Komponentensuche mit Begriffen

Christian Lindig
Institut für Programmiersprachen, Abteilung Softwaretechnologie
Technische Universität Braunschweig, 38106 Braunschweig
lindig@ips.cs.tu-bs.de

Zusammenfassung

Mit Schlüsselwörtern indexierte Komponenten einer Sammlung werden einmalig durch formale Begriffsanalyse zu einem Begriffsverband strukturiert. Dieser erlaubt dann, Komponenten durch die Angabe von Schlüsselwörtern einfach, effizient, inkrementell und mit Unterstützung für den Benutzer zu suchen. Der Suchprozeß garantiert dabei, daß bei einer Suche mindestens eine Komponente gefunden wird. Da der Suchprozeß unabhängig vom Inhalt der verwalteten Komponenten ist, ist er auch für inhomogene und multimediale Sammlungen geeignet. Eine prototypische Implementierung zur Verwaltung einer Unix Online-Dokumentation belegt die Anwendbarkeit des Verfahrens, dessen Komplexität zusätzlich experimentell untersucht wurde. Es zeigt sich, daß das Verfahren eine schnelle und genaue Navigation in Sammlungen mit mehreren tausend Komponenten ermöglicht.

Stichworte: Softwarebibliothek, Wiederverwendung, Inkrementelle Suche, Begriffsverband

1 Einleitung

In der Praxis haben sich Information-Retrieval-Ansätze [11, 7, 9] zum Verwalten von wiederverwendbaren Software-Komponenten als geeignet erwiesen. Im Gegensatz zu formal-logischen Ansätzen [8, 12, 13, 2] sind sie nicht auf spezielle Programmiersprachen festgelegt und ihre Anfragesprache ist einfacher, wenn auch zum Teil mit unklarer Semantik. Die in dieser Arbeit präsentierte Komponentensuche mit Begriffen verbindet eine einfache und klare Anfragesprache mit einer effizienten, inkrementellen Suche, die den Benutzer kontextsensitiv unterstützt. Dazu bestimmt *formale Begriffsanalyse* zu einer indexierten Sammlung einmalig die formalen Begriffe, die dann für spätere Suchen verwendet werden.

Jede Komponente wird zunächst mit einer beliebigen Anzahl von Schlüsselwörtern indexiert. Die berechneten formalen Begriffe der Sammlung bilden einen Verband aus Ober- und Unterbegriffen. Die Suche erfolgt inkrementell durch die Angabe von den Schlüsselwörtern, die die gesuchten Komponenten mindestens aufweisen sollen. Dabei wählt der Benutzer die Schlüsselwörter aus einer präsentierten Menge aus, wobei der Auswahlprozeß garantiert, daß mindestens eine Komponente selektiert wird – das Ergebnis also nicht leer ist. Mit jedem Schritt verringert sich sowohl die Zahl der selektierten Komponenten, als auch der im nächsten Schritt wählbaren Schlüsselwörter, so daß der Benutzer präzise und schnell zu den gesuchten Komponenten geführt wird.

Der nächste Abschnitt gibt zunächst einen Überblick über das Verfahren in Form eines Beispiels. Die sich daran anschließenden Abschnitte definieren Begriffsanalyse formal und zeigen, wie sie zur Suche eingesetzt wird. Die Vorstellung eines Prototyps, Experimente zur Aufwandsabschätzung und ein Vergleich mit anderen Ansätzen bilden den Schluß.

2 Ein Beispiel: Unix-Dokumentation

Unix-Systeme besitzen traditionell eine Online-Dokumentation ihrer Programme, Systemaufrufe, Bibliotheken und Dateiformate, die zusammen etwa 1 800 Dokumente umfaßt. Ein kleiner Ausschnitt aus der Dokumentation der Betriebssystemaufrufe (System Calls) soll demonstrieren, wie Dokumente, stellvertretend für Komponenten bestehend aus Programmcode, einer Signatur und andere für ihre Verwendung relevanten Informationen, indexiert und gesucht werden können. Abbildung 1 zeigt die Namen von 12 Unix System-Calls zusammen mit einer einzeiligen Beschreibung aus ihrer Dokumentation [15] und einer Menge von Schlüsselwörtern als Indexierung. Im Fall der Unix-Dokumentation ist die Indexierung oft naheliegend, da bereits eine Nomenklatur existiert: Dateien sind *files*, Zugriffrechte *permissions*. Im allgemeinen muß aber zunächst ein Katalog von Schlüsselwörtern und ihrer Semantik erstellt werden.

Nachdem die Komponenten indexiert sind, wird die Sammlung durch eine formale Begriffsanalyse für den späteren Zugriff zu Begriffen strukturiert. Der zugehörige Begriffsverband (siehe nächster Abschnitt) ist lediglich die unterliegende Datenstruktur für die Suche, und ist dem Benutzer nicht sichtbar.

Die Suche nach Komponenten verläuft schrittweise: in jedem Schritt wählt der Benutzer eines der Attribute, die die gesuchten Komponenten *mindestens* aufweisen sollen. Sofort werden ihm alle Komponenten präsentiert, die mindestens die bisher ausgewähl-

Sys.-Call	Kurzbeschreibung	Indexierung
`chmod`	*change mode of file*	change mode permission file
`chown`	*change owner and group of a file*	change owner group file
`stat`	*get file status*	get file status
`fork`	*create a new process*	create new process
`chdir`	*change current working directory*	change directory
`mkdir`	*make a directory file*	create new directory
`open`	*open or create a file for reading or writing*	open create file read write
`read`	*read input*	read file input
`rmdir`	*remove a directory file*	remove directory file
`write`	*write output*	write file output
`creat`	*create a new file*	create new file
`access`	*determine accessibility of file*	check access file

Abbildung 1: Unix-System-Calls mit Kurzbeschreibung und Indexierung

Schritt	Attribut	Komponenten	noch mögliche Attr.
1	–	alle	alle
2	*change*	`chdir chmod chown`	*directory file group mode owner permission*
3	*change file*	`chmod chown`	*group mode owner permission*
4	*change file mode*	`chmod`	–
1	–	alle	alle
2	*create*	`creat fork mkdir open`	*directory file new open process read write*
3	*create file*	`creat open`	*new read open write*
4	*create file read*	`open`	–

