7.5 Semantik von Rechteanfragen

7.5.1 Rechteanfragen

Syntaktisch ist eine **elementare Anfrage** ein explizites Recht. Eine **Anfrage** kann darüber hinaus Klassennamen anstelle von Objektnamen enthalten.

Definition:
Die Menge aller elementaren Anfragen ist EQ:=TAG×EA;
die Menge aller Anfragen ist Q:=TAG×A.

Eine elementare Anfrage ist also ein Tupel eq=(t,ea) TAG×EA;
eine Anfrage ist ein Tupel q=(t,sa) TAG×A.

Um die Bedeutung einer Anfrage q bezüglich einer gegebenen Menge von spezifizierten Rechten SR' festzulegen, benutzen wir das Prädikat is_valid(q,SR').

Elementare Anfragen an unsere Rechtedatenbank können z.B. zum Zwecke der Zugriffskontrolle gestellt werden. Darüberhinaus können Rechteanfragen die Arbeit des Rechteverwalters unterstützen und nicht zuletzt Benutzer über Erlaubnisse und Verbote informieren.

Die Bedeutung einer elementaren Anfrage eq kann sehr einfach dadurch beschrieben werden, dass man in der Menge der expliziten Rechte nachschaut, die zu einer Menge von spezifizierten Rechten SR' gehören: Das Ergebnis einer elementaren Anfrage eq ist gerade is_valid(eq,SR') : ⇔ eq explicit_rights*(SR').

Im Folgenden führen wir zwei unterschiedliche Anfrage-Semantiken ein.
Die erste, sog. **Zustandssemantik**, entspricht exakt der Semantik wie sie bisher eingeführt wurde.
Die zweite, sog. **Strukturesemantik**, ist u.a. aus Effizienzgründen leicht modifiziert, unterscheidet sich aber für elementare Anfragen nicht von der Zustandssemantik.

Der Unterschied liegt vielmehr in der Behandlung von Klassennamen.
Die Struktursemantik zielt darauf ab zu beantworten, welche Rechte für eine Klasse in der Regel gelten, z.B. für eine bestimmte Position einer Subjektklasse in der darf-mehr-Hierarchie.

7.5.2 Zustandssemantik

In der Zustandssemantik steht ein Klassenname exakt für die augenblickliche Menge seiner Klassenmitglieder.

Soweit Klassennamen von der Anfrage betroffen sind, beantwortet diese Semantik die Frage:

"Welche Rechte sind gültig
für *alle (gegenwärtigen) Objekte* dieser Klasse?"

Erlaubt man Klassen in Anfragen, so wird die Situation komplizierter.

Wir bearbeiten für solche Anfragen die Klassenhierarchien und Klassenmitgliedschaften in der gleichen Art wie bisher beschrieben und erhalten eine Menge von elementaren Anfragen mit einer einheitlichen Rechtekennung.

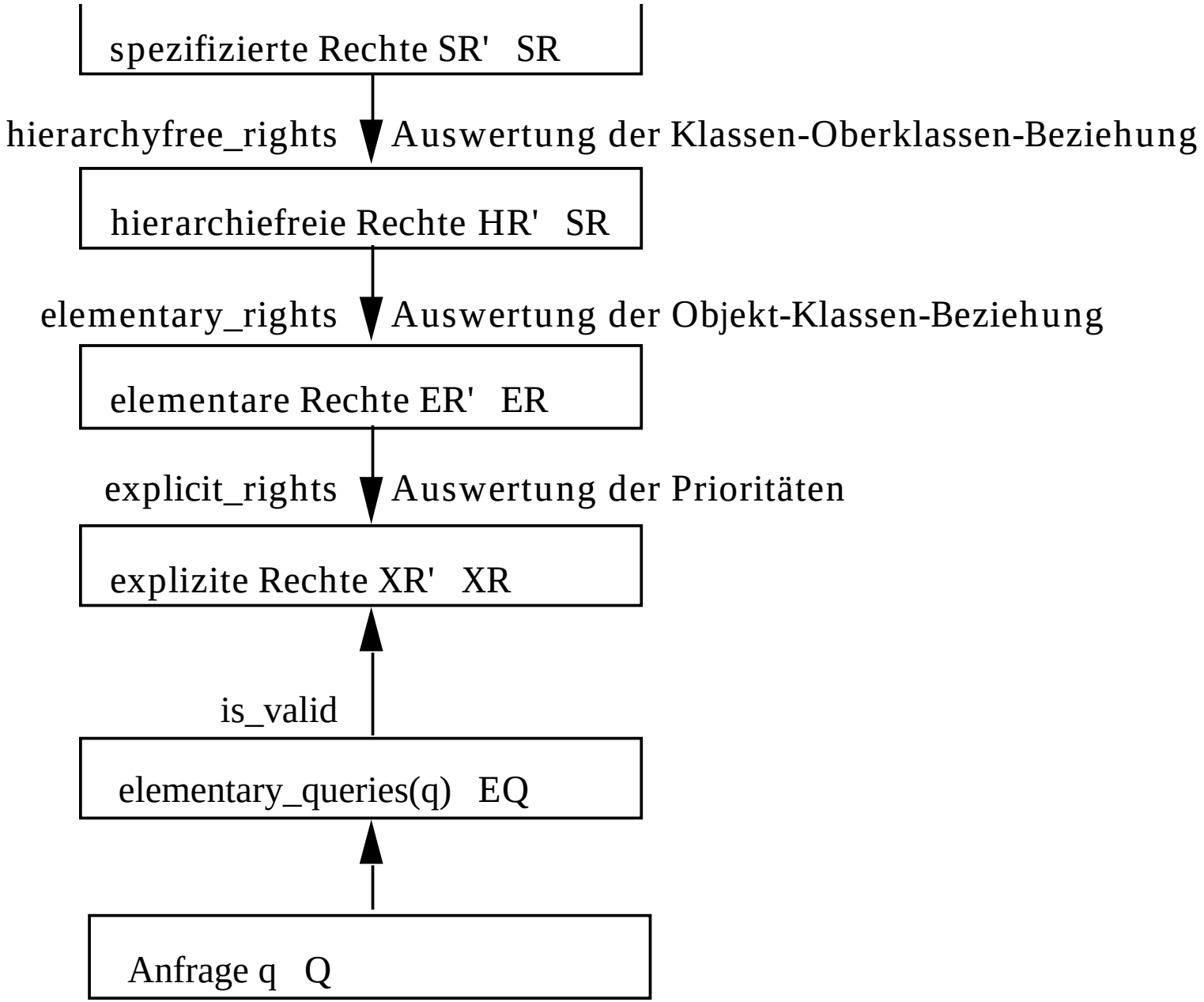
Definition: Sei q=(t,sa) eine Anfrage.

elementary_queries(t,sa) := {(t,ea) | ea elementary_actions*(t,sa)} = {t} × elementary_actions*(q)

ist die Menge der elementaren Anfragen, die durch Bearbeitung der Klassenhierarchie und Substitution der Klassennamen erzeugt wird.

