Analyse von Softwarevarianten

Christian Lindig

Informatik-Bericht Nr. 98-04
Januar 1998

© Abteilung Softwaretechnologie
Institut für Programmiersprachen
Technische Universität Braunschweig
Bültenweg 88
D-38092 Braunschweig
Germany

Analyse von Softwarevarianten

Christian Lindig
TU Braunschweig
Institut für Programmiersprachen und
Informationssysteme, Abt. Softwaretechnologie
D–38 100 Braunschweig

Januar 1998

Zusammenfassung

Software-Quelltexte werden oft durch den Einsatz eines Präprozessors an verschiedenen Zielplattformen angepaßt. Aus einem Quelltext entstehen dabei durch den Präprozessor verschiedene Varianten der Software, die einen Variantenverband bilden. Formale Begriffsanalyse ist eine mathematische Theorie, mit deren Hilfe der Variantenverband von Quelltexten effizient bestimmt werden kann. Darüberhinaus können Redundanzen in der Beschreibung der Variantenstruktur entdeckt und entfernt werden.

Ein Softwareprodukt ist oft nur ein Teil einer größeren Produktfamilie, deren Mitglieder durch Kombination von bestehenden Grundkomponenten entstehen. Die Mitglieder der Familie sind Varianten, die sich in ihrer Funktionalität oder den Plattformen, auf denen sie ablaufen unterscheiden. Bedingte Übersetzung von Quelltexten ist die gebräuchlichste Methode des Programmieren-im-Kleinen um Varianten zu erzeugen. Das Werkzeug dazu ist der C-Präprozessor (CPP): logische Ausdrücke über Variablen des Präprozessors bestimmen, welche Teile eines Quelltextes Eingang in eine Variante finden. Eine unterschiedliche Belegung der Variablen erzeugt aus einem Quelltext eine Menge von Varianten. Die so aus einem Quelltext erzeugbaren Varianten und ihre Beziehungen untereinander sind in der Praxis bislang wenig beachtet worden – zu Unrecht. Sie bieten eine von der konkreten Implementierung abstrahierte Sicht auf die Variantenstruktur eines Quelltextes. Durch die Analyse bestehender Quelltexte können sowohl Differenzen zu der intendierten Struktur der Varianten aufgedeckt werden, als auch Redundanzen in ihrer Beschreibung durch CPP-Ausdrücke. Ungetestete Varianten werden so entdeckt und die sie erzeugenden CPP-Ausdrücke können vereinfacht werden.

1 Einführendes Beispiel

Ein einfaches Beispiels soll zeigen, was die Analyse von Quelltexten leistet: Abbildung 1 enthält einen Quelltext, bestehend aus verschiedenen *Codesegmente* ($S_1, \ldots, S_6$), die durch CPP-Ausdrücke kontrolliert werden. Abhängig von den Werten der beteiligten Variablen $E_1, \ldots, E_8$ entstehen Varianten; jede Variante ist eine Teilmenge der Codesegmente $S_1, \ldots, S_6$.

Da Varianten Untermengen der Codemenge $S_1, \ldots S_6$ sind, bilden die sie einen Untermengen-Verband – den *Variantenverband* (Abbildung 1). Der Verband ist das Ergebnis der Analyse

```
#if defined(E_1) && defined(E_6) && defined(E_8)
  S_3
#if defined (E_3) && defined(E_5) && defined(E_7)
  S_1
#endif
#endif
  S_6
#if defined(E_2) && defined(E_5) && defined(E_7)
  S_2
#endif
#ifdef E_3
#if defined(E_4) && defined(E_6)
  S_4
#ifdef E_8
  S_5
#endif
#endif
```

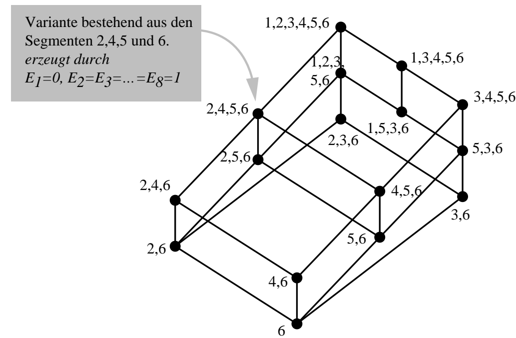


Abbildung 1: Quelltext und Verband seiner möglichen Varianten.

des Quelltextes. Die Elemente des Verbandes sind die Varianten und jede Kante zeigt eine Untermengenbeziehung zwischen ihnen an. Jede Variante wird durch eine eindeutige minimale Belegung der Variablen erzeugt – diese Information ist ebenfalls ein Ergebnis der Analyse, ist allerdings in Abbildung 1 nicht an den Varianten angetragen.

Die Variantenstruktur gibt Softwareentwicklern wertvolle Hinweise: enthält sie ungeplante Varianten, so können diese entweder entfernt werden oder sollten zumindest getestet werden. Die Zahl aller Varianten ist ein Maß für den zu erwartenden Testaufwand, wenn alle Varianten unabhängig voneinander getestet werden. Zudem ist klar, wie jede Variante erzeugt werden kann. Wenn Varianten nicht einzeln getestet werden, kann die Teilmengenbeziehung zwischen den Varianten zur Aufstellung eines Testplanes herangezogen werden.

CPP-Ausdrücke in einem Quelltext können Redundanzen enthalten. Dann können die Ausdrücke vereinfacht werden und trotzdem erzeugen die gleichen Variablenbelegungen die gleichen Varianten. Redundanzen sind nicht in jedem Fall unerwünscht, eine Analyse hilft zumindest, sie zu entdecken. Im Falle des Beispiels ist wahlweise die Variable E_5 oder E_7 redundant und zusätzlich die Variable E_6: entfernt man sie aus allen beteiligten Termen, so bleibt die Struktur der erzeugbaren Varianten identisch. Diese Vereinfachung kann auf Wunsch automatisch ausgeführt werden.

2 Begriffsbildung

Die Berechnung des Variantenverbandes ist nicht ein prinzipielles Problem. Durch eine systematische Belegung aller Variablen können alle Varianten erzeugt, und ihre Beziehung zueinander analysiert werden. Der Nachteil dieses Vorgehens ist sein *durchschnittlich* exponentieller algorithmischer Aufwand. Formale Begriffsanalsye ist ein effizientes mathematisches Verfahren zur Analyse binärer Relationen, das stattdessen für die Analyse von Varianten verwendet werden kann.

Formale Begriffsanalyse ist eine informationserhaltende Transformation, deren Ergebnis die Eigenschaften der Eingabe oft besser erkennen läßt. Die Eingabe ist eine binäre Relation, das Ergebnis ein sogenannter *Begriffsverband*. Für die Untersuchung der Variantenstruktur muß zunächst also eine Quelldatei mit CPP-Ausdrücken als eine binäre Relation dargestellt werden, die die wesentlichen Eigenschaften für die Variantenbildung erfaßt.

