Projekt MinCover
Algorithmen für die minimale Überdeckung von funktionalen Abhängigkeiten

Denis T. Wirries

12. August 2006

Zusammenfassung

Die Definitionen und Algorithmen dienen zur Berechnung der minimalen Überdeckung von funktionalen Abhängigkeiten und ähnlichen Anwendungen. Der erste Abschnitt führt mit einen kurzen Überblick über die wichtigsten Definition in die Thematik der funktionalen Abhängigkeiten ein. Der zweite und dritte Abschnitt beschäftigt mit der verwendete Datenstruktur und Algorithmen, die Definitionen der Algorithmen erfolgt in Pseudocode und beschreibt die grundlegenden Ideen, die hinter ihnen stecken. Die Datenstrukturen und Algorithmen wurden dann in C# implementiert und als Anwendung, die im vieren Abschnitt beschreiben wird, veröffentlicht. Das entsprechende Projekt finden Sie unter http://www.netpage-germany.de/projects/mincover oder unter SourceForge.Net, es steht unter der GNU Library oder „Lesser“ General Public License (LGPL).

Das Projekt „MinCover“ ist ein akademisches Framework (Bibliothek) für funktionale Abhängigkeiten (FA) mit Algorithmen zur Berechnung der minimalen Überdeckung, kompletten Überdeckung und ähnlicher Überdeckung. Das Framework implementiert das volle Spektrum der funktionalen Abhängigkeiten mit Attributen, Attributlisten und Listen von funktionalen Abhängigkeiten. Als Programmiersprache wurde C# verwendet. Die Beispielanwendung „MinCover“ zeigt die Anwendung dieser Algorithmen und nutzt zur graphischen Visualisierung der Ergebnisse die ”Netron Graph Bibliothek”(auch Projekt bei SF.Net).

Vorabversion zur Framework Version 2.0.2006.0809

1 Definitionen

Grundlegende Definitionen über die minimale Überdeckung von funktionalen Abhängigkeiten. [1]

1.1 Was sind funktionale Abhängigkeiten?

Funktionale Abhängigkeiten (Abk. *FA* , englisch functional dependency , FD) sind die Grundlage für die Normalisierung von Relationenschemata in der Datenbankwelt. Sie bilden somit die Grundvoraussetzung für die Analyse und Verbesserung von gegeben Schemata.

Eine Relation wird durch *Attribute* definiert. Einige dieser Attribute können nun andere Attribute eindeutig bestimmen, diese Beziehung nennt man funktionale Abhängigkeit. So kann man sich zum Beispiel vorstellen das in einer Kundendatenbank, eine Kundennummer genau zu einem Kundenname führt. Dies würde man wie folgt aufschreiben:

$$[\ \text{Kundennummer} -> \text{Kundenname}\ ]\ ^{2}$$

Weiter kann man sich nun aber auch überlegen, das z.B. der Straßename und der Ort die Kundennummer eindeutig bestimmen. Dies lässt aber nun auch die Folgerung zu, das der Straßenname und der Ort den Kundennamen eindeutig bestimmen. In diesem Fall hängen der Straßename und der Ort zusammen, so spricht von einem *zusammengesetzten Attribut* oder *Attributmenge* . Diese funktionale Abhängigkeit schreibt man wie folgt auf:

$$[\ \text{Straßenname Ort} -> \text{Kundennummer}\ ]$$
$$[\ \text{Straßenname Ort} -> \text{Kundenname}\ ]$$

oder zusammengeschrieben[3]:

$$[\ \text{Straßenname Ort} -> \text{Kundennummer Kundenname}\ ]$$

Bestimmen nun einige Attribute einer Relation eindeutig die Werte aller Attribute der Relation, so spricht man von einem Superschlüssel , d.h. jedes einzelne Attribut der gesamten Relation ist durch diese Attribute eindeutig bestimmt. (2) Einen minimalen Superschlüssel nennt man Schlüsselkandidat , wenn er eine minimale Attributmenge ist, von der alle Attribute der Relation funktional abhängig sind. (1)

Beispiel 1.1 *Gegeben sei folgenden Relation R:*

R	*A*	*B*	*C*
	1	*1*	*3*
	1	*1*	*3*
	1	*2*	*4*

[1] Im weiteren Verlauf sind funktionale Abhängigkeiten und Funktionalabhängigkeiten äquivalent und wird abgekürzt mit FA.

[2] Notation in Anlehnung in die spätere Anwendung des Frameworks

[3] weitere Regeln für FA im Satz 1.1 Axiome von Armstrong

In diesem Beispiel ist C funktional abhängig von A und B, geschrieben:

$$[\ A\ B\ ->\ C\ ]$$

Der Pfeil kann somit gelesen werden als „bestimmt eindeutig". Aber C ist nicht funktional abhängig von A allein.