Abbildung 2: Zwei Beispiele für schrittweise Suche

	access	*change*	*check*	*create*	*directory*	*file*	*get*	*group*	*input*	*mode*	*new*	*open*	*output*	*owner*	*permission*	*process*	*read*	*remove*	*status*	*write*
`access`	×		×			×														
`chdir`		×			×															
`chmod`		×				×				×						×				
`chown`		×				×		×						×						
`creat`				×		×					×									
`fork`				×							×					×				
`fstat`						×	×												×	
`mkdir`				×	×						×									
`open`				×		×						×					×			×
`read`						×			×								×			
`rmdir`					×	×												×		
`write`						×							×							×

Abbildung 3: Kontexttabelle *Unix-System-Calls* für Systemaufrufe des Betriebssystems Unix

ten Attribute aufweisen und gleichzeitig werden ihm die Attribute präsentiert, die er im nächsten Schritt wählen kann. Die zur Auswahl stehenden Attribute sind genau so bestimmt, daß jede Wahl die Menge der selektierten Komponenten verkleinert, sie aber *garantiert* nicht leer ist. Mit anderen Worten: der Benutzer kann keine Attribut-Kombinationen wählen, die keine der Komponenten erfüllt. Bei der Auswahl der Attribute ist die Reihenfolge beliebig: der Benutzer kann sich den gesuchten Komponenten unter verschiedenen Aspekten nähern. Auch bei großen Sammlungen erfolgt die Berechnung des Ergebnisses nach einem Suchschritt praktisch verzögerungsfrei.

Abbildung 2 zeigt zwei Beispiele für schrittweise Suchen; die in Abbildung 1 vorgenommene Indexierung ist in Abbildung 3 nochmals als Tabelle dargestellt. Solange kein Attribut ausgewählt ist, sind alle Komponenten der Sammlung selektiert und alle Attribute stehen noch zur Auswahl. Nachdem der Anwender im ersten Beispiel das Attribut *change* gewählt hat, werden dadurch die Komponenten `chdir`, `chmod` und `chown` selektiert, da sie mit *change* indexiert sind. Für den nächsten Schritt stehen nur noch die Attribute zur Auswahl, die zusammen mit *change* eine nicht leere Selektion ergeben. Der Anwender wählt im Beispiel *file* aus, wodurch sich sowohl die Menge der selektierten Komponenten, als auch der noch möglichen Attribute weiter verkleinert. Es sind jetzt genau die Komponenten selektiert, die mindestens mit *change*

und *file* indexiert sind. Im letzten Schritt wird durch die Auswahl von *mode* noch genau die Komponente `chmod` selektiert. Natürlich kann der Benutzer seine Suche auch schon vorher beenden, und so mehrere selektierte Komponenten erhalten.

Die Zahl der noch möglichen Attribute verringert sich in jedem Schritt im allgemeinen stark, so daß der Benutzer nach wenigen Schritten nur noch aus einer überschaubaren Menge auswählen muß. Könnte er in einem Schritt ein nicht angegebenes Attribut wählen, würde dieses mit den zuvor gewählten Attributen im Widerspruch stehen – keine Komponente weist die entsprechenden Attribute gemeinsam auf.

Die Grundlage für die effiziente Berechnung der selektierten Komponenten und der im nächsten Schritt einer Suche möglichen Attribute ist die vorhergehende Bestimmung der *Begriffe* der Sammlung. Ein Begriff ist ein maximales Paar aus einer Menge von Attributen und der zugehörigen Menge von selektierten Komponenten. So bilden im dritten Schritt des zweiten Beispiels *create*, *file* und `creat`, `open` einen Begriff – allerdings bildet nicht jedes Paar aus Anfrage und selektierten Komponenten einen Begriff, da die zu einem Ergebnis gehörende Anfrage nicht alle die den Komponenten des Ergebnisses gemeinsamen Attribute enthalten muß.

3 Formale Begriffsanalyse

Formale Begriffsanalyse wurde von R. Wille begründet und untersucht die Beziehung zwischen einer Menge von Objekten und diesen Objekten zugeordneten Attributen [16, 1].

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

Die Relation zwischen Objekten und Attributen läßt sich anschaulich in einer *Kontexttabelle* darstellen. Dies ist genau die schon in Abbildung 3 gezeigte Tabelle mit System-Calls als Objekten und Schlüsselwörtern als Attributen.

Definition 2 *Für Objekt-Mengen $O \subseteq \mathcal{O}$ und Attributmengen $A \subseteq \mathcal{A}$ eines Kontextes $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ werden die Mengen der ihnen gemeinsamen Attribute $\omega(O)$ bzw. Objekte $\alpha(A)$ definiert:*

$$\alpha(A) := \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in \mathcal{R}\}$$
$$\omega(O) := \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in \mathcal{R}\}$$

Zum Beispiel besitzen die Objekte `chdir` und `chmod` nur *change* als gemeinsames Attribut, also gilt $\omega(\{\texttt{chdir}, \texttt{chmod}\}) = \{change\}$.

Definition 3 *Ein formaler Begriff eines Kontextes $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ist ein Paar (O, A) aus Objekt- und Attributmengen mit $O \subseteq \mathcal{O}, A \subseteq \mathcal{A}$, wobei gleichzeitig gilt:*

$$\alpha(A) = O \quad \textit{und} \quad \omega(O) = A$$

Für einen Begriff $b = (O, A)$ bezeichnen $\pi_o(b)$ die Objekte (Umfang) und $\pi_a(b)$ die Attribute (Inhalt) des Begriffs: $\pi_o(b) := O$ und $\pi_a(b) := A$. Die Menge aller Begriffe eines Kontextes wird mit $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ bezeichnet.

Die Objekte eines Begriffs (in einem gegebenen Kontext) sind synonym für eine Menge von Attributen und umgekehrt – die Objektmenge eines Begriffs bestimmt die Attributmenge eindeutig. Ein Beispiel für einen Begriff des Kontexts Unix-System-Calls ist das Paar $(\{\texttt{chmod}, \texttt{chown}\}, \{file, change\})$. In diesem Kontext sind die Attribute *file* und *change* ein Synonym für die Objekte `chmod` und `chown` – und umgekehrt.

Zur Berechnung aller Begriffe $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ eines Kontextes existieren mehrere Algorithmen [3], deren Komplexität im ungünstigsten Fall exponentiell ist, da ein Kontext maximal 2^n mit $n = min(|\mathcal{O}|, |\mathcal{A}|)$ Begriffe enthalten kann. Der typische Aufwand ist aber wesentlich günstiger und wird unten noch genauer untersucht.