Definition 1 (Segment) *Ein Codesegment s ist ein Element aus einer endlichen Menge $\mathcal{S}$ von Segmenten. Jedes Codesegment s steht in Relation zu einer endlichen Menge $E \subseteq \mathcal{E}$ von kontrollierenden Ausdrücken.*

Jedes Codesegment einer Quelldatei wird zu den CPP-Ausdrücken in Beziehung gesetzt, die es kontrollieren. Der Gesamtzusammenhang zwischen Codesegmenten und Ausdrücken wird für einen Quelltext in der Konfigurationstabelle festgehalten:

Definition 2 (Konfigurationstabelle) *Eine* Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ *beschreibt den Zusammenhang zwischen einer Menge $\mathcal{S}$ von Segmenten und ihren Ausdrücken $\mathcal{E}$: $T \subseteq \mathcal{S} \times \mathcal{E}$.*

Abbildung 2 zeigt die zum Beispiel in Abbildung 1 gehörige Konfigurationstabelle. Sie ist eine binäre Relation, wie sie später für die Begriffsanalyse benötigt wird. In ihr sind für jedes Segment s jeder Ausdruck e als (s, e) verzeichnet, der einen Einfluß auf die Auswahl des Segments für eine Konfiguration besitzen.

	Ausdruck							
Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
S_4			•	•		•		
S_5			•					•
S_6								

Abbildung 2: Relation zwischen Ausdrücken und Codesegmenten

Der C-Präprozessor wertet die Ausdrücke zur Variantenbildung aus. Der Wert eines Ausdrucks hängt im Wesentlichen von den Belegungen der Variablen ab. Für die Analyse beschränken wir uns auf die Auswertung logischer Ausdrücke und berücksichtigen die anderen Aspekte wie Wertzuweisung und Vergleich von Werten nicht. Das abstrakte Gegenstück zum CPP ist die Konfigurationsfunktion.

Definition 3 (Konfigurationsfunktion) *Eine Konfigurationsfunktion f wertet Ausdrücke $e \in \mathcal{E}$ aus: $f : \mathcal{E} \to \{0, 1\}$. Sie ist eindeutig durch die Menge $E_f = \{e \in \mathcal{E} \mid f(e) = 1\}$ charakterisiert.*

An Hand der Werte der einzelnen Ausdrücke bestimmt der C-Präprozessor, welche Segmente zu einer Variante gehören. Für die Formalisierung verlangen wir, daß alle kontrollierenden Ausdrücke eines Segmentes einer Variante durch die Konfigurationsfunktion zu dem Wert 1

ausgewertet werden. Dies bedeutet, daß die kontrollierenden Ausdrücke Konjunktionen sind. Der Bedeutung dieser technischen Annahme wird später noch diskutiert. Das Beispiel ist gerade so gewählt, daß diese Bedingung erfüllt ist, und deshalb der Zusammenhang zwischen Quelltext, Definition einer Variante und der Konfigurationsfunktion einfach ist.

Definition 4 (Variante) *Eine Variante ist eine durch eine Konfigurationsfunktion f aus einen Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ ausgewählte Menge von Komponenten:* $f(\mathcal{S}, \mathcal{E}, \mathcal{T}) = \{s \in \mathcal{S} \mid f(e) = 1$ *für alle* $(s, e) \in \mathcal{T}\}$.

Die Segmente einer Variante lassen sich an der Konfigurationstabelle ablesen, wenn man weiß, welche Ausdrücke durch die Konfigurationsfunktion zu 1 ausgewertet werden: genau die Segmente, deren sämtliche Ausdrücke den Wert 1 besitzen.

Durch die oben stehenden Definitionen sind Varianten und ihre Entstehung aus Quelltexten abstrahiert worden. Formale Begriffsanalyse kann jetzt dazu benutzt werden, die Eigenschaften von Varianten zu untersuchen. Vorher ist es allerdings eine knappe Einführung in die Begriffsanalyse nötig.

3 Formale Begriffsanalyse

Der Ausgangspunkt der Begriffsanalyse ist eine binäre Relation über Objekten und ihnen zugeordneten Attributen, ein sogenannter *formaler Kontext*. Die Konfigurationstabelle eines Quelltextes ist ein Beispiel für einen Kontext, wobei Objekte Segmente sind und Attribute Ausdrücke, die Segmenten zugeordnet sind. Ein Kontext läßt sich als *Kontexttabelle* darstellen, wie in Abbildung 2.

Definition 5 (Kontext) *Ein formaler Kontext ist ein Tripel $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ aus den folgenden endlichen*[1] *Mengen: eine Objektmenge $\mathcal{O}$, eine Attributmenge $\mathcal{A}$ und einer Relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ zwischen ihnen. Die Schreibweise $(o, a) \in \mathcal{R}$ wird gelesen als: das Objekt o besitzt das Attribut a.*

Jede Menge von Objekten besitzt eine (möglicherweise leere) Menge von gemeinsamen Attributen: die Menge $\{C_4, C_5\}$ besitzt das gemeinsame Attribut E_3, geschrieben: $\{C_4, C_5\}' = \{E_3\}$. Analog besitzen Attribute gemeinsame Objekte, zum Beispiel $\{E_5, E_7, E_8\}' = \{C_1\}$.

Definition 6 (Gemeinsame Objekte/Attribute) *Für eine Menge $A \subseteq \mathcal{A}$ eines Kontextes $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ werden die gemeinsamen Objekte definiert als* $A' = \{o \in \mathcal{O} \mid (o, a) \in \mathcal{R}$ *für alle* $a \in A\}$ *und analog die gemeinsamen Attribute für eine Menge $O \subseteq \mathcal{O}$:* $O' = \{a \in \mathcal{A} \mid (o, a) \in \mathcal{R}$ *für alle* $o \in \mathcal{O}\}$

Die zentrale Definition ist die des Begriffes: ein *Begriff* ist ein spezielles Paar aus einer Objekt- und einer Attributmenge. Die gemeinsamen Attribute der Objektes des Begriffes sind seine Attribute und umgekehrt sind deren gemeinsame Objekte die Objekte des Begriffes.

[1] Die allgemeine Begriffsanalyse setzt keine endlichen Mengen voraus. Im Fall der hier präsentierten Anwendung ist diese Einschränkung aber nötig.

Definition 7 (Begriff) *Ein* Begriff *eines Kontextes* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *ist ein Paar* $(O, A) \in 2^{\mathcal{O}} \times 2^{\mathcal{A}}$ *wobei gleichzeitig gilt:* $O' = A$ *und* $A' = O$*. Die Menge aller Begriffe eines Kontextes wird mit* $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *bezeichnet.*

Jeder Begriff läßt sich in der Kontexttabelle als ein ausgefülltes, maximales Rechteck darstellen, wenn man Zeilen und Spalten der Tabelle entsprechend umordnet. In Abbildung 3 ist ein Begriff mit je zwei Objekten und Attributen als ein solches Rechteck markiert.

	Ausdrücke							
Segmente	E_1	E_2	E_3	E_4	E_5	E_7	E_6	E_8
C_1	○		○		●	●	○	○
C_2		○			●	●		
C_3	○						○	○
C_4			○	○			○	
C_5			○					○
C_6								

Abbildung 3: Kontext von Ausdrücken und Codesegmenten mit markiertem Begriff.

Die Begriffe eines Kontextes sind (partiell) geordnet: ein Oberbegriff umfaßt mehr Objekte, die aber weniger gemeinsame Eigenschaften besitzen. Wegen der partiellen Ordnung sind nicht alle Begriffe mit einander vergleichbar.