Dann testen wir, ob die so erzeugte Menge enthalten ist in der Menge explicit_rights*(SR'):

Definition: $is\_valid_{zst}(q,SR') :\Leftrightarrow$ elementary_queries(q) explicit_rights*(SR').



Prozess der Rechteerzeugung und Anfragebearbeitung bei der Zustandsemantik

Aus pragmatischen Gründen schlagen wir vor, die Antwort auf folgende Art zu geben:

Um den Anfragenden nicht zu verwirren
(die Antwort "nein" hätte die Bedeutung von "Menge der elementaren Anfragen ist nicht voll enthalten in der Menge der expliziten Rechte" und ein nicht-pedantischer Benutzer könnte leicht "nicht voll erlaubt" mit "voll verboten" verwechseln)
sowie um eine aussagekräftigere Antwort zu geben, halten wir es in diesem Fall für richtig, ein nummerisches Ergebnis zurückzugeben:

Den Quotienten aus der Anzahl der erzeugten elementaren Anfragen, die wirklich in der Menge der expliziten Rechte (zu SR') liegen und der Anzahl der erzeugten elementaren Anfragen insgesamt.

D.h., "100%" steht für "ja" und alle anderen Zahlen sind verschiedene Versionen von "nein".
Und nur das Ergebnis "0%" stellt sicher, dass die Anfrage mit invertierter Rechtekennung eine Antwort mit der Bedeutung "ja" hätte.

Das Ergebnis ist also

$$| \text{elementary\_queries}(q) \cap \text{explicit\_rights}^*(SR') | \; / \; | \text{elementary\_queries}(q) |$$

bzw. eine entsprechende Fehlermeldung, falls $|\text{elementary\_queries}(q)|=0$.

Abschließend zeigen wir noch im Vorgriff auf Kapitel 7.3 einen Satz (7.6) über die Zustandssemantik, der weitgehende Analogien zu Satz 7.18 über die Struktursemantik zeigt.

Lemma 7.4: Seien alle Klassen nicht-leer.

Sei q=(t, sa') eine Anfrage mit sa'=(ss',so',sg'),
(t,ea) elementary_queries(q) mit ea=(es,eo,eg).

Dann gilt:

ea elementary_actions*(t*,sa*) mit sa*=(ss*,so*,sg*)

⇔

if is_object(ss*)
then if is_object(ss') then ss'=ss*
else ss* members(subclasses$_{,t}$(ss')) fi
else if is_object(ss') then ss' members(subclasses$_{,t*}$(ss*))
else es members(subclasses$_{,t}$(ss'))
∩ members(subclasses$_{,t*}$(ss*)) fi fi

if is_object(so*)
then if is_object(so') then so'=so*
else so* members(subclasses$_{,t}$(so')) fi
else if is_object(so') then so' members(subclasses$_{,t*}$(so*))
else eo members(subclasses$_{,t}$(so'))
∩ members(subclasses$_{,t*}$(so*)) fi fi

if is_object(sg*)
then if is_object(sg') then sg'=sg*
else sg* members($subclasses_{,t}$(sg')) fi
else if is_object(sg') then sg' members($subclasses_{,t*}$(sg*))
else eg members($subclasses_{,t}$(sg'))
$\cap$ members($subclasses_{,t*}$(sg*)) fi fi

Beweis: Zunächst gilt:

(t,es,eo,eg) elementary_queries(t,ss',so',sg')

$\Leftrightarrow$ (Def. 7.2)

(t,es,eo,eg) {t}× elementary_actions*(q)

$\Leftrightarrow$ (q=(t,ss',so',sg'))

(es,eo,eg) elementary_actions*(t,ss',so',sg')

$\Leftrightarrow$ (Def.5.14 (elementary_actions*))

es covered_objects*$_t$(ss') eo covered_objects*$_t$(so')
eg covered_objects*$_t$(sg')

Sei jetzt außerdem:

(es,eo,eg) elementary_actions*(t*,ss*,so*,sg*)

$\Leftrightarrow$ (Def.5.14 (elementary_actions*))

es covered_objects*$_{t*}$(ss*) eo covered_objects*$_{t*}$(so*)
eg covered_objects*$_{t*}$(sg*)

Insgesamt erhalten wir also, dass unter den Voraussetzungen des Satzes die linke Seite des Satzes äquivalent ist zu:

es covered_objects*$_t$(ss') eo covered_objects*$_t$(so')
eg covered_objects*$_t$(sg')
es covered_objects*$_{t*}$(ss*) eo covered_objects*$_{t*}$(so*)
eg covered_objects*$_{t*}$(sg*)

Für Subjekte (und für Operationen und Granule analog) erhalten wir:

es covered_objects*$_{t*}$(ss*) es covered_objects*$_t$(ss')

⇔

es covered_objects*$_{t*}$(ss*) ∩ covered_objects*$_t$(ss')

⇔ (Beweis Lemma 5.24)

if is_object(ss*)
then if is_object(ss') then ss'=ss*
else ss* members(subclasses$_{,t}$(ss')) fi
else if is_object(ss') then ss' members(subclasses$_{,t*}$(ss*))
else es members(subclasses$_{,t}$(ss'))
∩ members(subclasses$_{,t*}$(ss*)) fi fi •

Definition 7.5: Sei q=(t,sa) eine Anfrage und SR' SR eine Menge von spezifizierten Rechten. Dann affected_rights$_{zst}$(q, SR') :=
{(t'',p'',ea) elementary_rights*(SR') | (t,ea) elementary_queries(q)
sr''=(t'',p'',sa'') SR': ea elementary_actions*(t'',sa'')}

Die Menge affected_rights$_{zst}$(q, SR') kann mithilfe des letzten Lemmas effizient berechnet werden.