Definition 4 *Zwei Begriffe (O_1, A_1), $(O_2, A_2) \in B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ eines Kontextes werden durch die Unterbegriffs-Relation $\leq$ geordnet:*

$$(O_1, A_1) \leq (O_2, A_2) \overset{\text{def}}{\Leftrightarrow} O_1 \subseteq O_2 \quad (\overset{\text{def}}{\Leftrightarrow} A_1 \supseteq A_2)$$

Die Relation wird gelesen als: (O_1, A_1) ist Unterbegriff von (O_2, A_2). Die Struktur aus Kontext und Ordnungsrelation wird abgekürzt als $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R}) = (B(\mathcal{O}, \mathcal{A}, \mathcal{R}), \leq)$.

In dem gegebenen Beispiel ist der Begriff $(\{\texttt{mkdir}, \texttt{creat}, \texttt{fork}\}, \{create, new\})$ ein Oberbegriff des Begriffes $(\{\texttt{fork}\}, \{create, new, process\})$. Ein Oberbegriff hat größeren Umfang und kleineren Inhalt als seine Unterbegriffe; speziell enthält ein Unterbegriff mindestens die Attribute aller seiner Oberbegriffe und ein Oberbegriff mindestens die Objekte aller seiner Unterbegriffe.

Satz 1 (Hauptsatz der Begriffsanalyse [16]) *Sei $K = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ ein Kontext, dann ist $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ein vollständiger Verband, der* Begriffsverband *von K. Infimum und Supremum sind gegeben durch:*

$$\bigwedge_{i \in I}(O_i, A_i) = \left(\bigcap_{i \in I} O_i \,,\, \omega\left(\alpha\left(\bigcup_{i \in I} A_i\right)\right)\right)$$
$$\bigvee_{i \in I}(O_i, A_i) = \left(\alpha\left(\omega\left(\bigcup_{i \in I} O_i\right)\right), \bigcap_{i \in I} A_i\right)$$

Dieser Satz besagt, daß mehrere Begriffe einen eindeutig bestimmten kleinsten Oberbegriff und größten

Unterbegriff besitzen und beschreibt, wie diese Begriffe berechnet werden. Der größte gemeinsame Unterbegriff (Infimum) entsteht aus dem Schnitt der Objektmengen und der Vereinigung der Attributmengen der beteiligten Begriffe, wobei die vereinigten Attributmengen noch erweitert werden müssen, damit sie mit dem Schnitt der Objektmengen wirklich einen Begriff bilden. Mit dem Supremum verhält es sich umgekehrt analog.

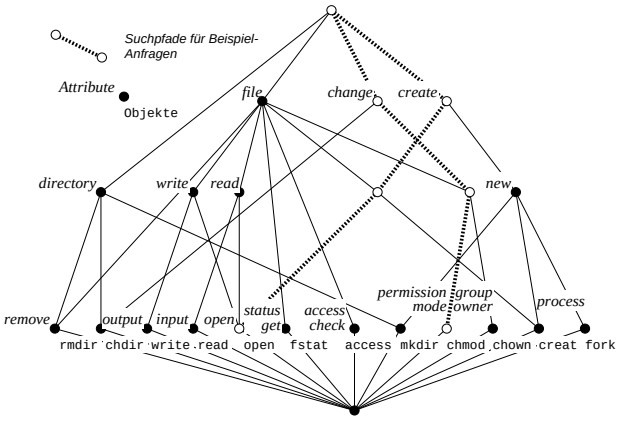

Abbildung 4: Begriffsverband für Unix-System-Calls

Abbildung 4 zeigt das Hasse-Diagramm des Begriffsverbandes des Beispiels aus Abbildung 1. Jeder Knoten des Graphen ist ein Begriff: ein Paar aus einer Menge von System-Calls und einer Menge von Schlüsselwörtern. Statt dieser Paare sind in Abbildung 4 vereinfachte Beschriftungen an den Knoten angetragen, aus denen die jeweiligen Mengen rekonstruiert werden können:

- Attribute werden in Begriffsverbänden an Unterbegriffe vererbt. Alle kleineren Begriffe als der mit *file* beschriftete Begriff tragen ebenfalls *file* in ihrer Attributmenge.
- Objekte werden in Begriffsverbänden an Oberbegriffe vererbt. Alle Oberbegriffe des mit `open` beschrifteten Begriffs tragen dieses Objekt ebenfalls in ihrer Objektmenge.

Jedes Attribut und jedes Objekt wird also durch einen Begriff eingeführt und dann an Unter- bzw. Oberbegriffe vererbt. Für jedes Objekt und jedes Attribut kann dieser Begriff bestimmt werden:

Satz 2 *Sei $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ein Begriffsverband und $o \in \mathcal{O}$, $a \in \mathcal{A}$. Sei $\sigma(o)$ der kleinste Begriff, in dessen Umfang o enthalten ist und sei $\mu(a)$ der größte Begriff, in dessen Inhalt a enthalten ist. Es gilt:*

$$\begin{aligned} \sigma(o) &= (\alpha(\omega(\{o\})), \omega(\{o\})) \\ \mu(a) &= (\alpha(\{a\}), \omega(\alpha(\{a\}))) \end{aligned}$$

Der mit *write* beschriftete Begriff ist also das Paar $(\{\texttt{write}, \texttt{open}\}, \{write, file\})$. Die Beschriftung der Knoten im Diagramm ergibt sich durch die Anwendung von σ und μ auf Objekte und Attribute: so ist $\mu(write)$ ist der mit *write* beschriftete Begriff im Hasse-Diagramm.

Die Grundlage für eine effiziente Suche und die Berechnung der verbleibenden Wahlmöglichkeiten ist der Begriffsverband einer Sammlung, bestehend aus Ober- und Unterbegriffen. Er stellt eine natürliche Strukturierung dar und ergibt sich automatisch durch die Analyse des Kontextes aus Komponenten und ihrer Indexierung.