Es gelten auch die funktionalen Abhängigkeiten (und sonst keine weiteren mehr):

$$[\ C\ ->\ A\ ]$$
$$[\ B\ ->\ A\ ]$$
$$[\ B\ ->\ C\ ]$$

Definition 1.1 (Formale Definition einer funktionalen Abhängigkeit) *Sei $r(R)$[4] eine Relation mit Schemata R und seien α und β Teilmengen von Attributen aus dem Schemata von R. Sei $t \in r(R)$ ein Tupel aus der Relation R. Dann ist $t.\alpha$ die Einschränkung (Projektion) des Tupels t auf die Attribute aus α. Die funktionale Abhängigkeit $[\alpha -> \beta]$ (β ist funktional abhängig von α) gilt auf R, wenn für jedes Tupel aus $r(R)$ gilt:*

$$\forall t_1, t_2 \in r(R) : t_1.\alpha = t_2.\alpha \Rightarrow t_1.\beta = t_2.\beta$$

Das heißt, für alle $t_1, t_2 \in r(R)$ mit gleichen α-Attributen gilt auch, dass ihre β-Attribute über einstimmen. Die Werte der Attribute aus der Attributmenge α bestimmen somit die Werte aus der Attributmenge β eindeutig. β heißt nun voll funktional abhängig von α, wenn aus α kein Attribut mehr entfernt werden kann, so dass die Bedingung immer noch gilt.

Satz 1.1 (Axiome von Armstrong) *Mit Hilfe der Axiome von Armstrong (auch Armstrong Axiome) lassen sich aus einer Menge von funktionalen Abhängigkeiten, die auf einer Relation gelten, weitere funktionale Abhängigkeiten ableiten. Die folgenden drei Regeln reichen aus, um alle funktionalen Abhängigkeiten herzuleiten:*

1. *Eine Menge von Attributen bestimmt eindeutig die Werte einer Teilmenge dieser Attribute (triviale Abhängigkeit), d.h. $\beta \subseteq \alpha \Rightarrow \alpha \rightarrow \beta$ (Reflexivitätsregel).*
2. *Gilt $\alpha \rightarrow \beta$, so gilt auch $\alpha\gamma \rightarrow \beta\gamma$ für jede Menge von Attributen γ (Erweiterungsregel, Verstärkung).*
3. *Gilt $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$, so auch $\alpha \rightarrow \gamma$ (Transitivitätsregel).*

Die Armstrong-Axiome sind korrekt und vollständig [5], d.h. sie beschreiben nur semantisch gültige Implikationen und mit ihnen sind alle semantisch gültigen Implikationen herleitbar.

Auch sind folgende Herleitungsregeln oft nützlich, sie folgen bereits aus den Armstrong-Axiomen:

4. *Gelten $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$, so gilt auch $\alpha \rightarrow \beta\gamma$ (Vereingungsregel).*
5. *Gilt $\alpha \rightarrow \beta\gamma$, so gelten auch $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$ (Dekompositions-/Zerlegungsregel).*
6. *Gilt $\alpha \rightarrow \beta$ und $\gamma\beta \rightarrow \delta$, so auch $\alpha\gamma \rightarrow \delta$ (Pseudotransitivitätsregel).*

[4] $r(R)$ ist die Ausprägung (bzw. Zustand) der Relation mit dem Schemata $R =< A_1, \ldots, A_n >$, wobei $A_1, \ldots A_n$ Attribute sind.

[5] siehe (1)

1.2 Abschluss, Hülle und Transitivität

Um nun die minimale Überdeckung berechnen zu können, müssen wir noch ein paar Eigenschaften definieren, die wir im Zusammenhang mit Mengen von funktionalen Abhängigen brauchen. Diese Eigenschaften sind ziemlich grundlegend und werden von vielen weiteren Algorithm genutzt.

Definition 1.2 (Abschluss bzgl. einer Attributmenge) *X^+ ist der Abschluss einer Attributmenge X bezüglich einer Menge von funktionalen Abhängigkeiten F. Sie umfasst die Menge aller Attribute Y, für die gilt $X \rightarrow Y$ in $F^+(X)$ bzw. F^*.*

Sei die Attributmenge U gegeben durch alle Attribute, die in F vorkommen, so wird der Abschluss bzgl. der Attributmenge $X \subseteq U$ wie folgt definiert:

$$X^+ := \{Y \in U | [X \rightarrow Y] \in F^+(X)\}$$

Definition 1.3 (Hülle bzgl. einer Attributmenge) *$F^+(X)$ ist die Hülle einer Menge von funktionalen Abhängigkeiten F bezüglich einer Attributmenge X. Sie erweitert die vorhandenen funktionalen Abhängigkeiten F mit funktionalen Abhängigkeiten, die sich transitiv (nach Regel 3 aus Satz 1.1) aus F berechnen lassen und auf der linken Seite der funktional Abhängigkeiten die Attributmenge X enthält.*

Somit erhält man einen rekursiven Aufbau der gesuchten Menge $F^+(X)$:

$$\begin{aligned} F^+(X) \quad := \quad & \{F\} \\ & \cup \{[X \rightarrow Y_1] | [X \rightarrow Y_0] \in F \wedge [Y_0 \rightarrow Y_1] \in F\} \\ & \dots \\ & \cup \{[X \rightarrow Y_n] | [X \rightarrow Y_0] \in F \wedge \dots \wedge [Y_0 \rightarrow Y_n] \in F\} \end{aligned}$$

