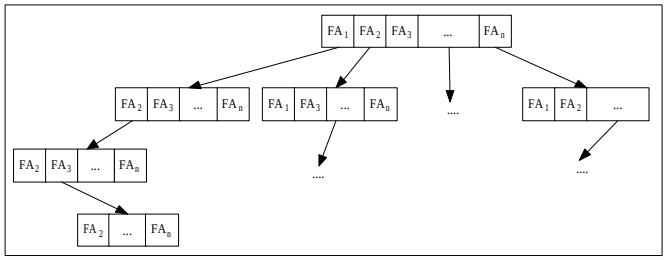

Abbildung 1: Beispiel zum Algorithmus MINCOVER(2)

Algorithm 5 MINCOVER(2) (Dipl.-Math. Mazeyar E. Makoui, DBS2a-Übung)

Eingabe: eine Menge F von FAs
Ausgabe: eine Tabelle T zum Verfolgen der Änderungen
{0. Zeile der Tabelle aufbauen (Startbedingung)}
1: $i \leftarrow 0$
2: $T \leftarrow$ neue Tabelle
3: **for all** FA $X \rightarrow Y \in F$ **do**
4: **if** MEMBER($\{X \rightarrow Y\}$, $F - \{X \rightarrow Y\}$) **then**
5: Füge ($F - \{X \rightarrow Y\}$) in T_i ein
6: **end if**
7: **end for**
{Baue die Tabelle auf, bis keine neue Ebene mehr hinzukommt}
8: **while** T verändert sich **do**
9: **for** jeden Eintrag (Liste) von FAs $F' \in T_i$ **do**
10: **for all** FA $X \rightarrow Y \in F'$ **do**
11: **if** MEMBER($\{X \rightarrow Y\}$, $F' - \{X \rightarrow Y\}$) **then**
12: Füge ($F' - \{X \rightarrow Y\}$) in T_{i+1} ein
13: **end if**
14: **end for**
15: **end for**
16: $i \leftarrow i + 1$
17: **end while**
18: **for** jeden Eintrag (Liste) von FAs $F' \in T_i$ **do**
19: **if** linke Seiten minimal **then**
20: Füge ($F' - \{X \rightarrow Y\}$) in T_{i+1} ein
21: **end if**
22: **end for**
23: **return** T

2.6 Algorithmus Gleiche Überdeckung (SAMECOVER)

Berechnet von zwei Mengen von FAs F und G, ob sie die gleiche Überdeckung erzeugen. Dabei werden zunächst alle möglichen Determinaten der beiden Mengen erzeugt und verglichen. Voraussetzung für dieses Verfahren ist, dass die Eingabemengen kanonisch (siehe Algorithmus 7 (KANONIZE)) und vorkorrigiert (siehe Algorithmus 10 (DETCREATION)) sind, damit die Mengen möglichst einfach sind.

2.7 Algorithmen zur Optimierung von funktionalen Abhängigkeitsmengen

2.7.1 Algorithmen zur Kanonisierung bzw. zur Dekanonisierung

Kanonisierung bzw. Dekanonisierung einer Menge von funktionalen Abhängigkeiten. Nach der Kanonisierung darf auf der rechten Seite einer FA nur noch ein Attribut stehen, bei FA, die

Algorithm 6 SAMECOVER

Eingabe: zwei Mengen F und G von FAs
Ausgabe: `TRUE`, wenn die Überdeckung gleich ist, sonst `FALSE`

```
1: F' ← DETCREATION(F)
2: G' ← DETCREATION(G)
3: b ← TRUE
   {F' < G' Testet, welche Menge größer ist.}
4: if F' < G' then
5:    for all FA X → Y ∈ F' do
6:       b ← FA ⊆ G'
7:    end for
8: else
9:    for all FA X → Y ∈ G' do
10:      b ← FA ⊆ F'
11:   end for
12: end if
13: return b
```

mehr als ein Attribut auf der rechten Seite haben, wird die FA auf mehrere FAs aufgeteilt. Bei der Dekanonisierung wird dieser Prozess wieder rückgängig gemacht, d. h. bei gleichen linken Seiten, sofern vorhanden, werden die rechten Seiten wieder zusammengefasst.

Algorithm 7 KANONIZE

Eingabe: Menge F von FAs
Ausgabe: kanonisierte Menge von FAs

```
1: G ← {}
2: for all FA X → Y ∈ F do
3:    if zusammengesetztes_Attribut(Y) then
4:       for all Z ∈ Y do
5:          G ← G ∪ {X → Z}
6:       end for
7:    else
8:       G ← G ∪ {X → Y}
9:    end if
10: end for
11: return G
```

Bemerkung 10 *Der Algorithmus 7 (KANONIZE) sollte immer im Vorfeld der Berechung der minimale Überdeckung ausgeführt werden. Dies reduziert die Komplexität der Algorithmen 4 und 5 (MINCOVER), man muss nicht darauf achten, ob die rechte Seite ein zusammengesetztes Attribut ist.*

Bemerkung 11 *Der Algorithmus 8 (DEKANONIZE) dient nur zur besseren Darstellung der Ergebnismenge von FAs. Er hat keine weitere Bedeutung in der Berechung der minimalen Überdeckung der funktionalen Abhängigkeiten.*

2.7.2 Algorithmen zur Optimierung der Menge von FAs

Verschiedene Algorithmen zur Optimierung der Eingabemenge von FAs.