4 Suche mit Begriffen

Die Komponenten $\mathcal{O}$ einer Sammlung bilden mit der Menge $\mathcal{A}$ aller Schlüsselwörter und der Relation $\mathcal{R}$ zwischen ihnen einen formalen Kontext $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ und damit einen Begriffsverband $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. Eine Anfrage ist eine Menge von Attributen und selektiert Komponenten der Sammlung:

Definition 5 (Anfrage und Ergebnis)
Eine Menge $A \subseteq \mathcal{A}$ von Attributen ist eine Anfrage *in einem Begriffsverband $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. Eine Komponente $o \in \mathcal{O}$* erfüllt *eine Anfrage A genau dann, wenn $\omega(o) \supseteq A$ gilt*[1]*, sie also mindestens die Attribute der Anfrage besitzt. Alle Komponenten, die eine Anfrage erfüllen, bilden das* Ergebnis $[\![A]\!]$ *einer Anfrage A:*

$$[\![A]\!] := \{o \mid o \in \mathcal{O}, \omega(o) \supseteq A\}$$

Die Berechnung des Ergebnises einer Anfrage geschieht über die einzelnen Attribute der Anfrage; zu jedem Attribut $a \in A$ existiert ein größter Begriff $\mu(a)$, in dessen Inhalt a enthalten ist. Das Infimum dieser Begriffe ist ein Begriff, dessen Objekte genau das Ergebnis der Anfrage sind: alle Objekte von Unterbegriffen des Infimums sind im Ergebnis enthalten. Eine Anfrage induziert also genau einen Begriff, dessen Objekte das Ergebnis bilden.

Satz 3 *Sei $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ein Begriffsverband und $A \subseteq \mathcal{A}$, dann ist $[\![A]\!] = \pi_o\left(\bigwedge_{a \in A} \mu(a)\right)$.*

Das Infimum der größten Begriffe, die die Attribute der Anfrage enthalten, enthält mindestens die geforderten Attribute und damit auch seine Objekte. Da das Infimum der größte Begriff mit dieser Eigenschaft ist, gibt es keine größere Menge von Objekten, die diese Bedingung erfüllt. Die Attribute der Anfrage A sind in der Attributmenge $\pi_a\left(\bigwedge_{a \in A} \mu(a)\right)$ des Infimums enthalten – die darüberhinaus im Infimum enthaltenen Attribute könnten ebenfalls vom Benutzer noch ausgewählt werden, ohne daß sich das Ergebnis verändert. Deshalb kann vor dem nächsten Such-

[1] Genauer wäre $\omega(\{o\})$, aber hier und im Folgenden werden o und $\{o\}$ identifiziert.

schritt davon ausgegangen werden, daß die Anfrage diese Attribute bereits enthält.

Abbildung 5 zeigt graphisch, wie das Ergebnis einer Anfrage in einem Begriffsverband bestimmt wird. Für die Attribute a_1 und a_2 einer Anfrage werden die zugehörigen größten Begriffe bestimmt – sie sind mit a_1 und a_2 beschriftet. Von diesen Begriffen wird das Infimum gebildet; in der Grafik geschieht dies durch Verfolgen der stärker gezeichneten Linien zum ersten gemeinsamen Begriff (Pfeil). Dieser Begriff enthält alle die Anfrage erfüllende Objekte o_1,o_2 und o_3. Dies bedeutet auch, das diese Objekte durch Begriffe des induzierten Unterverbandes (grau) eingeführt werden.

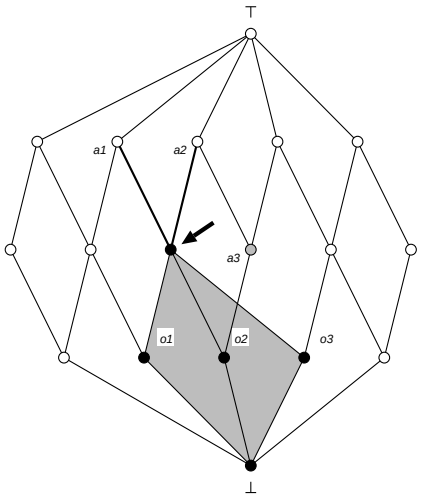

Abbildung 5: Suche im Begriffsverband

Aus welchen *sinnvollen* Attributen kann ein Benutzer auswählen, um seine Anfrage zu spezialisieren, so daß das Ergebnis der neuen Anfrage nicht leer ist? Die Objekte des Begriffes zu einer Anfrage enthalten Attribute, die nicht zu diesem Begriff gehören. In Abbildung 5 ist a_3 Attribut des Objektes o_2, aber a_3 ist nicht Attribut des bisherigen Infimums (Pfeil), da $\mu(a_3)$ kein Oberbegriff des Infimums ist. Alle Attribute, die nicht Attribut des Infimums sind, aber Attribut eines Objektes des Infimums, können eine Anfrage weiter spezialisieren. Das Attribut a_3 kann die Anfrage aus a_1 und a_2 also spezialisieren: das neue Ergebnis besteht nur aus dem Objekt o_2.

Definition 6 (Sinnvolle Attribute) *Sei $A \subseteq \mathcal{A}$ eine Anfrage in $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. Dann bezeichnet $\langle\langle A \rangle\rangle$ die Menge der* sinnvollen Attribute *für die Spezialisierung von A:*

$$\langle\langle A \rangle\rangle := \{a \in \mathcal{A} \mid \emptyset \subset [\![A \cup \{a\}]\!] \subset [\![A]\!]\}$$

Die Definition von $\langle\langle A \rangle\rangle$ zu einer Anfrage A ist also so gewählt, daß jedes Attribut aus $\langle\langle A \rangle\rangle$ A derart erweitert, daß das Ergebnis der neuen Anfrage eine echte Spezialisierung der alten, aber nicht leer ist. Die Attribute der Objekte von $[\![A]\!]$, die noch nicht durch die Anfrage A selbst impliziert werden, sind die Kandidaten für die Spezialisierung der Anfrage: sinnvolle Attribute werden durch Begriffe eingeführt, die mit dem Infimum zur Anfrage unvergleichlich sind oder ein echter Unterbegriff des Infimums sind und deren Objekte zum Ergebnis der Anfrage beitragen.

Satz 4 *Sei $A \subseteq \mathcal{A}$ eine Anfrage in $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. Dann gilt für die Attribute $\langle\langle A \rangle\rangle$ zur Spezialisierung von A:*

$$\langle\langle A \rangle\rangle = \Big(\bigcup_{o \in [\![A]\!]} \omega(o) \Big) \setminus \pi_a \Big(\bigwedge_{a \in A} \mu(a) \Big)$$

Der Benutzer kann nach jeder Anfrage A ein Attribut $a_1 \in \langle\langle A \rangle\rangle$ auswählen, um seine Anfrage weiter zu spezialisieren. Unabhängig von dem gewählten Attribut ist garantiert, daß das neue Ergebnis nicht leer ist. Zur algorithmischen Bestimmung von $\langle\langle A \rangle\rangle$ werden die Attribute der Objekte aus $[\![A]\!]$ vereinigt, wobei die Attribute ausgeschlossen werden, die bereits in dem durch die Anfrage A induzierten Begriff enthalten sind. Da $\omega(o) = \pi_a(\sigma(o))$ für $o \in \mathcal{O}$ gilt, kann $\omega(o)$ ebenfalls leicht aus dem Begriffsverband gewonnen werden, so daß eine Implementierung der Suche nicht zusätzlich den Kontext benötigt.