Definition 1.4 (Transitive Hülle) *...*

1.3 Äquivalenz von FA-Mengen

Um nun noch Überdeckungen zu vergleichen, brauchen wir folgende Definition:

Definition 1.5 (Äquivalenz) *Seien F und G zwei Mengen von funktionalen Abhängigkeiten mit gleichen Attributmengen. F und G sind äquivalent, wenn $F^+ = G^+$, d.h. wenn ihre Hüllen gleich sind. F heißt auch **Überdeckung** von G und umgekehrt.*

Anschaulich ausgedrückt, beide Mengen müssen die gleichen Abhängigkeiten enthalten, entweder direkt oder durch die transitive Hülle.

1.4 Determinanten

...

1.5 Minimale Überdeckung

Die minimale Überdeckung lässt sich nun wie folgt definieren:

Definition 1.6 (Minimale Überdeckung) *Eine Menge von funktionalen Abhängigkeiten F* ***ist minimal****, wenn gilt:*

1. *Jede rechte Seite von einer FA in F besteht aus einem Attribut (Kanonisierung).*
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.*

2 Datenstruktur

In diesem Abschnitt werde ich die grundlegenden Datenstruktur und -typen des Framework definieren. Die Struktur wird in UML angeben und die Definition der Datentypen erfolgt in Pseudo-Code, angelehnt an die Microsoft C# Notation.

2.1 Objekt-Modell

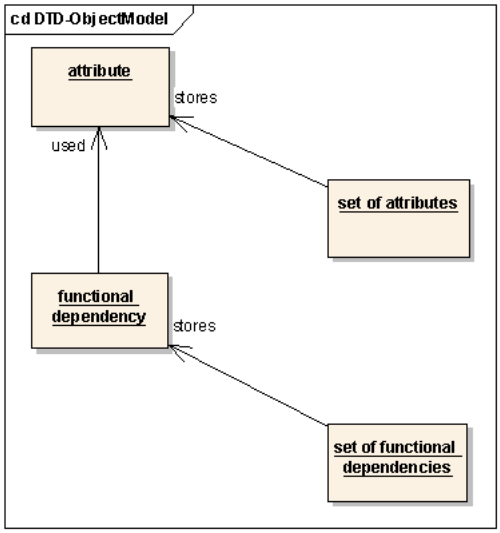


Abbildung 2.1: Objekt-Modell

Wie in Abbildung 2.1 zu sehen ist, gibt es vier grundlegende Datentypen: Attribute (attribute), Menge von Attributen (set (list) of attribute), funktionale Abhängigkeiten (functional dependency) und Menge von funktionalen Abhängigkeiten (set (list) of functional dependencies).

Wie auch in der Abbildung 2.1 zu sehen ist, besteht ein enger Zusammenhang zwischen diesen Datentypen. Ein Attribut ist so zu sagen der Grundbaustein, da alle anderen Datentypen ihn verwenden. Ein Attribut kann sich aus einem einzelnen Element, z.B. A, oder einer Menge von Elementen zusammensetzen, z.B. A B C ... .

Auf diesen Datentyp Attribut baut nun die Menge (oder auch Liste) von Attributen auf. Der Begriff Menge kann hier im mathematischen Sinne auf gefasst werden, da die Reihenfolge der Attribute keine Rolle spielt und doppelte Attribute ignoriert werden. Die Menge von Attributen enthält, wie der Name schon sagt, einzelne Elemente vom Datentyp Attribut.

Der Datentyp funktionale Abhängigkeit realisiert nun mit dem Datentyp Attribut eine funktionale Abhängigkeit wie man sie kennt. Sie setzt sich aus einer linken und rechten Seite zusammen, die jeweils durch ein Attribut repräsentiert wird.

Zum Schluss gibt es noch den Datentyp Menge von funktionalen Abhängigkeiten, der das Gegenstück zu dem Datentyp Menge von Attributen ist. Die Menge von funktionalen Abhängigkeiten unterstützt die meisten Operationen und ist daher für die Berechnung von großer Bedeutung.

2.2 Attribute und ihre Mengen

Datentyp Attribute

Wie in Abbildung 2.2 zu sehen ist, definiert das Interface IAttribute die notwendigen Schnittstellen für das Objekt Attribute. Die Klasse Attribute ist dann die konkrete Implementierung des Interface.

Ein Attribut wird eindeutig über seinen Namen identifiziert. Ein Attribut kann einfach, d.h. nur einen Namen haben, oder zusammengesetzt sein, d.h. mehrere Namen haben, z.B. A B C ... . Das Attribut speichert seine Namen alphabetisch.