Theorem 7.6: Sei q=(t,sa) eine Anfrage und SR' SR eine Menge spezifizierter Rechte. Dann gilt:

is_valid$_{zst}$(q,SR')

⇔

er=(t,p,ea) affected_rights$_{zst}$(q, SR')
er**=(t**,p**,ea) affected_rights$_{zst}$(q, SR'): ¬(p**>p)

Beweis:

is_valid$_{zst}$(q,SR')

⇔ (Def. 7.3 (is_valid))

elementary_queries(q) explicit_rights*(SR')

⇔ (Def. 5.14 (explicit_rights*))

elementary_queries(t,sa)
forget_priority(maximal_priority(elementary_rights*(SR')))

⇔

(t,ea) elementary_queries(t,sa)
(t,ea) forget_priority(maximal_priority(elementary_rights*(SR')))

⇔ (Def. 5.10 (forget_priority))

(t,ea) elementary_queries(t,sa)
p PRIO: (t,p,ea) maximal_priority(elementary_rights*(SR'))

⇔ (Def. 5.10 (maximal_priority))

(t,ea) elementary_queries(t,sa)
p PRIO: er=(t,p,ea) elementary_rights*(SR')
er**=(t**,p**,ea) elementary_rights*(SR'): ¬(p**>p)

⇔ (Def. 7.5)

er=(t,p,ea) affected_rights$_{zst}$(q, SR')
er**=(t**,p**,ea) affected_rights$_{zst}$(q, SR'): ¬(p**>p) •

Obwohl die Zustandssemantik zunächst sehr intuitiv scheint, hat sie eine recht unangenehme Eigenschaft:

Bei gleicher Anfrage und gleicher Rechtemenge kann sie unterschiedliche Ergebnisse liefern, wenn sich zwischenzeitlich die Klassenmitgliedschaften ändern!

Während man i.a. davon ausgeht, dass die Klassen strukturelle Information abbilden und daher recht stabil sind, erwartet man dagegen für die Objekte eine größere Veränderungshäufigkeit.
Daher fällt diese Eigenschaft ins Gewicht.

Beispiel 7.7:
Sei SR'={(Erlaubnis, 20, s, untersuchen, g), (Erlaubnis, 20, s, röntgen, g)} eine Menge von spezifizierten Rechten und
q=(Erlaubnis, s, Diagnose, g) eine Anfrage.
Dann gilt:
$is\_valid_{zst}(q,SR')$=wahr, weil mc_O(Diagnose)={untersuchen, röntgen}.

Fügen wir aber ultraschall als neue Operation in die Klasse Diagnose ein, so gilt $is\_valid_{zst}(q,SR')$=falsch!

7.5.3 Strukturssemantik

7.5.3.1 Abgrenzung zur Zustandssemantik

Die Strukturssemantik ist eine Modifikation der Zustandssemantik.

Benutzt man die Zustandssemantik, so kann die gleiche Anfrage zu unterschiedlichen Ergebnissen führen, selbst wenn zwischendurch sich nur Klassenmitgliedschaften (nicht aber Rechte!) geändert haben.

In der Strukturssemantik setzen wir daher nicht die individuellen (Objekt-) Rechte für alle Mitglieder einer Klasse mit dem (strukturellen) Recht für diese Klasse gleich, insbesondere erlauben wir nicht, von einem Recht, dass gültig ist für die Menge aller Klassenmitglieder einer Klasse, auf das Recht zu schließen, dass gültig ist für eine Klasse.

Die Strukturssemantik ist insbesondere von Nutzen, wenn die Entwurfsphilosophie benutzt wird, dass die strukturellen Rollen einer Anwendung durch Klassen modelliert und Objekte für diese Klassen erst in einem späteren, zweiten Schritt zugeordnet werden.

Dann kann man auch schon in der ersten Phase Anfragen bzgl. der strukturellen Rechte stellen.

Soweit Klassennamen von der Anfrage betroffen sind, beantwortet die Strukturssemantik die Frage:

"Welche (strukturellen) Rechte *erhält* ein Objekt,
wenn es Mitglied dieser Klasse wird?"

Wird in einer Kategorie nach einer Klasse gefragt, so betrachtet die Strukturesemantik ausschließlich (hierarchiefreie) Rechte, die in dieser Kategorie Klassen enthalten.

Es gibt daher folgende Unterschiede zur Zustandssemantik.

1. Die Strukturesemantik unterscheidet zwischen **strukturellen Rechten** (für Klassen) und **individuellen Rechten** (für Objekte).
   Insbesondere sind individuelle Rechte für alle (echten) Objekte einer Klasse nicht identisch mit dem strukturellen Recht für diese Klasse.
   Ein Klassenname in einem spezifizierten Recht steht für mehr als für alle (gegenwärtigen) Klassenmitglieder.
   Das heißt insbesondere, dass ein Recht, das für alle Mitglieder einer Klasse gilt, nicht automatisch für die Klasse selbst gilt.

   Beispiel 7.8:
   Sei SR'={(Erlaubnis, 20, s, untersuchen, g), (Erlaubnis, 20, s, röntgen, g)} eine Menge von spezifizierten Rechten und q=(Erlaubnis, s, Diagnose, g) eine Anfrage.

   Benutzen wir die Zustandssemantik, so liefert $is\_valid_{zst}(q,SR')$ den Wert wahr, weil $mc_O(\text{Diagnose})=\{\text{untersuchen, röntgen}\}$.

   Die Strukturesemantik berücksichtigt die Tatsache, dass sich Klassenmitgliedschaften ändern können und dass die spezifizierten Rechte die Gültigkeit von (Erlaubnis, s, Diagnose, g) -trotz invarianter Rechte- nicht auf Dauer garantieren können (z.B. könnte die Klasse Diagnose weitere Klassenmitglieder erhalten). Daher liefert $is\_valid_{str}(q,SR')$ den Wert falsch.

Durch die Eigenschaft, dass ein Klassenname in einem Recht für mehr als seine augenblicklichen Klassenmitglieder steht, wird die anfängliche Zuweisung eines **charakteristischen Objekts** an jede Klasse gerechtfertigt.
Da diesen charakteristischen Objekten individuell kein Recht gegeben werden kann, kann eine Menge individueller Rechte niemals äquivalent zu einem strukturellen Recht sein.