Das Ergebnis $[\![A_1]\!]$ der neuen Anfrage $A_1 = A \cup \{a_1\}$ wird schließlich inkrementell durch Berechnung eines neuen Infimums $[\![A_1]\!] = \pi_o(b_{[\![A]\!]} \wedge \mu(a_1))$ bestimmt. Dabei ist $b_{[\![A]\!]} = \bigwedge_{a \in A} \mu(a)$ bereits durch die vorhergehende Berechnung von $[\![A]\!]$ bekannt. Anschließend kann der nächste Zyklus zur Spezialisierung der Anfrage durchlaufen werden.

Die Berechnung des Ergebnisses $[\![A]\!]$ einer Anfrage und die exakt verbleibenden Möglichkeiten $\langle\langle A \rangle\rangle$ zur Spezialisierung der Anfrage lassen sich also elegant auf die Begriffe des Begriffsverbandes zurückführen. Der Begriffsverband kann als eine übersetzte Form aller Wahlmöglichkeiten des Benutzers aufgefaßt werden. Der Verband enthält gegenüber dem zugrundeliegenden Kontext keine wesentlich neuen Informationen, erlaubt aber eine effizientere Bearbeitung von Anfragen. Wie bei der Übersetzung von Programmen ist dies von Vorteil, wenn die Ausgangsdaten (Quelltext/Kontext) selten geändert, aber oft abgearbeitet (ausgeführt, durchsucht) werden.

5 Prototyp

Das beschriebene Verfahren wurde als Prototyp zur Verwaltung einer Unix Online-Dokumentation implementiert. Insgesamt 1658 Dokumente wurden mit mehreren Attributen aus einer Menge von 92 Attributen versehen – der zugehörige Begriffsverband enthält 714 Begriffe. Abbildung 6 zeigt die Schnittstelle des Prototyps zum Benutzer, nachdem die Attribute *change* und *file* ausgewählt wurden.

In der mittleren Liste werden die ausgewählten Attribute A angezeigt und rechts die noch wählbaren At-