Definition 8 (Ordnung) *Die folgende Relation* $\leq$ *definiert eine Halbordnung auf Begriffen* $(O_1, A_1), (O_2, A_2) \in B(\mathcal{O}, \mathcal{A}, \mathcal{R})$*:* $(O_1, A_1) \leq (O_2, A_2) \Leftrightarrow O_1 \subseteq O_2$*.*

Die zentrale Eigenschaft der Begriffe eines Kontextes ist, daß sie einen *Begriffsverband* bilden: zu jeder Menge B von Begriffen existiert ein eindeutiger kleinster Begriff (das *Supremum* $\bigvee B$), der nicht kleiner als die Begriffe der Menge ist. Umgekehrt existiert zu jeder Menge von Begriffen ein eindeutiger größter Begriff (das *Infimum* $\bigwedge B$), der nicht größer als jeder Begriff der Menge ist.

Theorem 1 (Hauptsatz der Begriffsanalyse [1]) *Sei* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *ein Kontext,* $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *ist dann ein vollständiger Verband, der* Begriffsverband *von* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$*. Infimum und Supremum sind gegeben durch:*

$$\bigwedge_{i \in I} (O_i, A_i) = (\bigcap_{i \in I} O_i \, , \, (\bigcup_{i \in I} A_i)'') \quad \textit{und} \quad \bigvee_{i \in I} (O_i, A_i) = ((\bigcup_{i \in I} O_i)'', \, \bigcap_{i \in I} A_i)$$

Der Begriffsverband eines Kontextes kann als Graph dargestellt werden: Abbildung 4 zeigt den Begriffsverband des Beispiels, der übrigens noch nicht der gesuchte Variantenverband ist. Dieser Begriffsverband beschreibt die Abhängigkeiten zwischen Segmenten und den sie kontrollierenden Ausdrücken.

Jeder Knoten des Graphen ist ein Begriff (also ein Mengen-Paar) und Kanten zeigen die Ober- und Unterbegriffsbeziehung an. Die Begriffe sind nicht mit ihren vollständigen Objekt- und

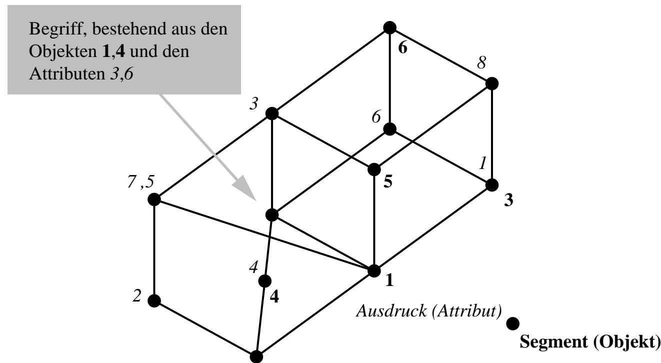


Abbildung 4: Begriffsverband des Beispiel-Kontextes aus Abbildung 3. Attribute sind die kontrollierenden Ausdrücke, Objekte sind Codesegmente.

Attributmengen gekennzeichnet, sondern nur mit den *Veränderungen* gegenüber ihren Unter- und Oberbegriffen: ein Begriff besitzt alle Attribute seiner Oberbegriffe und die Objekte seiner Unterbegriffe. Um die Attributmenge eines Begriffs zu ermitteln, muß man alle an ihm selbst und an seinen Oberbegiffen angetragenen Attribute vereinigen. Umgekehrt erhält man die Objektmenge, wenn man alle angetragenen Objekte der Unterbegriffe eines Begriffs vereinigt. Diese Systematik dient nur einer übersichtlichen Beschriftung von Begriffsverbänden und verändert nicht die Begriffe selbst.

Der Begriffsverband eines Kontextes kann mit dem Algorithmus von Ganter [1] effizient berechnet werden. Das ist der Grund, warum er zur Analyse von Variantenstrukturen besser geeignet ist, als ein brute-force-Ansatz. Zwar kann die Anzahl der Begriffe exponentiell mit der Anzahl der Attribute wachsen, so daß auch seine Berechnung im ungünstigsten Fall exponentiellen Aufwand besitzt. Dies gilt aber nicht im Mittel; tatsächlich sind Verbände mit bis zu einigen tausend Begriffen leicht zu bestimmen [2].

4 Analyse von Varianten

Die bisherige Darstellung der Begriffsanalyse scheint wenig mit Varianten von Softwarequelltexten zu tun zu haben. Die Verbindung zwischen beiden wird in diesem Abschnitt durch eine Reihe von Definitionen und darauf aufbauenden Sätzen hergestellt.

Konfigurationsfunktionen heißen äquivalent, wenn sie die gleiche Variante erzeugen. Nicht jede Variablenbelegung erzeugt nämlich eine eigene Variante, so daß nicht triviale Äquivalenzklassen existieren. Die Anzahl der möglichen Varianten wird durch die Anzahl der Äquivalenzklassen der Konfigurationsfunktionen bestimmt.

Definition 9 ($\doteq$) *Zwei Konfigurationsfunktionen f_1 und f_2 heißen in einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ äquivalent, geschrieben $f_1 \doteq f_2$, wenn gilt:*

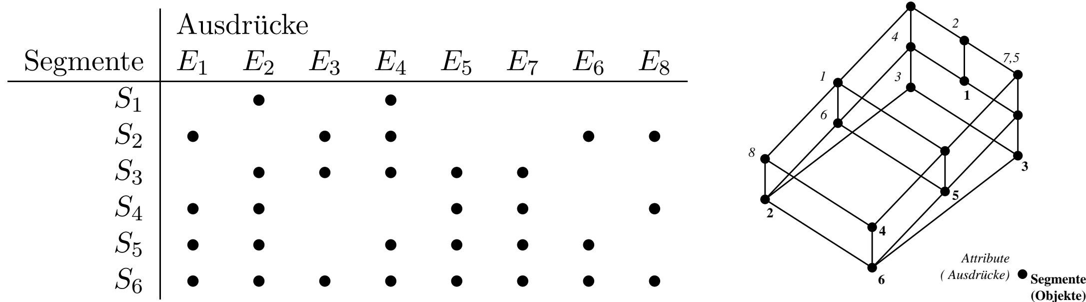
$$f_1(\mathcal{S}, \mathcal{E}, \mathcal{T}) = f_2(\mathcal{S}, \mathcal{E}, \mathcal{T})$$

Die Klasse der zu f äquivalenten Funktionen wird mit $[f]$ bezeichnet.

Welche und damit wieviele Äquivalenzklassen von Konfigurationsfunktionen existieren sagt der zentrale Satz zur Analyse von Variantenstrukturen:

Theorem 2 (Äquivalenzklassen) *In der Konfigurationstabelle* $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ *gilt:* $[f]$ *ist eine Äquivalenzklasse einer Konfigurationsfunktion* f *mit* $f(\mathcal{S}, \mathcal{E}, \mathcal{T}) = S$ *genau dann, wenn* $(S, \overline{E}) \in B(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$ *gilt, mit* $e \in \overline{E} \iff e \notin E$ *und* $(s, e) \in \overline{\mathcal{T}} \iff (s, e) \notin \mathcal{T}$*.*

Jeder Begriff $(S, \overline{E})$ der invertierten Konfigurationstabelle $\overline{\mathcal{T}}$ beschreibt eine Äquivalenzklasse von Konfigurationsfunktionen und damit eine Variante. Die Menge S ist dabei die Menge der Codesegmente, also die Variante, und die Ordnungsbeziehung zwischen den Begriffen ist genau die Teilmengenbeziehung zwischen Varianten. Der Begriffsverband von $\overline{\mathcal{T}}$ ist somit der Variantenverband, wenn man nur die S-Komponente jedes Begriffes betrachtet. Die Menge E mit $e \in E \iff e \notin \overline{E}$ des Begriffes $(S, \overline{E})$ beschreibt die minimale Menge von Ausdrücken, die von einer Konfigurationsfunktion zu 1 ausgewertet werden müssen, um die Variante S zu erzeugen. Abbildung 5 zeigt den invertierten Kontext des Beispiels und seinen Begriffsverband.