2. Ausnahmen für Objekte (und zwar sind hier nur die Mitglieder von Klassen relevant) werden von der Struktursemantik ignoriert (obwohl die Ausnahme eine höhere Priorität hat!), da sie für die Beschreibung der strukturellen Rechte einer Klasse irrelevant sind.

   Ausnahmen für Klassen werden jedoch berücksichtigt.

   Beispiel 7.9:
   Angenommen es sei eine Ausnahme für ein Klassenmitglied spezifiziert: SR' = {(Erlaubnis, 50, Chirurg, Med. Operation, Innere Organe), (Verbot, 60, hendrik, Med. Operation, herz)}.

   Für die Anfrage

   q=(Erlaubnis, Chirurg, Med. Operation, Innere Organe)

   liefert $is\_valid_{str}(q,SR')$ den Wert wahr, aber der Chirurg hendrik hat nicht die vollen Rechte.

3. Dürfen Objekte mehreren Klassen angehören, so gibt es (sofern man auf charakteristische Objekte verzichtet hat) einen weiteren Unterschied:

   Die Zustandssemantik berücksichtigt auch solche Rechte, bei denen die Objekte der Klasse, für die das Recht vergeben ist, auf verschiedene Unterklassen der Klasse verteilt sind, die angefragt wird; die Strukturseman tik nicht.

   Beispiel 7.10: Sei classes(obj_1)={cl_1, cl_2}, classes(obj_2)={cl_2, cl_3}, subclasses(cl_4)={cl_1, cl_3}:

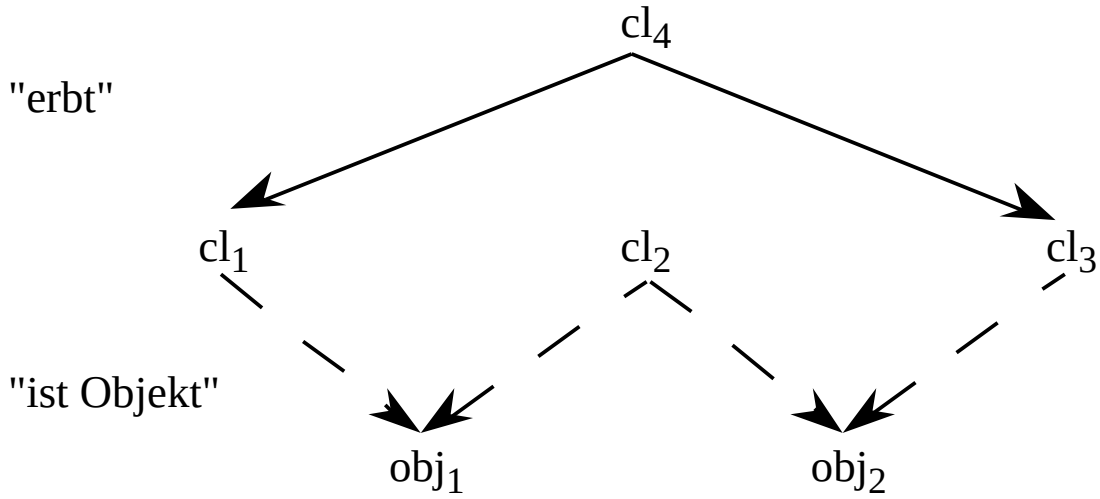


Bild 7.2: Unterschied zwischen Zustands- und Strukturseman tik bei mehrfachen Klassenzugehörigkeiten

Sei SR'={(Erlaubnis, 50, cl_2, o, g)} und q=(Erlaubnis, cl_4, o, g).
Dann ist is_valid_{zst}(q, SR')=true und is_valid_{str}(q, SR')=false.

4. Falls man die Struktursemantik auch zur Erkennung von Widersprüchen benutzen will, muss man beachten, dass (soweit Klassen in Rechten betroffen sind) Widersprüche, die auf der mehrfachen Klassenmitgliedschaft eines Objekts beruhen, von der Struktursemantik nicht erkannt werden.

Beispiel 7.11: Seien cl_1 und cl_2 zwei Subjektklassen und s $cl_1 \cap cl_2$ ein Objekt, das Mitglied beider Klassen ist, o O, g G, p PRIO.

Seien die beiden spezifizierten Rechte

sr_1=(Erlaubnis, p, cl_1, o, g),

sr_2=(Verbot,p,cl_2,o,g) gegeben.

Dann liegt ein Konflikt bzgl. der Handlung (s, o, g) vor, der mit den Algorithmen für die Struktursemantik nicht erkannt wird.[10]

---

10 Dieses Problem kann z.B. dadurch gelöst werden, dass man sich merkt, welche Paare von Klassen gemeinsame Objekte haben und welche Unverträglichkeiten zwischen Klassen auf Grund der spezifizierten Rechte bestehen und dann bei Rechteänderungen oder Änderungen der Klassenzuordnung vergleicht, ob die Rechte-Unverträglichkeiten zwischen Klassen zusammen mit den Klassenzuordnungen der Objekte zu einem **Widerspruch wegen mehrfacher Klassenmitgliedschaft** führen.

7.5.3.2 Definition der Struktursemantik

Für die Berechnung der Struktursemantik betrachten wir die Klassennamen nicht mehr nur als Abkürzungen, sondern als eigenständige Einheiten.

Zur Berechnung der Struktursemantik stellt man sich am besten folgendes vor:
Es wird (im Gegensatz zur bisherigen Beschreibung) die Objekt-Klassen-Beziehung nicht ausgewertet, sondern die Prioritäten werden unmittelbar bzgl. der erzeugten hierarchiefreien Rechte ausgewertet.

Wir erhalten damit Rechte, die außer der Handlung nur noch eine Rechtekennung enthalten; die Handlung muss aber nicht elementar sein.

Die Objekt-Klassen-Beziehung berücksichtigen wir dann bei Anfragen, indem wir Anfragen, die Objekte enthalten, automatisch dahin erweitern, dass wir nach diesem Objekt und nach den Klassen fragen, zu denen das Objekt gehört.

Zunächst definieren wir **verallgemeinerte explizite Rechte**. Diese dürfen Klassennamen enthalten.