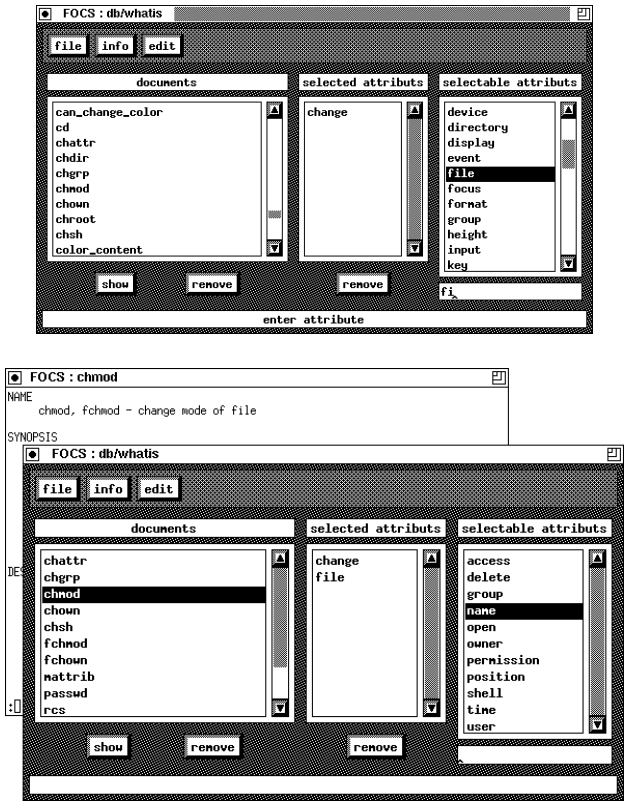

Abbildung 6: Zwei Schritte einer Suche mit dem Prototyp

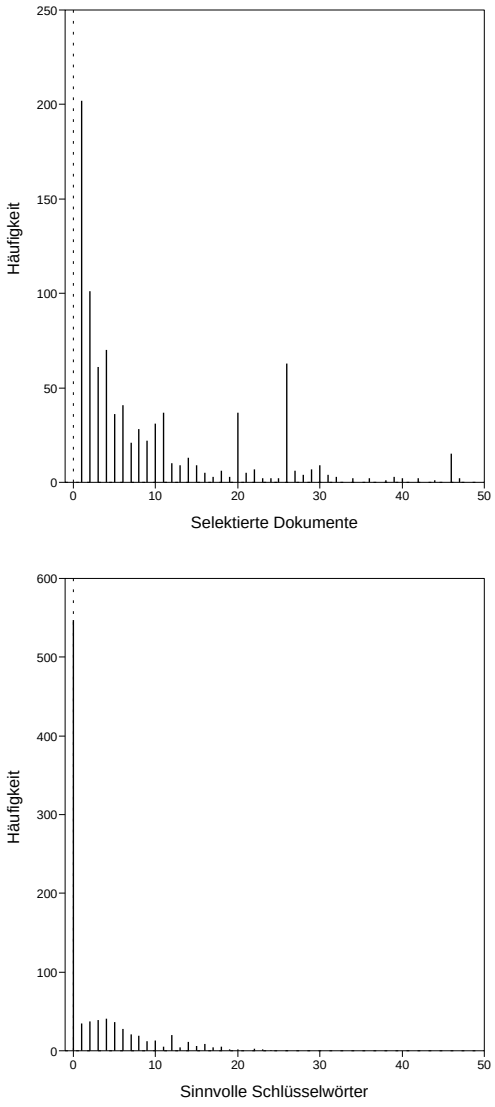

Abbildung 7: Verteilung von Dokumenten und Schlüsselwörtern nach 2 Suchschritten

tribute, also die Menge $\langle\langle A \rangle\rangle$. Die linke Liste enthält die durch die Attribute A ausgewählten Dokumente $[\![A]\!]$; durch Anklicken kann das entsprechende Dokument angesehen werden, wie es im Hintergrund für das Kommando *chmod*(1) zu sehen ist. Ein rechts ausgewähltes Attribut wird in die mittlere Liste übernommen und führt zu einer Verringerung der ausgewählten Dokumente und der noch wählbaren Attribute. Die Auswahl eines Attributs geschieht entweder direkt mit der Maus oder durch schriftliche Eingabe, wobei die Angabe eines Präfixes schon genügt, um den Listen-Cursor an das entsprechende Attribut springen zu lassen. Ein bereits ausgewähltes Attribut kann durch nochmaliges Auswählen mit der Maus zurückgenommen werden, woraufhin die Darstellung entsprechend aktualisiert wird. Dabei kann eine völlig neue Situation entstehen, wenn die Menge der dann noch ausgewählten Schlüsselwörter neu ist. Die Reaktion des Prototyps auf die Aktionen des Benutzers ist verzögerungsfrei – der Benutzer kann sehr schnell durch den Datenbestand navigieren. Durch die Präsentation seiner verbleibenden Wahlmöglichkeiten wird er dabei stark kontextsensitiv unterstützt.

Das Berechnen des Begriffsverbandes aus einer Indexierung von Objekten mit Attributen benötigt 150 Sekunden auf einer SPARCstation ELC. Diese Berechnung ist allerdings nur nötig, wenn sich die Ausgangsdaten geändert haben, ansonsten wird der vorberechnete Begriffsverband benutzt.

Für die mit dem Prototyp verwalteten Dokumente wurde untersucht, wie sich die Zahl der selektierten Dokumente $[\![A]\!]$ und der dann noch wählbaren Schlüsselwörter $\langle\langle A \rangle\rangle$ nach zwei Suchschritten ($|A| = 2$) verteilen. Von den $\binom{92}{2} = 4186$ theoretisch möglichen Kombinationen von Schlüsselwörtern sind 903 tatsächlich erreichbar. Für diese wurden jeweils die Zahl der selektierten Dokumente und im dritten Schritt wählbaren Schlüsselwörter bestimmt. Abbildung 7 zeigt die beiden Verteilungen: oben für die Dokumente, unten für die Schlüsselwörter.

Bereits nach zwei Suchschritten ist die Zahl der selektierten Dokumente in 52% aller Fälle von anfänglich 1658 auf weniger als 6 gefallen. Gleichzeitig ist die Zahl der noch zusätzlich wählbaren Schlüsselwörter in 68% aller Fälle kleiner als 2. Dies zeigt, wie das Verfahren den Suchraum eingrenzt und damit die Navi-

gation zielgerichtet und schnell wird.

Eine empirische Untersuchung von *Precision* und *Recall* beim Einsatz der begriffsbasierten Suche zur Verwaltung von Komponenten oder ihrer Dokumentation steht noch aus. Es ist allerdings zu bemerken, daß das Ergebnis $[\![A]\!]$ einer Anfrage A exakt und vollständig ist; Precision und Recall hängen also nur von der Qualität der Indexierung ab.

6 Berechnung von Begriffsverbänden

Unglücklicherweise kann die Zahl der Begriffe in einem Begriffsverband exponentiell mit der Zahl der Objekte und Attribute einer Sammlung wachsen. Für die praktische Anwendbarkeit der Suche muß deshalb untersucht werden, ob dieses Verhalten typisch ist, oder ob es vielmehr eine pathologische Ausnahme ist. Dazu wurden formale Kontexte zufällig generiert und die zugehörigen Begriffsverbände berechnet. Die erzeugten Kontexte wiesen eine variable Anzahl von Objekten auf, die mit je fünf Attributen versehen waren, um eine facettierte Klassifikation mit 5 Facetten nachzubilden [11]. Die fünf Attribute wurden dabei zufällig aus einer variabel großen Menge von Attributen ausgewählt. Abbildung 8 zeigt die Größe der Verbände und den Aufwand zu ihrer Berechnung durch den Algorithmus von Ganter [3] auf einer SPARCstation ELC. Die Zeitkomplexität des Algorithmus ist $O(n \cdot |\mathcal{A}| \cdot |\mathcal{O}|^2)$, wobei n die Zahl der Begriffe von $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ist.

Der Zeitaufwand bei den untersuchten Kontexten ist im Wesentlichen durch die Zahl der Objekte, also die Größe einer Sammlung, bestimmt. Die Größe des Wortschatzes zur Indexierung hat dagegen geringeren Einfluß. Die gemessenen Rechenzeiten liegen zwischen einigen Sekunden für mehrere hundert Objekte und steigen bis zu mehreren Minuten für Sammlungen von fast 2000 Objekten an. Die Größe der enstandenen Begriffsverbände wird ebenfalls durch die Zahl der Objekte dominiert, allerdings entstehen auffallend große Verbände bei vielen Objekten mit vergleichsweise wenigen Attributen zur Auswahl. Die Zahl der Begriffe liegt typischerweise bei einigen tausend – das präsentierte Verfahren ist damit für kleine und mittlere Sammlungen mit mehreren hundert bis einigen tausend Objekten ideal. Damit ergibt sich im typischen Fall eine polynomiale Zeit-Komplexität für die Bestimmung aller Begriffe eines Kontextes.