Segmente	Ausdrücke E_1	E_2	E_3	E_4	E_5	E_7	E_6	E_8
S_1		•		•				
S_2	•		•	•			•	•
S_3		•	•	•	•	•		
S_4	•	•			•	•		•
S_5	•	•		•	•	•	•	
S_6	•	•	•	•	•	•	•	•

Abbildung 5: Invertierter Kontext zum Beispiel aus Abbildung 1 und sein Begriffsverband

Die Beweisidee zu Theorem 2 ist: (1) jeder Begriff der invertierten Konfigurationstabelle ist eine Variante und (2) alle Varianten entsprechen einem Begriff. Die erste Aussage ist leicht einzusehen, wenn man den prinzipiellen Aufbau der Begriffe in Abbildung 6 ansieht: ein Begriff $(S, \overline{E})$ ist ein maximales, mit Markierungen gefülltes Rechteck in der invertierten Tabelle. Also ist dieses Rechteck in der Konfigurationstabelle vollständig frei von Markierungen. Die Menge S der Codesegmente hängt nur von Ausdrücken in E ab und umgekehrt sind alle von E abhängenden Codesegmente Elemente von S. Eine Konfigurationsfunktion, die die Elemente in E zu 1 auswertet erzeugt die Variante S. Weil jeder Begriff eindeutig ist, ist auch jede zugehörige Variante eindeutig – keine Variante wird doppelt gezählt.

Für einen indirekten Beweis der zweiten Aussage muß man nach Varianten suchen, die nicht durch einen Begriff beschrieben werden. Ein Begriff ist ein (1) maximales und (2) ausgefülltes Rechteck in der invertierten Konfigurationstabelle. In einem indirekten Beweis ist jetzt eine Variante S gesucht, erzeugt durch eine Konfigurationsfunktion f, die die Elemente E zu 1 auswertet, so daß $(S, \overline{E})$ *kein* Begriff ist. Der Fall (2), daß das Rechteck $(S, \overline{E})$ nicht vollständig ausgefüllt ist, ist unmöglich. Dann würde nämlich ein Codesegment $s \in S$ existieren, das von einem Ausdruck abhängt, der zu 0 ausgewertet wird. Dies ist ein Widerspruch zur Definition 4 einer Variante. Es bleibt, daß $(S, \overline{E})$ nicht maximal ist. Dies ist nur möglich, wenn E

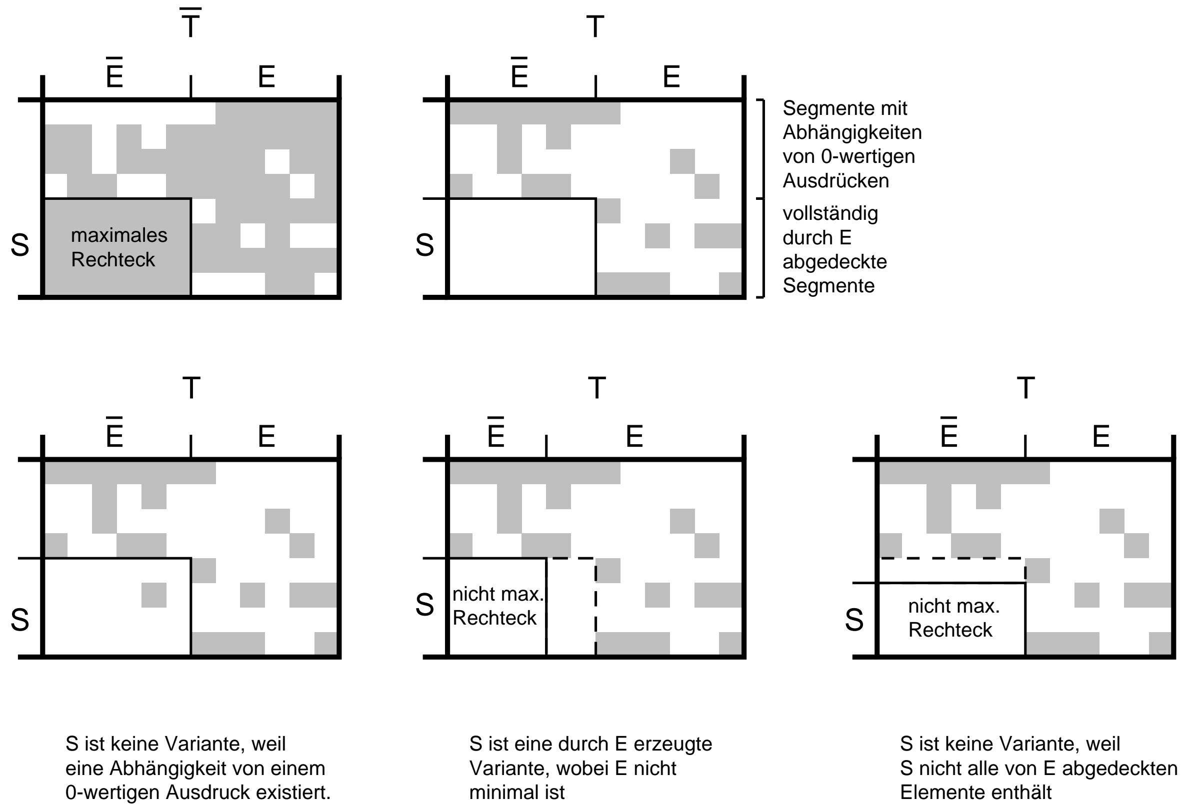


Abbildung 6: Jeder Begriff beschreibt eine Variante

Ausdrücke enthält, die für die Erzeugung von S nicht nötig sind (weil die Elemente in S von ihnen nicht abhängen). Dieser Fall wird aber durch einen anderen Begriff (eine äquivalente Konfigurationsfunktion) abgedeckt, bei dem E zu E_2 entsprechend verkleinert und $\overline{E}$ zu $\overline{E_2}$ vergrößert wird, so daß ein Begriff $(S, \overline{E_2})$ entsteht. Zu jeder Variante existiert also ein Begriff und zusammen ergibt sich, daß Varianten und Begriffe sich entsprechen.

5 Redundanzen

Die CPP-Strukturen zur Erzeugung von Varianten eines Quelltextes können redundant sein. Damit ist gemeint, daß eine einfachere Struktur exakt den gleichen Effekt hat, wie die redundante Struktur. Redundanzen drücken sich ebenfalls in der Konfigurationstabelle und ihrem Begriffsverband aus und können dort auch leicht identifiziert werden. Formal ist ein Ausdruck $e \in \mathcal{E}$ redundant, wenn er aus der Konfigurationstabelle entfernt werden kann, ohne daß Ergebnis einer beliebigen Konfigurationsfunktion zu verändern.