Datentypdefinition 2.1

Interface: IAttribute
Class: Attribute
Namespace: NetPage_Germany.MinCover.Framework
Sichtbarkeit: Public

public void Add(String name);

public void Add(String[] names);

public void Remove(String name);

public void Remove(String[] names);

public Boolean Equals(IAttribute attribute);

public Boolean Contains(IAttribute attribute);

public String GetAttributeName();

public String[] GetAttributeNames();

public IAttribute GetCopyOf();

public Boolean IsJoinedAttribute();

public int Size
{
get;
}

public String ToString();

Datentyp AttributeList

Wie in Abbildung 2.2 zu sehen ist, definiert das Interface IAttributeList die notwendigen Schnittstellen für das Objekt Menge von Attributen. Die Klasse AttributeList ist dann die konkrete Implementierung des Interface.

Die Menge von Attributen kann Attribute aufnehmen und diese dann auch wieder entfernen. Die Attribute werden in alphabetischer Reihenfolge gespeichert, man kann abfragen, ob ein Attribut in der Liste enthalten ist oder nicht.

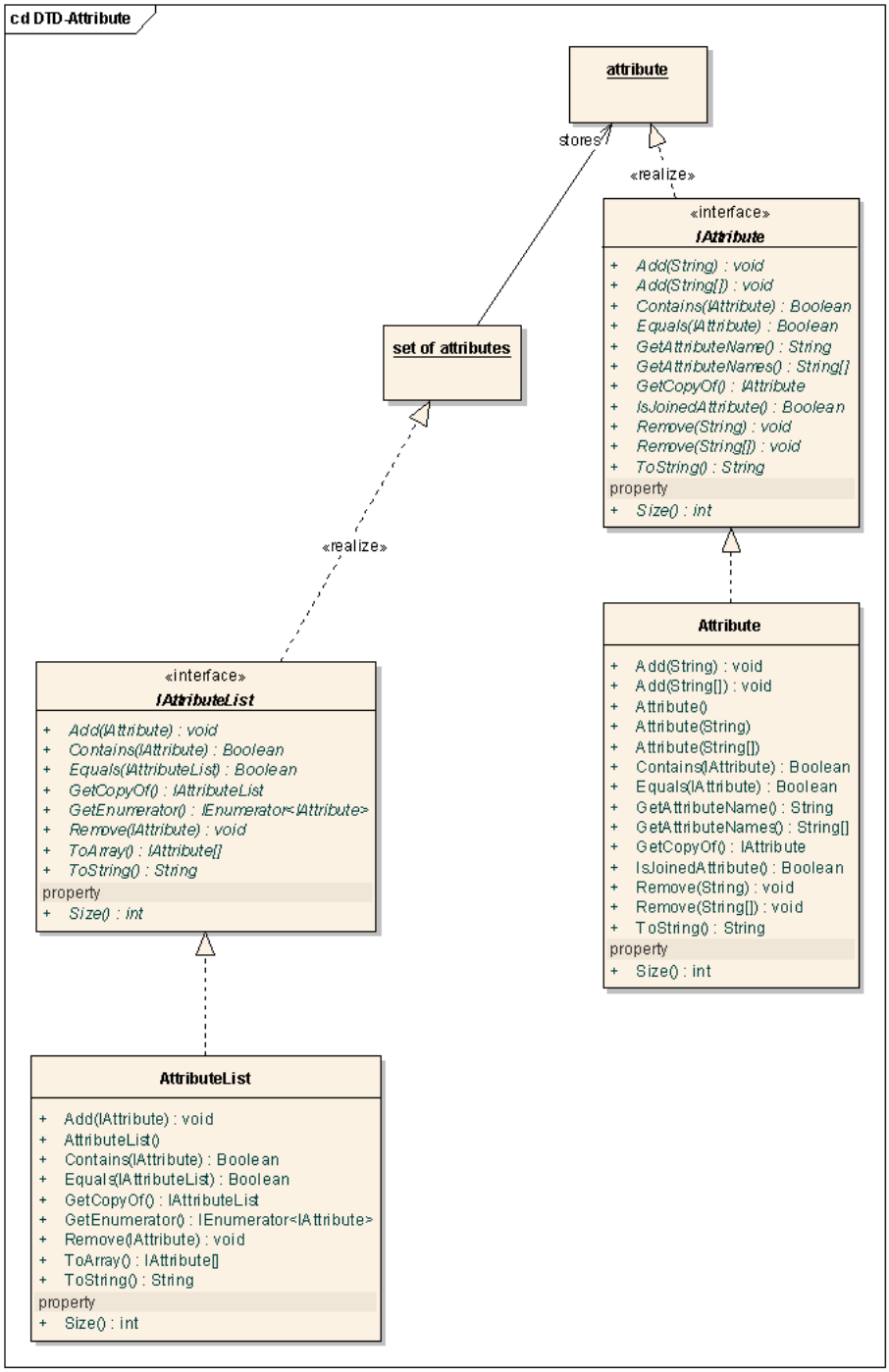


Abbildung 2.2: Attribut und Menge von Attributen

Datentypdefinition 2.2

Interface: IAttributeList
Class: AttributeList
Namespace: NetPage_Germany.MinCover.Framework
Sichtbarkeit: Public

public void Add(IAttribute attribute);

public void Remove(IAttribute attribute);

public void Boolean Contains(IAttribute attribute);

public Boolean Equals(IAttributeList attributeList);

public IEnumerator<IAttribute> GetEnumerator();

public IAttributeList GetCopyOf();

public int Size
{
get;
}

public IAttribute[] ToArray();

public String ToString();