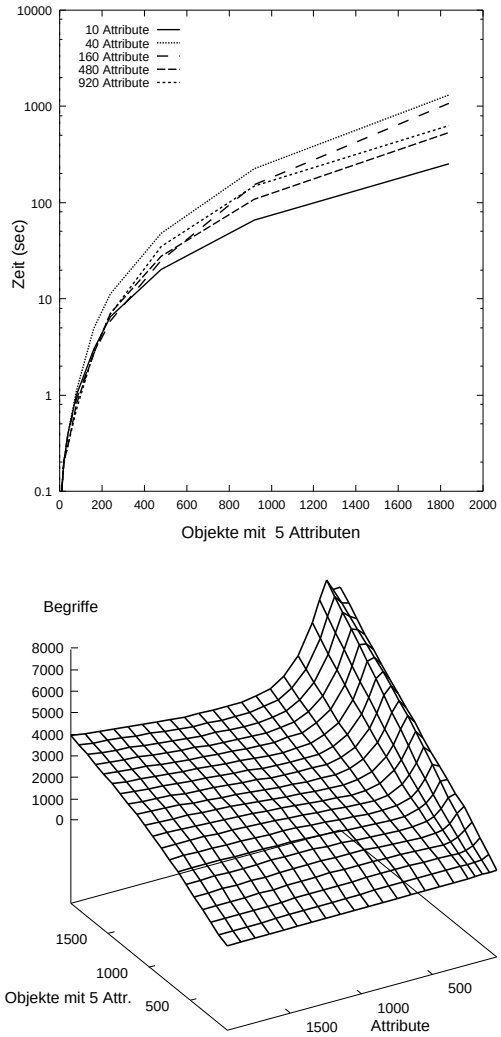

Abbildung 8: Zeitaufwand zur Berechnung von Begriffsverbänden und ihre Größe

7 Andere Ansätze

Die begriffsbasierte Suche ist ein Information-Retrieval-Ansatz, im Gegensatz zu den formallogischen Verfahren [8, 12, 13, 2]. Information-Retrieval-Ansätze werden von Maarek et al. [7] und Wood et al. [17] wegen ihrer oft leichteren Bedienung und Durchschaubarkeit bevorzugt. Die facettierte Klassifikation [11] ist ein bekannter Information-Retrieval-Ansatz, dessen Indexierung aus einem Vektor von Schlüsselwörtern für jede Komponente besteht. Damit ist bei der Suche die Reihenfolge der Schlüsselwörter signifikant und sogar fest vorgegeben; zusätzlich sind Platzhalter nur sehr eingeschränkt zugelassen. Der Benutzer kann sich einer gesuchten Komponente deshalb nur über *einen* Pfad nähern, wobei er nicht kontextsensitiv unterstützt wird. Airs [9] ist ein Verallgemeinerung dieses Ansatzes; durch manuell spezifizierte Beziehungen zwischen Komponenten wird die Navigation erleichtert, allerdings auf Kosten der Wartbarkeit der Sammlung.

Guru [7] verwaltet ebenfalls die Dokumentation eines Unix-Systems. Es indexiert den Inhalt von Dokumenten und Anfragen statistisch und erlaubt so natürlichsprachliche Anfragen. Zusätzlich stehen Dokumente in einer hierarchischen Beziehung, durch die sie ebenfalls inspiziert werden können. Die natürlichsprachliche Anfrage ist zunächst für einen Benutzer angenehm, allerdings bleibt das Ergebnis undurchschaubar, da das System nicht den Sinn einer Anfrage

versteht, sondern statistisch arbeitet. Im Gegensatz dazu ist die nicht-operationale Semantik einer Anfrage beim begriffsbasierten Ansatz dem Benutzer einsichtig.

Godin et al. [5] schlagen ebenfalls Begriffsverbände als Navigationsgrundlage vor. Bei ihnen erhält ein Benutzer aber nur eine Teilmenge aller sinnvollen Attribute einer Anfrage zur Unterstützung – nur die, die durch Begriffe unmittelbar kleiner als der Ergebnis-Begriff eingeführt werden. Sie schlagen eine Benutzeroberfläche vor, die dem Benutzer den Begriffsverband darstellt. Der vorgestellte Prototyp benutzt dagegen den Verband nur als für den Benutzer unsichtbare unterliegende Datenstruktur.

8 Ausblick

Die begriffsbasierte Suche erscheint nicht nur für die Verwaltung von Softwarekomponenten attraktiv. Damit nicht für jede weitere Anwendung die Suche vollständig neu implementiert werden muß, entsteht zur Zeit ein anwendungsunabhängiger Kern, der die zentralen Algorithmen der begriffsbasierten Suche implementiert. Dazu wird die Sprache Tcl/Tk [10] um begriffsbasierte Kommandos erweitert, die dann zusammen mit dem graphischen Toolkit Tk einen Ausgangspunkt für weitere begriffsbasierte Anwendungen bietet.

Zur Zeit muß bei einer Veränderung der Sammlung anschließend der gesamte Begriffsverband neu berechnet werden. Durch den Einsatz inkrementeller Algorithmen [4] zur Berechnung von Begriffsverbänden kann der Aufwand nach einer Änderung deutlich gesenkt werden, womit das Verfahren weiter an Attraktivität gewinnt.

9 Ergebnis

Mit Schlüsselwörtern indexierte Komponenten bilden einen formalen Kontext, aus dem durch formale Begriffsanalyse ein Begriffsverband berechnet werden kann. Dieser erlaubt dann eine effiziente, inkrementelle Suche von Komponenten zu implementieren, die den Benutzer durch die vollständige Präsentation seiner Wahlmöglichkeiten unterstützt. In Verbindung mit der Effizienz des Verfahrens ergibt sich eine schnelle und sichere Navigation durch eine Sammlung von Komponenten. Da zwischen Komponenten keine Konsistenz erfordernden expliziten Verbindungen spezifiziert werden, ist die Wartung einer Sammlung einfach. Die nach einer Veränderung der Sammlung nötige Neuberechnung des Begriffsverbandes hat typischerweise polynomialen Aufwand, wie Experimente gezeigt haben. Eine prototypische Implementierung als Teil von Nora [14, 6] demonstriert die praktische Anwendung und die Eignung des Verfahrens zur Verwaltung einer Unix Online-Dokumentation. Da das Suchverfahren unabhängig vom Inhalt der verwalteten Komponenten ist, können auch inhomogene und multimediale Sammlungen begriffsbasiert verwaltet werden.

Danksagung: Peter Dettmer implementierte den Prototyp in C++. Gregor Snelting, Franz-Josef-Grosch und Bernd Fischer trugen durch Diskussionen und Anmerkungen viel zum Verständnis des Verfahrens bei.

Literatur

[1] Davey, B. A. & Priestley, H. A. (1990), *Introduction to Lattices and Order*, 2nd ed., Cambridge University Press, Cambridge, GB, Formal Concept Analysis, 221–236.

[2] Fischer, B., Kievernagel, M. & Struckmann, W. (1995) VCR: A VDM-based software component retrieval tool. In Proc. ICSE-17 Workshop *Formal Methods Appl. in SW Eng. Practice*, Seattle, 30–38.

[3] Ganter, B. (1986), Algorithmen zur Formalen Begriffsanalyse, *in* B. Ganter, R. Wille & K. E. Wolff, Hrsg., 'Beiträge zur Begriffsanalyse', BI Wissenschaftsverlag, Mannheim, pp. 241–254.

[4] Godin R. & Missaoui R. (1994), An incremental concept formation approach for learning from databases. *Theoretical Computer Science*, 133 (1994), 387–419.

[5] Godin, R., Gecsei, J. & Pichet, C. (1989), Design of a Browsing Interface for Information Retrieval, *in* N. J. Belkin & C. J. van Rijsbergen, Eds, '12th Int. SIGIR Conference', Cambridge, Massachusetts, 32–39.