Definition 10 *In einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ heißt ein Ausdruck $e \in \mathcal{E}$ redundant, wenn für alle Konfigurationsfunktionen f gilt: $f(\mathcal{S}, \mathcal{E}, \mathcal{T}) = f(\mathcal{S}, \mathcal{E}^-, \mathcal{T}^-)$ mit $\mathcal{E}^- = \mathcal{E} \setminus \{e\}$ und $\mathcal{T}^- = \mathcal{T} \cap (\mathcal{S} \times \mathcal{E}^-)$.*

Redundante Ausdrücke können mit Hilfe des folgenden Satzes wiederum in der invertierten Konfigurationstabelle entdeckt werden:

Theorem 3 (Redundante Ausdrücke) *Ein Ausdruck $e \in \mathcal{E}$ in einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ ist redundant, wenn im Kontext $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$ eine Menge $X \subseteq \mathcal{E}, e \notin X$ existiert, mit $\{e\}' = X'$. Dabei ist $(s, e) \in \overline{\mathcal{T}} \iff (s, e) \notin \mathcal{T}$.*

Die in dem Satz geforderte Bedingung $X \subseteq \mathcal{E}, e \notin X$ ist sehr technisch und für eine unmittelbare Anwendung ungeeignet. Betrachtet man allerdings den Begriffsverband von $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$, so drücken sich redundante Ausdrücke als sogenannte $\wedge$-reduzible Begriffe aus. Ein Begriff b ist $\wedge$-reduzibel, wenn er sich als Infimum zweier Oberbegriffe b_1, b_2 darstellen läßt: $b = b_1 \wedge b_2$. In Abbildung 7 ist nochmals der Begriffsverband des Beispiels zu sehen. Wie bereits im Abschnitt erläutert 3 enthält ein Begriff alle Ausdrücke seiner Oberbegriffe, so daß sich die vollständigen Ausdrucksmengen für jeden Begriff rekonstruieren lassen.

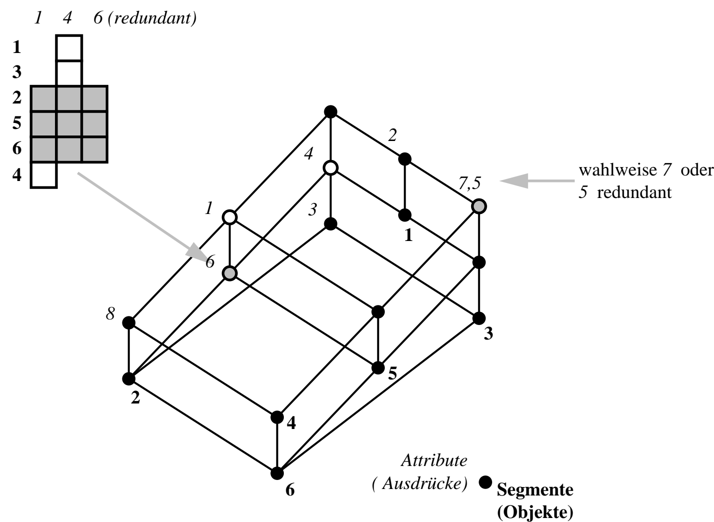


Abbildung 7: Begriffsverband des invertierten Kontextes des Beispiels

Ein Ausdruck e_1 ist redundant, wenn er an einem Begriff angetragen ist, an den entweder ein weiterer Ausdruck e_2 angetragen ist, oder der mindestens zwei direkte Oberbegriffe besitzt (dann ist er $\wedge$-reduzibel). Für das Beispiel bedeutet dies, daß E_6 und wahlweise E_5 oder E_7 redundant sind und entfernt werden können. Die Redundanz läßt sich im zugehörigen Kontext (Abbildung 5) ablesen: die Spalten der Ausdrücke E_5 und E_7 sind gleich, die Spalte E_6 ist der Schnitt (die Konjunktion) der Spalten E_1 und E_4. Entfernt man einen redundanten Ausdruck, so stimmt die dann entstehende Variantenstruktur mit der alten vollständig überein. Auf Wunsch könnte eine solche Vereinfachung durch ein Werkzeug automatisch vorgenommen werden.

In Abbildung 8 zeigt eine Skizze die beiden Fälle eines redundanten Ausdrucks. An Hand der Skizze ist zu erkennen, wie die Bedingung $X \subseteq \mathcal{E}, e \notin X$ des Theorems erfüllt ist. Der Beweis des Theorems selbst wird mit Hilfe der Begriffsanalsye geführt. Er zeigt, daß das Entfernen des redundanten Ausdrucks aus einem Kontext nicht zu einem strukturell veränderten Begriffsverband führt und die Äquivalenzklassen der Konfigurationsfunktionen erhalten bleiben.

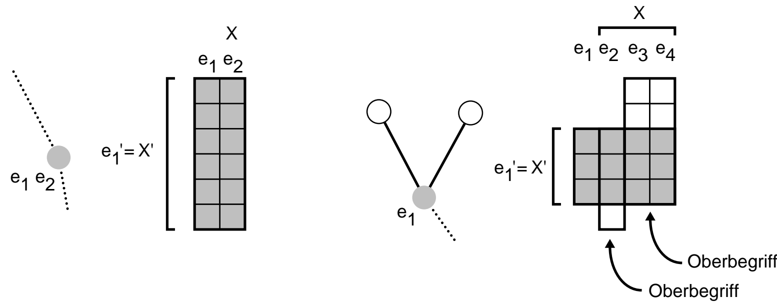


Abbildung 8: Der prinzipielle Aufbau von Begriffen mit redundanten Ausdrücken

6 Ausblick

Damit die durch die Analyse gewonnen Aussagen eine Bedeutung haben, muß das durch Konfigurationsfunktionen modellierten Verhalten auch dem des tatsächlichen Präprozessor entsprechen. Eine Konfigurationsfunktion wählt ein Segment für eine Variante aus, wenn alle seine kontrollierenden Ausdrücke zu 1 ausgewertet werden. Die Ausdrücke müssen deshalb im Quelltext konjunktiv verknüpft sein, weil sie ebenfalls genau dann durch den C-Präprozessor in eine Variante aufgenommen werden, wenn sie alle zu 1 ausgewertet werden.

```
#if defined(a) || defined(b)
  C1
#if defined (c)
  C2
#endif
#else
  C3
#endif
```

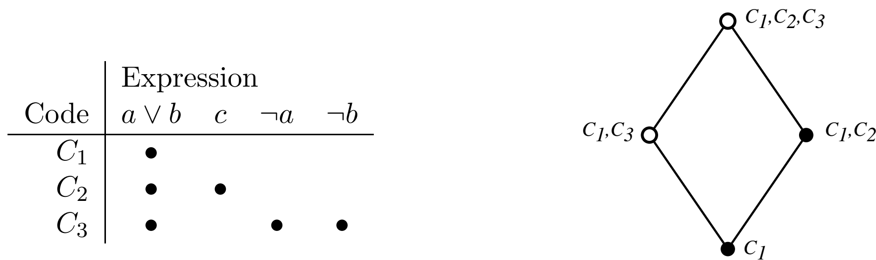
```
#if defined(a) || defined (b)
  C1
#endif
#if (defined(a) || defined(b)) && defined (c)
  C2
#endif
#if !defined(a) && !defined(b)
  C3
#endif
```

Abbildung 9: Beispiel für eine problematische Variantenstruktur

Die konjunktiv verknüpften Ausdrücke der Normalform bestehen aber in Wirklichkeit nicht immer nur aus einer Variablen, sondern können selbst Disjunktionen und zusätzlich auch Negationen enthalten. Abbildung 9 zeigt dafür ein Beispiel: der linke Quelltext wurde in disjunktive Normalform gebracht, die rechts zu sehen ist. Daraus ensteht dann die in Abbildung 10 gezeigte Konfigurationstabelle und der Variantenverband.