2.3 Funktionale Abhängigkeiten und ihre Mengen

Datentyp FunctionalDep

Wie in Abbildung 2.3 zu sehen ist, definiert das Interface IFunctionalDep die notwendigen Schnittstellen für das Objekt funktionale Abhängigkeit. Die Klasse FunctionalDep ist dann die konkrete Implementierung des Interface.

Die funktionale Abhängigkeit verhält sich ganz wie erwahrtet. Die Attribute der linken bzw. rechten Seite sind nach ihrem Namen alphabetisch geordnet.

Datentypdefinition 2.3

Interface: IFunctionalDep
Class: FunctionalDep
Namespace: NetPage_Germany.MinCover.Framework
Sichtbarkeit: Public

public Boolean Equals(IFunctionalDep FA);

public IAttribute LeftSide
{
get;
}

public IAttribute RightSide
{

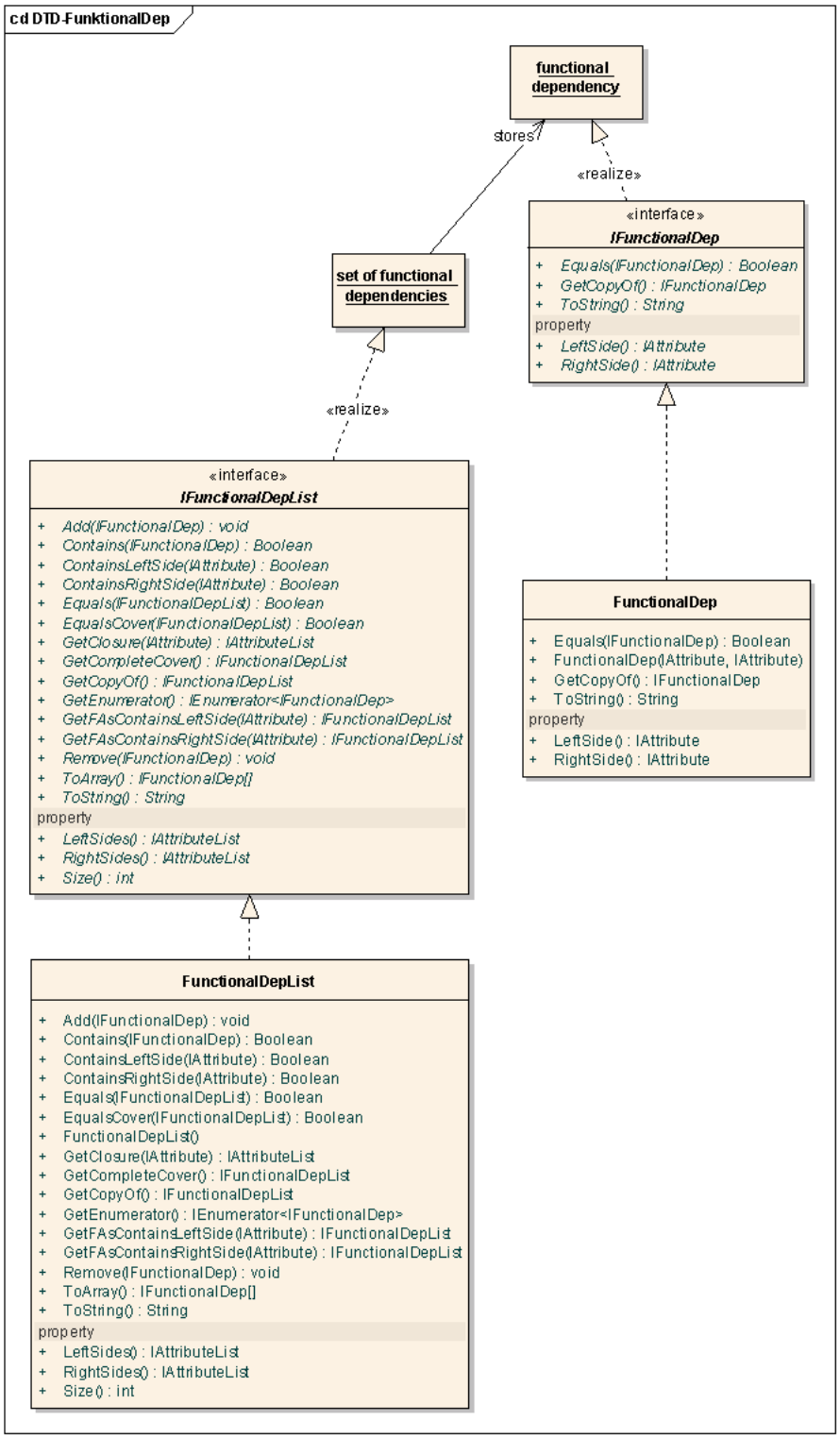


Abbildung 2.3: Attribut und Menge von Attributen

get;
}

public IFunctionalDep GetCopyOf();

public String ToString();

Datentyp FunctionalDepList

Wie in Abbildung 2.3 zu sehen ist, definiert das Interface IFunctionalDepList die notwendigen Schnittstellen für das Objekt Menge von funktionale Abhängigkeit. Die Klasse FunctionalDepList ist dann die konkrete Implementierung des Interface.