Definition 7.12: Die Menge aller verallgemeinerten expliziten Rechte ist GXR:=TAG×A.
Ein **verallgemeinertes explizites Recht** ist also ein Tupel gxr=(t,sa)= (t,(ss,so,sg)) TAG×A.

Definition 7.13: Sei SR' SR eine Menge von spezifizierten Rechten und HR'=hierarchyfree_rights(SR').
maximal_priority(HR') := {(t,p,ha) HR' | hr' HR': hr'=(t',p',ha) ¬(p'>p)}
forget_priority(HR') := {(t,ha) GXR | p PRIO: (t,p,ha) HR'}

Die Menge der verallgemeinerten expliziten Rechte, die aus einer Menge von spezifizierten Rechten SR' erzeugt wird, ergibt sich dann durch:

explicit-class-rights(SR') := forget_priority(maximal_priority(hierarchyfree_rights(SR')))

Definition 7.14: Für eine Anfrage q=(sa,t) mit sa=(ss,so,sg) definieren wir:

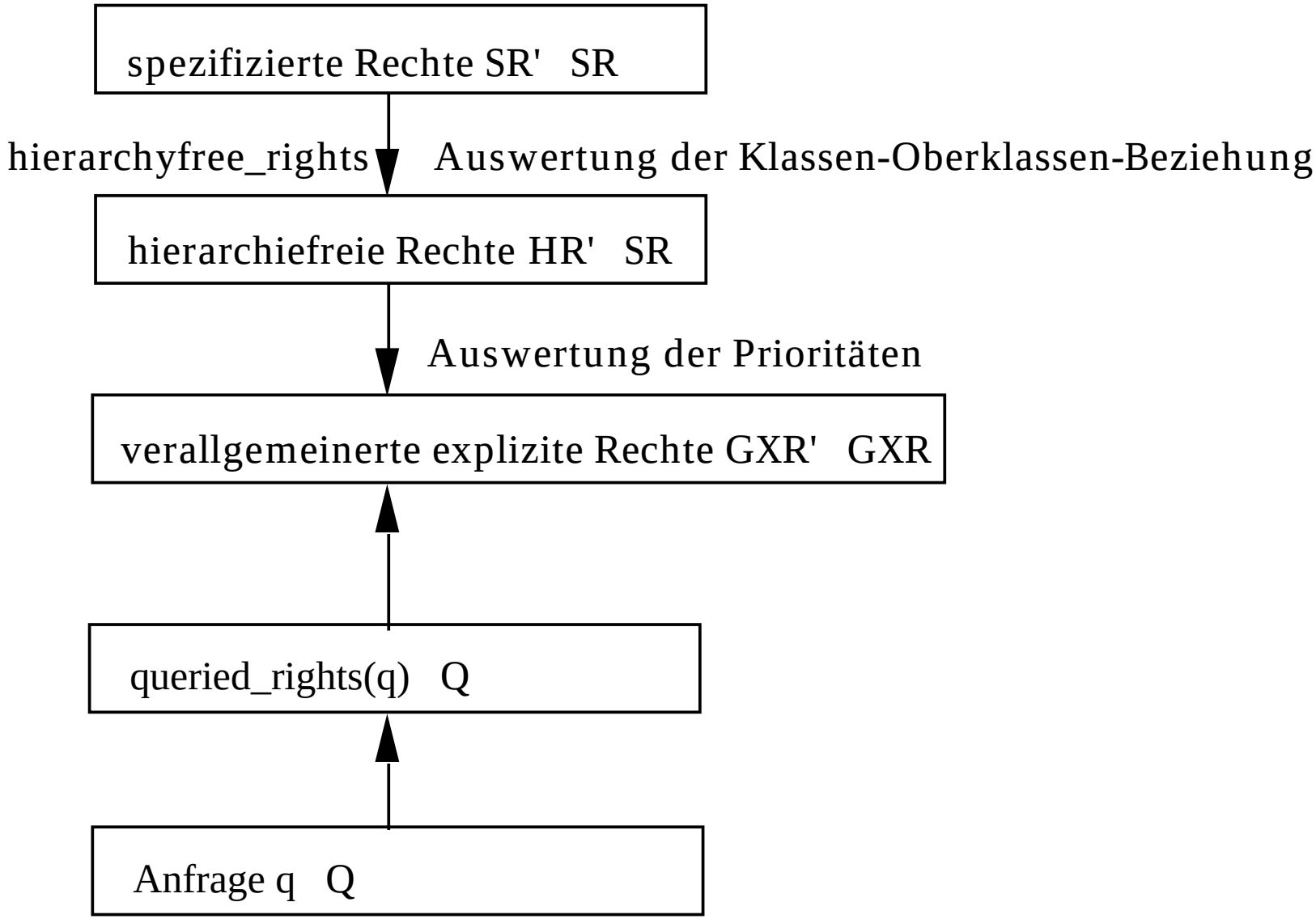
queried_items(ss) := if is_object(ss) then {ss} classes(ss) else {ss} fi
queried_items(so) := if is_object(so) then {so} classes(so) else {so} fi
queried_items(sg) := if is_object(sg) then {sg} classes(sg) else {sg} fi

queried_actions(ss,so,sg)= (queried_items(ss), queried_items(so), queried_items(sg))

queried_rights(t,ss,so,sg)= (t, queried_items(ss), queried_items(so), queried_items(sg))

Beispiel 7.15: Die Anfrage q=(Erlaubnis, john, röntgen, Kiefer) führt bei punktweiser Auswertung zu folgenden verallgemeinerten expliziten Rechten:
queried_rights(q)={(Erlaubnis, john, röntgen, Kiefer), (Erlaubnis, john, Diagnose, Kiefer), (Erlaubnis, Arzt, röntgen, Kiefer), (Erlaubnis, Arzt, Diagnose, Kiefer)}.
Gehört jedes Objekt zu genau einer Klasse und enthält die Handlung sa n Objekte, n {0,1,2,3}, so enthält queried_rights(sa,t) 2^n verallgemeinerte explizite Rechte, also 1, 2, 4 oder 8.