Der bislang vorgestellte Formalismus betrachtet die kontrollierenden Ausdrücke als unabhängig von einander mit 0 oder 1 belegbar. Dies trifft für Variablen auch zu, aber nicht für beliebige Ausdrücke. Nach Satz 2 wird durch die Belegung $a \vee b = 1, c = 1, \neg a = 1, \neg b = 1$ die Variante $\{C_1, C_2, C_3\}$ erzeugt. Dies aber widerspricht dem wirklichen Verhalten des CPP, da sich dort eine gleichzeitige Belegung von $a = 1$ und $\neg a = 1$ ausschließt. Die Folge ist, daß der Variantenverband zu viele Varianten enthält, wenn die Ausdrücke der Konfigurationstabelle nicht unabhängig sind. In Abbildung 10 sind die unmöglichen Varianten gekennzeichnet.

Die zusätzlich zwischen den Attributen bestehenden Beziehungen lassen sich durch Implikationen zwischen ihnen ausdrücken ($a \Rightarrow a \vee b, \neg a \wedge a \Rightarrow \{a, b, c\}$). Ein Hüllenoperator kann zu einer Menge von Attributen und Implikationen die kleinste Menge von Attributen bestim-



Code	Expression $a \vee b$	c	$\neg a$	$\neg b$
C_1	•			
C_2	•	•		
C_3	•		•	•

Abbildung 10: Konfigurationstabelle und Variantenverband. Die beiden hellen Begriffe markieren Varianten, die nicht praktisch erzeugt werden können.

men, die die Implikationen respektiert. Dieser Hüllenoperator kann auf alle Attributmengen angewendet werden, die die mit Hilfe der Begriffsanalyse bestimmten Varianten erzeugen. Die dann erweiterten Attributmengen erzeugen die tatsächlich möglichen Varianten. Zusätzliches Wissen, das nicht in den CPP-Ausdrücken des Quelltextes codiert ist, könnte ebenfalls durch Implikationen ausgedrückt werden und bei der Bestimmung der möglichen Varianten berücksichtgt werden.

7 Andere Ansätze

Die Struktur von CPP-Dateien wurde erstmals von Krone und Snelting [5, 4] mit Begriffsanalyse untersucht. Abhängigkeiten zwischen kontrollierenden Ausdrücken und Codesegmenten werden an Hand des Begriffsverbandes der Konfigurationstabelle ermittelt. Infima im Begriffsverband beschreiben Codesegmente, die von verschiedenen Ausdrücken kontrolliert werden. Gehören diese Ausdrücke zu semantisch verschiedenen Aspekten ist dies ein Anzeichen dafür, daß das softwaretechnische Prinzip der schwachen Koppelung verletzt ist.

Der hier vorgestellte Ansatz ist eine Fortsetzung der Arbeit von Krone und Snelting: aus CPP-Ausdrücken eines Quelltextes wird eine Konfigurationstabelle erstellt, die mit Begriffsanalyse untersucht wird. Während bei Krone und Snelting der Einflußbereich von CPP-Ausdrücken als eine Menge von Segmenten bestimmt wird, versucht die vorgestellte Theorie den globalen syntaktischen Effekt des CPP auf eine Quelldatei zu erfassen.

Der in [4] verfolgte Ansatz, Querbeziehungen durch Begriffsanalyse sichtbar zu machen wird in [3] auf der Ebene von Software-Modulen und -Architekturen wiederholt. Die Gemeinsamkeit zu dieser Arbeit liegt eher im generellen Thema und der Verwendung von Begriffsanalyse als in technischen Details. Ein weiterer Ansatz auf der Ebene von Modulen ist [6]: die Menge von Funktionen und die von ihnen verwendete Typen bilden einen Kontext. Mittels Begriffsanalyse wird die Menge der Funktionen in Module partitioniert. Wiederum liegt die Gemeinsamkeit in der Verwendung von Begriffsanalyse im Software Engineering.

8 Ergebnisse

Quelltexte müssen für verschiedene Plattformen in unterschiedlichen Varianten vorliegen. Die Varianten können mit Hilfe eines Präprozessors aus einem Quelltext erzeugt werden, in dem

einzelne Segmente durch logische Ausdrücke kontrolliert werden. Die aus einem Quelltext generierbaren Varianten bilden einen Variantenverband den die vorgestellte Theorie effizient bestimmt. Jeder Variante wird durch eine Klasse äquivalenter Konfigurationsfunktionen erzeugt. Jede Variante, und damit jede Klasse von Konfigurationsfunktionen, besitzt eine minimale Belegung von Ausdrücken, die sie erzeugt. Sowohl die Variante als die sie erzeugende Belegung werden berechnet.

Die logischen Ausdrücke zur Beschreibung der verschiedenen Varianten können Redundanzen aufweisen. Sie sind gerade so definiert, daß ihre Entfernung aus der Beschreibung keinen Einfluß auf die generierbaren Varianten und damit den Variantenverband hat. Redundanzen können ebenfalls mit Hilfe des durch formale Begriffsanalyse bestimmten Begriffsverbandes gefunden und dann auch entfernt werden.

In bestimmten Fällen können in der Theorie auftretende Varianten praktisch nicht erzeugt werden. Der Grund sind Abhängigkeiten zwischen logischen Ausdrücken, die von der Theorie noch nicht berücksichtigt werden. Die entsprechende Erweiterung der Theorie ist das Ziel der zukünftigen Arbeit.

Literatur

[1] B. Ganter and R. Wille: *Formale Begriffsananlyse - Mathematische Grundlagen.* Springer, Berlin, 1996.

[2] C. Lindig: *Komponentensuche mit Begriffen.* In *Softwaretechnik '95*, Braunschweig, Oktober 1995.

[3] C. Lindig and G. Snelting: *Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis.* Proc. International Conference on Software Engineering (ICSE 97), pp. 349-359, Boston, USA, May 1997.

[4] G. Snelting: *Reengineering of Configurations Based on Mathematical Concept Analysis.* ACM Transactions on Software Engineering and Methodology 5(2), 146-189, April 1996.

[5] M. Krone and G. Snelting: *On the Inference of Configuration Structures from Source Code.* Proc. 16th International Conference on Software Engineering, pp. 49-57., IEEE Comp. Soc. Press, Mai 1994.

[6] M. Siff and T. Reps: *Identiyfing Modules Via Concept Analysis.* Proc. International Conference on Software Maintenance, Bari 1997, pp. 170 – 179.

A Beweise

Theorem 4 (Eigenschaften) *Ist* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *ein Kontext und sind* $O_i \subseteq \mathcal{O}$ *Mengen von Objekten und* $A_i \subseteq \mathcal{A}$ *Mengen von Attributen, so gilt:*

1. $A_1 \subseteq A_2 \Rightarrow A'_2 \subseteq A'_1$ *und* $B_1 \subseteq B_2 \Rightarrow B'_2 \subseteq B'_1$
2. $A \subseteq A''$ *und* $B \subseteq B''$

3. $A' = A'''$ *und* $B' = B'''$

4. $A \subseteq B' \iff B \subseteq A'$

5. $(\bigcup_{i \in I} O_i)' = \bigcap_{i \in I} O_i'$ *und* $(\bigcup_{i \in I} A_i)' = \bigcap_{i \in I} A_i'$

Beweis. Siehe Hilfsatz 10 und 11 in [1].