Algorithmus CORRECTION Der Algorithmus 9 (CORRECTION) reduziert die Eingabeliste für den Algorithmus 4 und 5 (MINCOVER), indem er unnötige FA streicht. Dies hat zur folge das nach der Korrektur nur noch notwendige FAs existieren und die linken Seiten so kurz wie möglich sind. Im Vorfeld der Korrektur, muss der Algorithmus 7 (KANONIZE) und 10 (DETCORRECTION) ausgeführt werden, weil die Korrektur bei der rechten Seite nicht auf

Algorithm 8 DEKANONIZE

Eingabe: Menge F von FAs
Ausgabe: dekanonisierte Menge von FAs

```
1: G ← {}
2: for all FA X → Y ∈ F do
3:    for all FA X' → Y' ∈ F do
4:       if X = X' then
5:          Y ← Y ∪ Y'
6:       end if
7:    end for
8:    G ← {X → Y}
9: end for
10: return G
```

zusammengesetzte Attribute achtet und nicht die rechte Seite weiter ableitet.
Allgemeine Arbeitsweise des Algorithmus (siehe auch Abbildung 2):

1. wähle die erste FA aus
2. prüfe, ob es zu dieser FA eine kürze FA in F gibt, die die selbe Funktionalabhängigkeit beschreibt,
3. wenn dies der Fall ist lösche diese gewählte FA und wähle die nächste FA aus F und gehe zu Schritt 2, dies wird solange ausgeführt, bis die Liste der FAs abgearbeitet ist.

Beispiel 12 (Beispiel für Algorithmus CORRECTION)

$$F = (\{A \to D\}, \{AB \to D\}, \{ABC \to D\}) \Rightarrow F' = (\{A \to D\})$$

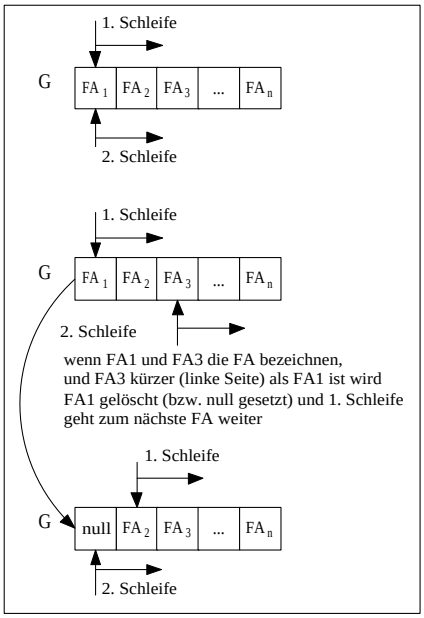

Abbildung 2: Arbeitsweise des Algorithmus KORRECTION

Algorithmus DETCREATION Der Algorithmus 10 erzeugt alle Determinanten einer Menge von FAs, d. h. es werden zu jedem Attribut X einer FA $X \to Y$ alle FA hinzugefügt, die von X aus erreicht werden können. Die Arbeitsweise von DETCREATION ist relativ simpel, es wird zu jeder FA $X \to Y$ mit Hilfe des Algorithmus 1 CLOSURE(X,F) der Abschluss berechnet. Mit

Algorithm 9 CORRECTION

Eingabe: eine Menge F von FAs (kanonisiert)
Ausgabe: eine reduzierte Menge G von FAs ohne unnötige FAs

```
1:  G ← F
2:  for all FA1 X → Y ∈ G do
3:     for all FA2 X' → Y' ∈ G do
4:        if FA1 ≠ FA2 and Y = Y' and X' ⊆ X then
5:           G ← G − {X → Y}
6:           break
7:        end if
8:     end for
9:  end for
10: return G
```

Hilfe der so bestimmten Menge von (erreichbaren) Attributen aus dem Abschluss, lässt sich jetzt zu jeder FA $X \rightarrow Y$ alle rechten Seiten, die von X erreichbar sind, herstellen.
Führe für jede FA $X \rightarrow Y$ in F folgende Schritte durch:

1. bestimme den Abschluss von X bzgl. F
2. dann werden neue FAs erzeugt, in der Form FA $X \rightarrow Y'$ für alle $Y' \in$ CLOSURE(X,F)

Algorithm 10 DETCREATION

Eingabe: Menge F von FAs
Ausgabe: Menge aller Determinaten der Eingabemenge

```
1:  A ← {}
2:  G ← {}
3:  for all FA X ← Y ∈ F do
4:     A ← CLOSURE(X,F)
5:     for all Y' ∈ A and Y' ≠ X do
6:        G ← G ∪ {X → Y'}
7:     end for
8:  end for
9:  ...ERWEITERUNG...
10: return G
```

Bemerkung 13 *Die Variable A sei in dem Algorithmus 10 eine Menge von Attributen.*