Dieser Datentyp ist wichtigste Datentyp, um Berechnungen auf funktionalen Abhängigkeiten durchzuführen. Er wird später von den Algorithmen in Abschnitt 3 verwendet. Grundsätzlich speichert er die einzelnen Abhängigkeiten, doppelte FAen werden ignoriert. Die FAen werden in alphabetischer Reihenfolge gespeichert. Man kann prüfen, ob eine FA enthalten ist, den Abschluss bzgl. eines Attributes berechnen, gleiche Überdeckung ...

Datentypdefinition 2.4
Interface: IFunctionalDepList
Class: FunctionalDepList
Namespace: NetPage_Germany.MinCover.Framework
Sichtbarkeit: Public

public void Add(IFunctionalDep FA);

public void Remove(IFunctionalDep FA);

public Boolean Contains(IFunctionalDep FA);

public Boolean ContainsRightSide(IAttribute attribute);

public Boolean ContainsLeftSide(IAttribute attribute);

public Boolean Equals(IFunctionalDepList FAList);

public Boolean EqualsCover(IFunctionalDepList FAList);

public int Size
{
get;
}

public IEnumerator<IFunctionalDep> GetEnumerator();

public IFunctionalDepList GetFAsContainsLeftSide(IAttribute attribute);

public IFunctionalDepList GetFAsContainsRightSide(IAttribute attribute);

public IAttributeList LeftSides
{
get;
}

public IAttributeList RightSides
{

```
  get;
 }
public IFunctionalDepList GetCopyOf();
public IFunctionalDepList GetCompleteCover();
public IAttributeList GetClosure(IAttribute attribute);
public IFunctionalDep[] ToArray();
public String ToString();
```

---

3 Algorithmen

Die Definition der Algorithmen erfolgt in Pseudo-Code, angelehnt an die Microsoft C# Notation. Auf die Definition der Datenstruktur wurde bereits in Abschnitt 2 eingegangen.

Allgemeine Variablen und ihre Bedeutung[6]*:*

Variable	Bedeutung
X, Y, Z	Variable für Attribute (können auch zusammengesetzte Attribute sein), siehe Datentyp 2.2 Attribut
FA	eine funktionale Abhängigkeit mit den Attributen X und Y, siehe Datentyp 2.3 funktionale Abhängigkeit
$X \rightarrow Y$	
F, G	Variablen für die Mengen von funktional Abhängigkeiten, siehe Datentyp 2.3 Menge von funktionalen Abhängigkeiten

Bemerkung 3.1 *Die Datenstrukturen für die Listen von Attributen und FAs ignorieren mehrfach gleiche Einträge, z.B.* $F = (\{X \rightarrow Y\}) \Rightarrow F \leftarrow F \cup \{X \rightarrow Y\} \Rightarrow F = (\{X \rightarrow Y\})$*, genauso bei Attributlisten.*

3.1 Algorithmus Closure (Abschluss)

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 3.1 CLOSURE

Eingabe: Ein Attributen 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 3.2 *Die Variablen OLDDEP und NEWDEP sind in dem Algorithmus 3.1 Mengen von Attributen.*

[6] wenn nichts anderes definiert ist

3.2 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 3.2 MEMBER

Eingabe: FA $X \rightarrow Y$, F
Ausgabe: `TRUE`, wenn $F \models X \rightarrow Y$, sonst `FALSE`

```
1: if Y ⊆ CLOSURE(X, F) then
2:     return(TRUE)
3: else
4:     return(FALSE)
5: end if
```

Bemerkung 3.3 *Der Algorithmus verwendet den Algorithmus 3.1 Closure (Abschluss).*

3.3 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 3.3 MINCOVER

Eingabe: eine Menge F von FAs (siehe Bemerkung 3.5)
Ausgabe: eine minimale Überdeckung für F

```
1: G ← F
2: for all FA {X → Y} ∈ F do
3:     if MEMBER({X → Y}, G − {X → Y}) then
4:         G ← G − {X → Y}
5:     end if
6: end for
7: return G
```

Bemerkung 3.4 *Dieser Algorithmus verwendet den Algorithmus 3.2 (MEMBER) und implizit auch der Algorithmus 3.1 (CLOSURE).*

Bemerkung 3.5 *Die Eingabemenge sollte bereits voroptimiert sein, dann liefert der Algorithmus bessere minimale Überdeckungen, siehe dazu die Algorithmen zur Optimierung der Eingabemenge.*

Bemerkung 3.6 *Zeile 2: Bei MEMBER($\{X \rightarrow Y\}$, $G - \{X \rightarrow Y\}$) soll (und darf!) die Menge G nicht verändert werden.*

3.4 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 Menge erzeugt und verglichen. Voraussetzung für diese Verfahren ist, das die Eingabemengen kanonisch (siehe

Algorithmus 3.5 (KANONIZE)) und vorkorrigiert (siehe Algorithmus 3.8 (DETCREATION)) sind, damit die Mengen möglichst einfach sind.