Die Struktursemantik wird dann wie folgt definiert:
Definition 7.16:
$is\_valid_{str}(q, SR') : \Leftrightarrow$ gxr queried_rights(q): gxr explicit-class-rights(SR')



Prozess der Rechteerzeugung und Anfragebearbeitung bei der Struktursemantik

Wenn wir die Zustandssemantik benutzen, ist es nicht einfach, die "Ersetzen und Nachgucken"-Berechnungsmethode abzukürzen.

Wenn z.B. eine Anfrage einen Klassennamen enthält, so ist es nicht hinreichend, nur diejenigen spezifizierten Rechte zu betrachten, die Klassennamen enthalten (vgl. Beispiel in Abschnitt 7.2).

Die Struktursemantik liefert dagegen die Möglichkeit, eine effizientere Anfragebearbeitung zu erreichen.

Die Schlüsselidee dazu ist, die spezifizierten Rechte eins nach dem anderen zu überprüfen und sofort zu entscheiden, ob die Handlung eines Rechts von Bedeutung für die Anfrage ist oder nicht.

Dies ist aus der Sicht der Berechnungskomplexität ein entscheidender Vorteil.
Diese Vorgehensweise ist möglich, weil es bei der Struktursemantik keine spezifizierten Rechte gibt, die eine Anfrage nur teilweise betreffen (wie es bei der Zustandssemantik der Fall sein kann).
Das zeigt nachfolgender Satz.
Zunächst definieren wir jedoch die Menge der Rechte, die bzgl. der Handlung der Rechteanfrage von Bedeutung sind.

Definition 7.17: Sei q=(t, sa') eine Anfrage und SR' SR eine Menge von spezifizierten Rechten. Dann affected_rights$_{str}$(q, SR') :=

{sr''=(t'',p'', sa'') SR' |
sa A: (t,sa) queried_rights(t,sa') sa hierarchyfree_actions(t",sa")}

Insbesondere hängt affected_rights$_{str}$(q, SR') nur von der Handlung und nicht von der Rechtekennung der Anfrage ab.

Theorem 7.18: Sei q=(t,sa') eine Anfrage mit sa'=(ss',so',sg') und SR' SR eine Menge spezifizierter Rechte. Dann gilt:

is_valid$_{str}$(q, SR') $\Leftrightarrow$

sr*=(t,p,sa*) affected_rights$_{str}$(q, SR')

sr**=(t**,p**,sa**) affected_rights$_{str}$(q, SR'): ¬ (p**>p)

Beweis:

is_valid$_{str}$(q, SR')

$\Leftrightarrow$ (Def. 7.16)

gxr queried_rights(q): gxr explicit-class-rights(SR')

$\Leftrightarrow$ (Def. 7.13 (explicit-class-rights))

gxr queried_rights(q):

gxr forget_priority(maximal_priority(hierarchyfree_rights(SR')))

$\Leftrightarrow$ (gxr=(t,sa))

(t,sa) queried_rights(q):

(t,sa) forget_priority(maximal_priority(hierarchyfree_rights(SR')))

$\Leftrightarrow$ (Def. 7.13 (forget_priority))

(t,sa) queried_rights(q): p PRIO:

(t,p,sa) maximal_priority(hierarchyfree_rights(SR'))

$\Leftrightarrow$ (Def. 7.13 (maximal_priority))

(t,sa) queried_rights(q): p PRIO:
(t,p,sa) hierarchyfree_rights(SR')
(t***,p***,sa) hierarchyfree_rights(SR'): ¬(p***>p)

⇔ (Def. 5.5 (hierarchyfree_rights))

(t,sa) queried_rights(q): p PRIO: sr*=(t*,p*,sa*) SR':
(t,p,sa) hierarchyfree_rights(sr*)
sr**=(t**,p**,sa**) SR':
(t***,p***,sa***) hierarchyfree_rights(sr**) sa=sa*** ¬(p***>p)

⇔ (Def. 5.4 (hierarchyfree_rights))

(t,sa) queried_rights(q): p PRIO: sr*=(t*,p*,sa*) SR':
t=t* p=p* sa hierarchyfree_actions(t*,sa*)
sr**=(t**,p**,sa**) SR':
t**=t*** p**=p*** sa*** hierarchyfree_actions(t**,sa**) sa=sa***
¬(p***>p)

⇔ (Vereinfachung)

(t,sa) queried_rights(t,sa'): p PRIO: sr*=(t,p,sa*) SR':
sa hierarchyfree_actions(t,sa*)
sr**=(t**,p**,sa**) SR':
sa hierarchyfree_actions(t**,sa**) ¬(p**>p)

⇔ (Def. 7.17)

sr*=(t,p,sa*) affected_rights$_{str}$(t,sa',SR')
sr**=(t**,p**,sa**) affected_rights$_{str}$(t,sa',SR'): ¬(p**>p) •

Der letzte Satz legt nahe, eine Anfrage wie folgt zu berechnen:

- berechne die Menge der spezifizierten Rechte, die die Handlung sa' der Anfrage betreffen,
- aus dieser Menge wähle diejenigen Rechte aus, die eine maximale Priorität haben (dann ist bei Abwesenheit von aktuellen Konflikten sichergestellt, dass alle ausgewählten Rechte die gleiche Rechtekennung t haben!),
- vergleiche die Rechtekennung t' der Anfrage mit der Rechtekennung t der eben ermittelten Menge.

Operational ergibt sich dann bei der Struktursemantik das Ergebnis einer Anfrage q=(t',sa') bezüglich einer Rechtemenge SR' SR offensichtlich durch:

$$is\_valid_{str}(q,SR') \Leftrightarrow tag(maxpriority(affected\_rights_{str}(q,SR')))=t'$$

mit den Definitionen:

Definition 7.19:
maxpriority(SR') := {(t,p,sa) SR' | (t'',p'',sa'') SR': ¬(p''>p)}

Die Funktion maxpriority wählt aus einer Menge von spezifizierten Rechten SR' diejenigen spezifizierten Rechte mit einer maximalen Priorität aus.

$$tag(SR') := \begin{cases} \text{Erlaubnis} & \text{falls } SR' \neq \emptyset \text{ und } sr \ SR': sr=(t,p,sa) & t=\text{Erlaubnis} \\ \text{Verbot} & \text{falls } SR' \neq \emptyset \text{ und } sr \ SR': sr=(t,p,sa) & t=\text{Verbot} \\ \text{undefiniert} & \text{sonst} & \end{cases}$$

Das Prädikat tag liefert die Rechtekennung einer Menge von spezifizierten Rechten SR', sofern alle ihre Elemente die gleiche Rechtekennung haben.