Definition 11 (Abdeckung) *Für eine Menge $A \subseteq \mathcal{A}$ eines Kontextes $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ werden die von A abgedeckten Objekte A° definiert als:*

$$A^\circ = \{o \in \mathcal{O} \mid a \in A \text{ für alle } (o, a) \in \mathcal{R}\}$$

Für eine Objektmenge $O \subseteq \mathcal{O}$ wird die abdeckende Menge $O^\bullet$ definiert als:

$$O^\bullet = \{a \in \mathcal{A} \mid (o, a) \in \mathcal{R} \text{ existiert mit } o \in O\}$$

Theorem 5 (Abdeckung) *Sei $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ein endlicher Kontext und $(\mathcal{O}, \mathcal{A}, \overline{\mathcal{R}})$ ein weiterer Kontext mit $(o, a) \in \mathcal{R} \iff (o, a) \notin \mathcal{R}$. Das Paar $(O, \overline{A}) \in \mathcal{O} \times \mathcal{A}$ mit $a \in A \iff a \notin \overline{A}$ ist genau dann ein Begriff in $B(\mathcal{O}, \mathcal{A}, \overline{\mathcal{R}})$, wenn im Kontext $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ gleichzeitig gilt: $O^\bullet = A$ und $A^\circ = O$.*

Beweis. Sei $(O, \overline{A}) \in B(\mathcal{O}, \mathcal{A}, \overline{\mathcal{R}})$, also $\overline{A} = O'$ und $(\overline{A})' = O$. Es gilt $\overline{A} = O' = \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in \overline{\mathcal{R}}\} = \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \notin \mathcal{R}\} \iff A = \{a \in \mathcal{A} \mid \exists o \in O : (o, a) \in \mathcal{R}\} = O^\bullet$. Außerdem gilt: $O = (\overline{A})' = \{o \in \mathcal{O} \mid \forall a \in \overline{A} : (o, a) \in \overline{\mathcal{R}}\} = \{o \in \mathcal{O} \mid \forall a \in \overline{A} : (o, a) \notin \mathcal{R}\} = \{o \in \mathcal{O} \mid \forall (o, a) \in \mathcal{R} : a \in A\} = A^\circ$.

Lemma 1 *Die Relation $\doteq$ ist eine Äquivalenzrelation. Die Klasse der zu f äquivalenten Funktionen wird mit $[f]$ bezeichnet.*

Beweis. Die Gleichheit von Mengen ist bekanntlich eine Äquivalenzrelation.

Definition 12 (Minimale Konfigurationsfunktion) *Eine Konfigurationsfunktion f heißt in einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ minimal, genau dann wenn gleichzeitig gilt: $E_f^\circ = f(\mathcal{S}, \mathcal{E}, \mathcal{T})$ und $(f(\mathcal{S}, \mathcal{E}, \mathcal{T}))^\bullet = E_f$.*

Definition 13 *Zu einer Konfigurationsfunktion f wird in einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ eine abgeleitete Konfigurationsfunktion f^* definiert:*

$$f^*(e) = 1 \iff \exists s \in f(\mathcal{S}, \mathcal{E}, \mathcal{T}) : (s, e) \in \mathcal{T}, f^*(e) = 0 \text{ sonst}$$

Theorem 6 *Die von einer Konfigurationsfunktion f in einer Konfigurationstabelle $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ abgeleitete Konfigurationsfunktion f^* besitzt folgende Eigenschaften:*

1. *Aus $f^*(e) = 1$ folgt $f(e) = 1$*

2. $f(\mathcal{S},\mathcal{E},\mathcal{T}) = f^*(\mathcal{S},\mathcal{E},\mathcal{T})$

3. f^* *ist minimal in* $(\mathcal{S},\mathcal{E},\mathcal{T})$

4. Für alle $f_1 \in [f_2]$ *gilt:* $f_1^* \doteq f_2^*$

Beweis.

1. Sei $e \in \mathcal{E}$ mit $f^*(e) = 1$. Also existiert $s \in f(\mathcal{S},\mathcal{E},\mathcal{T})$ mit $(s,e) \in \mathcal{T}$. Daraus folgt $f(e) = 1$ nach der Definition von $f(\mathcal{S},\mathcal{E},\mathcal{T})$.

2. Sei $s \in f(\mathcal{S},\mathcal{E},\mathcal{T})$, also $f(e) = 1$ für alle $(s,e) \in T$. Betrachte ein beliebiges e mit $(s,e) \in \mathcal{T}$. Es folgt nach der Konstruktion $f^*(e) = 1$, also $s \in f^*(\mathcal{S},\mathcal{E},\mathcal{T})$ und damit $f^*(\mathcal{S},\mathcal{E},\mathcal{T}) \supseteq f(\mathcal{S},\mathcal{E},\mathcal{T})$.

   Es gilt $f^*(\mathcal{S},\mathcal{E},\mathcal{T}) = \{s \in \mathcal{S} \mid f^*(e) = 1 \text{ für alle } (s,e) \in \mathcal{T}\}$. Weil $f(e) = 1$ aus $f^*(e)$ folgt, gilt $f^*(T) \subseteq \{s \in \mathcal{S} \mid f(e) = 1 \text{ für alle } (s,e) \in \mathcal{T}\}$.

   Insgesamt gilt also $f(\mathcal{S},\mathcal{E},\mathcal{T}) = f^*(\mathcal{S},\mathcal{E},\mathcal{T})$.

3. Es gilt $E_{f^*} = \{e \in \mathcal{E} \mid \exists s \in f(\mathcal{S},\mathcal{E},\mathcal{T}) : (s,e) \in \mathcal{T}\}$. Weiter gilt $(E_{f^*})^\circ = \{s \in \mathcal{S} \mid e \in E_{f^*} \text{ für alle } (s,e) \in \mathcal{T}\} = \{s \in \mathcal{S} \mid f^*(e) = 1 \text{ für alle } (s,e) \in \mathcal{T}\} = f^*(\mathcal{S},\mathcal{E},\mathcal{T})$.

   $(f^*(\mathcal{S},\mathcal{E},\mathcal{T}))^\bullet = \{e \in \mathcal{E} \mid (s,e) \in \mathcal{T} \text{ existiert mit } s \in f^*(\mathcal{S},\mathcal{E},\mathcal{T})\}$. Aus $e \in (f^*(\mathcal{S},\mathcal{E},\mathcal{T}))^\bullet$ folgt $f^*(e) = 1$ und damit $e \in E_{f^*}$. Andererseits folgt aus $e \in E_{f^*} \iff \exists s \in f(\mathcal{S},\mathcal{E},\mathcal{T}) = f^*(\mathcal{S},\mathcal{E},\mathcal{T}) : (s,e) \in T$, daß $e \in (f^*(\mathcal{S},\mathcal{E},\mathcal{T}))^\bullet$. Insgesamt also $E_{f^*} = (f^*(\mathcal{S},\mathcal{E},\mathcal{T}))^\bullet$ und $(E_{f^*})^\circ = f^*(\mathcal{S},\mathcal{E},\mathcal{T})$.

4. Diese Eiegenschaft gilt nach Konstruktion von $[f]$, denn $f_i(\mathcal{S},\mathcal{E},\mathcal{T}) = f_j(\mathcal{S},\mathcal{E},\mathcal{T})$ für $f_i, f_j \in [f]$ und damit $f_i^* \doteq f_j^* \doteq f^*$.