[6] Krone, M. & Snelting, G. (1994). On the inference of configuration structures from source code. In *Proc. 16th International Conference on Software Engineering*, Sorrento, Italy, 49–57.

[7] Maarek, Y. S., Berry, D. M. & Kaiser, G. E. (1991), 'An Information Retrieval Approach For Automatically Constructing Software Libraries', *IEEE Transactions on Software Engineering* **SE-17**(8), 800–813.

[8] Moorman Zaremski, A. & Wing, J. M. (1993), Signature Matching: A Key to Reuse, *in* D. Notkin, Ed., 'Proc. of the 1st ACM SIGSOFT Symposium on the Foundation of Software Engineering', Los Angeles, 182–190.

[9] Ostertag, E., Hendler, J., Prieto-Diáz, R. & Braun, C. (1992), 'Computing Similarity in a Reuse Library System: An AI-Based Approach', *ACM Transactions on Programming Languages and Systems* **1**(3), 205–228.

[10] Ousterhout, J. K., (1991), An X11 Toolkit Based on the TCL Language, In *Proc. 1991 Winter USENIX Conf.*, 105–115.

[11] Prieto-Diáz, R. (1991), 'Implementing Faceted Classification for Software Reuse', *Journal of the ACM* **34**(5), 89–97.

[12] Rittri, M. (1991), 'Using types as search keys in function libraries', *Journal of Functional Programming* **1**(1), 71–89.

[13] Rollins, E. J. & Wing, J. M. (1991), Specifications as Search Keys for Software Libraries, *in* K. Furukawa, ed., 'Logic Programming: Proc. of the 8th Int. Conference', Paris, 173–187.

[14] Snelting G., Fischer, B., Grosch, F.-J., Kievernagel M. & Zeller, A. (1994). Die inferenzbasierte Softwareentwicklungsumgebung NORA. *Informatik—Forschung und Entwicklung*, 9(3):116–131

[15] Sun Microsystems. *SunOS Reference Manual*. Sun Microsystems, Inc., SunOS 4.1.1, 1990.

[16] Wille, R. (1982), Restructuring lattice theory: An approach based on hierarchies of concepts. *in* I. Rival, ed., *Ordered Sets*, Reidel, 445–470.

[17] Wood, M. & Sommerville, I. (1988), 'An Information Retrieval System for Software Components', *SIGIR Forum* **22**(3), 11–25.

A Beweise

A.1 Satz 2

Beweis: Davey, B. A. & Priestley, H. A. (1990), *Introduction to Lattices and Order*, 2nd ed., Cambridge University Press, Cambridge, 221–236.

A.2 Satz 3

$$\begin{aligned}\pi_o\Big(\bigwedge_{a\in A}\mu(a)\Big) &= \pi_o\Big(\bigwedge_{a\in A}(\alpha(a),\omega(\alpha(a)))\Big)\\ &= \bigcap_{a\in A}\alpha(a)=\bigcap_{a\in A}\{o\in\mathcal{O}\mid(o,a)\in\mathcal{R}\}\\ &= \{o\in\mathcal{O}\mid\forall a\in A:(o,a)\in\mathcal{R}\}\\ &= \{o\in\mathcal{O}\mid\omega(o)\supseteq A\}=[\![A]\!]\end{aligned}$$

A.3 Satz 4

1. Sei $a_1\in\langle\!\langle A\rangle\!\rangle$. Zu zeigen ist:

   - $a_1\in\bigcup_{o\in[\![A]\!]}\omega(o)$
   - $a_1\notin\pi_a(\bigwedge_{a\in A}\mu(a))$

   (a) Sei $a_1\in\langle\!\langle A\rangle\!\rangle$. Also existiert $o_1\in[\![A\cup\{a_1\}]\!]\subset[\![A]\!]$ mit $\omega(o_1)\supseteq A\cup\{a_1\}\supseteq\{a_1\}$. Dies ist äquivalent zu $a_1\in\omega(o_1)$ und daraus folgt $a_1\in\bigcup_{o\in[\![A]\!]}\omega(o)$.

   (b) Annahme: $a_1\in\pi_a(\bigwedge_{a\in A}\mu(a))$. Daraus folgt $\mu(a_1)\geq\bigwedge_{a\in A}\mu(a)$ und damit $\bigwedge_{a\in A\cup\{a_1\}}\mu(a)=\bigwedge_{a\in A}\mu(a)$. Also gilt $[\![A]\!]=[\![A\cup\{a_1\}]\!]$, was im Widerspruch zur Voraussetzung $[\![A\cup\{a_1\}]\!]\subset[\![A]\!]$ steht.

2. Sei $a_1\in\bigcup_{o\in[\![A]\!]}\omega(o)$ und $a_1\notin\pi_a(\bigwedge_{a\in A}\mu(a))$. Zu zeigen ist:

   - $[\![A\cup\{a_1\}]\!]\neq\emptyset$
   - $[\![A\cup\{a_1\}]\!]\subseteq[\![A]\!]$
   - $[\![A]\!]\neq[\![A\cup\{a_1\}]\!]$

   (a) Aus $a_1\in\bigcup_{o\in[\![A]\!]}\omega(o)$ folgt, daß $o_1\in[\![A]\!]$ existiert, mit $a_1\in\omega(o_1)$ und damit $o_1\in\alpha(a_1)$. Betrachte nun:

   $$\begin{aligned}[\![A\cup\{a_1\}]\!] &= \bigcap_{a\in A}\alpha(a)\cap\alpha(a_1)\\ &\supseteq \{o_1\}\neq\emptyset\end{aligned}$$

   (b) Betrachte: $[\![A\cup\{a_1\}]\!]=\bigcap_{a\in A}\alpha(a)\cap\alpha(a_1)\subseteq\bigcap_{a\in A}\alpha(a)=[\![A]\!]$.

   (c) Annahme: $\forall o\in[\![A]\!]:\omega(o)\cap a_1\neq\emptyset$. Daraus folgt $a_1\in\omega([\![A]\!])=\pi_a(\bigwedge_{a\in A}\mu(a))$. Dies ist ein Widerspruch; also gilt: $\exists o_1\in[\![A]\!]:\omega(o_1)\cap a_1=\emptyset$ und damit $o_1\notin[\![A\cup\{a_1\}]\!]$. Also $[\![A]\!]\neq[\![A\cup\{a_1\}]\!]$.