7.5.3.3 Effiziente Berechnung der Struktursemantik

Die effiziente Berechnung der Struktursemantik hängt weitgehend von der effizienten Berechnung der Menge $affected\_rights_{str}$ ab.
Zur Ermittlung der effizienten Berechnung von $affected\_rights_{str}$ benötigen wir zunächst ein paar Hilfssätze:

Lemma 7.20: Seien i', i* Objekte oder Klassen einer Kategorie und t* eine Rechtekennung. Dann gilt:

queried_items(i') ∩ $covered\_items_{,t*}$(i*) ≠ Ø
⇔ if is_object(i*) then i*=i'
else if is_object(i') then i $subclasses_{,t*}$(i*) ∩ classes(i')
else i' $subclasses_{,t*}$(i*) fi fi

Beweis: i: i queried_items(i') ∩ $covered\_items_{,t*}$(i*)

⇔ (Def. 5.4 (covered_items))
i: i queried_items(i')
if is_object(i*) then i=i* else i $subclasses_{,t*}$(i*) fi

⇔ (Def. 7.14 (queried_items))
i: if is_object(i') then i {i'} classes(i') else i {i'} fi
if is_object(i*) then i=i* else i $subclasses_{,t*}$(i*) fi

⇔
i: if is_object(i') then i {i'} classes(i') else i=i' fi
if is_object(i*) then i=i* else i $subclasses_{,t*}$(i*) fi

⇔ (Fallunterscheidung)
i: if is_object(i*) then

if is_object(i') then i=i* i {i'} classes(i')
else i=i* i=i' fi
else if is_object(i') then i subclasses$_{,t*}$(i*) i {i'} classes(i') [11]
else i subclasses$_{,t*}$(i*) i=i' fi fi

⇔ (Vereinfachung)

if is_object(i*) then
if is_object(i') then i* {i'} classes(i') [12]
else i*=i' fi
else if is_object(i') then i subclasses$_{,t*}$(i*) ∩ classes(i')
else i' subclasses$_{,t*}$(i*) fi fi

⇔ (Vereinfachung)

if is_object(i*) then
if is_object(i') then i*=i'
else i*=i' fi [13]
else if is_object(i') then i subclasses$_{,t*}$(i*) ∩ classes(i')
else i' subclasses$_{,t*}$(i*) fi fi

⇔ (Vereinfachung)

if is_object(i*) then i*=i'
else if is_object(i') then i subclasses$_{,t*}$(i*) ∩ classes(i')
else i' subclasses$_{,t*}$(i*) fi fi •

---

11 Da i covered_classes(t*,i*) eine Klasse ist und i' ein Objekt, entfällt die Möglichkeit, dass i=i' ist.

12 Da i* Objekt ist, entfällt die Möglichkeit, dass i* classes(i').

13 Diese Bedingung ist unerfüllbar; wir nutzen sie zur Vereinfachung der Fallunterscheidung.

Unter der zusätzlichen Voraussetzung, dass jede Klasse ein charakteristisches Objekt hat, an das ja kein Recht vergeben werden kann, gilt sogar nach Lemma 5.28:

Lemma 7.21: Seien i', i* Objekte, für die Rechte vergeben werden können, oder Klassen der gleichen Kategorie und t* eine Rechtekennung und enthalte jede Klasse ein charakteristisches Objekt. Dann gilt:

queried_items(i') ∩ covered_items$_{,t*}$(i*) ≠ Ø

⇔ covered_objects*$_{t*}$(i') covered_objects*$_{t*}$(i*)

Lemma 7.22: Sei q=(t,sa') eine Anfrage mit sa'=(ss',so',sg'),

(t,sa) queried_rights(q) mit sa=(ss,so,sg).

Dann gilt:

sa hierarchyfree_actions(t*,sa*) mit sa*=(ss*,so*,sg*)

⇔ if is_object(ss*) then ss*=ss'
else if is_object(ss') then subclasses$_{,t*}$(ss*) ∩ classes(ss')≠Ø
else ss' subclasses$_{,t*}$(ss*) fi fi
if is_object(so*) then so*=so'
else if is_object(so') then subclasses$_{,t*}$(so*) ∩ classes(so')≠Ø
else so' subclasses$_{,t*}$(so*) fi fi
if is_object(sg*) then sg*=sg'
else if is_object(sg') then subclasses$_{,t*}$(sg*) ∩ classes(sg')≠Ø
else sg' subclasses$_{,t*}$(sg*) fi fi

Beweis: Zunächst einmal gilt:

(t,ss,so,sg) queried_rights(t,ss',so',sg')

⇔ (Def. 7.14 (queried_rights))

(ss,so,sg) queried_actions(ss',so',sg')

⇔ (Def. 7.14 (queried_actions))

ss queried_items(ss') so queried_items(so') sg queried_items(sg')

Sei jetzt außerdem:

(ss,so,sg) hierarchyfree_actions(t*,ss*,so*,sg*)

⇔ (Def. 5.4 (hierarchyfree_actions))

ss covered_items$_{,t*}$(ss*) so covered_items$_{,t*}$(so*)

sg covered_items$_{,t*}$(sg*)

Insgesamt erhalten wir also, dass unter den Voraussetzungen des Satzes die linke Seite des Satzes äquivalent ist zu:

ss queried_items(ss') so queried_items(so') sg queried_items(sg')

ss covered_items$_{,t*}$(ss*) so covered_items$_{,t*}$(so*)

sg covered_items$_{,t*}$(sg*)

Mit dem Lemma 7.20 erhalten wir für Subjekte (und für Operationen und Granule analog):

ss queried_items(ss') ss covered_items$_{,t*}$(ss*)

⇔ (Lemma 7.20)

if is_object(ss*) then ss*=ss'

else if is_object(ss') then subclasses$_{,t*}$(ss*) ∩ classes(ss')≠Ø

else ss' ∈ subclasses$_{,t*}$(ss*) fi fi •

Die Menge affected_rights$_{str}$(q, SR') kann mithilfe des Lemmas 7.22 effizient berechnet werden:

affected_rights$_{str}$((t,ss',so',sg'), SR') = {(t*,p*,ss*,so*,sg*) ∈ SR' |
if is_object(ss*) then ss*=ss'
else if is_object(ss') then subclasses$_{,t*}$(ss*) ∩ classes(ss')≠Ø
else ss' ∈ subclasses$_{,t*}$(ss*) fi fi
if is_object(so*) then so*=so'
else if is_object(so') then subclasses$_{,t*}$(so*) ∩ classes(so')≠Ø
else so' ∈ subclasses$_{,t*}$(so*) fi fi
if is_object(sg*) then sg*=sg'
else if is_object(sg') then subclasses$_{,t*}$(sg*) ∩ classes(sg')≠Ø
else sg' ∈ subclasses$_{,t*}$(sg*) fi fi }

Hat jede Klasse ein charakteristisches Objekt, so lässt sich $affected\_rights_{str}$ auch einfach mit Hilfe der überdeckten Handlungen charakterisieren:

Theorem 7.23: Sei q=(t,sa') eine Anfrage und SR' SR eine Menge von spezifizierten Rechten und enthalte jede Klasse ein charakteristisches Objekt. Dann gilt:

$sr^*=(t^*,p^*,sa^*)$ $affected\_rights_{str}(q,SR')$

$\Leftrightarrow$ $elementary\_actions^*(t^*,sa')$ $elementary\_actions^*(t^*,sa^*)$

Beweis:

(t^*,p^*,sa^*) $affected\_rights_{str}(q,SR')$

$\Leftrightarrow$ (Def. 7.17)

$sa=(ss,so,sg)$ A: (t,sa) queried_rights(t,sa')
sa hierarchyfree_actions(t*,sa*)

$\Leftrightarrow$ (sa*=(ss*,so*,sg*), Def. 7.14 (queried_rights), Def. 5.4 (hierarchyfree_actions))

ss S SC: ss queried_items(ss') ss $covered\_items_{,t^*}(ss^*)$
so O OC: so queried_items(so') so $covered\_items_{,t^*}(so^*)$
sg G GC: sg queried_items(sg') sg $covered\_items_{,t^*}(sg^*)$

$\Leftrightarrow$ (Lemma 7.21)

$covered\_objects^*_{t^*}(ss')$ $covered\_objects^*_{t^*}(ss^*)$
$covered\_objects^*_{t^*}(so')$ $covered\_objects^*_{t^*}(so^*)$
$covered\_objects^*_{t^*}(sg')$ $covered\_objects^*_{t^*}(sg^*)$

$\Leftrightarrow$ (Def. 5.14 (elementary _actions*))

$elementary\_actions^*(t^*,sa')$ $elementary\_actions^*(t^*,sa^*)$ •

Für die Berechnung der Rechte, die eine Anfrage betreffen, gilt das folgende

Theorem 7.24: Sei q=(t',ss',so',sg') eine Anfrage, SR' eine Menge von spezifizierten Rechten, die frei ist von aktuellen Konflikten. Seien alle Klassen nichtleer und jedes Objekt gehöre zu nur einer Klasse.
Die Hierarchien für Subjekt- und Operationsklassen seien gegenläufig zu behandeln, die für Granulklassen gleichläufig. Dann gilt:

$affected\_rights_{str}$(q,SR') = {(t,p,ss,so,sg) SR' |
if is_object(ss')
then if t=Erlaubnis then ss {ss'} superclasses(classes(ss'))
else ss {ss'} subclasses(classes(ss')) fi
else if t=Erlaubnis then ss superclasses(ss') else ss subclasses(ss') fi fi

if is_object(so')
then if t=Erlaubnis then so {so'} superclasses(classes(so'))
else so {so'} subclasses(classes(so')) fi
else if t=Erlaubnis then so superclasses(so') else so subclasses(so') fi fi

if is_object(sg')
then sg {sg'} superclasses(classes(sg'))
else sg superclasses(sg') fi }

Aufwandsabschätzung:
Jedes spezifizierte Recht braucht also zur Berechnung der affected_rights$_{str}$-Funktion nur einmal betrachtet zu werden.
Die Betrachtung umfasst für jedes Recht einen Vergleich der drei Handlungskomponenten mit einer Menge, deren Kardinalität beschränkt ist durch die Höhe des zugehörigen Hierarchiewaldes plus 1 (im Fall, dass jede Klasse nur eine Oberklasse hat und jedes Objekt nur eine Klasse) bzw. durch die Anzahl der Knoten im zugehörigen Hierarchiewald plus 1 (im allgemeinen Fall).

Die Klasse eines Objektes kann sehr einfach berechnet werden.
Die Mengen der Ober- und Unterklassen einer Klasse können zur Initialisierungszeit vorberechnet werden.

Beispiel 7.25: Wir stellen die Rechteanfrage

q=(Erlaubnis, Zahnarzt, injizieren, Gliedmaßen).

Dafür haben wir zu berechnen:

affected_rights$_{str}$(Erlaubnis, Zahnarzt, injizieren, Gliedmaßen, SR_1)

=

{(t,p,ss,so,sg) SR_1 | (t=Erlaubnis
ss { Zahnarzt, Arzt, Krankenschwester, Zivildienstleistender, Zahnarzthelfer}
so {injizieren, Therapie, Med. Operation}
sg {Gliedmaßen, Körper})
(t=Verbot ss {Zahnarzt} so {injizieren, Therapie, Pflege}
sg {Gliedmaßen, Körper})}

=

{ (Erlaubnis, 10, Arzt, Therapie, Körper),
(Verbot, 20, Zahnarzt, Therapie, Gliedmaßen),
(Erlaubnis, 30, Krankenschwester, injizieren, Gliedmaßen) }

$is\_valid_{str}(q,SR_1)$

⇔

$tag(maxpriority(affected\_rights_{str}(q,SR_1)))=t'$

⇔

tag(maxpriority({(Erlaubnis, 10, Arzt, Therapie, Körper),
(Verbot, 20, Zahnarzt, Therapie, Gliedmaßen),
(Erlaubnis, 30, Krankenschwester, injizieren, Gliedmaßen) }))
= Erlaubnis

⇔

tag({(Erlaubnis, 30, Krankenschwester, injizieren, Gliedmaßen) }) = Erlaubnis

⇔

Erlaubnis=Erlaubnis

⇔

wahr

Also ist das angefragte Recht vorhanden.