Beweis zu Theorem 2 Sei $[f]$ eine Äquivalenzklasse mit $f(\mathcal{S},\mathcal{E},\mathcal{T}) = S$. Es existiert $f^* \in [f]$ mit $f^*(\mathcal{S},\mathcal{E},\mathcal{T}) = S$. f^* ist minimal in $(\mathcal{S},\mathcal{E},\mathcal{T})$, also gilt $E_{f^*}^\circ = f(\mathcal{S},\mathcal{E},\mathcal{T}) = C$ und $(f(\mathcal{S},\mathcal{E},\mathcal{T}))^\bullet = E_{f^*}$. Nach Theorem 5 ist dies äquivalent mit $(S,\overline{E}) \in B(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$.

Sei umgekehrt $(S,\overline{E}) \in B(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$. Nach Theorem 5 ist dies äquivalent mit $S^\bullet = E$ und $E^\circ = S$ in der Konfigurationstabelle $(\mathcal{S},\mathcal{E},\mathcal{T})$. Nach der Definition minimaler Konfigurationsfunktionen bedeutet dies, daß f definiert durch $E_f = E$ eine minimale Konfigurationsfunktion mit $f(\mathcal{S},\mathcal{E},\mathcal{T}) = S$ ist, also $[f]$ eine entsprechende Äquivalenzklasse.

Beweis zu Theorem 3 Zum Beweis bestimmen wir die Äquivalenzklassen der Konfigurationsfunktionen in der vereinfachten Konfigurationstabelle und zeigen, daß sie genau die Konfigurationen wie im ursprünglichen Kontext berechnen. Es gilt nämlich: wenn (S,E) ein Begriff im Kontext $(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$ ist, dann ist $(S,E) \cap (\mathcal{S} \times \mathcal{E}^-) = (S, E \setminus \{e\})$ ein Begriff in $B(\mathcal{S},\mathcal{E}^-,\overline{\mathcal{T}^-})$. Dies bedeutet insbesondere, daß für jede Äquivalenklasse in der ursprünglichen Konfigurationstabelle eine eineindeutige Äquivalenzklasse in der vereinfachten Konfigurationstabelle existiert, also keine Konfiguration verlorengeht oder dazukommt.

Sei $(S,E) \in B(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$ ein Begriff, gelte also $S' = E$ und $E' = S$ in $(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$. Wenn der redundante Ausdruck $e \in E$ Teil des Begriffes ist, dann ist auch die Menge X Teil des Begriffes: $X \subset E$, denn: mit Hilfe von Theorem 4 gilt: $\{e\} \subseteq E \Rightarrow \{e\}' \supseteq E'$ im Kontext

$(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$. Weiter gilt $X' = \{e\}' \supseteq E' = S$, also $S \subseteq X' \iff X \subseteq S' = E$. Weil $e \in E$, aber $e \notin X$ muß sogar gelten $X \subset E$.

Es ist zu zeigen, daß $(S, E \setminus \{e\})$ ein Begriff im Kontext $B(\mathcal{S}, \mathcal{E}^-, \overline{\mathcal{T}^-})$ ist.

1. Sei $e \notin E$, dann folgt $(E \setminus \{e\})' = E' = S$ sowohl in $B(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$, als auch in $B(\mathcal{S}, \mathcal{E}^-, \overline{\mathcal{T}^-})$.
2. Sei $e \in E$. Dann gilt $X \subset E$. Betrachte nun im Kontext $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$: $(E \setminus \{e\})' = (E \setminus \{e\})' \cap X'$, weil $X \subset E \Rightarrow X \subseteq (E \setminus \{e\}) \Rightarrow X' \supseteq (E \setminus \{e\})'$ gilt. Wegen $X' = \{e\}'$ gilt weiter: $(E \setminus \{e\})' \cap X' = (E \setminus \{e\})' \cap \{e\}' = (E \setminus \{e\}) \cup \{e\})' = E'$. Also gilt $(E \setminus \{e\})' = E' = S$ auch im Kontext $(\mathcal{S}, \mathcal{E}^-, \overline{\mathcal{T}^-})$.
3. Betrachte ein beliebiges $e_1 \in (E \setminus \{e\})$: wegen $S' = E$ im ursprünglichen Kontext $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$ gilt: $\forall s \in S : (s, e_1) \in \overline{\mathcal{T}}$ und weil $e_1 \neq e$, gilt $\forall s \in S : (s, e_1) \in \overline{\mathcal{T}^-}$. Insgesamt gilt also $(E \setminus \{e\}) \subseteq S'$ im vereinfachten Kontext.
4. Wir nehmen an, es gäbe ein $e_2 \in \mathcal{E}, e_2 \notin (E \setminus \{e\})$ mit $e_2 \in S'$ im vereinfachten Kontext $(\mathcal{S}, \mathcal{E}^-, \overline{\mathcal{T}^-})$. Dies würde bedeuten, daß e_2 wegen $\forall s \in S : (c, e_2) \in \overline{\mathcal{T}^-} \subseteq \overline{\mathcal{T}} \Rightarrow e_2 \in C'$ auch im urpsrünglichen Kontext Element von $C' = E$ wäre. Dies ist ein Widerspruch. Also gilt im vereinfachten Kontext $S' \subseteq (E \setminus \{e\})$ und damit $S' = (E \setminus \{e\})$.

Technische Universität Braunschweig

Informatik-Berichte ab Nr. 96-01

96-01	A. Zeller, G. Snelting	Unified Versioning Through Feature Logic
96-02	M. Goldapp, U. Grottker, G. Snelting	Validierung softwaregesteuerter Meßsysteme durch Program Slicing und Constraint Solving
96-03	C. Lindig, G. Snelting	Modularization of Legacy Code Based on Mathematical Concept Analysis
96-04	J. Adámek, J. Koslowski, V. Pollara, W. Struckmann	Workshop Domains II (Proceedings)
96-05	F.-J. Grosch	A Syntactic Approach to Structure Generativity
96-06	E. H. A. Gerbracht, W. Struckmann	Zur Diskussion elementarer Funktionen aus algorithmischer Sicht
97-07	H.-D. Ehrich	Object Specification
97-01	A. Zeller	Versioning Software Systems through Concept Descriptions
97-02	K. Neumann, R. Müller	Implementierung von Assertions durch Oracle7-Trigger
97-03	G. Denker, P. Hartel	TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics
97-04	F.-J. Grosch	M - eine typisierte, funktionale Sprache für das Programmieren-im-Grossen
97-05	J. Küster Filipe	Putting Synchronous and Asynchronous Object Modules together: an Event-Based Model for Concurrent Composition
97-06	J. Küster Filipe	A categorical Hiding Mechanism for Concurrent Object Systems
97-07	G. Snelting, U. Grottker, M. Goldapp	VALSOFT Abschlussbericht
98-01	J. Krinke, G. Snelting	Validation of Measurement Software as an application of Slicing and Constraint Solving
98-02	S. Petri, M. Bolz, H. Langendörfer	Transparent Migration and Rollback for Unmodified Applications in Workstation Clusters
98-03	M. Cohrs, E. H. A. Gerbracht, W. Struckmann	DISKUS - Ein Programm zur symbolischen Diskussion reeller elementarer Funktionen
98-04	C. Lindig	Analyse von Softwarevarianten