Algorithm 3.4 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
```

3.5 Algorithmen zur Optimierung von funktionalen Abhängigkeitsmengen

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

Algorithm 3.5 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 3.7 *Der Algorithmus 3.5 (KANONIZE) sollte immer im Vorfeld der Berechnung der minimale Überdeckung ausgeführt werden. Dies reduziert die Komplexität der Algorithmen*

3.3 (MINCOVER), man muss nicht darauf achten, ob die rechte Seite ein zusammengesetztes Attribut ist.

Algorithm 3.6 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
```

Bemerkung 3.8 *Der Algorithmus 3.6 (DEKANONIZE) dient nur zur besseren Darstellung der Ergebnismenge von FAs. Er hat keine weitere Bedeutung in der Berechnung der minimalen Überdeckung der funktionalen Abhängigkeiten.*

Algorithmen zur Optimierung der Menge von FAs

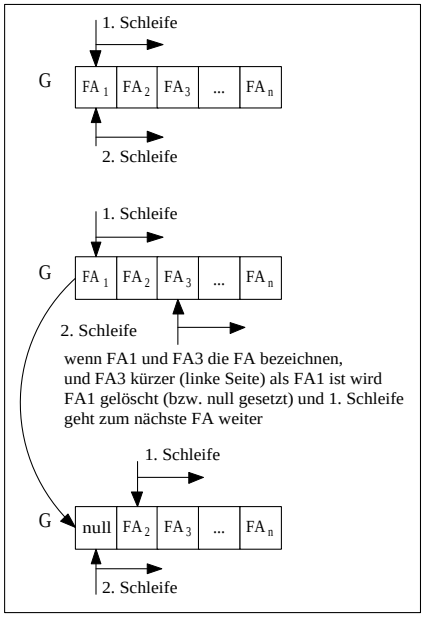
Verschiedene Algorithmen zur Optimierung der Eingabemenge von FAs.

Algorithmus CORRECTION Der Algorithmus 3.7 (CORRECTION) reduziert die Eingabeliste für den Algorithmus 3.3 (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 3.5 (KANONIZE) und 3.8 (DETCORRECTION) ausgeführt werden, weil die Korrektur bei der rechten Seite nicht auf zusammengesetzte Attribute achtet und nicht die rechte Seite weiter ableitet.
Allgemeine Arbeitsweise des Algorithmus (siehe auch Abbildung):

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 aus geführt, bis die Liste der FAs abgearbeitet ist.

Beispiel 3.1 (Beispiel für Algorithmus CORRECTION)

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



Algorithm 3.7 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
```

Algorithmus DETCREATION Der Algorithmus 3.8 erzeugt alle Determinanten einer Menge von FAs, d.h. es werden zu jedem Attribut X einer FA $X \rightarrow Y$ alle FA hinzugefügt, die von X aus erreicht werden können. Die Arbeitsweise von DETCREATION ist relativ simpel, es wird zu jeder FA $X \rightarrow Y$ mit Hilfe des Algorithmus 3.1 CLOSURE(X,F) der Abschluss berechnet. Mit 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)

Bemerkung 3.9 *Die Variable A sei in dem Algorithmus 3.8 eine Menge von Attributen.*

[6] Arbeitsweise des Algorithmus KORRECTION

Algorithm 3.8 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
```

4 Programm

4.1 Verwendete Frameworks - Netron Graph Library Version 2.2

Siehe dazu die Netron Graph Libray

Summary of essential features - Solid open source C# code, totally free - Fully CLS compliant, can be used in any .Net language - Easy to use - Visual Studio toolbox controls - Drag-drop support - Extensive code comments, tutorials, complete applications and documentation online - Integrates into the Netron framework (NAF) but can be used outside it too - Programming interface to handle graphs, node and connections - Interactive editing and context menu's - Plugable and extensible model (uses .Net reflection) of shapes, connections and shape libraries - Layout algorithms - Support for visual data flow programming paradigm and development environments - Support for GraphML - Zoom in and out of the canvas - Continuously improved and used in our own applications - Fully supports the CLR and 100% managed code, no extra assemblies required - Full interactive access to traditional graph algorithms (Dijkstra, Floy, Prim,...) - Ships with advanced applications showing in detail how to use in real-world applications

Solid open source code The source code was written and architectured with care and contains a lot of coding comments in case you want to tweak the library or simply understand the mechanics. You can find extensive help, tutorials and other stuff online on the Netron site. Support is available via the forums on the Netron site, we would be glad to hear from you and receive some feedback to improve the library.

Layout algorithms Layout algorithms try to organize graphs in a visually pleasing way. The Netron graph control contains the spring embedder algorithm and the tree or rectangular algorithm. In future version more algorithms will be supplied and more graph mathematics will be fused into the graph control.

What's in the box? If you start up the Visual Studio solution under the 'GraphApplications' directory, you will see the following list of projects. See also the Setup topic below. the AutomataShapes library is a collection of graph shapes for the Automatron application the Automatron application shows some advanced custom shapes the BasiShapes library is a collection of general purpose graph shapes the Biotron application (aka Visual Logo) shows how to implement the visual programming paradigm with the graph library the BiotronCore library contains the essential classes and interface of the Biotron application the BiotronShapes library is a collection of graph shapes to give life to the bionts of the Biotron application the GenericGraphApplication application is a simple demonstration of the graph library accessing some of its most basic properties and features the GraphAnalysis application shows how you can benefit from the graph library and the data structures library to do some traditional graph analysis the GrooglerII application is an example of how you can use the graph layout algorithm to represent and lay out knowledge graphs the NetronDataStructures library is a collection of well-known data structures (queue, graph, stack, linked list and so on) the NetronGraphLib library is the actual graph assembly with the graph control and other elements related to it the NetronMaths library is a collection of classes related to mathematical operations

What can I do with it? This application is in the first place a drawing application with which you can create basic diagrams and flowcharts (similar to Microsoft Visio). It is fairly easy to extend the shipped drawing shapes with your own, see the online tutorial ' How to implement and use automata shapes' for details. Some advanced features like the data flow and the layout algorithm are not used by this application, if you wish to try these out see (respectively) the Biotron application above and the Groogler application below.

This application shows how to use the various traditional graph algorithms from the new Netron data structures library and how the results can be shown interactively using the graph library.

The Netron graph library can be used for doing traditional graph analysis, i.e. testing graph-like structures for:

shortest path

connectedness

cyclic property

Hamiltonian property

and so on. Since the graph data structure (as well as other) is widely used and emphasis was put on data flow aspects of the graph library, the graph analysis part was taken to a separate assembly called the 'NetronDataStructures'. The graph analysis application shows you how you can combine the aforementioned library and the graph control to explore graphs and their mathematical properties.

This application and the algorithms behind require some mathematical background though the basic ideas are simple to grasp. It would require a textbook to explain all the machinery behind and the current assemblies are by no means complete since the field of graph theory is really a world on its own. Despite this, this current version should allow a wider audience of users to use the graph control and to implement more refined layout algorithms since these algorithms often depend on the precise mathematical form of graphs.

Recently a lot of research is going on in graph theory and random networks in particular. The so-called small-world phenomenon, also called six degrees of freedom, pervades both technology and natures: the way our brain is connected, the way the internet hangs together, the social relationships of people...it seems to underly a lot of systems. In this graph analysis demo you can experiment with various random graph algorithm, among which the Watts-Strogatz small-world algorithm.

The Netron data structures contains a lot of structures, among which the 'graph' class and the 'algorithms' class. First, a random graph is generated corresponding to a random adjacency matrix. This matrix is used to feed the graph class and the algorithm class, containing the various graph algorithms. Once, these class have outputted their calculations, a loop is going over the newly created adjacency matrix to draw the directed graph. In this way, both libraries (the graph library and the data structures) are independent and can be used independently of each other.

How the algorithms work and the theory behind it lies outside the scope of this document but you will find more information on the Netron site.

Index

Literatur

[1] Lipeck, Prof. Dr. Udo: *Datenbanksysteme IIa Wintersemester 2005/06*. Vorlesung Datenbanksysteme, 2005/06.

[2] Wikipedia: *http://de.wikipedia.org/wiki/Funktionale_Abhängigkeit*, Juli 2006.

A Project Summary

The project „MinCover“ is an academic framework for functional dependencies (FD) with algorithms for the computation of the minimum cover, complete cover and equal cover. The framework implements the full spectrum of functional dependencies with attributes, lists on attributes and lists on functional dependencies.

As programming language Microsoft C# was used. The application of examples „MinCover“ shows the application of these algorithms and uses for the graphic visualization of the results the „Netron Graph Library“ (see project Netron Reloaded on SF.Net).

The project is under GNU Library or „Lesser“ General Public License (LGPL).

B GNU Lesser General Public License

Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser"General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the libraryänd a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called ”this License”). Each licensee is addressed as ”you”.

A ”library”means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The ”Library”, below, refers to any such software library or work which has been distributed under these terms. A ”work based on the Library”means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term ”modification”.)

SSource code”for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any

application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library"with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Libraryüses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though

the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library"with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library"must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and äny later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY ÄS IS"WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR

A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Libraries If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ”copyright”line and a pointer to where the full notice is found.

MinCover Copyright (C) 2006 Denis T. Wirries

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA