Dissertation

# Ein Bedingungsmodell für Planungsprobleme in strukturierten Domänen

von
*Petra Schwaiger*

eingereicht an der

*Fakultät für Informatik und Mathematik*
*Universität Passau*

Januar 2008

Erstgutachter:
*Prof. Dr. Burkhard Freitag*

Zweitgutachter:
*Prof. Dr. Harald Kosch*

# Zusammenfassung

Computerunterstützte Beratungssysteme finden sowohl in der Industrie als auch im akademischen Bereich eine zunehmende Bedeutung. Die Anforderungen an solche Systeme sind hinsichtlich der abbildbaren Strukturen, der Flexibilität der Anfragen und der Vollständigkeit und Korrektheit der Antworten hoch. Dies gilt insbesondere für Planungsprobleme in strukturierten Domänen.

Derartige Probleme treten beispielsweise bei der Erstellung von Tests auf der Grundlage einer Menge von Fragen und gewissen Anforderungen an den Test, bei der Konsistenzprüfung von Studienordnungen und bei der computerunterstützen Studienberatung auf.

In der vorliegenden Arbeit wird ein Framework zur Behandlung eben genannter Probleme präsentiert. Die vorgestellte Lösung bietet durch den modellbasierten Ansatz und die entwickelte anwendungsnahe Modellierungssprache – gerade auch im Vergleich zu existierenden Ansätzen – einen sehr hohen Grad an Abstraktion, Allgemeingültigkeit, Ausdrucksstärke, Flexibilität und Integrierbarkeit. Im Rahmen des entwickelten Modells wird eine geeignete Verzahnung von strukturellen und constraintbasierten Aspekten erreicht. Der hierbei in Syntax und Semantik definierte Constraintbegriff kann darüber hinaus als Formalisierung und Verallgemeinerung von Pfadconstraints bzw. Pfadanfragen in hierarchischen Datenmodellen aufgefasst werden.

Für die interne Repräsentation erweist sich ein logikbasierter Ansatz mit Constraints, nämlich Answer Set Programming mit Gewichten, als eine ausgezeichnete Methode bezüglich der Ausdrucksstärke, Mächtigkeit und Adäquatheit. Die Praxistauglichkeit des verfolgten Ansatzes im Hinblick auf Performanz und Skalierbarkeit wird in verschiedenen realen Anwendungsfällen demonstriert.

# Danksagung

Diese Doktorarbeit entstand im Wesentlichen während meiner dreijährigen Voll- und einjährigen Teilbeschäftigung als wissenschaftliche Mitarbeiterin am Lehrstuhl für Informationsmanagement bei Prof. Dr. Burkhard Freitag an der Universität Passau.

Ganz besonderer Dank gilt Herrn Prof. Dr. Burkhard Freitag für die Möglichkeit der wissenschaftlichen Tätigkeit an seinem Lehrstuhl und für seine Unterstützung, Betreuung und Ermutigung. Er begleitete meine wissenschaftliche Arbeit und bereicherte sie durch zahlreiche Fachgespräche und wertvolle Anregungen.

Herrn Prof. Dr. Harald Kosch danke ich für die Übernahme des Zweitgutachtens.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Motivation

Computerunterstützte Beratungssysteme finden in unserer Informationsgesellschaft einen immer breiteren Einsatzbereich. Der Wunsch nach Automatisierung von Beratungsaufgaben ist sowohl in der Industrie als auch im akademischen Bereich zu beobachten. Allerdings werden je nach Einsatzbereich derartige Informationssysteme immer komplexer und es werden immer höhere Anforderungen an sie gestellt.

Diese Problematik soll exemplarisch im Rahmen zweier konkreter Anwendungsbereiche aufgezeigt, diskutiert und gelöst werden, nämlich zum einen bei der automatisierten Erstellung von Tests auf der Grundlage einer Menge von Fragen und gewissen Anforderungen an den Test und zum anderen bei der Konsistenzprüfung von Studienordnungen und der Studienberatung.

Diesen Anwendungsbereichen ist gemein, dass neben einer mehr oder weniger stark strukturierten Domäne Anforderungen an die Lösungen des jeweiligen Problems auftreten, welche mit Hilfe von Constraints gewisser „Constraintklassen“ formuliert werden können. Diese Anforderungen lassen sich als „Planungsprobleme in strukturierten Domänen“ verallgemeinern.

Da die Komplexität solcher Planungsprobleme im Allgemeinen hoch ist, scheiden Ansätze aus, welche Standard- und Trivialmethoden verwenden. Für die interne Repräsentation erscheinen hingegen Methoden, welche sowohl mit strukturellen Gegebenheiten als auch mit Constraints „gut umgehen“ können, als geeignet. Logikbasierte Ansätze mit Constraints wie beispielsweise Answer Set Programming mit Gewichten (vgl. Abschnitt 4.2) erweisen sich hierfür als gut geeignet. Auch gängige pfadbasierte Ansätze (vgl. Abschnitt 4.3) lassen sich bis zu einem gewissen Grad gut verwenden, stoßen jedoch beispielsweise schon aufgrund der auftretenden Anzahlbedingungen an die Grenzen der Ausdrucksmöglichkeit.

In keinem der betrachteten Anwendungsbereiche kann eine vernünftige Handhabung für den Administrator oder den Benutzer des jeweiligen Systems erreicht werden, wenn Änderungen und Anpassungen im internen Formalismus der entsprechenden Programmiersprache vorgenommen werden müssen. Darüber hinaus kann die Offenheit und Flexibilität der Systeme nur dann gewährleistet werden, wenn geeignete Schnittstellen bereitgestellt werden.

Im Rahmen dieser Arbeit wird daher ein modellgetriebener Ansatz verfolgt. Das entwickelte Modell, welches ein zentrales Element der Problemlösung ist, verwirklicht eine Verzahnung von strukturellen und constraintbasierten Aspekten und damit einen hohen Grad an Allgemeingültigkeit und Abstraktion. Es bildet die Schnittstelle sowohl zur internen Repräsentation als auch zur Bedienoberfläche und stellt daher die Basis sowohl für die Definition einer benutzernahen Spezifikationssprache als auch für die Transformationen in interne Formalismen dar.

Darüber hinaus wird hiermit die Plattformunabhängigkeit sichergestellt. Verschiedene interne Formalismen sind möglich. Einstellungen und Anpassungen, die der Administrator bzw. der Benutzer vorzunehmen hat, sollen möglichst benutzernah und deklarativ sein. Der verwendete Ansatz unterstützt dies. Eine definierte Transformationsvorschrift lässt eine automatisierte Transformation einer Probleminstanz in den verwendeten internen Formalismus zu.

## 1.2 Überblick

Die Arbeit ist wie folgt aufgebaut: In Kapitel 2 erfolgt die Beschreibung des zu lösenden Problems. Neben der Grundstruktur der Problemstellung werden die betrachteten Anwendungsbereiche in informeller Weise definiert. Darüber hinaus wird auf die wesentlichen Aspekte des Beitrags dieser Arbeit in allgemeiner Form eingegangen.

Darauf aufbauend wird in Kapitel 3 der verfolgte Lösungsansatz aufgezeigt. Neben der Grundidee methodischen Vorgehens werden die Methodenwahl für die Modellierung und die interne Repräsentation diskutiert und Überlegungen zur Softwarearchitektur angestellt.

Kapitel 4 zeigt einen kurzen Überblick über verwendete und alternative Methoden. Der Schwerpunkt liegt hierbei in der Darstellung der Grundlagen eines logikbasierten Ansatzes mit Constraints, nämlich Answer Set Programming mit Gewichten. Ergänzend werden einige Grundlagen zu einem pfadbasierten Ansatz, nämlich XML mit XPath/XQuery beleuchtet.

Im Anschluss daran wird in Kapitel 5 das zentrale Modell definiert. Die Elemente und Komponenten der Modellierung für die beiden betrachteten Anwendungsbereiche werden festgelegt und es erfolgt eine Definition der Syntax und Semantik betrachteter Constraints. Schließlich wird die Modellierung der gewählten Probleminstanzen vorgenommen, wobei die Gemeinsamkeiten der Lösungen herausgearbeitet werden.

In Kapitel 6 wird eine musterbasierte Transformation für den favorisierten logikbasierten Ansatz vorgenommen. Darüber hinaus werden Überlegungen zur Transformation in den schon erwähnten pfadbasierten Ansatz angestellt.

Schließlich erfolgt in Kapitel 7 eine ausführliche Bewertung des eigenen Ansatzes nach verschiedenen Gesichtspunkten, wobei sowohl funktionale als auch nichtfunktionale Anforderungen und Zielkriterien eine Rolle spielen.

Abgeschlossen wird die Arbeit in Kapitel 8 mit einem Resumee und Ausblick.

# Kapitel 2

# Problembeschreibung

## 2.1 Grundstruktur der Problemstellung

Einhergehend mit der zunehmenden Bedeutung der Bildung für den individuellen, sozialen und gesellschaftlichen Erfolg wächst – sowohl im akademischen als auch im industriellen Kontext – die Notwendigkeit einer ständigen Weiterentwicklung der Lern- und Bildungsangebote.

Im Rahmen des Bologna-Prozesses [Eura, Eurb, Eurf] und durch die zunehmende Internationalisierung der akademischen Bildung wurden und werden an europäischen Hochschulen viele neue Studiengänge eingerichtet. Zahlreiche komplexe Bedingungen bestimmen die Möglichkeiten für ein gemäß der Studienordnung gültiges Studium. Sowohl für die Studienberatung als auch für die Prüfung der Adäquatheit (Konsistenz und Anwendbarkeit) von Studienordnungen sind entsprechende Dienste erforderlich.

Auch in der Industrie spielen Weiterbildungsmaßnahmen und Tests eine zunehmende Rolle. Durch entsprechende Trainingsprogramme kann sich das Personal weiter bilden und sich so für veränderte Anforderungen qualifizieren. Durch einschlägige (Eingangs-)Tests lässt sich der Kenntnisstand von Bewerbern und Mitarbeitern überprüfen. Je nach Art und Zweck der Prüfung[1] wird diese auf der Grundlage von gewissen Anforderungen und Bedingungen von Lehrern und Trainern erstellt. Teilweise werden sie hierbei von sogenannten Prüfungsmanagementsystemen zur automatischen Generierung dieser Tests unterstützt.

Bekanntlich gilt: Liegt ein Regelwerk nur informell vor, so bedarf es umfangreichen Expertenwissens, um Anfragen an die Wissensbasis beantworten zu können. Dies geschieht durch direkte Kommunikation zwischen Laien und Experten. Eine Anfrage des Laien kann der Experte aufgrund seines

[1] Die Bezeichung „Prüfung“ suggeriert, dass eine Benotung der Arbeit des Prüflings erforderlich ist; im Gegensatz dazu werden Tests nicht notwendigerweise bewertet, sondern können auch einfach zu Übungszwecken eingesetzt werden. Eine Unterscheidung nach dem hier angesprochenen Verwendungszweck ist im Rahmen dieser Arbeit nicht bedeutsam. Daher werden diese beiden Begriffe „Prüfung“ und „Test“ synonym verwendet.

erworbenen Wissens über das informelle Regelwerk oder durch dessen zusätzliches Studium – mit möglicherweise vorherigen Rückfragen an den Laien – beantworten. Diese Situation wird in Abbildung 2.1 dargestellt. In der Regel sind jedoch diese Beratungsvorgänge sehr zeit- und kostenintensiv.

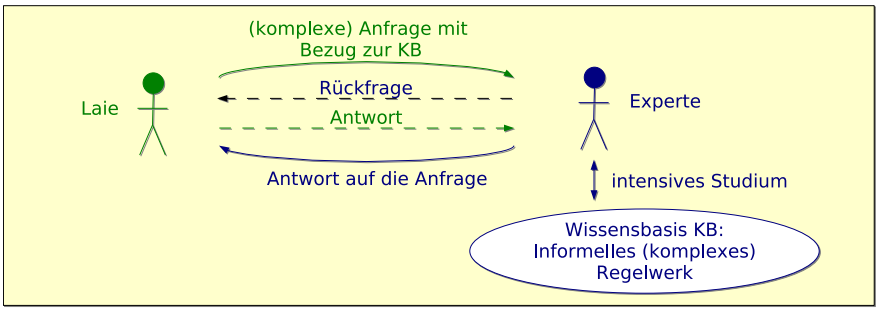



Abbildung 2.1: Grundstruktur der Problemstellung: Ausgangssituation

Moderne, umfassende Informationssysteme können den Experten unterstützen oder ihn in manchen Fällen sogar ersetzen. Ein Laie, der Kompetenzen in der Bedienung des Informationssystems besitzt, stellt an dieses über eine Bedienschnittstelle eine Anfrage. Nach möglicher weiterer Interaktion liefert das System – unter Zuhilfenahme eines Mechanismus zur automatisierten und intelligenten Berechnung („Reasoning-Mechanismus“) – die passende Antwort. In Abbildung 2.2 wird die gewünschte Zielsituation umrissen.

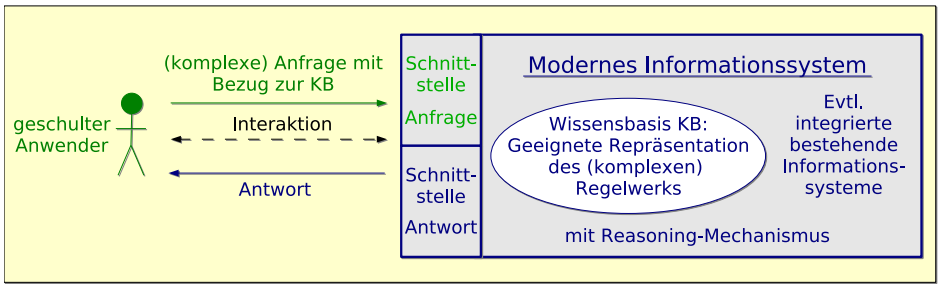



Abbildung 2.2: Grundstruktur der Problemstellung: gewünschte Zielsituation

Der Einsatz eines modernen Informationssystems kann folgende Vorteile bringen:

- wesentliche Erhöhung und Erweiterung des Informationsangebotes in Qualität und Quantität,
- beschleunigte Anfragebearbeitung,
- Schonung oder Einsparung personeller Ressourcen und

- wesentliche Kostenersparnis.

In obigen Beispielen ist ein System zur Studienberatung und Konsistenzprüfung von Studienordnungen, kurz StARC-System[2] genannt, bzw. ein modernes Testmanagement-System zur Behandlung komplexer Anfragen und Prüfung komplexer Konsistenzbedingungen wünschenswert. Hierbei spielen neben den strukturellen Gegebenheiten der Domänen vor allem auch die auftretenden Bedingungen und Anforderungen in Form von Constraints eine zentrale Rolle.

Zu bemerken ist ferner, dass im Hinblick auf elektronische Daten und Informationssysteme schon eine gewisse Infrastruktur vorliegen kann. In nachfolgenden Ausführungen soll dies allerdings keine bzw. nur periphär eine Rolle spielen.

Im Rahmen dieser Arbeit werden zwei Anwendungsbereiche betrachtet, nämlich die „Generierung von Tests“ (Kürzel: GT) und die „Studienberatung und Konsistenzprüfung von Studienordnungen“ (Kürzel: SO). Vor der informellen Beschreibung dieser Anwendungsbereiche in den Abschnitten 2.3 und 2.4 erfolgt eine Definition der Ziele dieser Arbeit und des eigenen Beitrags.

## 2.2 Eigener Beitrag

In der vorliegenden Arbeit wird ein methodisch-technisches Framework zur Behandlung von Planungsproblemen in strukturierten Domänen anhand konkreter Anwendungsbereiche präsentiert. Hierdurch wird für die beiden Anwendungsbereiche „Generierung von Tests“ und „Studienberatung und Konsistenzprüfung von Studienordnungen“ eine Methodik für den Übergang der skizzierten Situation von Abbildung 2.1 nach Abbildung 2.2 definiert.

Die wesentlichen innovativen Aspekte im Rahmen des vorgestellten Frameworks sind folgende:

1. In der vorliegenden Arbeit wird ein modellgetriebener Ansatz verfolgt. Als ein zentraler Aspekt der Problemlösung erfolgt die Entwicklung und Bereitstellung eines geeigneten Modells, welches sowohl die Schnittstelle zur Bedienoberfläche als auch zur internen Repräsentation darstellt. Es bildet daher die Basis sowohl für die Definition einer formalen benutzernahen Spezifikationssprache als auch für die Transformationen in interne Formalismen.

2. Gängige Datenmodelle und Anfragesprachen für strukturierte Domänen wie XML+XPath reichen für die Modellierung von komplexen Planungsproblemen nicht aus. Als ein zentraler Bestandteil der vorgestellten Modellierungsmethodik werden im Constraintmodell Modellierungselemente definiert, welche gängige Pfadanfragen in Anfragesprachen für baum- bzw. netzartige Datenmodelle wie XML+XPath formalisieren und verallgemeinern. Jene werden hierdurch für allgemeine Constraintaufgaben nutzbar, wobei komplexe Anfragen in einfacher und kompakter Weise abgebildet werden können.

[2] StARC-System: „**St**udy **A**dvice **R**egulation **C**onsistencychecking System“

3. Darüber hinaus wird „Answer Set Programming mit Gewichten“[3] als ein adäquater Formalismus für die interne Repräsentation festgelegt und eine Transformationsvorschrift vom Modell zur internen Repräsentation definiert. Hierdurch erfolgt die Nutzbarmachung und der Nachweis der Relevanz eines nicht gängigen Spezifikations- und Ausführungsformalismus, gerade auch im Vergleich mit modernen Sprachen.

4. Evaluation[4], prototypische Implementierungen (Demonstratoren) und Benchmarks liefern den Nachweis der Praktikabilität und Relevanz des eigenen Ansatzes.

Detailliertere Ausführungen bezüglich der Bewertung des eigenen Ansatzes und Erläuterungen hinsichtlich der innovativen Aspekte des vorgestellten Frameworks finden sich in den Kapiteln 7 und 8.

## 2.3 Generierung von Tests

Wird in herkömmlicher Weise ein Test konzipiert, so müssen zunächst die einzelnen Aufgaben oder Fragen erstellt werden. Hierbei orientiert sich der Autor an den Kriterien und Anforderungen, die an den Test gestellt werden – wie etwa abzudeckende Themen, der geforderte Schwierigkeitsgrad und die Gesamtbearbeitungsdauer. In der Regel ist dies ein sehr arbeits- und zeitintensiver Vorgang. Genügt der Testentwurf den Anforderungen, so kann er in eine druckreife Form gebracht und schließlich gedruckt werden. Ist dies jedoch nicht der Fall, so muss gegebenenfalls ein Teil der Fragen verworfen und neue Aufgaben erstellt werden. Dieser Prozess ist in Abbildung 2.3 dargestellt.

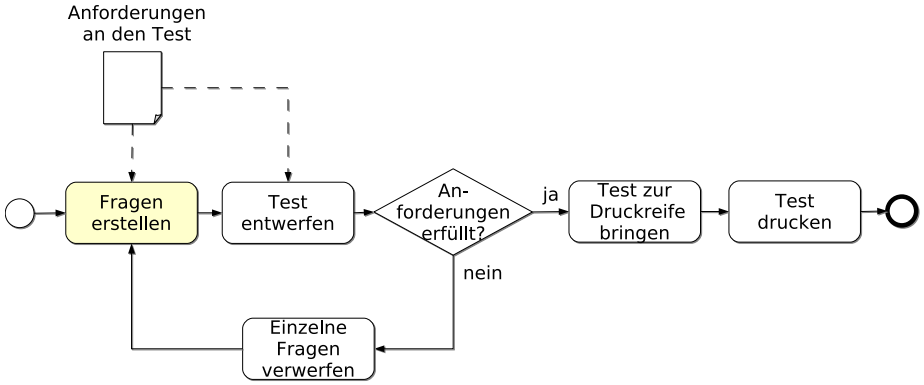



Abbildung 2.3: Prozess der Testerstellung mit Fragenerstellung

[3] Vgl. Abschnitt 4.2
[4] Vgl. auch Kapitel 7

Möglicherweise wurde schon mehrmals ein Test bestimmter Art abgehalten. Hat nun ein Prüfer Zugang zu den darin enthaltenen Aufgaben und Fragen, so kann er auf diesen Fragenbestand bei erneuter Prüfungserstellung zurückgreifen. Hierbei muss er versuchen, die Fragen so auszuwählen, dass sämliche Anforderungen an den Test erfüllt werden. Sind die Bedingungen sehr komplex, so bedarf dies in der Regel einer wiederholten Revision der Auswahl, was ebenso – wenn es „per Hand“ durchgeführt wird – sehr arbeits- und zeitintensiv ist (vgl. Abbildung 2.4).

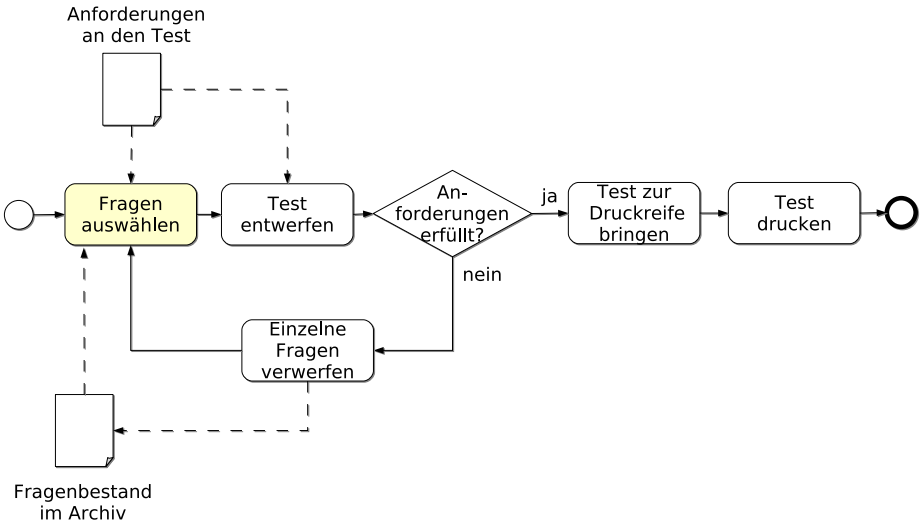



Abbildung 2.4: Prozess der Testerstellung bei vorhandenem Fragenpool

In der Regel möchte der Prüfer vermeiden, dass sich Fragen in einem Test ähneln. Möchte man darüber hinaus verhindern, dass der gleiche Test nochmals erstellt wird, so hat man bei der Aufgabenauswahl auch darauf zu achten, dass eine entsprechende Variation stattfindet. Ähnliche Fragen und ähnliche Tests – Kriterien für die „Ähnlichkeit“ sind jeweils vorher festzulegen – sollten auf diese Weise vermieden werden.

Nun ist sowohl in der industriellen Praxis als auch im universitären Umfeld eine zunehmende Bedeutung verschiedener Aspekte des e-learning zu beobachten. Assessment Tests gehören mittlerweile zu den Standardmethoden, um das Wissen und die Fähigkeiten von Bewerbern und Mitarbeitern festzustellen. Manche Unternehmen nutzen mittlerweile sogar ausschließlich elektronische Medien für Qualifizierungsprogramme und Prüfungen. Hierbei liegen die Testfragen in einem sogenannten Fragenpool in elektronischer Form vor. Nach Angabe der Anforderungen an die Prüfung erfolgt die Fragenauswahl automatisiert. Diese sogenannten Testmanagement-Systeme wie beispielsweise [IFI, ExB, Dea] unterstützen also den Trainer bei der Zusammenstellung von Übungen und Tests.

### 2.3.1 Anwendungsfälle für ein Testmanagement-System

Typischerweise können bei der Generierung von Tests folgende Anwendungsfälle auftreten:

1. *Verändern des Fragenbestandes:*
   - Ein Autor erstellt neue Aufgaben und Fragen und ordnet diesen bestimmte Eigenschaften zu.
   - Bestehende Fragen bzw. deren Eigenschaften werden geringfügig verändert.
   - Veraltete Fragen werden gelöscht.
2. *Zusammenstellen von Tests (Generieren, Checken):*
   - Ein Trainer oder Prüfer konzipiert eine neue Prüfung nach bestimmten Kriterien.
   - Ein Lerner erstellt sich zu Übungszwecken einen (beliebigen) Test.
   - Ein Trainer möchte wissen, ob ein vorliegender Testentwurf den geforderten Anforderungen genügt.
   - Für einen bestimmten Prüfungstyp wird eine Spezifikation von Anforderungen durch das Prüfungskomitee definiert. Es stellt sich die Frage, ob die Anforderungen erfüllbar sind.
3. *Administration des Systems:*
   - Ein Administrator führt bei konzeptuellen Veränderungen im Datenbestand oder den Anforderungen geeignete Anpassungen im System durch.

Neben diesen für das informatische Kernproblem der „Generierung von Tests“ relevanten Anwendungsfällen spielen im Umfeld der Verwendung eines Testmanagement-Systems möglicherweise noch folgende Aspekte eine Rolle:

- *Form, Durchführung und Auswertung einer Prüfung:*
  - Die Prüfung wird entweder in eine geeignete elektronische oder in eine druckbare Form gebracht.
  - Ein Lerner führt online einen Test durch und erhält Übungshinweise.
  - Ein Prüfling bearbeitet eine Prüfung.
  - Zu einer Prüfung wird eine Korrekturschablone erstellt.
  - Eine Prüfung wird (evtl. automatisiert) korrigiert.
  - Das Ergebnis eines Tests wird ausgewertet und eine Statistik erstellt.

Natürlich könnte diese Liste auch noch weiter ausgebaut werden, wie beispielsweise um die Erstellung interaktiver Übungsformen und die Verknüpfung von Inhalten mit Lernsoftware. Da sich die vorliegende Arbeit auf die infomatischen Kernfragestellungen im Rahmen dieses Anwendungsbereiches konzentriert, werden die Aspekte der Durchführung und Auswertung nicht vertieft behandelt.

Die Grundlage eines Testmanagement-Systems bildet also ein „intelligentes“ System, in welchem ein großer Pool von Aufgaben und Fragen geeignet verwaltet werden kann. Neben der Administration spielen prinzipiell zwei Hauptaspekte bei dessen Verwendung eine Rolle: Auf der einen Seite müssen die Fragen in das System gebracht, also erstellt werden; auf der anderen Seite kann sich ein Prüfer (evtl. auch ein Lerner) mit Hilfe des Systems einen Test generieren lassen, in der die im Pool vorhandenen Fragen und die entsprechenden Anforderungen an den Test Berücksichtigung finden. Dieser Sachverhalt ist in Abbildung 2.5 skizziert.

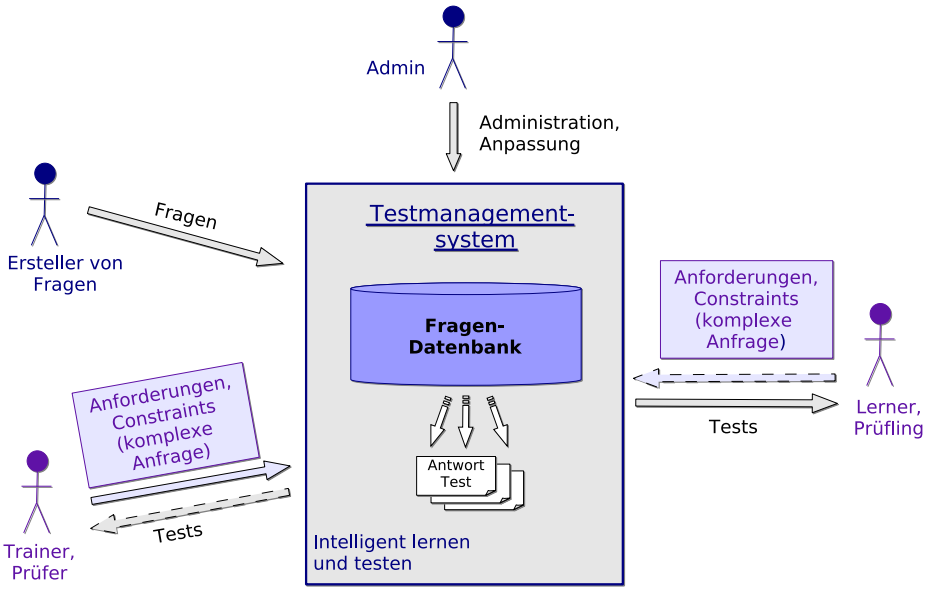



Abbildung 2.5: Testmanagement-System und Benutzerrollen

### 2.3.2 Funktionale, systembezogene Anforderungen

Unter Berücksichtigung der relevanten Anwendungsfälle ergeben sich folgende funktionale, systembezogene Anforderungen an ein Testmanagement-System:

1. *Geeignete Repräsentation und Speicherung der Daten, insbesondere der Fragen und ihrer Eigenschaften*, wobei update-, insert- und delete-Vorgänge möglich sein sollen.

2. *Repräsentation der Anforderungen an einen Test*, wobei diese sich auf den Fragenbestand und deren Eigenschaften beziehen. Eine breite Variation der Anforderungen an die Prüfung soll möglich sein.
3. *Bearbeitung der Anfragen* durch eine „intelligente“ Engine. Hierbei sollen alle bzw. beliebige Lösungen gefunden werden können.
4. *Ausgabe der Anfrageergebnisse*, vor allem der Tests.
5. *Geeignete Bedienschnittstellen* sowohl für die Autoren der Fragen als auch für die Prüfer und die Lerner.[5]

Hierbei sind auch folgende Aspekte zu beachten:

6. *Berücksichtigung des generierenden Aspekts (Modelgenerating, Planen, Suchproblem)*
7. *Berücksichtigung eines prüfenden Aspekts (Modeltesting)*
8. *Umgang mit loser Spezifikation und vorgegebenen Teillösungen (lose Spezifikation, Teillösungen)*
9. *Beachtung der strukturierten Domäne (Strukturproblem)*

### 2.3.3 Nichtfunktionale Anforderungen

Neben den allgemeinen nichtfunktionalen Anforderungen für beide Anwendungsbereiche, welche im Abschnitt 2.5 aufgezeigt werden, können noch folgende notiert werden:

- Für die interaktive Bearbeitung der Anfragen sind Laufzeiten von unter einer halben Sekunde wünschenswert. Falls es nicht erforderlich ist, dass das System als Just-in-Time-Applikation läuft, so sind Laufzeiten im Minutenbereich akzeptabel.[6]
- Datenmengen im Bereich von 100 bis 3000 Fragen sollten möglich sein.[7]
- Normale Verfügbarkeit des Systems genügt.[8]

[5] Die Entwicklung der graphischen Bedienoberflächen wird im Rahmen dieser Arbeit nicht behandelt, jedoch die Schnittstellen hierzu.

[6] Laufzeiten im Minutenbereich stellen in dem beschriebenen Fall immer noch eine erhebliche Zeitersparnis im Vergleich zu einer Testerstellung „per Hand“ dar.

[7] Realistische industrielle Fragenpools besitzen etwa 2000 bis 3000 Fragen. Akademische Fragenpools sind eher kleiner, dafür aber spezieller. Zu bemerken ist ferner, dass in der Regel große Fragenpools in kleinere disjunkte Teilmengen aufgeteilt werden können.

[8] In der Regel sind die Anzahl der Anfragen pro Tag nicht besonders groß, so dass ein zwischenzeitlicher Ausfall des Systems normalerweise keinen großen Schaden anrichtet.

- Bei Inkonsistenzen sollten aussagekräftige Fehlermeldungen für den Testersteller ausgegeben werden.
- Die Spezifikation der Anforderungen sollte möglichst deklarativ und leicht verständlich sein.[9]

### 2.3.4 Kernaspekte

Bei der Erstellung einer Testfrage erhält diese einen eindeutigen Identifier, etwa einen Namen oder eine Nummer, und neben dem Fragentext sind auch mögliche Antworten zu spezifizieren. Üblicherweise können die Fragen durch eine Art Fragen-Browser graphisch visualisiert werden. Es gibt eine große Bandbreite von Fragenarten, die ein gewisses Maß an Benutzerinteraktion zulassen, beginnend bei einfachen Single-Choice-Fragen über Pick-and-Place-Fragen zu Multimedia-Fragen. Darüber hinaus sind die Fragen durch Eigenschaften wie `Thema` (mögliche Werte z. B. *angewandte Geographie*, *Politik*), `Schwierigkeitsgrad` (z. B. *leicht*, *mittel*, *schwer*) und `Fragenart` (z. B. *single-choice*, *multiple-choice*, *pick-and-place*) und `Bearbeitungsdauer` (z. B. *1*, *5*, *10*) näher zu spezifizieren. Jeder Frage ist mindestens ein Eigenschaftswert-Vektor zugeordnet. Möglicherweise wird den Eigenschaftskategorien durch formalisierte Ontologien, wie zum Beispiel durch eine Themen-Hierarchie, mehr Struktur verliehen.

**Beispiel 2.3.1 (Frage)** Das Beispiel einer Frage ist in Abbildung 2.6 gezeigt. Dieser Frage mit dem möglichen Identifier „frage374“ und dem Fragentext „Welche Stadt ist die Hauptstadt von Deutschland?“ sind die falschen Antwortmöglichkeiten „Bonn“, „Potsdam“ und „Wien“ und die richtige Antwortmöglichkeit „Berlin“ zugeordnet. Der Attributwert des Attributs `Thema` könnte durch *angewandte Geographie* festgelegt sein. Der `Schwierigkeitsgrad` könnte durch *leicht* und die `Bearbeitungsdauer` durch *1 min* gegeben sein.

Für die interne Repräsentation einer Frage gibt es verschiedene Ansätze. Ein in diesem Zusammenhang oft verwendeter Standard ist der IMS-QTI-Standard [IMS].

**Beispiel 2.3.2 (Interne Repräsentation von Fragen)** In der Auflistung 2.1 wird ein möglicher Ausschnitt einer XML-Repräsentation der Frage aus dem Beispiel 2.3.1 gezeigt. Um die Darstellung übersichtlich zu halten, wird hier eine naive Repräsentation gewählt. In diesem Fall ist es nicht möglich, einer Frage – je nach Kontext – unterschiedliche Schwierigkeitsgrade zuzuordnen, was in der Realität durchaus sinnvoll wäre. Ähnliches gilt für die Zuordnung zu Themen bzw. Themenhierarchien.

[9] Aus Gründen der Benutzerfreundlichkeit wünscht man sich eine Spezifikationsmöglichkeit, die nahe an der informellen Beschreibung des Anwendungsbereiches liegt. Dies spricht für einen deklarativen Ansatz.

**Identifier:** frage374

Thema: angewandte Geographie
Schwierigkeitsgrad: leicht
Bearbeitungsdauer: 1 min

Fragetext:
*Welche Stadt ist die Hauptstadt von Deutschland?*

Falsche Antwortmöglichkeiten:
Bonn
Potsdam
Wien
Richtige Antwortmöglichkeit:
Berlin

Abbildung 2.6: Beschreibung einer Frage

```
<assessmentItem ...>
   <identifier>frage374</identifier>
   <properties>
      <topics>
         <topic>angewandte Geographie</topic>
      </topics>
      <difficulty>leicht</difficulty>
      <questiontype>single-choice</questiontype>
      <duration>1</duration>
   </properties>
   ...
   <itemBody>
      <prompt>
         Welche Stadt ist die Hauptstadt von Deutschland?
      </prompt>
      <simpleChoice identifier="Bo">Bonn</simpleChoice>
      <simpleChoice identifier="P">Potsdam</simpleChoice>
      <simpleChoice identifier="B">Berlin</simpleChoice>
      <simpleChoice identifier="W">Wien</simpleChoice>
   </itemBody>
   ...
   <responseDeclaration ...>
      <correctResponse>
         <value>B</value>
      </correctResponse>
      ...
   </responseDeclaration>
</assessmentItem>
```

Listing 2.1: Vereinfachte XML-Repräsentation der Frage aus Beispiel 2.3.1

**Bemerkung 2.3.3** Für die Untersuchungen des zu lösenden informatischen Problems spielt die eigentliche Ausprägung der Frage, also der Fragentext und mögliche Antworten, keine Rolle, da es unabhängig davon zu lösen ist. Deshalb wird von nun an hiervon abstrahiert. Der eindeutige Name der Frage dient als Bezeichner und zur Identifikation.

Typischerweise werden an die Zusammensetzung der Fragen in einem Test bestimmte Anforderungen gestellt. Diese können zum Beispiel durch Bestimmungen aus der Personalabteilung für interne Trainings oder von didaktischen Überlegungen der Trainer beeinflusst werden. Beispielsweise könnte eine Mindesbearbeitungsdauer der Prüfung oder eine maximale Anzahl von Fragen zu einem bestimmten Thema vorgegeben werden.

**Bemerkung 2.3.4** Nach Bemerkung 2.3.3 spielt für die weiteren Überlegungen der Fragentext keine Rolle. Sollte ein Prüfer eine Anforderung in der Form stellen wollen, dass in der Prüfung Fragen mit bestimmten Schlüsselwörtern vorkommen sollen, so kann dies mit Hilfe einer zusätzlichen Suchdimension `Schlüsselwort` modelliert werden. Die in Bemerkung 2.3.3 getroffene Vereinbarung stellt also keine Einschränkung der Allgemeinheit dar.

**Bemerkung 2.3.5 (Gültige Prüfung und Auswahl)** Jede Auswahl von Fragen aus dem Fragenpool, die die vom Prüfer gestellten Anforderungen erfüllt, stellt eine sogenannte „gültige Prüfung“ dar. In der Regel wird von der Menge der gültigen Prüfungen nur eine beliebig gewählte benötigt. Die in ihr enthaltenen Fragen nennen wir die „gewählten Fragen der Prüfung“ bzw. oft nur kurz „gewählte Fragen“. In diesem Zusammenhang sprechen wir auch vom „Wählen von Fragen“.

**Definition 2.3.6 (Kernaspekte „Generierung von Tests“)** Die nachfolgenden Aspekte des Anwendungsbereichs „Generierung von Tests“ sind für die weiteren Untersuchungen relevant:

- **Fragenkatalog, Fragenpool:** eine Menge von Fragen mit eindeutigen Identifiers
- **Beschreibung (Eigenschaften) der Fragen:** Attributnamen (bezeichnen die Kategorien der Suchdimensionen) und Attributwerte (bestimmen die Beschreibung der einzelnen Frage)
- **Benutzerdefinierte Anforderungen:** müssen von einer Prüfung erfüllt werden
- **Test, Prüfung:** Menge von Fragen aus dem Fragenpool in Übereinstimmung mit den benutzerdefinierten Anforderungen.

Zu bemerken ist, dass existierende industrielle Lösungen für Testmanagement- oder ähnliche Systeme wie beispielsweise [Exa, TeB, Okt, Dea, ExB, IFI] je nach Einsatzbereich Einschränkungen unterliegen: Es gibt oftmals nur wenige, festgelegte Spezifikations-Parameter, so dass die Möglichkeiten für

die Konfiguration eines Tests oft sehr gering, wenn überhaupt vorhanden, sind. Auch der Einsatzbereich unterliegt möglicherweise starken Einschränkungen. Darüber hinaus sind die zugrunde liegenden Algorithmen teilweise nicht vollständig, d. h. es können nicht alle Lösungen einer gewissen Spezifikation gefunden werden. Zu bemerken ist, dass laut [IPR05] für das System [ExB] eine ASP-Spezifikation[10] zugrunde liegt. Die Autoren verwenden mit dem Reasoner DLV [CFL+01] einen disjunktiven Ansatz, um die logischen Regeln zu repräsentieren. Die dort angegebene Spezifikation ist jedoch nicht für strukturierte Domänen gedacht bzw. geeignet.

Diese Systeme lösen daher das Kernproblem der Generierung von Tests aus einer Menge von Fragen, wobei alle benutzerdefinierten Anforderungen Berücksichtigung finden, nicht zufriedenstellend, was die Entwicklung eines solchen Testmanagement-Systems wünschenswert macht.

## 2.4 Studienberatung und Konsistenzprüfung von Studienordnungen

In Lissabon hat im März 2000 der Europäische Rat das Ziel aufgestellt [Eure, Eurd], die Europäische Union zum „dynamischsten wissensbasierten Wirtschaftsraum“ zu machen – einem „Wirtschaftsraum, der fähig ist, ein dauerhaftes Wirtschaftswachstum mit mehr und besseren Arbeitsplätzen und einem größeren sozialen Zusammenhalt zu erzielen“. Um dieses Ziel zu erreichen, haben die Staats- und Regierungschefs im Jahr 2002 festgelegt, dass Europa bis zum Jahr 2010 im Bezug auf die Qualität der Bildungssysteme weltweit führend sein sollen. Hierzu soll die allgemeine und berufliche Bildung in Europa grundsätzlich umgestaltet werden. Durch die Initiative „Allgemeine und berufliche Bildung 2010“ werden alle Aktionen im Bildungsbereich auf europäischer Ebene integriert.

Hierzu ist auch der sogenannte Bologna-Prozess zu zählen, der für den Hochschulbereich eine wichtige Rolle spielt: In Bologna haben sich 1999 insgesamt 29 europäische Nationen das Ziel gesetzt, einen europäischen Hochschulraum zu schaffen [Eura, Eurb, Eurf]. Seither befindet sich Europa inmitten einer bedeutenden Hochschulreform. Zu den Schwerpunkten dieses sogenannten Bologna-Prozesses gehören

- die Einführung eines Systems *leicht verständlicher und vergleichbarer* Hochschulabschlüsse,
- die Schaffung eines *zweistufigen Systems von Studienabschlüssen* (konsekutive Studiengänge),
- die Einführung eines Leistungspunktesystems, dem *European Credit Transfer System (ECTS)*
- die Einführung einer *Modularisierung*,
- die Förderung der europäischen Dimension der Hochschulausbildung und
- die Unterstützung der Mobilität der Studierenden, Lehrenden und Forschenden.

[10] ASP: Answer Set Programming

In Deutschland wurden konsekutive Studiengänge als Bachelor und Master realisiert. Der Bachelor soll bereits nach drei bis vier Jahren zu einem berufsfähigen Abschluss führen, so dass früher als bisher ein Berufseinstieg möglich ist. Durch den zweiten Zyklus wird ein höherer Abschluss erreicht. Nimmt man die Promotion in die Betrachtungen mit dazu, so liegt ein dreigliedriges System vor.

Entscheidend bei dieser Entwicklung ist die Reform der Studieninhalte und damit eine Schaffung neuer und besser strukturierter Curricula. Im Zuge dieser Entwicklungen wurden und werden eine Vielzahl neuer Studiengänge mit neuen Studienordnungen eingerichtet. Die neuen Studiengänge werden in Modulen angeboten und enthalten in der Regel studienbegleitende Prüfungen und ein Leistungspunktesystem.

Das Europäische System zur Übertragung und Akkumulierung von Studienleistungen, ECTS [Eurc], hat als Grundlage das *Arbeitspensum*, das die Studierenden für ein erfolgreiches Studium (Lernergebnisse, Kompetenzen) absolvieren müssen. Hierbei werden den Bildungskomponenten eines Studiengangs, beipielsweise den Veranstaltungen, Kursen, Praktika, Abschlussarbeiten und Modulen, bestimmte Credits zugeordnet, die ein Maß für den Umfang der Lernprogramme darstellen. Es wurde die Übereinkunft getroffen, dass das *Arbeitspensum* von Vollzeitstudierenden während eines akademischen Jahres 60 ECTS-Punkte ergibt.

In der Regel sind die Bildungskomponenten eines Studiengangs in Form eines sogenannten Modulkatalogs strukturiert. Diese werden in der entsprechenden Studienordnung beschrieben. In dieser sind außerdem eine Vielzahl von Bedingungen festgelegt, die für ein gültiges Studium erfüllt werden müssen. Typische Anforderungen sind beispielsweise: „aus Modulgruppe A sind zwischen 5 und 7 Veranstaltungen mit mindestens 37 ECTS-Punkten zu belegen“ oder „aus Modulgruppe B ist eine Fächergruppe als Schwerpunktfächergruppe zu wählen; in dieser sind drei Basismodule und zwei Prüfungsmodule zu absolvieren“. Darüber hinaus sind Abhängigkeiten zwischen den Bildungskomponenten möglich. So können beispielsweise bestimmte Module andere Module zur Voraussetzung haben. Prinzipiell unterscheiden sich die Studienordnungen stark in der Wahlfreiheit: Manche lassen den Studierenden kaum, manche große Freiheit in der Auswahl von Studieneinheiten und Bildungskomponenten. Abbildung 2.7 zeigt einen Ausschnitt aus einer Studien- und Prüfungsordnung eines möglichen Studiengangs.[11]

Die Studierenden müssen nun ihren Studienverlauf gemäß der Studienordnung planen. Hierzu haben sie die Veranstaltungen und Kurse gemäß den geforderten Anforderungen zu wählen. Besonders bei sehr großen Variationsmöglichkeiten und individuellen Studienverläufen ist eine gute und umfassende Studienberatung unumgänglich. Auch die Universitäten sind durch diese Hochschulreform in die Pflicht genommen. Sie haben ihr Lehrangebot nach den Anforderungen der Studienordnung zu richten – im Rahmen ihrer personellen und finanziellen Ressourcen. Eine gute Planung ist unumgänglich, was geeignete Beratungsdienste wünschenswert macht. Viele Universitäten befinden sich inmitten des Prozesses der Einführung neuer Studiengänge. Neue Studienordnungen sind zu erstellen und alte teilweise zu verbessern und zu revidieren. Die Komitees zur Erstellung von Studienordnungen benötigen hierzu eine entsprechende Unterstützung. Wie in Abschnitt 2.1 schon angedeutet, sind allerdings

[11] Dieses Beispiel lehnt sich an den Studiengang European Studies der Universität Passau (vgl. [SoP]) an.

**STUDIEN- UND PRÜFUNGSORDNUNG INTERNATIONAL STUDIES**

**I. Allgemeine Bestimmungen**

...

**II. Besondere Bestimmungen über die einzelnen Modulgruppen (Modulkatalog)**

Abkürzungen: V = Vorlesung, PS = Proseminar, HS = Hauptseminar, WÜ = wiss. Übung,
Ü = sprachpraktische Übung,
SWS = Semesterwochenstunde, ECTS = ECTS-Leistungspunkt

**§ 17 Modulgruppe A: Internationale Basismodule**

| **(1) Basismodul Europäische Kulturwissenschaft** | | **SWS** | **ECTS** |
|---|---|---|---|
| V | Europäische Kulturwissenschaft | 3 | 5 |
| PS/WÜ | Europäische Kulturwissenschaft | 2/2 | 2/2 |

| **(2) Basismodul Internationales Recht** | | **SWS** | **ECTS** |
|---|---|---|---|
| V | Grundzüge des internationalen Rechts | 2 | 3 |
| V | Verfassungsrecht | 2 | 2 |

**§ 18 Modulgruppe B: Internationale Schwerpunktmodule**

Eine der folgenden Fächergruppen ist als Schwerpunkt 1, die andere als Schwerpunkt 2 zu wählen. Der Student absolviert im Schwerpunkt 1 zwei Basis- und zwei Prüfungsmodule und im Schwerpunkt 2 zwei Basis- und ein Prüfungsmodul. In einem Prüfungsmodul des Schwerpunkts 1 ist ein Hauptseminar zu belegen.

Fächergruppe I: Historisch-sozialwissenschaftliche Fächer

Fach: Geographie

| **(1) Basismodul Angewandte Regionalforschung** | | **SWS** | **ECTS** |
|---|---|---|---|
| V | Regionale Geographie | 3 | 5 |
| WÜ | Geländepraktikum | 3 | 5 |

| **(2) Prüfungsmodul Geographische Methoden** | | **SWS** | **ECTS** |
|---|---|---|---|
| WÜ | Kartenkunde und -interpretation | 2 | 3 |
| WÜ | Geographische Informationssysteme | 2 | 3 |

...

Fach: Kunstgeschichte

| **(1) Basismodul Arbeiten am Original** | | **SWS** | **ECTS** |
|---|---|---|---|
| PS | Restaurierungsmethoden | 1 | 2 |
| WÜ/PS | Geländepraktikum | 3/3 | 5/5 |

| **(2) Prüfungsmodul Christliche Archäologie** | | **SWS** | **ECTS** |
|---|---|---|---|
| V | Christliche Archäologie | 2 | 5 |
| PS/HS | Christliche Archäologie | 2/2 | 5/5 |

...

Fächergruppe II: Fremdsprachliche Philologien und Kulturen

...

**§ 19 Modulgruppe C: Internationale Sprachmodule**

...

Abbildung 2.7: Ausschnitt aus einer möglichen Studien- und Prüfungsordnung

personelle Beratungsdienste und Experten in der Regel sehr teuer, wozu den Universitäten oftmals die finanziellen Mittel fehlen.

Aus diesem Grunde sind weitere Hilfmittel zu entwickeln. Insbesondere ist ein System zur Studienberatung der Studierenden, der Beratung der Studiendekane und zur Prüfung der Konsistenz und Anwendbarkeit von Studienbedingungen, kurz StARC-System[12], wünschenswert.

An den verschiedenen Universitäten ist die Infrastruktur bezüglich Informationen und Datenverwaltung unterschiedlich ausgebaut und automatisiert. In der Regel sind aber folgende Komponenten und Institutionen vorhanden:

- *Zentrale Studierendenverwaltung:* persönliche Daten der Studierenden (z. B. Name, Wohnort, Geburtsdatum, Matrikelnummer, Studiengang, Semester)
- *Prüfungssekretariat und Prüfungsverwaltung:* offizielle Anmeldung der Studierenden zu Prüfungen, Speicherung der Prüfungsergebnisse
- *Studien- und Prüfungsordungen:* legen die Bedingungen für ein korrektes und bestandenes Studium fest
- *Vorlesungsverzeichnis:* Auflistung der in einem konkreten Semester angebotenen Lehrveranstaltungen, als Buch oder elektronisch
- *Lernmanagement:* Skripten, Unterlagen und spezielle Hinweise zu konkreten Veranstaltungen von Dozenten für die Studierenden
- *Studienberatungsstelle:* Information und Beratung der Studierenden
- *Raum- und Zeitplanung(ssystem):* zur Planung der Veranstaltungen in einem bestimmten Semester

In diese Infrastukur ist das StARC-System zu integrieren, um eine flexiblere und umfangreichere Beratung zu gewährleisten. Der Bezug zu den einzelnen Komponenten der an der Universität vorhandenen Infrastruktur ist unterschiedlich stark. Ein zentrale Rolle für ein StARC-System spielen die Studienordnungen und auch die an der Universität angebotenen konkreten Lehrveranstaltungen (Vorlesungsverzeichnis). Im Rahmen der Studienberatung findet das System eine wichtige Anwendung. Es benötigt eventuell einige Daten der Studierenden und Informationen über Prüfungsergebnisse. Eine Ankopplung des StARC-Systems an das Lernmanagementsystem und die Raum- und Zeitplanung hat nicht zu erfolgen. Abbildung 2.8 deutet diese Sachverhalte an.

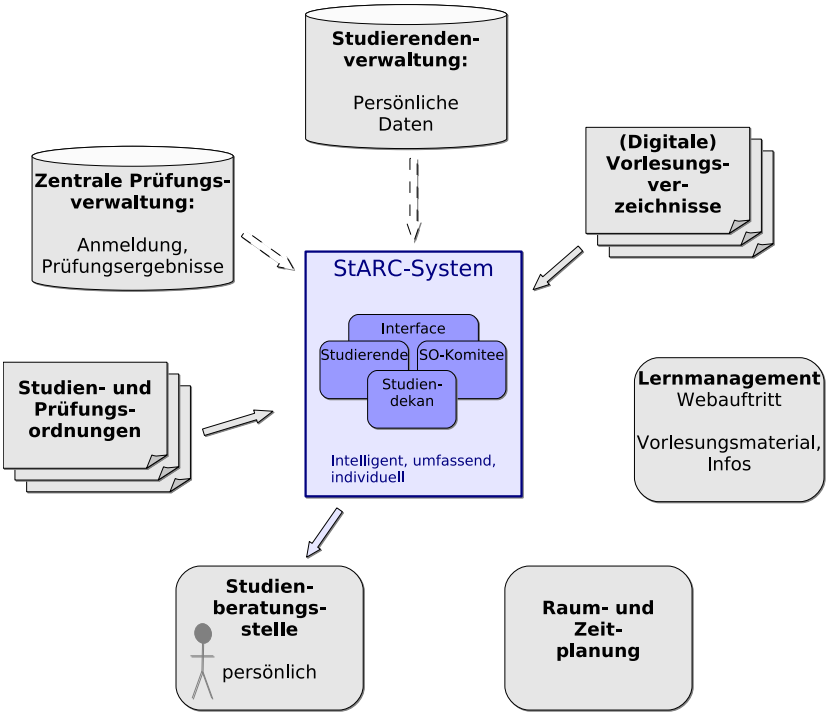



Abbildung 2.8: Einordnung des StARC-Systems in die vorhandene Infrastruktur der Universität

### 2.4.1 Anwendungsfälle für ein StARC-System

Für ein solches StARC-System sind nachfolgend beschriebene Anwendungsfälle, wie auch in den Abbildungen 2.9, 2.10 skizziert, besonders relevant:

Die Hauptbereiche der Anwendungsfälle sind die Konsistenzprüfung von Studienordnungen, die Studienberatung und die Beratung für den Studiendekan der Fakultät. Genauer zeigt dies folgende Übersicht und die Abbildung 2.10, in welchen die wesentlichen Anwendungsfälle aufgelistet sind:

- *Konsistenzprüfung von Studienordnungen*
  - An einer Universität soll ein neuer Studiengang eingeführt werden. Das Komitee zur Erstellung der Studienordnung hat verschiedene Module und Veranstaltungen zusammen mit Studienbedingungen definiert. Es möchte wissen, ob der aktuelle Vorschlag widerspruchsfrei ist, die Studienordnung also erfüllbar ist. Im negativen Fall sollen Hinweise auf mögliche Ursachen der Inkonsistenz angegeben werden.

[12] StARC-System: „**St**udy **A**dvice **R**egulation **C**onsistencychecking System“

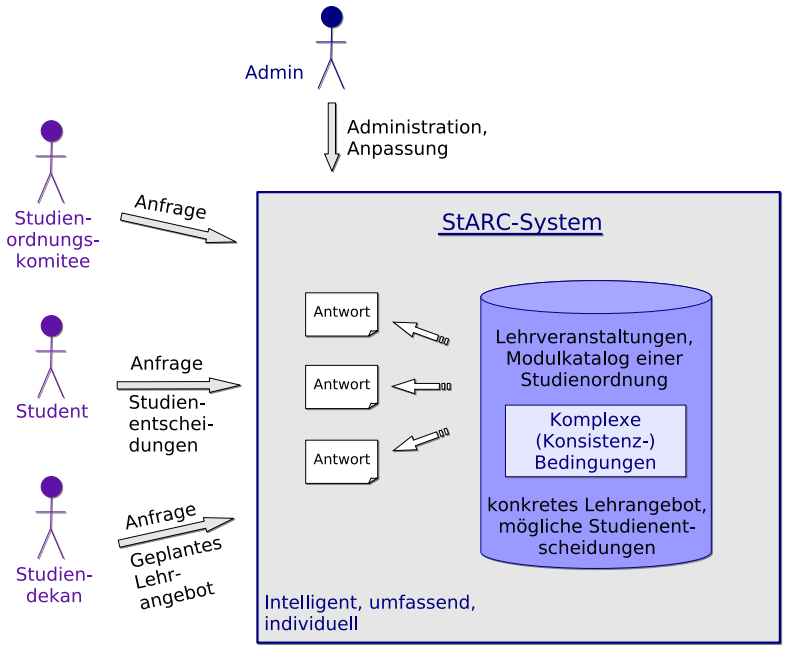



Abbildung 2.9: Studienordnungsberatungssystem und Benutzerrollen

- *Studienberatung*
  - Generierung sämtlicher Möglichkeiten für ein gültiges Studium:
    - Zu Beginn ihres Studiums möchte Studentin A wissen, welche Möglichkeiten sie hat, ihr Studium zu absolvieren. Sie möchte also wissen, welche Möglichkeiten sie hat, Lehrveranstaltungen, Fächer, Module usw. zu belegen.
  - Generierung sämtlicher Möglichkeiten für ein gültiges Studium unter bestimmten Voraussetzungen bzw. Entscheidungen:
    - Student B1 hat schon gewisse Kurse und Fächer belegt. Er möchte wissen, welche Möglichkeiten er hat, sein Studium zu beenden.
    - Studentin B2 hat schon gewisse Veranstaltungen belegt. Sie möchte wissen, für welche Fächer sie sich diese anrechnen lassen kann.
    - Student B3 hat sein Hauptfach gewählt. Er möchte wissen, welche Möglichkeiten er hat, zugehörige Veranstaltungen zu belegen.
  - Generierung der Möglichkeiten für ein gültiges Studium unter bestimmten Voraussetzungen bzw. „zeitlichen Aspekten“:
    - Student C1 beginnt gerade sein Studium und möchte wissen, welche Kurse er ohne Voraussetzungen besuchen kann.

    - Studentin C2 hat schon bestimmte Prüfungen bestanden und möchte wissen, welche Kurse sie damit als nächste besuchen dürfte.
  - Generierung der notwendigen Voraussetzungen für bestimmte Veranstaltungen („teilweise zeitlicher Aspekt“)
    - Student C3 möchte gerne die Veranstaltung „xy“ besuchen und daher wissen, welche Voraussetzungen er erfüllen muss, damit dies möglich ist.
  - Konsistenzprüfung von Studienentscheidungen:
    - Studentin D möchte gewisse Kurse, Module etc. belegen. Sie möchte wissen, ob diese Wahl mit der Studienordnung konform ist.

- *Beratung für den (die) Studiendekan(e) der Fakultät(en):*
  - Der Studiendekan E1 einer Fakultät möchte wissen, ob für einen bestimmten Studiengang das geplante Lehrangebot ausreichend ist, d. h. ob damit der Studiengang studierbar ist.
  - Die Studiendekanin E2 einer Fakultät möchte für ihre weiteren Planungen wissen, welche Möglichkeiten sie hat, konkrete Veranstaltungen in den nächsten Semestern anzubieten, damit ein bestimmter Studiengang an der Universität auch studierbar ist. Möglicherweise möchte sie sich auch einen Musterstudienplan erstellen lassen.

- *Weitere Aspekte:*
  - Ein Student möchte sich die eigenen Leistungen anzeigen lassen. Er fragt sich, welche Scheine er schon erworben hat.
  - Eine Studentin hat während eines Auslandsaufenthaltes Studienleistungen erbracht. Für welche Studienleistungen kann sie sich diese an einer bestimmten Universität (z. B. Passau) anerkennen lassen?

### 2.4.2 Funktionale, systembezogene Anforderungen

Es ergeben sich folgende funktionale, systembezogene Anforderungen an ein StARC-System:

1. *Repräsentation der Studienordnungen:* Verschiedene, in der Regel sehr komplexe, Studienordnungen müssen im System repräsentiert werden. Die wichtigsten Bestandteile sind:
   (a) *Veranstaltungen und Modulkatalog:* Die Veranstaltungen sind durch Eigenschaften wie ECTS-Punkte und Semesterwochenstunden beschrieben. Im sogenannten Modulkatalog sind sie hierarchisch durch Kategorien wie z. B. Modulgruppe, Fach, Modul und Veranstaltung strukturiert.

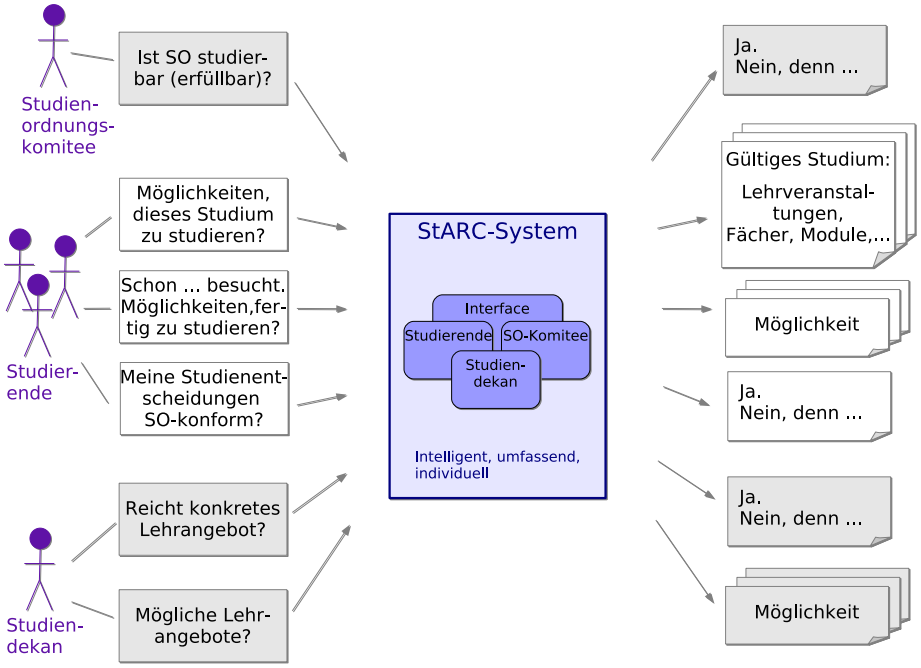



Abbildung 2.10: StARC-System: Anfragen und Antworten

(b) *Bedingungen an ein Studium, mögliche Studienentscheidungen:* Ein Studierender muss nur eine Teilmenge der Veranstaltungen belegen, die im Modulkatalog beschrieben sind. Zahlreiche und sehr komplexe Bedingungen sind bei einem korrekten Studium zu erfüllen.

2. *Repräsentation der Studienentscheidungen der Studierenden:* Schon bestandene Veranstaltungen (Prüfungen) und getroffene Studienentscheidungen müssen repräsentiert werden.

3. *Bearbeitung von Anfragen* durch eine „intelligente“ Engine.

4. *Ausgabe der Anfrageergebnisse* in geeigneter Form.

5. *Geeignete Bedienschnittstellen* für sämtliche Nutzer des Systems.[13]

Analog zu Abschnitt 2.3.2 sind auch hier folgende Aspekte zu beachten:

8. *Berücksichtigung des generierenden Aspekts (Modelgenerating, Planen, Suchproblem)* beispielsweise bei der Berechnung sämtlicher Möglichkeiten für ein „gültiges Studium“.

[13] Die Entwicklung der graphischen Bedienoberfläche wird im Rahmen dieser Arbeit nicht behandelt, jedoch die Schnittstellen hierzu.

9. *Berücksichtigung eines prüfenden Aspekts (Modeltesting)* z. B. bei der Prüfung auf Konsistenz der Anforderungen oder der getroffenen Entscheidungen.

10. *Umgang mit loser Spezifikation und vorgegebenen Teillösungen (lose Spezifikation, Teillösungen):* Möglicherweise liegen gewisse Informationen nicht vollständig vor. Hat ein Studierender zum Beispiel noch keinen Schwerpunkt gewählt, so soll trotzdem eine Beratung auf der Grundlage der sonstigen Entscheidungen stattfinden können.

11. *Beachtung der strukturierten Domäne (Strukturproblem):* Der Modulkatalog ist strukturiert, die Anforderungen haben engen Bezug zum Modulkatalog.

### 2.4.3 Nichtfunktionale Anforderungen

Neben den allgemeinen nichtfunktionalen Anforderungen für beide Anwendungsbereiche (Abschnitt 2.5) können noch folgende notiert werden:

- Für die interaktive Bearbeitung der Anfragen sind Laufzeiten von unter einer halben Sekunde wünschenswert. Falls es nicht erforderlich ist, dass das System als Just-in-Time-Applikation läuft, so sind Laufzeiten im Minutenbereich akzeptabel.[14]
- Datenmengen im Rahmen realer Studienordnungen (Größenordnung: 30 - 500 Veranstaltungen pro Studienordnung) sollten möglich sein.[15]
- Normale Verfügbarkeit des Systems genügt.
- Die Spezifikation der Anforderungen sollte möglichst deklarativ und leicht verständlich sein.[16]

### 2.4.4 Kernaspekte

Es lassen sich nachfolgende Kernbestandteile des Anwendungsbereichs „Studienberatung und Konsistenzprüfung von Studienordnungen“ identifizieren:

Ein zentraler Aspekt stellt die Modellierung und Repräsentation der Studienordnungen mit ihren oftmals sehr komplexen Anforderungen und Bedingungen dar. Darüber hinaus sind die Entscheidungen der Studierenden, die Vorschläge der Studienordnungsersteller und das konkrete Lehrangebot einer Fakultät zu repräsentieren.

[14] Laufzeiten im Minutenbereich stellen in dem beschriebenen Fall immer noch eine erhebliche Zeitersparnis im Vergleich zu einer komplexen Konsistenzprüfung oder umfangreichen Studienberatung „per Hand“ dar.

[15] Beispielsweise enthält an der Universität Passau die momentane Studienordnung des Bachelorstudiengangs Infomatik etwa 30, die komplexere Studienordnung des Bachelorstudiengangs European Studies etwa 200 Veranstaltungen [SoP].

[16] Aus Gründen der Benutzerfreundlichkeit wünscht man sich eine Spezifikationsmöglichkeit, die nahe an der informellen Beschreibung des Anwendungsbereiches liegt. Dies spricht für einen deklarativen Ansatz.

**Definition 2.4.1 (Kernaspekte „Studienberatung und Konsistenzprüfung “)** Die relevanten Hauptaspekte des Anwendungsbereichs Studienberatung und Konsistenzprüfung von Studienordnungen sind:

- **Veranstaltungen der Studienordnung:** eine Menge von Platzhaltern für konkrete Veranstaltungen zusammen mit deren Beschreibung (Eigenschaften): Attributnamen und Attributwerte
- **Modulkatalog der Studienordnung:** Strukturierung der Veranstaltungen der Studienordnung (Strukturtypen, Hierarchie)
- **Studienbedingungen, mögliche Studienentscheidungen:** Bedingungen und Anforderungen an ein gültiges Studium (Auswahlen und Voraussetzungen)
- **Gültiges Studium:** Teilmenge bzw. Teilgraph des Modulkataloges in Übereinstimmung mit den Studienbedingungen
- **Benutzerentscheidungen:** absolvierte Leistungen und getroffene Entscheidungen der Studierenden, möglicher Vorschlag des Studienordnungskomitees, konkretes Lehrangebot der Fakultät

Im Hochschulbereich sind heutzutage schon verschiedene Assistenzsysteme wie Campus-, Kurs-, Lernmanagementsysteme und Studienassistentsysteme im Einsatz. Zu bemerken ist allerdings, dass industrielle Systeme oftmals Einschränkungen unterliegen und daher nicht ausdrucksstark genug sind:

In einigen modernen Campusmanagementsystemen wie [Dat, Ora06, SAP04] ist zwar eine modulare Modellierung möglich, der Repräsentation von komplexen Bedingungen sind allerdings oftmals Grenzen gesetzt. Die Kurs- und Lernmanagementsysteme wie [Bla, Ili, Moo, Web] legen ihren Fokus meist auf e-learning-Aspekte wie das Anbieten einer Plattform für den Austausch von Kursmaterial. Komplexe Studienordnungen mit ihren Bedingungen können nicht repräsentiert werden. Studienassistentsysteme wie [HW98, WHSB99, GW03] haben auch meist nur (auf bestimmte Studiengänge) eingeschränkte Lösungen.

Teilweise werden die Veranstaltungen flach bzw. linear repräsentiert, was der meist vorhandenen hierarchischen Struktur des Modulkataloges nicht gerecht wird. Manchmal wird nicht zwischen den Templates für die Kurse und den aktuell in einem bestimmten Semester angebotenen Kursen unterschieden, was zu einer inadäquaten und inflexiblen Repräsentation des Studienangebotes führt. Ferner können manche Systeme nicht mit semantisch äquivalenten, jedoch an verschiedenen Universitäten unterschiedlich bezeichneten Kursen umgehen.

Das Ziel der Entwicklung eines zeitgemäßen Campus-Management-Systems, in der die in Abbildung 2.8 benannten Komponenten der Infrastruktur einer Universität Berücksichtigung finden, wird von einem ab 2005 für drei Jahre vom deutschen Bundesministerium für Bildung und Forschung geförderten großen Projekt [Int, KLF06, RGF06] verfolgt.

Aktuell scheint es allerdings kein System zu geben, welches das komplexe Problem der Studienberatung und der Konsistenzprüfung von Studienordnungen gemäß der oben angeführten Anforderungen allgemein löst.

## 2.5 Allgemeine nichtfunktionale Anforderungen und Zielkriterien

Auf der Grundlage obiger Ausführungen lassen sich folgende allgemeine nichtfunktionale Anforderungen und Zielkriterien für Testmanagement-Systeme bzw. StARC-Systeme formulieren:

1. *Ausdruckskraft und Mächtigkeit*
   - Relevante Anfragen und Konsistenzbedingungen abbildbar
   - Komplexe Abhängigkeiten zwischen Datenobjekten realisierbar
   - Genau sämtliche Lösungen können jeweils gefunden werden (Vollständigkeit und Korrektheit)
2. *Adäquatheit*
   - Kompakte Spezifikationen nahe an der informellen Beschreibung des Anwendungsfalles
   - Lose, unvollständige Spezifikation
   - Benutzerfreundlichkeit (Erkennen und Lokalisieren von Inkonsistenzen)
   - Interaktionsmöglichkeit
3. *Erweiterbarkeit und Integrierbarkeit*
   - Modularität
   - Lokalität bei Änderungen und Erweiterungen
   - Geeignete Schnittstellen zur Integrierbarkeit in Informationssysteme
4. *Performanz und Skalierbarkeit*
   - Adäquate Antwort- und Verarbeitungszeiten bei der Anfragebearbeitung bei realistischen Datenmengen
   - Akzeptable Laufzeitfunktion bei Variation im Datenvolumen
5. *Programmverständnis und Wartbarkeit*
   - Leichtes Einarbeiten in das Programm für den Experten
6. *Relevanz*
   - Aktuelle Problematik

# Kapitel 3

# Lösungsansatz

## 3.1 Grundidee methodischen Vorgehens

Wie in der Problembeschreibung (Kapitel 2) erläutert, gilt es, informelle Regelwerke – im Rahmen dieser Arbeit für die Anwendungsbereiche „Generierung von Tests“ und „Studienberatung und Konsistenzprüfung von Studienordnungen“ – für ein modernes Informationssystem adäquat zu formalisieren und zu repräsentieren. Außerdem ist eine Auswahl geeigneter Komponenten, Methoden und Mittel zu treffen.

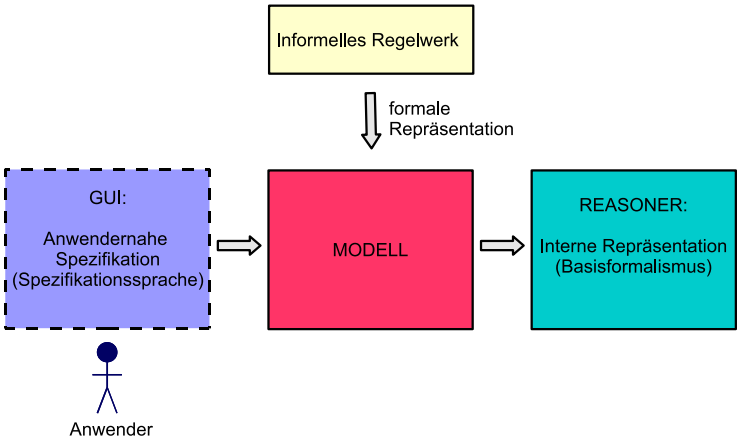



Abbildung 3.1: Grundidee der Lösung, Grobstruktur

Das gewählte prinzipielle Vorgehen wird in Abbildung 3.1 skizziert: Das informelle Regelwerk wird durch ein Modell, welches ein zentrales Element des Ansatzes ist (modellgetriebener Ansatz), re-

präsentiert.[1] Dieses stellt sowohl eine Schnittstelle zur internen Repräsentation der entsprechenden Plattform (Reasoner) als auch zur Bedienoberfläche (GUI) dar.

Die Abbildung 3.2 zeigt die Struktur des gewählten Ansatzes detaillierter.

Genauer betrachtet, besteht das Modell aus abstrakten und konkreten Modellen. Das abstrakte Modell eines Anwendungsbereichs gibt die Struktur und Komponenten der entsprechenden konkreten konzeptuellen Modelle vor. Diese modellieren die verschiedenen Problemausprägungen im Anwendungsbereich.

Ausgehend vom konkreten Modell erfolgt eine Transformation in den gewählten internen Formalismus. Im Rahmen dieser Arbeit wird ein logikbasierter Ansatz mit Constraints, genauer Answer Set Programming mit Gewichten (vgl. Abschnitt 4.2), favorisiert. Allerdings soll eine Offenheit zu verschiedenen Plattformen gewährleistet sein. Neben Answer Set Programming sind noch weitere interne Formalismen denkbar. Konkret wird in dieser Arbeit auch ein pfadbasierter Ansatz, nämlich XML, XPath und XQuery (vgl. Abschnitt 4.3) betrachtet.

Auf der Grundlage der plattformabhängigen, internen Repräsentation erfolgt mit Hilfe des entsprechenden Reasoners die Berechnung der Lösungen bzw. Modelle. Durch eine Nachbearbeitung kann die Antwort auf eine Anfrage des Anwenders generiert werden.

Für den Benutzer wünscht man sich eine Bedienoberfläche, in der er eine anwendernahe Spezifikation auf der Grundlage einer Spezifikationssprache vornehmen kann.

Im Rahmen dieser Arbeit soll allerdings eine formale Spezifikationssprache und die Realisierung einer graphischen Bedienoberfläche (GUI) nicht Gegenstand der Betrachtungen sein. Nichtsdestotrotz können die abstrakte Modellierung und die vorgenommenen Spezifikationsbeschreibungen als eine wesentliche Grundlage hierfür betrachtet werden.

[1] Der aktuell moderne modellgetriebene Ansatz der Model Driven Architecture (MDA) [KWB03, MSU04] erscheint hier als nicht adäquat und wird daher nicht verwendet: Im Rahmen von MDA wird das Gesamtmodell in mehrere Schichten (Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM)) unterteilt und die Transformation der Modelle definiert (Modell- und Codetransformation). Hierbei spielen die sogenannten UML-Profile eine wichtige Rolle, wobei die im UML-Metamodell vorgegebenen Meta-Typen durch Anwendung von Erweiterungsmechanismen weiter spezialisiert werden. Die Metamodelle der hier untersuchten Anwendungsbereiche sind jedoch für einen MDA-Ansatz zu unspezifisch.

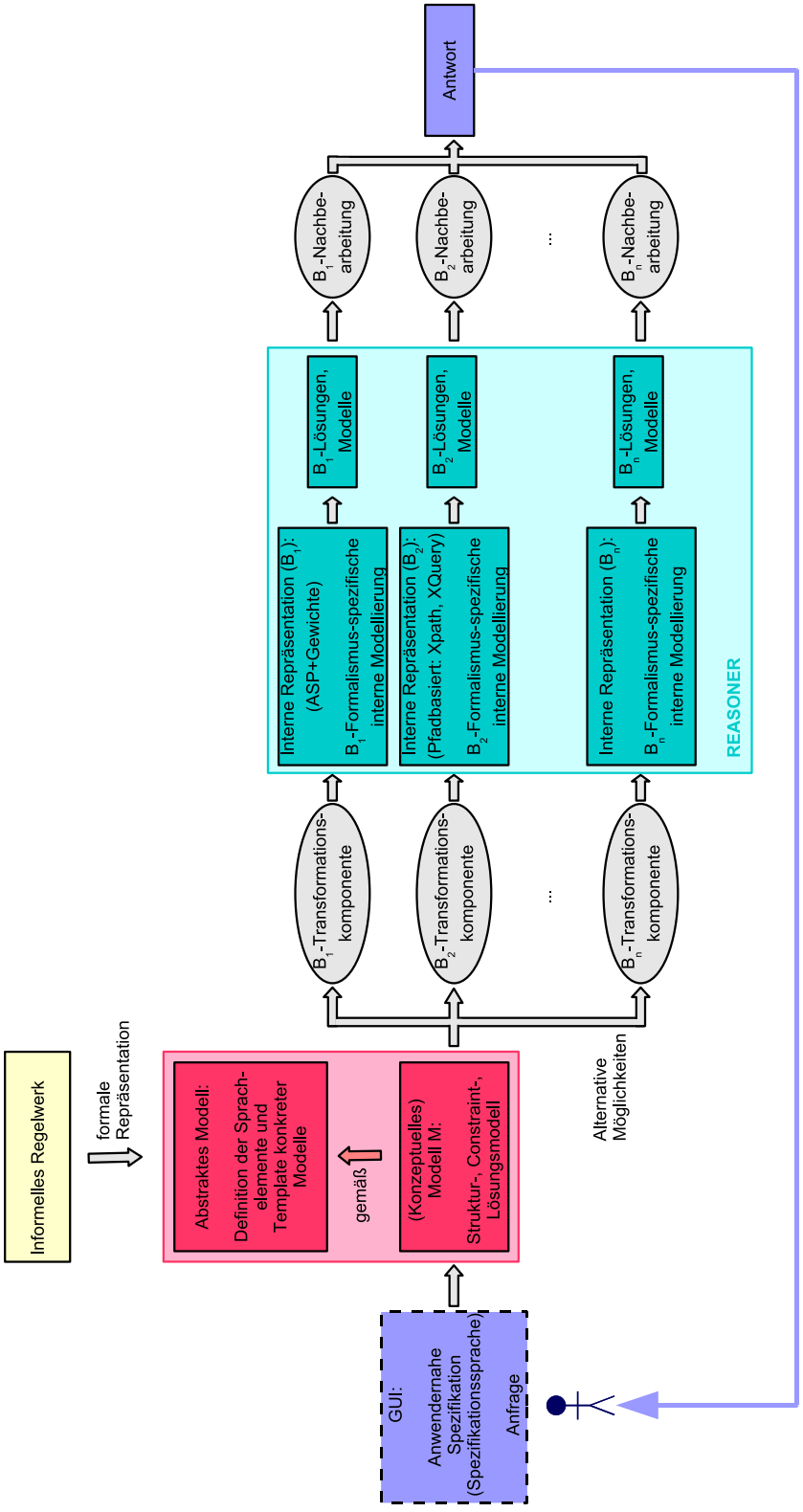



Abbildung 3.2: Grundidee der Lösung, Feinstruktur

## 3.2 Methodenwahl für die Modellierung

Wie in Abschnitt 3.1 schon erwähnt, wird ein modellgetriebener Ansatz verfolgt.[2] Grundsätzlich wird also vom Modell ausgegangen. Für die Wahl der geeigneten Methoden sollen folgende an die Modellierung gestellten Wunschkriterien in Betracht gezogen werden:

1. *Nähe zur informellen Beschreibung* des Anwendungsbereichs
   - Anwendungsnähe
   - Grundlage für die (leichte) Möglichkeit der Definition einer Spezifikationssprache für den Anwendungsbereich
2. Unabhängigkeit von internen Formalismen *(Plattformunabhängigkeit)*
3. Möglichkeit der (einfachen) Definition einer *Transformationsvorschrift* in möglicherweise unterschiedliche interne Repräsentationsarten (Basisformalismen)
4. Abstraktes Modell zur *Definition der Sprachelemente* bzw. als *Template* für die konkreten Modelle
5. *Adäquatheit*
   - Geeignete Ausdrucksstärke und Abstraktionsniveau
   - Klar definierte Semantik
   - Erweiterbarkeit
6. Verwendung gängiger und leicht verständlicher Methoden
   - Verständlichkeit für den „Nicht-Experten“
7. Analyse, Definition und Spezifikation des zu lösenden Problems

In Kapitel 5 wird ein Modell definiert, welches diesen Wunschkriterien Rechnung trägt. Es wird sich zeigen, dass es für beide Anwendungsbereiche bis zu einem gewissen (hohen) Grad eine gemeinsame Modellebene gibt. Methodisch gesehen finden folgende Aspekte Berücksichtigung:

1. *Verschiedene Ebenen der Modellierung:* Instanzmodell, Klassenmodell, abstraktes Modell
2. *Mehrere Modellkomponenten (-module):* Struktur-, Constraint-, Lösungsmodell
   - Strukturmodell: Graphkonzepte, Klassen und Instanzen
   - Constraintmodell: Relationen-, Mengenkonzepte

[2] Vergleiche auch die entsprechende Fußnote im vorherigen Abschnitt.

- Lösungsmodell: Definition einer Lösung, Verknüpfung von Struktur- und Constraintmodell

3. *Überlagerung der Graph- und Relationen-/Mengenkonzepte:*
   - Verzahnung der Modellkomponenten aufgrund gewisser Abhängigkeiten
   - Definition komplexer Constraintarten (z. B. Successorconstraints, Baumconstraints und Auswahl-Baumconstraints)
4. *Verschiedene „Spezifikationsebenen“* im Constraintmodell
   - Mengen- und Relationensicht
   - „Benutzerspezifikationssicht“ (Eigenschaften, Parameter)

## 3.3 Methodenwahl für die interne Repräsentation

Als Kernmethode interner Repräsentation wird ein logikbasierter Ansatz mit Constraints, genauer Answer Set Programming mit Gewichten (vgl. Abschnitt 4.2), gewählt. Folgende Aspekte sprechen für diesen Ansatz:

1. Hoher Grad an Deklarativität
2. Verbindung von Logik und Constraints
3. Nähe zur relationalen Modellierung (ER-Modellierung, Datenbanken)
4. Hohe Expressivität, Ausflüchte in die Logik 2. Ordnung, Möglichkeit der Definition rekursiver Funktionen
5. Geeignet für Suchprobleme mit
   - mehrdimensionalem Suchraum
   - $n : m$-Beziehungen
   - Hierarchien
6. Gewisses Maß an Modularität
7. Umgang mit späten Spezifikationen und Teillösungen
8. Gewisse Offenheit für andere Systeme

Die Modellierung in Answer Set Programming mit Gewichten (ASP+Gewichte oder nur kurz ASP) erfolgt im Wesentlichen wie folgt:

- *Anwendungsbereich Generierung von Tests:* Die Anforderungen an die Tests werden in ASP so modelliert, dass die Antwortmengen (stabile Modelle) des ASP-Programms genau die Codierungen der Tests darstellen, die die Anforderungen erfüllen.
- *Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen:* Die Anforderungen einer Studienordnung werden in ASP so modelliert, dass die Antwortmengen (stabile Modelle) des ASP-Programms genau sämtliche Möglichkeiten angeben, gemäß dieser Studienordnung zu studieren.

Ergänzend werden weitere Methoden betrachtet (vgl. Abschnitt 4.1), unter anderem XML in Verbindung mit XPath/XQuery. Grundsätzlich hat diese Methode folgende Vorteile (vgl. auch Abschnitt 7.4.1):

1. Gute und einfache Navigationsmöglichkeit in strukturierten Dokumenten (Domänen)
2. Möglichkeit der Definition von rekursiven Funktionen

Wie im Kapitel zur Evaluation des Ansatzes (Kapitel 7) erläutert wird, geht unter den betrachteten Methoden zur internen Repräsentation „Answer Set Programming mit Gewichten“ bei weitem als „Sieger“ hervor.

## 3.4 Architektur

Obwohl im Abschnitt 3.1 schon die Grundstruktur der Architektur für ein Testmanagement-System bzw. ein StARC-System zu erkennen ist, sollen diese hier der Deutlichkeit halber noch genauer betrachtet werden (Abbildungen 3.3 und 3.4).

Nach den Erläuterungen zu den betrachteten Methoden für die interne Repräsentation im nächsten Kapitel werden in den darauffolgenden Kapiteln das Modell definiert und die Transformation in interne Formalismen beschrieben. Eine Evaluation des Ansatzes erfolgt in Kapitel 7. Es wird ersichtlich, dass aufgrund des anwendungsnahen und intuitiven Modells der gewählte Ansatz eine große Offenheit in sich trägt. Die Transformation in die logik- und constraintbasierte interne Repräsentation erfolgt in hohem Maße modular. Aufgrund der Mächtigkeit von Answer Set Programming können sämtliche Problemstellungen der Anwendungsbereiche gut gelöst werden.

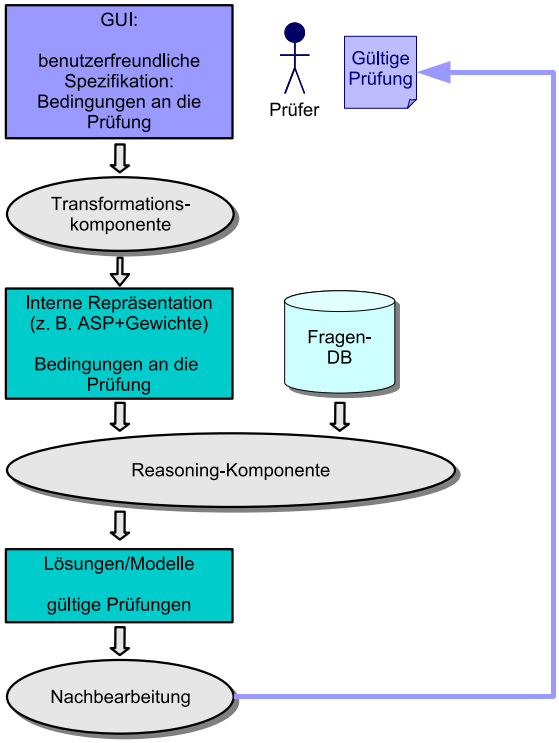



Abbildung 3.3: Architektur eines Testmanagement-Systems

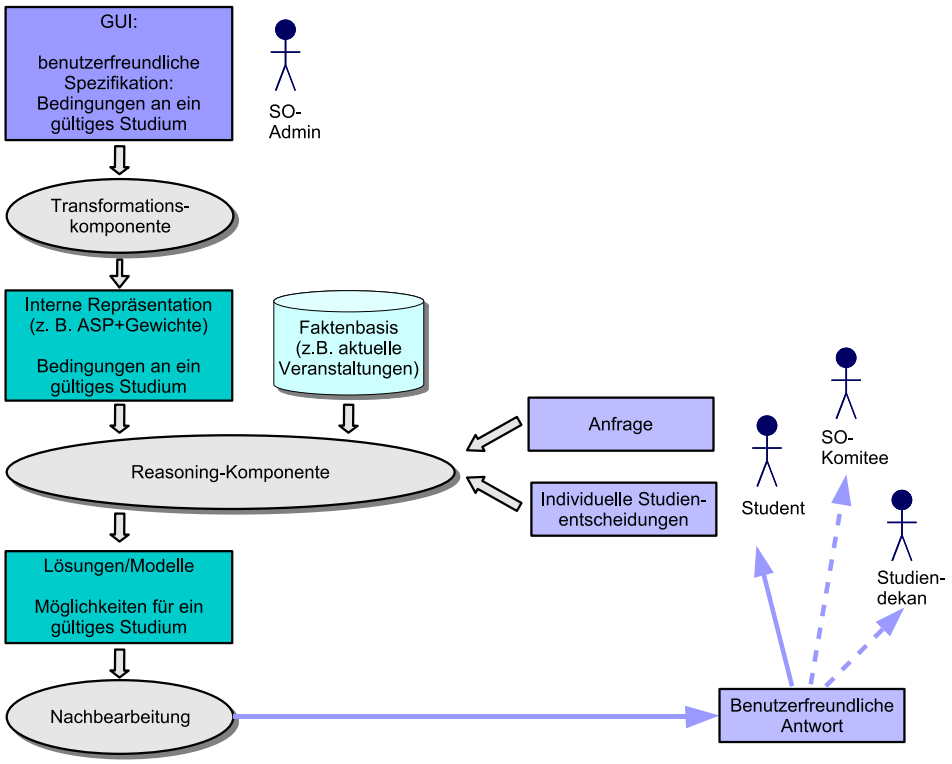



Abbildung 3.4: Architektur eines StARC-Systems

# Kapitel 4

# Grundlagen und Methoden

Die folgende Darstellung über die Grundlagen und Methoden dient der inhaltlichen Geschlossenheit der Arbeit. Der Großteil ist aus der Literatur übernommen. Nichtsdestotrotz erfolgt an manchen Stellen eine Anreicherung mit eigenen Beispielen. In der Regel sind diese gut erkenntlich und werden daher nicht eigens markiert.

## 4.1 Überblick betrachteter Methoden

Die folgende Auflistung zeigt einen kurzen Überblick über betrachtete interne Repräsentationsformalismen und -methoden. Diese beinhalten sowohl den favorisierten logikbasierten Ansatz mit Constraints als auch alternative Methoden.

1. *Logikbasierter Ansatz mit Constraints:*

   Dieser spielt im Rahmen dieser Arbeit eine zentrale Rolle. Ausgewählt wurde die Antwortmengenprogrammierung mit Gewichten. Die Grundlagen hierzu aus der wissenschaftlichen Literatur werden im Abschnitt 4.2 dargestellt.

   Als weiterer Ansatz, der Logik und Constraints verbindet, ist die Constraint Logic Programming CLP zu nennen. Die zugrundeliegenden deklarativen Paradigmen sind Constraint Satisfaction Probleme (CSP) [Tsa93] und Logikprogrammierung [SS86]. Eine grundlegende Idee ist dabei, bestimmte syntaktisch ausgezeichnete Prädikate (Constraints) durch spezielle Algorithmen über bestimmten Wertebereichen zu lösen [FA97]. Diese Methode wird nicht näher besprochen.

2. *Pfadbasierter Ansatz:*

   Ergänzend wird ein pfadbasierter Ansatz, genauer XML in Verbindung mit XPath und XQuery betrachtet. Einige Grundlagen hierzu werden im Abschnitt 4.3 aufgezeigt.

3. *Straight-forward-Lösung:*

   Imperativ, mit Java (z. B. [JAV]) o. Ä.

4. *Structured Query Language, SQL:*

   Eine Anfragesprache an relationale Datenbanken (z. B. [EN02, SQL]).

5. *Datalog:*

   Eine logische Anfragesprache an Datenbanken (z. B. [CGT90]).

6. *Operations Research, OR:*

   Eine Methode zur Optimierung bestimmter Verfahren oder Prozesse (z. B. [Hoo, DD07])

## 4.2 Logikbasierter Ansatz mit Constraints

Als logikbasierter Ansatz mit Constraints wird „Antwortmengenprogrammierung mit Gewichten“ verwendet. Die wesentlichen Grundlagen hierzu werden im Folgenden aufgezeigt. Für umfangreichere Informationen sei der interessierte Leser auf weiterführende Literatur verwiesen.

### 4.2.1 Normale logische Programme und stabile Modelle

Die folgenden Ausführungen sind größtenteils aus der Literatur übernommen und orientieren sich im Wesentlichen an [Llo87, Lif96, GL88] und zu großen Teilen an die Darstellung von [Fre05], angereichert mit eigenen Beispielen.

#### 4.2.1.1 Syntax

**Definition 4.2.1 (Alphabet)** Ein *Alphabet* $\Sigma$ einer Logiksprache erster Stufe enthält:

1. Abzählbar viele *Variablen*

   $$X, X_1, X_2, \ldots, Y, Y_1, Y_2, \ldots, Z, Z_1, Z_2, \ldots$$

2. Zu jedem $n \in \mathbb{N}_0$ abzählbar viele n-stellige *Funktionszeichen*

   $$f, f_1, f_2, \ldots, g, g_1, g_2, \ldots, h, h_1, h_2, \ldots$$

   Die 0-stelligen Funktionszeichen heißen *Konstanten*.

3. Zu jedem $n \in \mathbb{N}_0$ abzählbar viele n-stellige *Prädikatzeichen*

   $$p, p_1, p_2, \ldots, q, q_1, q_2, \ldots, r, r_1, r_2, \ldots$$

   Die 0-stelligen Prädikatzeichen heißen *aussagenlogische Konstanten*.

4. Die *logischen Junktoren* $\mathrm{not}$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), $\leftrightarrow$ (equivalent).

5. Die *Quantoren* $\forall$ (forall) und $\exists$ (exists).

6. Die Punktuationszeichen "(", ")" und ",".

Es wird stets angenommen, dass die Menge $\mathcal{V}$ der Variablen und die Menge $\mathcal{FUN} \cup \mathcal{PRED}$ der Funktions- und Prädikatzeichen disjunkt sind.

**Definition 4.2.2 (Term)** Ein *Term* ist wie folgt induktiv definiert:

1. Jede Variable $X$ ist ein Term.
2. Jede Konstante $a$ ist ein Term.
3. Sind $t_1, \ldots, t_n$ Terme und ist $f$ ein n-stelliges Funktionszeichen, so ist $f(t_1, \ldots, t_n)$ ein Term.

Die (syntaktische) Gleichheit zweier Terme wird mit $\equiv$ bezeichnet.

**Definition 4.2.3 (Atom, Atomare Formel)** Für jedes $k$-stellige Prädikatzeichen $p$ und Terme $t_1, \ldots, t_n$ ist $p(t_1, \ldots, t_n)$ eine *atomare Formel* oder kurz ein *Atom*.

**Definition 4.2.4 (Formel)** Die *Formeln* werden wie folgt induktiv definiert:

1. Jede atomare Formel ist eine Formel.
2. Sind $F$ und $G$ Formeln, so sind auch $(\mathrm{not}\, F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ und $(F \leftrightarrow G)$ Formeln.
3. Ist $F$ eine Formel und $X$ eine Variable, so sind $(\forall X\ F)$ und $(\exists X\ F)$ Formeln.

Statt $(F \rightarrow G)$ schreiben wir auch $(G \leftarrow F)$. Die (syntaktische) Gleichheit zweier Formeln wird mit $\equiv$ bezeichnet.

**Bemerkung 4.2.5 (Bindungshierarchie in Formeln)** Es soll auf Klammern verzichtet werden, wenn dies möglich ist. Dazu wird folgende Bindungshierarchie zwischen den logischen Junktoren und Quantoren verwendet:

1. not, $\forall$, $\exists$
2. $\wedge$, $\vee$
3. $\rightarrow$, $\leftrightarrow$ $\rightarrow$ ist rechtsassoziativ

**Definition 4.2.6 (Logiksprache erster Stufe)** Eine *Logiksprache erster Stufe* über einem Alphabet besteht aus der Menge aller Formeln, die gemäss Definition 4.2.4 der Formeln konstruiert werden können.

**Definition 4.2.7 (Gebundenes und freies Auftreten von Variablen, geschlossene Formel)** Der *Bindungsbereich* oder *Scope* des jeweiligen Quantors in einer Formel $\forall X\ F$ oder $\exists X\ F$ ist die Formel $F$. Ein *gebundenes Auftreten* einer Variablen $X$ ist ein Auftreten direkt hinter einem Quantor oder im Scope eines Quantors, der $X$ bindet. Jedes andere Auftreten einer Variablen ist *frei*. Eine *geschlossene Formel* ist eine Formel, in der keine freien Variablen auftreten.

**Definition 4.2.8 (Literal)** Ein *positives Literal* ist ein Atom $A$, ein *negatives Literal* ist die Negation $\text{not}\, A$ eines Atoms $A$. Ein *Literal* ist ein positives Literal oder ein negatives Literal.

**Definition 4.2.9 (Klausel)** Eine *Klausel* ist eine Formel der Gestalt

$$\forall X_1 \ldots \forall X_n (L_1 \vee \cdots \vee L_m)$$

wobei $L_i, 1 \leq i \leq m, m \in \mathbb{N}_0$ Literale und $X_1, \ldots, X_n$ sämtliche in $L_1 \vee \cdots \vee L_m$ auftretende Variablen sind.

**Definition 4.2.10 (Deduktive Regel, Programmklausel)** Eine *deduktive Regel* hat die Form

$$C_1, \ldots, C_k \leftarrow A_1, \ldots, A_n, \text{not}\, B_1, \ldots, \text{not}\, B_m$$

und bezeichnet die Klausel

$$\forall X_1 \ldots \forall X_s (\text{not}\, A_1 \vee \ldots \vee \text{not}\, A_n \vee B_1 \vee \ldots \vee B_m \vee C_1 \vee \ldots \vee C_k)$$

wobei $A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k, n, m, k \in \mathbb{N}_0$ Atome darstellen und $X_1, \ldots, X_s, s \in \mathbb{N}_0$ sämtliche Variablen sind, die in diesen Atomen auftreten. Falls $m = n = 0$ gilt, wird das Implikationszeichen $\leftarrow$ meistens weggelassen.

$C_1, \ldots, C_k$ ist der *Kopf* und $A_1, \ldots A_n, \text{not}\, B_1, \ldots, \text{not}\, B_m$ ist der *Rumpf* der Regel. Entsprechend heißen die Literale $A_i, 1 \leq i \leq n$, die *positiven* und $\text{not}\ B_j, 1 \leq j \leq m$, die *negativen Rumpfliterale* der Regel.

Eine deduktive Regel heißt *normal*, falls $k = 1$, sie heißt *definit*, falls $k = 1$ und $m = 0$. Sie heißt *Fakt*, falls $k = 1$ und $n = m = 0$.

**Definition 4.2.11 (Normales und definites logisches Programm)** Ein *normales* bzw. *definites logisches Programm* ist eine endliche Menge normaler bzw. definiter Regeln.

Als logische Formel ist ein Programm die *Konjunktion* aller durch seine Regeln bezeichneten Klauseln.

**Definition 4.2.12 (Sprache eines Programms)** Sei $P$ ein Programm über $L$. Die von $P$ *induzierte Logiksprache* $L_P$ ist die aus den in $P$ auftretenden Prädikatzeichen, Funktionssymbolen und Konstanten gebildete Teilsprache (Logiksprache 1. Ordnung) von $L$.

**Definition 4.2.13 (Grundierte Terme, Atome, Regeln, Programme)** Terme, Atome, Regeln und Programme, in denen keine Variablen auftreten, werden als *grundiert* oder *variablenfrei* bezeichnet.

Für den Umgang mit nicht-grundierten Programmen wird das Konzept der Substitution benötigt.

**Definition 4.2.14 (Substitution)** Sei $L$ eine Logiksprache und $\mathcal{T}$ eine beliebige Menge von Termen von $L$. Eine *Substitution über* $\mathcal{T}$ ist eine endliche Menge

$$\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$$

von Variablenbindungen, für die gilt:

1. $X_1, \ldots, X_n$ sind paarweise verschiedene Variablen von $L$.
2. $t_i \in \mathcal{T}$ für alle $i = 1, \ldots, n$

Falls die Termmenge $\mathcal{T}$ nicht von Belang ist, wird allgemein von *Substitutionen* gesprochen.

**Definition 4.2.15 (Ausdruck)** Ein *Ausdruck* ist entweder ein Term, ein Literal oder eine Konjunktion oder Disjunktion von Literalen. Ein *einfacher Ausdruck* ist entweder ein Term oder ein Atom.

**Definition 4.2.16 (Ersetzung von Variablen)** Seien $E$ ein Ausdruck, $t_1, \ldots, t_n$ Terme und $X_1, \ldots, X_n$ paarweise verschiedene Variablen.

Das Ergebnis $E[X_1/t_1, \ldots, X_n/t_n]$ der *Ersetzung der Variablen* $X_1, \ldots, X_n$ in $E$ durch die Terme $t_1, \ldots, t_n$ wird definiert durch

1. Falls alle $X_1, \ldots, X_n$ freie Variablen in $E$ sind und für alle $1 \leq i \leq n$ die freien Variablen in $t_i$ in $E$ *nicht* als gebundene Variablen auftreten, bezeichnet

   $$E[X_1/t_1, \ldots, X_n/t_n]$$

   den Term oder die Formel, die durch die simultane textuelle Ersetzung von $X_i$ durch $t_i$, $1 \leq i \leq n$, in $E$ entsteht.

2. Falls $X_i, 1 \leq i \leq n$ keine freie Variable in $E$ ist, so ist

$$E[X_1/t_1, \ldots, X_n/t_n] :\equiv E[X_1/t_1, \ldots, X_{i-1}/t_{i-1}, X_{i+1}/t_{i+1}, \ldots, X_n/t_n]$$

In allen anderen Fällen ist das Ergebnis der Ersetzung nicht definiert.

**Definition 4.2.17 (Instanz)** Sei $E$ ein Ausdruck der Sprache $L$. Sei $\mathcal{T}$ eine Menge von Termen von $L$ und $\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$ eine Substitution über $\mathcal{T}$.

$E\sigma$ heißt *$\mathcal{T}$-Instanz* von $E$ durch $\sigma$ oder einfach *Instanz* von $E$. Falls $E\sigma$ keine freien Variablen enthält ist $E\sigma$ eine *$\mathcal{T}$-Grundinstanz* von $E$ oder einfach eine *Grundinstanz* von $E$.

**Definition 4.2.18 (Instanz einer deduktiven Regel)** Sei $L$ eine Logiksprache, $\mathcal{T}$ eine Menge von Termen von $L$ und $\sigma$ eine Substitution über $\mathcal{T}$. Die *$\mathcal{T}$-Instanz der deduktiven Regel* $C_1, \ldots, C_k \leftarrow L_1, \ldots, L_n$ durch $\sigma$ ist definiert durch

$$(C_1, \ldots, C_k \leftarrow L_1, \ldots, L_n)\,\sigma :\equiv C_1\sigma, \ldots, C_k\sigma \leftarrow L_1\sigma, \ldots, L_n\sigma$$

mit $k \geq 0$ und $n \geq 0$.

Zur Definition der Semantik normaler Programme werden noch folgende Konzepte benötigt.

**Bemerkung 4.2.19 (Vereinbarung)** Es werden nur solche Logiksprachen $L$ betrachtet, die mindestens eine Konstante enthalten. Falls eine gegebene Logiksprache keine Konstanten enthält, wird ihr zunächst eine neue Konstante $c_0$ hinzugefügt und die resultierende Sprache $L$ zugrunde gelegt.

**Definition 4.2.20 (Herbrand-Universum)** Sei $L$ eine Logiksprache, die mindestens eine Konstante enthält. Das *Herbrand-Universum* $\mathcal{U}_L$ von $L$ ist die Menge aller Terme ohne freie Variablen (Grundterme), die aus den Konstanten und Funktionszeichen von $L$ nach Definition 4.2.2 der Terme gebildet werden können.

**Definition 4.2.21 (Herbrand-Basis)** Die *Herbrand-Basis* $\mathcal{B}_L$ einer Logiksprache $L$ ist die Menge aller Atome ohne freie Variablen (Grundatome), die aus den Prädikatzeichen von $L$ und den Grundtermen des Herbrand-Universums $\mathcal{U}_L$ gemäß Definition 4.2.3 syntaktisch korrekt gebildet werden können.

**Definition 4.2.22 (Herbrand-Universum, Herbrand-Basis eines Programms)** Sei $P$ ein Programm über $L$ und $L_P$ die von $P$ induzierte Sprache (vgl. Definition 4.2.12). Das *Herbrand-Universum von* $P$ wird durch

$$\mathcal{U}_P = \mathcal{U}_{L_P}$$

und die *Herbrand-Basis* von $P$ durch

$$\mathcal{B}_P = \mathcal{B}_{L_P}$$

festgelegt.

#### 4.2.1.2 Stabile Modelle normaler Programme

**Bemerkung 4.2.23** Der Einfachheit halber werden im Folgenden oft variablenfreie Programme betrachtet. Programme $P$ mit Variablen können durch eine entsprechende $\mathcal{U}_P$-Grundinstanziierung (vgl. Definition 4.2.18, 4.2.20 und 4.2.22) variablenfrei „gemacht“ werden. In diesem Zusammenhang wird auch einfach nur vom (zugehörigen) „*grundierten*“ Programm gesprochen.

**Beispiel 4.2.24** Gegeben sei das normale logische Programm $P$

$$\begin{aligned}
&\mathsf{question}(\mathsf{q}_1).\\
&\mathsf{question}(\mathsf{q}_2).\\
\mathsf{rejectQuestion}(\mathsf{Q}) &\leftarrow \mathsf{not\ chooseQuestion}(\mathsf{Q}).\\
\mathsf{chooseQuestion}(\mathsf{Q}) &\leftarrow \mathsf{not\ rejectQuestion}(\mathsf{Q}).
\end{aligned}$$

Das Herbrand-Universum ist

$$\mathcal{U}_P = \{\mathsf{q}_1, \mathsf{q}_2\}$$

da die Konstanten $q_1$, $q_2$ die einzigen grundierten Terme des Programms sind. Die Herbrand-Basis ist

$$\begin{aligned}
\mathcal{B}_P = \{&\\
&\mathsf{question}(\mathsf{q}_1), \mathsf{question}(\mathsf{q}_2),\\
&\mathsf{chooseQuestion}(\mathsf{q}_1), \mathsf{chooseQuestion}(\mathsf{q}_2),\\
&\mathsf{rejectQuestion}(\mathsf{q}_1), \mathsf{rejectQuestion}(\mathsf{q}_2)\\
\}&
\end{aligned}$$

Das zugehörige $\mathcal{U}_P$-grundinstanziierte (grundierte) Programm ist durch

$$\begin{aligned}
&\mathsf{question}(\mathsf{q}_1).\\
&\mathsf{question}(\mathsf{q}_2).\\
\mathsf{rejectQuestion}(\mathsf{q}_1) &\leftarrow \mathsf{not\ chooseQuestion}(\mathsf{q}_1).\\
\mathsf{rejectQuestion}(\mathsf{q}_2) &\leftarrow \mathsf{not\ chooseQuestion}(\mathsf{q}_2).\\
\mathsf{chooseQuestion}(\mathsf{q}_1) &\leftarrow \mathsf{not\ rejectQuestion}(\mathsf{q}_1).\\
\mathsf{chooseQuestion}(\mathsf{q}_2) &\leftarrow \mathsf{not\ rejectQuestion}(\mathsf{q}_2).
\end{aligned}$$

gegeben. Auch dieses wird oft einfach mit $P$ bezeichnet.

**Definition 4.2.25 (Immediate Consequence Operator)** Sei $P$ ein definites, grundiertes Programm. Sei $I \subseteq \mathcal{B}_P$ eine Menge von Atomen. Die Menge der *direkten Folgerungen von $I$ bezüglich $P$* ist wie folgt definiert:

$$\mathcal{T}_P(I) \ := \ \{C \mid \text{Es gibt eine Regel } C \leftarrow A_1, \ldots, A_n \text{ in } P \text{ mit } \{A_1, \ldots, A_n\} \subseteq I\}.$$

**Beispiel 4.2.26** Gegeben sei das definite Programm $P$

$$\begin{aligned}
&\mathsf{chooseQuestion}(\mathsf{q_1}).\\
&\mathsf{rejectQuestion}(\mathsf{q_3}) \leftarrow \mathsf{chooseQuestion}(\mathsf{q_1}), \mathsf{chooseQuestion}(\mathsf{q_2}).\\
&\mathsf{chooseQuestion}(\mathsf{q_2}) \leftarrow \mathsf{chooseQuestion}(\mathsf{q_1}).\\
&\mathsf{chooseQuestion}(\mathsf{q_3}) \leftarrow \mathsf{rejectQuestion}(\mathsf{q_2}).
\end{aligned}$$

Dann ist offensichtlich

$$\begin{aligned}
\mathcal{T}_P(\emptyset) &= \{\mathsf{chooseQuestion}(\mathsf{q_1})\} =: I_1\\
\mathcal{T}_P(I_1) &= \{\mathsf{chooseQuestion}(\mathsf{q_1}), \mathsf{chooseQuestion}(\mathsf{q_2})\} =: I_2\\
\mathcal{T}_P(I_2) &= \{\mathsf{chooseQuestion}(\mathsf{q_1}), \mathsf{chooseQuestion}(\mathsf{q_2}), \mathsf{rejectQuestion}(\mathsf{q_3})\} =: I_3\\
\mathcal{T}_P(I_3) &= \{\mathsf{chooseQuestion}(\mathsf{q_1}), \mathsf{chooseQuestion}(\mathsf{q_2}), \mathsf{rejectQuestion}(\mathsf{q_3})\} = I_3
\end{aligned}$$

Zu beachten ist, dass $\mathcal{T}_P(I_3) = I_3$ gilt und daher $I_3$ ein sogenannter *Fixpunkt* von $\mathcal{T}_P$ darstellt. Dieser ist sogar der sogenannte *kleinste Fixpunkt* von $\mathcal{T}_P$: Für jeden Fixpunkt $I$ von $\mathcal{T}_P$ ist $I_3 \subseteq I$.

**Definition 4.2.27 (Stabile Modelle definiter Programme)** Sei $P$ ein definites, grundiertes Programm. Eine Teilmenge $M \subseteq \mathcal{B}_P$ ist ein *stabiles Modell* von $P$, wenn gilt:

1. $M$ ist ein Fixpunkt von $\mathcal{T}_P$: $\mathcal{T}_P(M) = M$
2. $M$ ist der *kleinste Fixpunkt* von $\mathcal{T}_P$: Für jeden Fixpunkt $M'$ von $\mathcal{T}_P$ gilt: $M \subseteq M'$

$M$ wird auch als *kleinstes Herbrandmodell* oder als *deduktiver Abschluss* von $P$ bezeichnet. [1]

**Bemerkung 4.2.28** Wie oben schon erwähnt, können entsprechende Definitionen auch auf nicht variablenfreie Programme übertragen werden. Hierbei sind die zugehörigen grundierten Programme zu berücksichtigen. Wie der Literatur zu entnehmen ist, hat ein definites Programm $P$ genau ein stabiles Modell. Dieses wird oft einfach mit $M_P$ oder $cl(P)$ bezeichnet.

**Beispiel 4.2.29 (Stabiles Modell eines definiten Programms)** Gegeben sei das Programm $P$ aus Beispiel 4.2.26. Das stabile Modell von $P$ ist offensichtlich durch

$$M_P = I_3 = \{\mathsf{chooseQuestion}(\mathsf{q_1}), \mathsf{chooseQuestion}(\mathsf{q_2}), \mathsf{rejectQuestion}(\mathsf{q_3})\}$$

gegeben.

[1] In der Literatur werden in der Regel zunächst sogenannte Herbrandmodelle, minimale Herbrandmodelle und dann das kleinste Herbrandmodell definiert. Zur Definition der stabilen-Modell-Semantik wird dann auf diese zurückgegriffen: Das stabile Modell eines definiten Programms ist genau das kleinste Herbrandmodell des Programms. Um die Ausführungen kurz zu halten, wird hier dieser Weg nicht eingeschlagen.

**Definition 4.2.30 (Gelfond-Lifschitz Transformation)** Sei $P$ ein variablenfreies normales Programm. Sei $I \subseteq \mathcal{B}_P$ eine Herbrand-Interpretation. Die *Gelfond-Lifschitz Transformation* oder *GL-Transformation* $P^I$ von $P$ modulo $I$ ist das Programm, das aus $P$ wie folgt gebildet wird:

$$P^I = \{C \leftarrow A_1, \ldots, A_n \mid C \leftarrow A_1, \ldots, A_n, \text{not } B_1, \ldots, \text{not } B_m \in P \text{ und } \{B_1, \ldots, B_m\} \cap I = \emptyset\}$$

Oftmals wird $P^I$ auch als das *Redukt von $P$ bzgl. $I$* bezeichnet. Es ist stets definit.

**Beispiel 4.2.31 (Gelfond-Lifschitz-Transformation eines normalen Programms)** Gegeben sei das normale Programm $P$ aus Beispiel 4.2.24 in der grundierten Version. Die Gelfond-Lifschitz-Transformation $P^\emptyset$ von $P$ modulo $I_1 = \emptyset$ ist

$\mathsf{question}(\mathsf{q}_1).$
$\mathsf{question}(\mathsf{q}_2).$
$\mathsf{rejectQuestion}(\mathsf{q}_2).$
$\mathsf{chooseQuestion}(\mathsf{q}_1).$
$\mathsf{chooseQuestion}(\mathsf{q}_2).$

Für $I_2 = \{\mathsf{chooseQuestion}(\mathsf{q}_1), \mathsf{rejectQuestion}(\mathsf{q}_2)\}$ ist die Gelfond-Lifschitz-Transformation $P^{I_2}$:

$\mathsf{question}(\mathsf{q}_1).$
$\mathsf{question}(\mathsf{q}_2).$
$\mathsf{rejectQuestion}(\mathsf{q}_2).$
$\mathsf{chooseQuestion}(\mathsf{q}_1).$

Für $I_3 = \{\mathsf{question}(\mathsf{q}_1), \mathsf{question}(\mathsf{q}_2), \mathsf{chooseQuestion}(\mathsf{q}_1), \mathsf{rejectQuestion}(\mathsf{q}_2)\}$ ist $P^{I_3}$:

$\mathsf{question}(\mathsf{q}_1).$
$\mathsf{question}(\mathsf{q}_2).$
$\mathsf{rejectQuestion}(\mathsf{q}_2).$
$\mathsf{chooseQuestion}(\mathsf{q}_1).$

Sämtliche entstandenen Programme sind definit, sie bestehen sogar nur aus lauter Fakten.

**Definition 4.2.32 (Stabile Modelle)** Sei $P$ ein variablenfreies normales Programm und $I$ eine Teilmenge der Herbrand-Basis, $I \subseteq \mathcal{B}_P$. Dann ist $I$ ist ein *stabiles Model* (stable model) von $P$, wenn gilt

$$M_{P^I} = I$$

Dies heißt, dass $I$ genau dann ein stabiles Modell ist, wenn es mit dem kleinsten Herbrand-Modell (stabilen Modell) $M_{P'}$ des Programms $P'$ übereinstimmt, welches aus $P$ vermöge der Gelfond-Lifschitz-Transformation von $P$ modulo $I$, also $P^I = P'$, hervorgeht.

**Beispiel 4.2.33 (Stabile Modelle normaler logischer Programme)** Leicht zu sehen ist in Beispiel 4.2.31, dass $I_3$ ein stabiles Modell von $P$ ist, $I_1$ und $I_2$ hingegen sind dies nicht. Es ist außerdem leicht nachzuprüfen, dass $P$ noch weitere stabile Modelle besitzt.

**Bemerkung 4.2.34 (Stabile Modelle normaler logischer Programme)** Ein normales logisches Programm kann eines, keines oder mehrere stabile Modelle besitzen. Ein Nichtdeterminismus kann durch die Existenz mehrerer stabiler Modelle ausgedrückt werden. Die Berechnung stabiler Modelle ist auch für einfache Programmklassen schon NP-vollständig (vgl. [MT91]).

### 4.2.2 Antwortmengenprogrammierung (Answer Set Programming, ASP)

#### 4.2.2.1 Allgemeines

Die Antwortmengenprogrammierung hat ihre Wurzeln in der nicht-monotonen Logik [BDK97] und in der stabilen Modellsemantik normaler logischer Programme (vgl. Abschnitt 4.2.1, [GL88]). Sie stellt eine Erweiterung der stabilen Modellsemantik dar. Hierbei gibt es verschiedene Arten der Erweiterung. Je nach Anwendungsgebiet bzw. wissenschaftlicher Forschungsrichtung wird der Fokus auf die eine oder andere Richtung festgelegt. Zahlreiche Publikationen stellen die Prinzipien und grundlegenden Ideen der Antwortmengenprogrammierung dar. Um nur auf einige hinzuweisen, seien

- Klassische Negation und Disjunktion (z. B. [Min94, GL90, GL94])
- Anzahl- und Gewichtsconstraints (z. B. [NSS99, NS00, SNTS01, Syr04])
- Aggregate (z. B. [DFI$^+$03a, DFI$^+$03b, PDB03])
- Templates (z. B. [IIP$^+$04])
- Präferenzen (z. B. [Bre96, BE98, Bre04, DSTW02])
- Description Logic (z. B. [ELST04])
- Planen (z. B. [Lif99, Lif02, EFL$^+$03])

genannt. Es gibt verschiedene ASP-Systeme, unter denen SMODELS [NS97, Sim99, NSS00] und DLV [FP, CFL$^+$01, LPF$^+$02] als die bekanntesten zu nennen sind.

Im Folgenden soll nur auf die Aspekte bzw. Erweiterungen eingegangen werden, die für die weiteren Ausführungen von Bedeutung sind. Vor allem die Anzahl- und Gewichtsconstraints und der Solver SMODELS wird eine Rolle spielen.

Nachfolgend wird die syntaktische und semantische Erweiterung der normalen Programme und stabilen Modelle um Anzahl- und Gewichtsconstraints aufgezeigt. Näheres hierzu ist unter anderem in

[NS97, NSS99, NS00, SNS02, Syr03] zu finden. Die folgende Darstellung der Abschnitte 4.2.2.2 bis 4.2.2.6 – einige Beispiele ausgenommen – orientiert sich stark an [SNS02]. Ist nichts anderes bemerkt, so sind hier nicht aufgeführte Beweise diesem Artikel zu entnehmen.

Zunächst sollen grundierte Programme betrachtet werden.

#### 4.2.2.2 Antwortmengenprogrammierung grundierter Programme

**Syntax** Im Gegensatz zu den normalen Programmen, in denen Literale die Grundbausteine der Regeln darstellen, treten hier sogenannte Gewichtsconstraints als Grundelemente auf.

**Definition 4.2.35 (Gewichtsconstraint)** Sei $Atoms$ eine Menge von Grundatomen (variablenfreien Atomen). Ein *Gewichtsconstraint (weight constraint)* $C$ hat die Gestalt

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathrm{not}\, b_1 = w_{b_1}, \ldots, \mathrm{not}\, b_m = w_{b_m}\} \leq u$$

wobei $a_1, \ldots, a_n, b_1, \ldots, b_m; n, m \in \mathbb{N}_0$ Atome aus $Atoms$ sind. Jedes Literal im Constraint hat ein ihm zugeordnetes *Gewicht*: $w_{a_i}$ ist das Gewicht des Literals $a_i$, $1 \leq i \leq n$ und $w_{b_j}$ das Gewicht des Literals $\mathrm{not}\, b_j$, $1 \leq j \leq m$. Die Zahlen $l, u \in \mathbb{R}^{\infty}$ stellen die *untere* bzw. *obere Grenze* des Constraints $C$ dar. Wird die untere bzw. obere Grenze nicht notiert, so ist $l = -\infty$ bzw. $u = \infty$ gemeint. Die Menge der im Constraint $C$ auftretenden Literale wird durch $lit(C)$ abgekürzt und die Grenzen werden gelegentlich zur genaueren Beschreibung mit $l(C)$ bzw. $u(C)$ gekennzeichnet.

**Definition 4.2.36 (Gewichtsconstraint-Regel)** Eine *Gewichtsconstraint-Regel (weight constraint rule)* hat die Gestalt

$$C_0 \leftarrow C_1, \ldots, C_n$$

wobei jedes $C_i, 0 \leq i \leq n$ ein Gewichtsconstraint darstellt. Ein *Gewichtsconstraint-Programm* ist eine Menge von Gewichtsconstraint-Regeln.

**Definition 4.2.37 (Spezialfälle von Gewichtsconstraint-Regeln)** Ein *Anzahlconstraint (cardinality constraint)*

$$l\{a_1, \ldots, a_n, \mathrm{not}\, b_1, \ldots, \mathrm{not}\, b_m\}u$$

stellt eine Kurznotation für

$$l \leq \{a_1 = 1, \ldots, a_n = 1, \mathrm{not}\, b_1 = 1, \ldots, \mathrm{not}\, b_m = 1\} \leq u$$

dar. Für das Literal $l$ kann das Constraint

$$1 \leq \{l = 1\}$$

einfach mit $l$ abgekürzt werden.[2] Ein *Integritätsconstraint (integrity constraint)* hat die Form

$$\leftarrow C_1, \ldots, C_n$$

und ist eine Kurzform der entsprechenden Regel mit unerfüllbarem Kopfconstraint $C_0$, etwa

$$1 \leq \{\} \leftarrow C_1, \ldots, C_n$$

wobei jedes $C_i, 1 \leq i \leq n$ ein Gewichtsconstraint darstellt.

**Beispiel 4.2.38 (Gewichtsconstraint-Regel)** Mit Definition 4.2.37 ist folgende Regel in korrekter Syntax:

$$2 \leq \{\mathsf{a} = 8, \mathsf{b} = 2, \text{not } \mathsf{c} = 4\} \leq 11 \ \leftarrow \ \mathsf{b}, 2\{\text{not } \mathsf{d_1}, \text{not } \mathsf{d_2}, \text{not } \mathsf{d_3}\}.$$

In informeller Weise könnte diese Regel wie folgt interpretiert werden: Ist b erfüllt und mindestens zwei der drei Literale $\mathsf{d_i} (1 \leq i \leq 3)$ nicht, so muss das Gesamtgewicht der aus $\{\mathsf{a}, \mathsf{b}, \text{not } \mathsf{c}\}$ erfüllten Literale im Bereich von 2 und 11 liegen. Somit würden beispielsweise die Mengen $\{\mathsf{b}\}$, $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ die Regel erfüllen, $\{\mathsf{a}, \mathsf{b}\}$ jedoch nicht.

**Definition 4.2.39 (Gewichtsconstraint-Programm)** Ein *Gewichtsconstraint-Programm* ist eine Menge von Gewichtsconstraint-Regeln.

**Definition 4.2.40 (Grundlegende Notationen)** Das *Komplement* $not(l)$ eines Literals $l$ ist definiert durch

$$not(l) \ :\equiv \ \begin{cases} \text{not } a & \text{falls } l \equiv a, \quad a \in Atoms \\ a & \text{falls } l \equiv \text{not } a, \quad a \in Atoms \end{cases}$$

Für eine Menge $A$ von Literalen ist

$$\begin{aligned} not(A) &= \{not(a) \mid a \in A\} \\ A^+ &= \{a \in Atoms \mid a \in A\} \\ A^- &= \{a \in Atoms \mid not(a) \in A\} \\ Atoms(A) &= A^+ \cup A^- \end{aligned}$$

Für ein Programm $P$ definiert man

$$Atoms(P) = Atoms(L)$$

wobei $L$ die in $P$ vorkommende Literalmenge ist.[3] Ist keine Verwechslung zu befürchten, so notiert man $Atoms(P)$ oft auch einfach mit $Atoms$.

[2] In diesem Sinne sind Literale auch Gewichtsconstraint-Literale.

[3] Vergleiche Bemerkung 4.2.23

**Semantik** Die Modelle von Gewichtsconstraint-Regeln bzw. Gewichtsconstraint-Programmen sind genau wie bei den stabilen Modellen normaler Programme Mengen von grundierten Atomen. Im Umfeld der normalen Programme sagt man, dass eine Menge von Atomen, die in ihr enthaltenen Atome *erfüllt*.

**Definition 4.2.41 (Erfüllbarkeit von Literalen)** Sei $S \subseteq Atoms$ eine Menge von Grundatomen. $S$ *erfüllt* das Atom $a \in Atoms$, genau wenn $a$ ein Element von $S$ ist, kurz:

$$S \models a : \Longleftrightarrow \ a \in S$$

$S$ *erfüllt* das Literal $\mathrm{not}\, a$ mit $a \in Atoms$, genau wenn $a$ nicht in $S$ enthalten ist, kurz:

$$S \models \mathrm{not}\, a : \Longleftrightarrow \ a \notin S$$

In Erweiterung wird definiert, unter welcher Bedingung eine Menge $S$ ein Gewichtsconstraint erfüllt:

**Definition 4.2.42 (Erfüllbarkeit von Gewichtsconstraints, -regeln und -programme)** Seien $S \subseteq Atoms$ eine Menge von Grundatomen und $C$ das Constraint

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathrm{not}\, b_1 = w_{b_1}, \ldots, \mathrm{not}\, b_m = w_{b_m}\} \leq u$$

$S$ *erfüllt* $C$, kurz

$$S \models C,$$

genau dann, wenn

$$l \leq w(C, S) \leq u, \quad \text{wobei } w(C, S) = \sum_{a_i \in S} w_{a_i} + \sum_{b_i \notin S} w_{b_i}$$

die Summe der Gewichte der Literale aus $C$ darstellt, welche von $S$ erfüllt werden. $S$ *erfüllt* die Gewichtsconstraint-Regel $C_0 \leftarrow C_1, \ldots, C_n$, kurz

$$S \models C_0 \leftarrow C_1, \ldots, C_n,$$

genau dann, wenn

aus $S \models C_1$ und $\ldots$ und $S \models C_n$ folgt: $S \models C_0$.

Das Programm $P$ wird von $S$ *erfüllt*, kurz

$$S \models P,$$

genau dann, wenn

für jede Regel $r \in P$ gilt: $S \models r$

**Beispiel 4.2.43 (Erfüllbarkeit von Constraints und Regeln)** Für das Kopfconstraint $C_0 = 2 \leq \{\mathsf{a} = 8, \mathsf{b} = 2, \mathrm{not}\, \mathsf{c} = 4, \} \leq 11$ der Regel $r$ aus Beispiel 4.2.38 lässt sich

$$w(C_0, \{a, b, c\}) = 8 + 2 = 10, \text{ daher } \{a, b, c\} \models C_0 \text{ und}$$
$$w(C_0, \{a\}) = 8 + 4 = 12, \text{ daher } \{a\} \not\models C_0$$

und

$$\{a, b, c\} \models r,$$
$$\{a\} \models r$$

einfach feststellen.

Die grundlegende Idee der Definition der stabilen Modelle für Gewichtsconstraint-Programme ist es, die Konzepte der Defintion der stabilen Modelle von normalen Programmen auf den allgemeineren Fall zu übertragen. Stabile Modelle sind solche Mengen von Grundatomen, die vom entsprechenden Programm „gestützt“ sind. Auch hier werden zunächst spezielle, einfachere Gewichtsconstraint-Regeln bzw. Gewichtsconstraint-Programme – in Analogie zu den definiten Regeln oder Hornregeln und den definiten Programmen – betrachtet.

**Definition 4.2.44 (Hornconstraint-Regeln, deduktiver Abschluss)** Eine *Hornconstraint-Regel* ist eine Gewichtsconstraint-Regel der Gestalt

$$h \leftarrow C_1, \ldots, C_n,$$

wobei $h$ ein Atom ist und jedes Constraint $C_i, 1 \leq i \leq n$ nur positive Literale und nur eine Bedingung an die untere Grenze enthält (obere Grenze: unendlich). Der *deduktive Abschluss* eines Hornconstraint-Programms $P$ wird definiert als die kleinste Menge von Atomen, die unter $P$ geschlossen ist.[4] Die Eindeutigkeit folgt aus der Tatsache, dass Hornconstraint-Regeln monoton sind: Wird der Rumpf einer Regel von einer Menge $S$ erfüllt, so auch von einer Obermenge von $S$.

**Beispiel 4.2.45 (Deduktiver Abschluss eines Hornconstraint-Programms)** Gegeben sei das Hornconstraint-Programm $P$

$$\mathsf{a} \leftarrow 1 \leq \{\mathsf{a} = 1\}$$
$$\mathsf{b} \leftarrow 0 \leq \{\mathsf{b} = 100\}$$
$$\mathsf{c} \leftarrow 6 \leq \{\mathsf{b} = 5, \mathsf{d} = 1\}, 2 \leq \{\mathsf{b} = 2, \mathsf{a} = 2\}$$

Der deduktive Abschluss von $P$ ist $\{b\}$. Wird die Regel

$$\mathsf{d} \leftarrow 1 \leq \{\mathsf{a} = 1, \mathsf{b} = 1, \mathsf{c} = 1\}$$

hinzugefügt, so ist $\{b, d, c\}$ der deduktive Abschluss.

[4] Vgl. Definition 4.2.27.

**Bemerkung 4.2.46 (Äquivalente Constraints)** Negative Gewichte und negative Literale haben einen engen Bezug zueinander. Sind negative Literale in den Gewichtsconstraints erlaubt, so kann auf negative Gewichte verzichtet werden und umgekehrt. Um die Definition der Semantik stabiler Modelle möglichst einfach zu halten, verzichtet man auf negative Gewichte. Aus diesem Grunde sind zunächst Constraints mit negativen Gewichten in entsprechende Constraints ohne negative Gewichte zu transformieren. Ein Gewichtsconstraint $C$ der Form

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \text{not } b_1 = w_{b_1}, \ldots, \text{not } b_m = w_{b_m}\} \leq u$$

ist äquivalent zu dem Constraint $C'$ mit nicht-negativen Gewichten

$$l + \sum_{w_{a_i}<0} |w_{a_i}| + \sum_{w_{b_i}<0} |w_{b_i}| \leq$$

$$\{\ldots \underbrace{a_i = w_{a_i}}_{w_{a_i}\geq 0}, \ldots, \underbrace{b_j = |w_{b_j}|}_{w_{b_j}<0}, \ldots \underbrace{\text{not } b_k = w_{b_k}}_{w_{b_k}\geq 0}, \ldots \underbrace{\text{not } a_l = |w_{a_l}|}_{w_{a_l}<0}, \ldots\}$$

$$\leq u + \sum_{w_{a_i}<0} |w_{a_i}| + \sum_{w_{b_i}<0} |w_{b_i}|$$

wobei jedes negative Gewicht und das assoziierte Literal komplementiert wird und die Summe der negativen Gewichte zu den Grenzen addiert wird.

Die Äquivalenz von $C$ und $C'$ kann man sich wie folgt überlegen: Um ein *negatives* Gewicht $w_{a_l}$ zu eliminieren wird $C$ in ein äquivalentes Constraint transformiert, indem man das Paar

$$a_l = |w_{a_l}|, \text{not } a_l = |w_{a_l}|$$

hinzufügt und die Grenzen um $|w_{a_l}|$ erhöht. Das Constraint enthält also

$$a_l = w_{a_l}, a_l = |w_{a_l}|, \text{not } a_l = |w_{a_l}|,$$

was zu

$$\text{not } a_l = |w_{a_l}|$$

äquivalent ist. In analoger Weise können alle negativen Gewichte $w_{b_j}$ eliminiert werden. Dies führt zu obiger Darstellung. Hiermit kann ohne Verlust der Ausdrucksstärke eine Beschränkung auf Gewichtsconstraints mit nicht-negativen Gewichten erfolgen.

**Beispiel 4.2.47 (Äquivalente Constraints)** In der Regel

$$1 \leq \{\mathsf{a} = 2\} \leq 2 \leftarrow -1 \leq \{\mathsf{a} = -4, \text{not } \mathsf{b} = -1\} \leq 0$$

können die negativen Gewichte des Rumpfes mit Hilfe des eben beschriebenen Verfahrens eliminiert werden, wodurch man die äquivalente Regel

$$1 \leq \{\mathsf{a} = 2\} \leq 2 \leftarrow 4 \leq \{\text{not } \mathsf{a} = 4, \mathsf{b} = 1\} \leq 5$$

erhält.

Von nun an erfolgt eine Beschränkung auf Gewichtsconstraints mit nicht-negativen Gewichten. Auch im Falle von Gewichtsconstraint-Regeln wird ein Redukt definiert.[5] Dies erfolgt in zwei Schritten:

**Definition 4.2.48 (Redukt eines Gewichtsconstraints)** Sei $C$ ein Gewichtsconstraint (mit nicht-negativen Gewichten) der Form

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathrm{not}\, b_1 = w_{b_1}, \ldots, \mathrm{not}\, b_m = w_{b_m}\} \leq u$$

und $S \subseteq Atoms$ eine Menge von Atomen. Das *Redukt von $C$ modulo $S$*, kurz $C^S$ ist das Constraint

$$l' \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}\}$$

mit der unteren Grenze

$$l' = l - \sum_{b_i \notin S} w_{b_i}.$$

Die negativen Literale werden entfernt und die untere Grenze um die Summe der Gewichte der negativen Literale, die in $S$ erfüllt sind, vermindert. Dies trägt dem Anteil der in $S$ erfüllten negativen Literale an der unteren Grenze Rechnung. Die obere Grenze wird hier ignoriert bzw. auf $\infty$ gesetzt. Diesem Informationsverlust muss dann aber bei der Definition der stabilen Modelle noch Rechnung getragen werden.

**Beispiel 4.2.49 (Redukt eines Gewichtsconstraints)** Das Redukt des Constraints $C$

$$3 \leq \{a_1 = 1, \mathrm{not}\, b_1 = 2, \mathrm{not}\, b_2 = 3\} \leq 5$$

bezüglich der Menge $\{b_1\}$ ist

$$C^{\{b_1\}} : \; 0 \leq \{a_1 = 1\}$$

und bezüglich $\{b_2\}$ ist

$$C^{\{b_2\}} : \; 1 \leq \{a_1 = 1\}.$$

Im nächsten Schritt erfolgt die Definition des Reduktes eines Gewichtsconstraint-Programms. Informell ist das Redukt eines Programms $P$ bezüglich einer Menge $S$ nun eine Menge von Hornconstraint-Regeln, welche eine Regel $r'$ mit dem Kopfatom $p$ enthält, wenn $p$ ein Element von $S$ ist und es eine Regel $r$ in $P$ gibt, in der $p$ im Kopf vorkommt und die oberen Grenzen der Constraints im Regelrumpf von $r$ von $S$ erfüllt sind. Den Rumpf von $r'$ erhält man durch die Redukte der Constraints von $r$. Formal:

[5] Vgl. Definition 4.2.30

**Definition 4.2.50 (Redukt eines Gewichtsconstraint-Programms)** Sei $P$ ein Gewichtsconstraint-Programm und $S$ eine Menge von Atomen. Das *Redukt* $P^S$ von $P$ bezüglich $S$ wird durch

$$P^S = \{p \leftarrow C_1^S, \ldots, C_n^S \mid C_0 \leftarrow C_1, \ldots, C_n \in P,\ p \in lit(C_0) \cap S \text{ und } w(C_i, S) \leq u(C_i),\ 1 \leq i \leq n\}$$

definiert.

**Beispiel 4.2.51 (Redukt eines Gewichtsconstraint-Programms)** Gegeben ist das Programm $P$ in Form der Regel

$$2 \leq \{\mathsf{b} = 2, \mathsf{c} = 3\} \leq 4 \ \leftarrow\ 2 \leq \{\text{not } \mathsf{a} = 2, \mathsf{b} = 4\} \leq 5$$

Das Redukt $P^{\{c\}}$ von $P$ modulo $\{c\}$ ist durch

$$\mathsf{c} \ \leftarrow\ 0 \leq \{\mathsf{b} = 4\}$$

gegeben. Das Redukt $P^{\{b\}}$ von $P$ modulo $\{b\}$ ist leer.

Schließlich wird das stabile Modell eines Programms $P$ als Menge $S$ von Atomen definiert, die alle Regeln von $P$ erfüllt und mit dem deduktiven Abschluss von $P$ bezüglich $S$ übereinstimmt.

**Definition 4.2.52 (Stabile Modelle (Antwortmengen) eines Gewichtsconstraint-Programms)** Sei $P$ ein Gewichtsconstraint-Programm mit nicht-negativen Gewichten und $S$ eine Menge von Atomen. $S$ ist ein *stabiles Modell* von $P$ genau dann, wenn die zwei Bedingungen

$(i)$ $S \models P$,
$(ii)$ $S = cl(P^S)$

erfüllt sind.[6] Das stabile Modell eines Gewichtsconstraint-Programms wird oft synonym auch als *Antwortmenge* oder *anwer set* bezeichnet.

**Beispiel 4.2.53 (Stabile Modelle von Gewichtsconstraint-Programmen)** Gesucht sind die stabilen Modelle des Programms $P$ aus Beispiel 4.2.51. Aus der Forderung $(ii)$ von Definiton 4.2.52 folgt, dass ein stabiles Modell eines Gewichtsconstraints eine Teilmenge der Atome sein muss, die in den Regelköpfen der Regeln des Programms auftauchen, in diesem Fall also $\{b, c\}$

- Die leere Menge ist wegen $\emptyset \not\models P$ kein stabiles Modell von $P$.
- Die Menge $\{b\}$ erfüllt zwar $P$, ist aber wegen leerem Redukt $P^{\{b\}}$ ebenso kein stabiles Modell.
- $S = \{c\}$ ist ein stabiles Modell, da $S \models P$ und $cl(P^S) = cl(\{\mathsf{c} \ \leftarrow\ 0 \leq \{\mathsf{b} = 4\}\}) = S$.
- Tatsächlich ist $\{c\}$ das einzige stabile Modell, wie nun leicht zu sehen ist.

[6] Die erste Forderung ist notwendig, da im Redukt eines Constraints (vgl. Definition 4.2.48) die oberen Grenzen eliminiert werden.

#### 4.2.2.3 Optimierungs-Statements

In realen Anwendungen werden Optimierungsmöglichkeiten im Sinne der Optimierung einer Zielfunktion (Kostenfunktion) benötigt. Aus diesem Grunde werden die im Abschnitt 4.2.2.2 definierten Gewichtsconstraint-Programme um zwei Optimierungs-Statements erweitert.

**Definition 4.2.54 (Optimierungs-Statement)** Ein *minimize-Statement* $M$ der Gestalt

$$minimize\{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \text{not } b_1 = w_{b_1}, \ldots, \text{not } b_m = w_{b_m}\}$$

drückt aus, dass die stabilen Modelle $S$ mit dem kleinsten Gewicht

$$w(M, S) = \sum_{a_i \in S} w_{a_i} + \sum_{b_i \notin S} w_{b_i}$$

gefunden werden sollen. Ein *maximize-Statement* wird dual hierzu definiert.

**Bemerkung 4.2.55 (Induzierte Ordnung)** Durch ein oder mehrere minimize-Statements kann eine Ordnung auf den stabilen Modellen des Programms definiert werden. Ein maximize-Statement kann in ein äquivalentes minimize-Statement transformiert werden. Näheres hierzu siehe [SNS02].

#### 4.2.2.4 Gewichtsconstraint-Regeln mit Variablen und Funktionen

In vielen Anwendungen ist es umständlich, das Programm in grundierter Form zu verfassen. In [SNS02] wird ein Weg besprochen, wie Gewichtsconstraint-Regeln mit Variablen und Funktionen definiert werden können. Es erfolgt eine Einschränkung auf sogenannte *domänenbeschränkte (domain-restricted)* Programme.

Man kann sich vorstellen, dass ein domänenbeschränktes Programm $P$ aus zwei Teilen besteht: $P_{Do}$, der die Domänenprädikate definiert und $P_{Ot}$, der die restlichen Regeln enthält. In $P_{Do}$ muss die Gestalt der Regeln derart eingeschränkt werden, dass $P_{Do}$ ein einziges stabiles Modell $D$ besitzt, das relativ effizient bestimmbar sein sollte. Alle anderen Regeln in $P_{Ot}$ müssen in dem Sinne domänenbeschränkt sein, so dass alle Variablen einer Regel in einem positiven Rupfliteral mit einem Domänenprädikat auftreten. Dann hat $P$ die selben stabilen Modelle wie eine Teilmenge $P_D$ der Grundinstanziierung von $P$, in der für jede grundierte Regel die Instanzen der Domänenprädikat-Literale von $D$ erfüllt werden. Zu beachten ist, dass es in diesem Fall auch möglich ist, Funktionssymbole zuzulassen, ohne die Entscheidbarkeit zu verlieren, da das Programm $P_D$ endlich ist.

Es gibt mehrere Ansätze solche Domänen-Regeln zu definieren. Ein in SMODELS verfolgter Weg ist es, stratifizierte Regeln zu betrachten. Die Einschränkung besteht dann darin, dass jede Variable in einer Regel in einem positiven Rumpfliteral vorkommen muss, das sich in einem echt niedrigerem Stratum als der Kopf der Regel – bezüglich einer Stratifizierung der Domänenregeln bzw. deren Prädikate – befindet [Syr01, SNS02, Syr].

In diesem Zusammenhang werden zur Möglichkeit der kompakten Schreibweise sogenannte *Bedingungsliterale (conditional literal)* der Form

$$l : d$$

definiert, wobei $l$ ein Literal und im Bedingungsteil das zugehörige Prädikat von $d$ ein Domänenprädikat ist. Ein solches Bedingungsliteral stellt eine Abkürzung für die Sequenz aller Instanzen $l'$ des Literals $l$ dar, die durch Substitutionen mit solchen Variablenbelegungen sich entsprechender Variablen entstehen, für die die ensprechende Bedingungsinstanz $d'$ im stabilen Modell von $P_{Do}$ enthalten ist.

Eine Erweiterung um einige built-in-Funktionen wurde ebenso vorgenommen.

**Bemerkung 4.2.56** Die stabile-Modell-Semantik von Gewichtsconstraint-Programmen ist eine Erweiterung der stabilen Modellsemantik normaler Programme.

**Beispiel 4.2.57 (Auswahl von Fragen)** In Anlehnung an den Anwendungsbereich Generierung von Tests wird folgendes Beispielprogramm betrachtet. Die Fragen werden durch question-Fakten repräsentiert.

$$\begin{aligned}
&\mathsf{question}(\mathsf{q}_1, \mathsf{easy}).\\
&\mathsf{question}(\mathsf{q}_2, \mathsf{difficult}).\\
&\mathsf{question}(\mathsf{q}_3, \mathsf{difficult}).\\
&\mathsf{question}(\mathsf{q}_4, \mathsf{easy}).\\
&\mathsf{question}(\mathsf{q}_5, \mathsf{easy}).\\
&\mathsf{question}(\mathsf{q}_6, \mathsf{difficult}).\\
&\mathsf{difficulty}(\mathsf{D}) \leftarrow \mathsf{question}(\mathsf{Q}, \mathsf{D}).
\end{aligned}$$

Die Prädikate question und difficulty sind Domänenprädikate. Mit der zusätzlichen Regel

$$\mathsf{chosenQuestion}(\mathsf{Q}) \leftarrow \mathsf{question}(\mathsf{Q}, \mathsf{D}).$$

kann man ausdrücken, dass sämtliche Fragen gewählt werden sollen. Möchte man allerdings die Gesamtzahl der gewünschten Fragen auf mindestens eine und höchstens drei einschränken, so kann dies anstelle der obigen Regeln durch

$$\begin{aligned}
1\{&\mathsf{chosenQuestion}(\mathsf{q}_1), \mathsf{chosenQuestion}(\mathsf{q}_2), \mathsf{chosenQuestion}(\mathsf{q}_3),\\
&\mathsf{chosenQuestion}(\mathsf{q}_4), \mathsf{chosenQuestion}(\mathsf{q}_5), \mathsf{chosenQuestion}(\mathsf{q}_6)\}3.
\end{aligned}$$

oder mit Hilfe des Bedingungsliterals einfacher durch

$$1\{\mathsf{chosenQuestion}(\mathsf{Q}) : \mathsf{question}(\mathsf{Q}, \mathsf{D})\}3.$$

geschehen. Möchte man stattdessen allerdings von jedem Schwierigkeitsgrad genau zwei Fragen ziehen, so lässt sich dies in der Form

$$2\{\mathsf{chosenQuestion(Q)} : \mathsf{question(Q,D)}\}2 \leftarrow \mathsf{difficulty(D)}.$$

ausdrücken. Hier tritt $D$ als eine Art globale und $Q$ als lokale Variable auf. Die Instanziierung im Bedingungsliteral erfolgt nach der Belegung der Variablen $D$, die ja in dieser Regel auch außerhalb des Bedingungsliterals auftritt. Man kann sich dies wie folgt in zwei Schritten vorstellen: Zuerst wird die Variable $D$ geeignet belegt:

$$2\{\mathsf{chosenQuestion(Q)} : \mathsf{question(Q,easy)}\}2 \leftarrow \mathsf{difficulty(easy)}.$$
$$2\{\mathsf{chosenQuestion(Q)} : \mathsf{question(Q,difficult)}\}2 \leftarrow \mathsf{difficulty(difficult)}.$$

Im zweiten Schritt werden die Bedingungen aufgelöst:

$$2\{\mathsf{chosenQuestion(q_1)}, \mathsf{chosenQuestion(q_4)}, \mathsf{chosenQuestion(q_5)}\}2 \leftarrow \mathsf{difficulty(easy)}.$$
$$2\{\mathsf{chosenQuestion(q_2)}, \mathsf{chosenQuestion(q_3)}, \mathsf{chosenQuestion(q_6)}\}2 \leftarrow \mathsf{difficulty(difficult)}.$$

In vereinfachter Version könnte diese Anforderung auch durch die Regeln

$$2\{\mathsf{chosenQuestion(q_1)}, \mathsf{chosenQuestion(q_4)}, \mathsf{chosenQuestion(q_5)}\}2 \leftarrow$$
$$2\{\mathsf{chosenQuestion(q_2)}, \mathsf{chosenQuestion(q_3)}, \mathsf{chosenQuestion(q_6)}\}2 \leftarrow$$

modelliert werden. Die Formeln „$\mathsf{difficulty(easy)}$“ und „$\mathsf{difficulty(difficult)}$“ sind ja aufgrund der Regel „$\mathsf{difficulty(D)} \leftarrow \mathsf{question(Q,D)}.$“ und der vorgegebenen question-Fakten auf jeden Fall erfüllt.

#### 4.2.2.5 Basisconstraint-Regeln

Eine Teilklasse grundierter Gewichtsconstraint-Regeln bilden die sogenannten Basisconstraint-Regeln. Die Grundidee ist, dass sie als eine Art „Normalform“ der allgemeinen Gewichtsconstraint-Regeln dienen können. Die Basisconstraint-Regeln spielen bei der Implementierung von Gewichtsconstraint-Regeln eine wichtige Rolle. Die Autoren von [SNS02] verfolgen die Idee, eine effiziente Implementierung für die „Kernsprache“ zu entwickeln und allgemeinere Regeln durch die Transformation in die Kernsprache zu behandeln.

**Definition 4.2.58 (Basisconstraint-Regeln)** *Basisconstraint-Regeln (basic constraint rules)* beinhalten zwei Arten von Regeln:

- Eine *Gewichtsregel (weight rule)* hat die Gestalt

  $$h \leftarrow \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathrm{not}\, b_1 = w_{b_1}, \ldots, \mathrm{not}\, b_m = w_{b_m}\} \geq w,$$

  wobei alle auftretenden Gewichte nicht-negativ sind. Diese ist eine Kurzform der Regel

  $$1 \leq \{h = 1\} \leftarrow w \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathrm{not}\, b_1 = w_{b_1}, \ldots, \mathrm{not}\, b_m = w_{b_m}\}.$$

- Eine *Auswahlregel (choice rule)* hat die Gestalt

  $$\{h_1, \ldots, h_k\} \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$$

  und ist eine Kurzform für

  $$0 \leq \{h_1, \ldots, h_k\} \leftarrow n + m \leq \{a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m\}.$$

**Definition 4.2.59 (Kurzformen von Basisconstraint-Regeln)** Häufig auftretende Kurzformen von Basisconstraint-Regeln sind:

- Eine *Anzahlconstraint-Regel (cardinality rule)*

  $$h \leftarrow k\{a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m\}$$

  entspricht der Regel

  $$h \leftarrow \{a_1 = 1, \ldots, a_n = 1, \text{not } b_1 = 1, \ldots, \text{not } b_m = 1\} \geq k.$$

- Eine *normale Regel (normal rule)*

  $$h \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$$

  ist eine Kurzform für

  $$h \leftarrow \{a_1 = 1, \ldots, a_n = 1, \text{not } b_1 = 1, \ldots, \text{not } b_m = 1\} \geq n + m.$$

- Ein *Integritätsconstraint (integrity constraint)*

  $$\leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$$

  entspricht der Regel

  $$f \leftarrow \{a_1 = 1, \ldots, a_n = 1, \text{not } b_1 = 1, \ldots, \text{not } b_m = 1, \text{not } f = 1\} \geq n + m + 1,$$

  wobei $f$ ein neues Atom ist, das nur in Integritätsconstraints verwendet wird. Enthält ein Programm ein derartiges Integritätsconstraint, so kann es kein stabiles Modell $S$ des Programms mit $\{a_1, \ldots, a_n\} \subseteq S$ und $\{b_1, \ldots, b_n\} \cap S = \emptyset$ geben.

**Bemerkung 4.2.60** Ein Gewichtsconstraint-Programm kann in Basisconstraint-Regeln transformiert werden, wobei die Programmgröße nur linear wächst und die stabilen Modelle „in gewissem Sinne“ erhalten bleiben bzw. einander korrespondieren [SNS02].

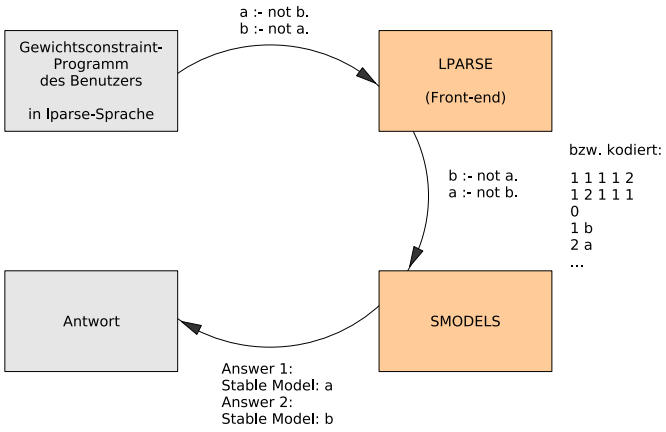



Abbildung 4.1: Datenweg durch LPARSE und SMODELS

#### 4.2.2.6 SMODELS und LPARSE

Die Implementierung von allgemeinen Gewichtsconstraint-Programmen in den Systemen SMODELS und LPARSE beruht auf einer Zwei-Schichten-Architektur:

1. *Front-end* LPARSE*:* Dieses übersetzt das von einem Anwender erstellte Programm mit Variablen in die Kernsprache (Basisconstraint-Regeln) des Systems SMODELS [Syr].
2. SMODELS*:* Als Engine berechnet SMODELS die stabilen Modelle des Programms und kann sie an den Anwender als Antwort ausgeben.

Das Prinzip wird in dem Schaubild der Abbildung 4.1 [Syr] veranschaulicht.

**Bemerkung 4.2.61 (Hinweise zur Notation der Regeln)** Die Darstellung der Regeln in den weiteren Abschnitten, vor allem in Kapitel 6.1, erfolgt einer LPARSE-nahen Notation, welche obiger sehr ähnlich ist – mit kleinen Abweichungen:

- Die allgemeinen Gewichtsconstraints werden mit einer eckigen statt einer runden Klammer notiert. Werden im Gewichtsconstraint keine Gewichte explizit angegeben, so wird das mit einer Gewichtsdeklaration zugeordnete Gewicht angenommen. Erfolgen diesbezüglich keinerlei Angaben, so wird vom Standardgewicht 1 ausgegangen.
- Bei den Integritätsconstraints wird in den Beispielen der Deutlichkeit halber das Symbol $\bot$ (entspricht false) statt dem „leeren“ Kopf notiert.
- Der Pfeil bei einer Regel mit leerem Rumpf kann fakultativ notiert werden.

**Beispiel 4.2.62 (Notation der Regeln in Beispielen)** Im nachfolgenden Datenbestand ist den Fragen eine Bearbeitungsdauer zugeordnet:

$$\begin{aligned}
&\mathsf{question}(\mathsf{q}_1, 5).\\
&\mathsf{question}(\mathsf{q}_2, 10).\\
&\mathsf{question}(\mathsf{q}_3, 1).\\
&\mathsf{question}(\mathsf{q}_4, 5).\\
&\mathsf{question}(\mathsf{q}_5, 1).\\
&\quad \mathsf{duration}(\mathsf{D}) \leftarrow \mathsf{question}(\mathsf{Q}, \mathsf{D}).
\end{aligned}$$

Die Festlegung einer „zu wählenden Gesamtbearbeitungsdauer“ im Bereich von 12 und 16 Minuten geschieht zum einen durch die Angabe der Gewichtung mit Hilfe des Gewichts-Statements

$$\#\mathsf{weight}\ \mathsf{chosenQuestion}(\mathsf{Q}, \mathsf{D}) = \mathsf{D}.$$

und zweitens durch das entsprechende Wählen über die Summe der Gewichte

$$12\ [\mathsf{chosenQuestion}(\mathsf{Q}, \mathsf{D}) : \mathsf{question}(\mathsf{Q}, \mathsf{D})]\ 16 \leftarrow$$

Möchte man die Frage $q_5$ ausschließen, so kann dies durch das Integritätsconstraint

$$\bot \leftarrow \mathsf{chosenQuestion}(\mathsf{q}_5).$$

modelliert werden.

Näheres zur Notation in LPARSE ist in [Syr] zu finden. Anzumerken ist ferner, dass in LPARSE einige built-in-Funktionen definiert sind, wie beispielsweise eq (equal) für den „Gleichheitstest“.

## 4.3 Pfadbasierter Ansatz

Neben der Antwortmengenprogrammierung wird ein pfadbasiertes Paradigma untersucht. Hier soll den strukturellen Gegebenheiten der Anwendungsbereiche Rechnung getragen werden. Als konkrete Methode dienen XML und XPath zusammen mit einigen Möglichkeiten von XQuery. Eine ausführliche Beschreibung der Konzepte wird hier unterlassen, es sollen nur einige grundlegende Aspekte angesprochen werden und dies zum Teil nur beispielbasiert. Der interessierte Leser sei auf die zahlreich vorhandene Literatur verwiesen, unter anderem auf [CDF+04, W3Ca, LS04].

### 4.3.1 XML

XML, die „eXtensible Markup Language“ ist eine Metasprache zur Definiton von Sprachen für Informationseinheiten, die sogenannten XML-Dokumente, und macht Vorgaben für deren Verarbeitung durch Programme. XML-Dokumente bestehen aus Speichereinheiten oder Entities, die entweder analysierte Daten oder nicht analysierte Daten (Stringdaten, Markup) enthalten. XML-Dokumente haben eine logische und eine physische Struktur. Markup beschreibt die logische und physische Struktur des Dokuments. Im Gegensatz zu den relationalen Datenbanken müssen die Daten im XML-Dokument nicht einem fest vorgegebenem Schema entsprechen. Man spricht auch von semistrukturierten Daten.

**Beispiel 4.3.1 (XML-Dokument für ein Bücherverzeichnis)** Als Beispiel wird in der Literatur häufig ein Bücherverzeichnis verwendet, etwa:

```
<bib>
   <book category="STUDIES">
      <title lang="en">Harry Potter</title>
      <author>J. K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
   </book>
   <book category="STUDIES">
      <title lang="de">XQuery</title>
      <author>D. Lehner</author>
      <author>H. Schöning</author>
      <year>2004</year>
      <price>33.00</price>
   </book>
   <book category="STUDIES">
      <title lang="en">Knowledge Representation, Reasoning and
                  Declarative Problem Solving
      </title>
      <author>C. Baral</author>
      <year>2003</year>
      <price>72.23</price>
   </book>
</bib>
```

Listing 4.1: Bücherverzeichnis als XML-Dokument

Das Dokument genügt der Struktur aus Abbildung 4.2. Zu beachten ist, dass hier je Buch eine unterschiedliche Anzahl an Autoren auftreten und Einheiten auch mit Attributen versehen sein können, wie beispielsweise die Kategorie des Buches (category) oder auch die Sprache (lang) als Attribut von Titel (title).

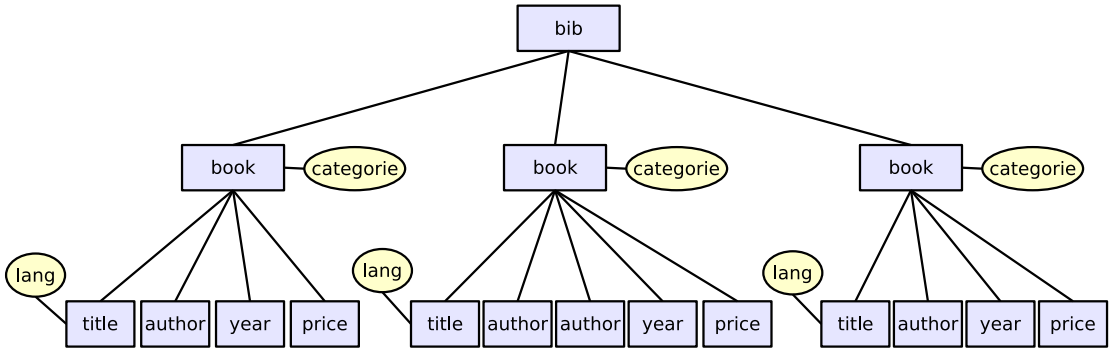



Abbildung 4.2: Struktur eines XML-Dokuments, Beispiel

### 4.3.2 XPath

XPath ist ein Standard, der die Spezifikation von XML-Pfadausdrücken und ihre Auswertung beschreibt. XPath dient der Adressierung von Teilen eines XML-Dokuments. Das Basiskonstrukt von XPath 2.0 ist der Ausdruck (Expression), welcher geschachtelt werden kann. Der Wert eines Ausdrucks ist eine Sequenz und abhängig von seinem Kontext. Der Ausdruckskontext eines gegebenen Ausdrucks besteht aus allen Informationen, die das Resultat des Ausdrucks beeinflussen können. Es wird zwischen dem sogenannten statischen und dem den statischen Kontext umfassenden Evaluationskontext unterschieden.

**Beispiel 4.3.2 (Pfadanfragen)** An das XML-Dokument von Beispiel 4.3.1 können unter anderem folgende Anfragen gestellt werden:

- Liste aller Autoren:

  `/bib/book/author` oder einfach `//author`

- Bücher, die weniger als 40.00 kosten:

  `//book[price<40.00]`

- Titel der Bücher in der Kategorie Studium, die deutschsprachig sind:

  `//book[@category="STUDIES"]/title[@lang="de"]`

- Der erste Autor der Bücher in der Kategorie Studium mit deutschsprachigem Titel:

  `//book[@category="STUDIES"]/title[@lang="de"]/parent::book/author[position()=1]`

Mit Hilfe der XPath-Ausdrücke kann innerhalb des Baumes (Graphen) navigiert werden und so Sequenzen von Ergebnisknoten (-teilbäumen) bestimmt bzw. adressiert werden.

### 4.3.3 XQuery

XQuery ist eine Anfragesprache für XML-Dokumente. Sie ist im Wesentlichen eine *funktionale Sprache mit geschachtelten Queryausdrücken und Variablen* und stellt eine *Erweiterung von XPath* 2.0 dar. Sie ist ferner eine stark typisierte Sprache. Das syntaktische Grundgerüst von XQuery ist der Ausdruck (expression), das Resultat von Ausdrücken ist eine Sequenz. Es gibt Funktionen für den Zugriff auf Eingabedaten.

Nähere Informationen sind der oben schon genannten Literatur [CDF+04, LS04] und auch den W3C-Seiten [W3Ca, W3Cb] zu entnehmen. In diesen Quellen sind unter anderem auch ausführliche Informationen zur Syntax und zur Definiton einer formalen Semantik von XQuery zu finden.

# Kapitel 5

# Modell

Im Folgenden werden die Modelle für die beiden Anwendungsbereiche definiert. Dies erfolgt in verschiedenen Ebenen: Die abstrakten Modelle geben die Sprachelemente, Komponenten und die Struktur der konkreten konzeptuellen Modelle vor. Sie stellen daher eine Art Template für diese dar. Darüber hinaus werden Gemeinsamkeiten und Besonderheiten der Modelle der betrachteten Anwendungsbereiche aufgezeigt. Das Vorgehen orientiert sich an den in Abschnitt 3.2 festgelegten Wunschkriterien.

## 5.1 Kernstruktur der Anwendungsbereiche

Die beiden betrachteten Anwendungsbereiche „Generierung von Tests“ und „Studienberatung und Konsistenzprüfung von Studienordnungen“ weisen im Kern eine gemeinsame Struktur auf. In Abbildung 5.1 ist deren Grobstruktur dargestellt. Sie zeigt das Grundmodell der Modelle, also eine Art Metamodell, in welchem die Bausteine der Modelle definiert werden.

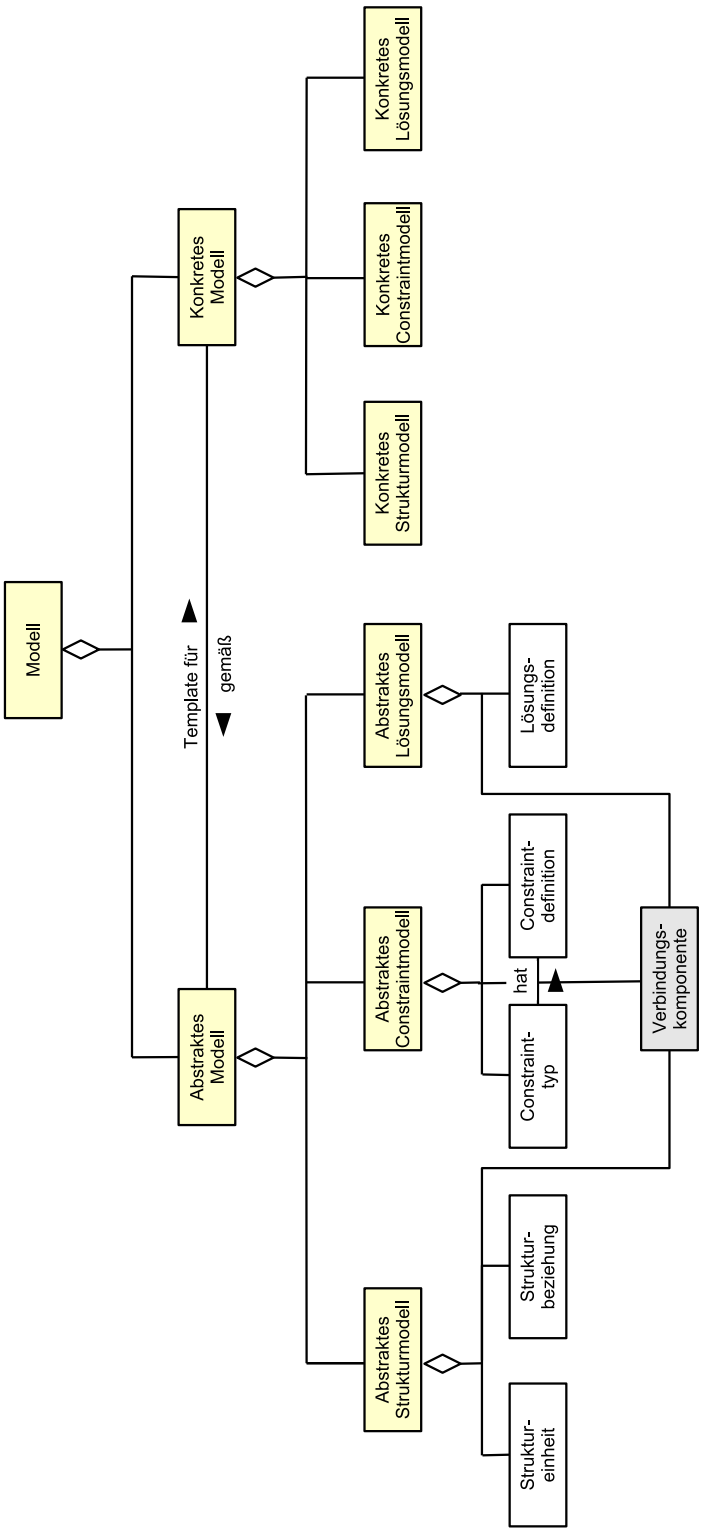



Abbildung 5.1: Grundstruktur des Modells

Wesentliche Aspekte hierbei sind:

- Bei der Modellierung wird zwischen *Abstraktem Modell* und *Konkretem Modell* unterschieden. Eine Instanz des *Abstrakten Modells* stellt ein Template für die Instanzen der *Konkreten Modelle* dar.
- Für die Anwendungsbereiche lassen sich drei Hauptbestandteile der Modellierung finden, nämlich das *Strukturmodell*, das *Constraintmodell* und das *Lösungsmodell*, die zunächst unabhängig voneinander gefunden werden können.
- Gewisse Verbindungsschnittstellen bestehen jedoch natürlicherweise, was durch die *Verbindungskomponente* ausgedrückt wird.
- Im *Strukturmodell* treten *Struktureinheiten* und *Strukturbeziehungen*, im *Constraintmodell* *Constrainttypen* und *Constraintdefinitionen* auf. Das *Lösungsmodell* beinhaltet eine *Lösungsdefinition*.

Grundsätzlich ist noch zu bemerken:

**Bemerkung 5.1.1 (Verschiedene Sichten der Modellierung)** Je nach Blickwinkel ist die objektorientierte, relationale oder mengenmäßige Modellierungsart die adäquate Sichtweise. Gelegentlich ist es von Vorteil, einen Blickrichtungswechsel vorzunehmen.

Bekanntlich können Klassen bei der Modellierung verschiedene Rollen spielen, so kann eine Klasse beispielsweise als Konzept oder auch als Objektmenge aufgefasst werden.

Je nach Ziel werden im Folgenden die verschiedenen Sichtweisen verwendet. Aus dem Kontext geht in der Regel die verwendete Betrachtungsweise hervor.

**Bemerkung 5.1.2 (Notation)** Gemäß Bemerkung 5.1.1 wird eine Klasse $K$ auch als Objektmenge $K = \{o \mid o : K\}$ aufgefasst. Außerdem soll für eine Menge $M$ neben der Notation $m \in M$ auch die gleichbedeutende Notation $M(m)$ erlaubt sein.

### 5.1.1 Gemeinsamkeiten im Strukturmodell

Die Struktur der Elemente der Anwendungsbereiche weisen Gemeinsamkeiten auf, welche im Folgenden skizziert werden sollen:

**Definition 5.1.3 (Elemente des Anwendungsbereichs)** Die Elemente oder Objekte der Anwendungsbereiche können durch Eigenschaften und Eigenschaftswerte näher spezifiziert werden. Möglicherweise können sie durch unterschiedliche Typen ausgezeichnet werden. Die Domänen sind mehr oder weniger stark strukturiert. Typischerweise taucht eine Graphstruktur auf.

Bei den Domänen kann eine Sichtweise als Graph adäquat erscheinen. Aus diesem Grunde werden hierzu einige Begriffe wiederholt, auf welche in späteren Abschnitten zurückgegriffen werden kann.

**Definition 5.1.4 (Begriffe und Notation Graphen)** Ein *Graph* $G = (V, E)$[1] besteht aus einer Menge $V$ von *Knoten* und einer Menge $E$ von *Kanten*. Jede Kante $e \in E$ verbindet zwei Knoten $v_1, v_2 \in V$. Ist die Kante $e \in E$ *gerichtet*, so gehört zu ihr ein geordnetes Paar $(v_1, v_2) \in V \times V$ von Knoten, wobei $e$ von $v_1$ ausgeht und in $v_2$ endet. Ein Graph heißt *gerichtet*, wenn seine Kanten gerichtet sind. Andernfalls heißt er *ungerichtet*. Ein Knoten $v \in V$ eines gerichteten Graphen $(V, E)$ ist ein *Blattknoten*, falls es keine Kante $e \in E$ gibt, die von $v$ ausgeht. Alle anderen Knoten sind *innere Knoten*.

Ein *Pfad* von $v_1 \in V$ nach $v_{n+1} \in V$ der Länge $n \in \mathbb{N}$ ist eine Folge

$$(v_1, v_2), (v_2, v_3), \ldots, (v_n, v_{n+1})$$

von Kanten. Ein *Zyklus* in $(V, E)$ ist ein Pfad, der von einem Knoten $v \in V$ ausgeht und auch in $v$ endet. Ein Graph ohne Zyklen heißt *zyklenfrei* oder *azyklisch*. Ein gerichteter, azyklischer Graph wird oft als *DAG* (directed acyclic graph) bezeichnet.

Ein Knoten $r \in V$ ist eine *Wurzel* eines Graphen $(V, E)$, falls es für jeden Knoten $v \neq r$ einen Pfad von $r$ nach $v$ gibt. Ein Graph mit einer Wurzel $r$ ist ein *Baum*, falls es für jeden Knoten $V \ni v \neq r$ genau einen Pfad von $r$ nach $v$ gibt. Ein Baum ist immer azyklisch und hat genau eine Wurzel. Ein *Wald* ist ein Graph, der aus einem oder mehreren Bäumen besteht.

Die Kanten eines Graphen $G = (V, E)$ sind *markiert*, wenn es eine Menge $\mathcal{L}_E$ von *Kantenmarkierungen* und eine *Markierungsfunktion*

$$f_E : E \to \mathcal{L}_E$$

gibt. $G$ heißt dann *kantenmarkiert*. Die Knoten von $G$ sind *markiert*, wenn es eine Menge $\mathcal{L}_V$ von *Knotenmarkierungen* und eine *Markierungsfunktion*

$$f_V : V \to \mathcal{L}_V$$

gibt. $G$ heißt in diesem Fall *knotenmarkiert*. Für einen Knoten $v \in V$ bezeichnet

$$\mathit{child}(v) = \{w \in V \mid (v, w) \in E\}$$

die Menge der *Kindknoten* von $v$ und

$$\mathit{parent}(v) = \{w \in V \mid (w, v) \in E\}$$

die Menge der *Elternknoten* von $v$. Für eine Menge $\widetilde{V} \subseteq V$ von Knoten lässt sich

$$\mathit{child}(\widetilde{V}) = \{\mathit{child}(v) \mid v \in \widetilde{V}\} \text{ bzw. } \mathit{parent}(\widetilde{V}) = \{\mathit{parent}(v) \mid v \in \widetilde{V}\}$$

[1] An einige Begriffe und Notationen aus der bekannten Graphentheorie soll hier erinnert werden (vgl. z. B. [Die06]).

als die Menge entsprechender Kind- bzw. Elternknoten von Knoten aus $\widetilde{V}$ definieren. Soll die entsprechende Knotenmenge $\widetilde{V}$ selbst auch noch betrachtet werden, so sind die Mengen

$$\mathit{selfChild}(\widetilde{V}) = \mathit{child}(\widetilde{V}) \cup \widetilde{V} \text{ bzw. } \mathit{selfParent}(\widetilde{V}) = \mathit{parent}(\widetilde{V}) \cup \widetilde{V}$$

hilfreich. Darüber hinaus bezeichnet

$$anc(v) = \{w \in V \mid \text{Es gibt einen Pfad von } w \text{ nach } v\}$$

die Menge der *Vorfahrenknoten* von $v$ und

$$desc(v) = \{w \in V \mid \text{Es gibt einen Pfad von } v \text{ nach } w\}$$

die Menge der *Nachfahrenknoten* von $v$. Analog zu $\mathit{selfChild}(\widetilde{V})$ lässt sich $\mathit{selfDesc}(\widetilde{V})$ und $\mathit{selfAnc}(\widetilde{V})$ definieren.

Im Hinblick auf mögliche Lösungen ist es bedeutsam, welche Objekte in ihnen prinzipiell enthalten sein können.

**Definition 5.1.5 (Wählbare Elemente)** Die Menge der *wählbaren Elemente* ist eine endliche – statisch definierte – Teilmenge $\mathit{Selectable} = \{s_i \mid 1 \leq i \leq l, l \in \mathbb{N}\}$ von Objekten des Anwendungs- bzw. Problembereichs.

**Bemerkung 5.1.6 (Wählbare Elemente und Lösungen)** Die wählbaren Elemente sind die Elemente, die in einer Lösung (Modell) des Problems enthalten sein können. Man kann daher „Modell“ auch als Teilmenge der Menge der wählbaren Elemente auffassen.

Andererseits kann eine Lösung auch die Struktur der Elemente (z. B. eine Graphstruktur) beinhalten.

### 5.1.2 Gemeinsamkeiten im Constraintmodell

An die Lösungen der Problemstellungen werden bestimmte Bedingungen gestellt. Bestimmte Arten von Anforderungen treten in beiden Anwendungsbereichen auf. Diese sind im Klassenmodell der Abbildung 5.2 skizziert und werden im Folgenden erläutert.

Die Constraints beziehen sich entweder auf eine Teilmenge der Gesamtheit der sogenannten „wählbaren Elementen“ (vgl. Definition 5.1.5, $\mathit{Selectable}$) oder auf eine Teilmenge aller Constraints ($\mathit{Constraint}$). $\mathit{Selectable}$ bzw. $\mathit{Constraint}$ stellen daher eine Art Grundmenge der Constraints dar. Die entsprechende „referenzierte“ Teilmenge wird im ersten Fall als *Constraintreferenzmenge* bzw. im zweiten Fall als *Constraintmenge* bezeichnet. Es kann eine Klassifikation der Constraints in

- Elementare Constraints ($\mathit{Elementarconstraint}$) und

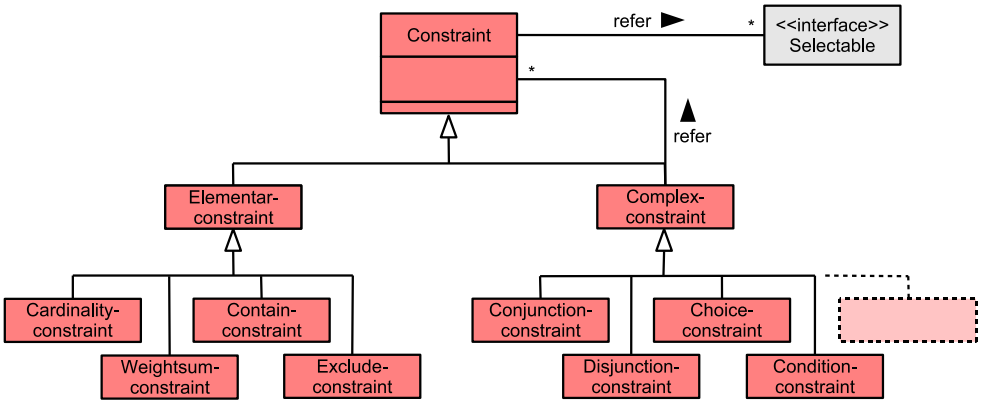



Abbildung 5.2: Gemeinsames Constraintmodell

- Komplexe Constraints (*Complexconstraint*)

erfolgen. Eine feinere Unterteilung für die beiden Anwendungsbereiche erfolgt in folgenden Definitionen:

**Definition 5.1.7 (Elementarconstraint)** Ein *Elementarconstraint* ist ein Constraint der folgenden Art:

- Anzahlconstraint (*Cardinalityconstraint*)
- Vorgabe von obligatorischen Elementen (*Containconstraint*)
- Gewichtssummenconstraint (*Weightsumconstraint*)
- Ausschluss von Elementen, Negation (*Excludeconstraint*)

**Definition 5.1.8 (Komplexconstraint)** Zu den *Komplexconstraints* gehören unter anderem folgende Arten von Constraints:

- Konjunktion von Constraints (*Conjunctionconstraint*)
- Disjunktion von Constraints (*Disjunctionconstraint*)
- Auswahlconstraint (*Choiceconstraint*)
- Bedingtes Constraint (*Conditionconstraint*)

Nun zu detaillierteren Definitionen, in welchen aus Gründen der Kompaktheit und Einfachheit Mengen und Relationen verwendet werden (vgl. auch Bemerkungen 5.1.1, 5.1.12). Um die Constraints möglichst intuitiv zu repräsentieren wird auch Overloading (Überladung) in Kauf genommen.

**Bemerkung 5.1.9 (Abkürzung: Wählbare Elemente und Constraints)** Im Folgenden repräsentiere die endliche Menge $\mathcal{S}$ die Gesamtheit der wählbaren Elemente, also die Menge aller Instanzen vom Typ $Selectable$:

„$\mathcal{S}$= $Selectable$“

In Analogie dazu stelle die endliche Menge $\mathcal{C}$ die Menge der Constraints, also die Menge der Instanzen vom Typ $Constraint$, dar:

„$\mathcal{C}$= $Constraint$“

#### 5.1.2.1 Anzahlconstraints

Ein Anzahlconstraint schränkt die Anzahl von Elementen aus einer vorgegebenen Menge, der Constraintreferenzmenge, ein. Eine relationale Modellierung der Klasse $Cardinalityconstraint$ geschieht in nachfolgender Definition. Zu beachten ist, dass die Notation $cardinalityCons$ überladen wird (Abbildung 5.3).

**Definition 5.1.10 (Relation $cardinalityCons$)** Mit Hilfe der *Anzahlconstraint-Relation*

$$cardinalityCons \subseteq \mathcal{P}(\mathcal{S}) \times (\mathbb{N}_0 \times \mathbb{N}_0^\infty \cup \mathbb{N}_0 \cup \{\text{all}\})$$

werden *Anzahlconstraints* mit folgender Semantik definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ wird festgelegt:

a) Ist $(C, n_1, n_2) \in \mathcal{P}(\mathcal{S}) \times \mathbb{N}_0 \times \mathbb{N}_0^\infty$, so gilt:

$$M \models cardinalityCons(C, n_1, n_2) :\Longleftrightarrow\ n_1 \leq |M \cap C| \leq n_2$$

$M$ erfüllt also genau dann das Constraint, wenn $M$ zwischen $n_1$ und $n_2$ Elemente aus der Constraintreferenzmenge $C$ enthält.

b) Ist $(C, n) \in \mathcal{P}(\mathcal{S}) \times \mathbb{N}_0$, so gilt:

$$M \models cardinalityCons(C, n) :\Longleftrightarrow\ |M \cap C| = n$$

$M$ erfüllt genau dann das Constraint, wenn $M$ genau $n$ Elemente aus der Constraintreferenzmenge $C$ enthält.

c) Ist $(C, \text{all}) \in \mathcal{P}(\mathcal{S}) \times \{\text{all}\}$, so gilt:

$$M \models cardinalityCons(C, \text{all}) : \Longleftrightarrow \; M \cap C = C \text{ (bzw. } C \subseteq M)$$

$M$ erfüllt genau dann das Constraint, wenn $M$ alle Elemente aus der Constraintreferenzmenge $C$ enthält.

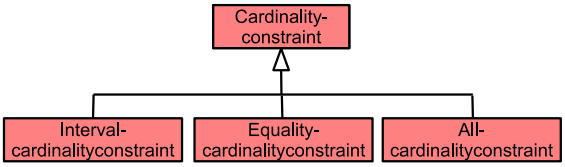



Abbildung 5.3: *Cardinalityconstraint*

**Beispiel 5.1.11 (Relation $cardinalityCons$)** Die Anforderung

„Es müssen alle Elemente aus der Constraintreferenzmenge $C \subseteq \mathcal{S}$ enthalten sein."

kann durch

$cardinalityCons(C, \text{all})$

modelliert werden. Sind hingegen aus dieser Menge nur

„zwischen 2 und 3 Elemente"

zu wählen, so lässt sich dies durch

$cardinalityCons(C, 2, 3)$

ausdrücken.

**Bemerkung 5.1.12 (Äquivalente Repräsentation von Constraints)** Konkrete Anzahlconstraints wie die Constraints $c_1 = cardinalityCons(C, \text{all})$ und $c_2 = cardinalityCons(C, 2, 3)$ aus Beispiel 5.1.11 können auch als konkrete Objekte der Klasse *Cardinalityconstraint* modelliert werden (vgl. Abbildungen 5.2, 5.4). Aus Gründen der Einfachheit in der Darstellung und Argumentation erscheint hier die mengenmäßige und relationale Darstellung der Constraints als der adäquatere Ansatz.

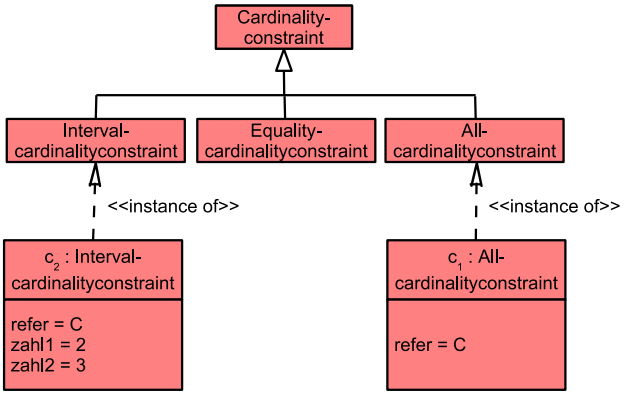



Abbildung 5.4: Objektdarstellung der Constraints aus Beispiel 5.1.11

#### 5.1.2.2 Obligatorische Elemente

Die Forderung, dass gewisse Elemente aus $\mathcal{S}$ auf jeden Fall im Modell enthalten sein müssen, lässt sich mit Hilfe der folgenden Relation (vgl. Klasse $Containconstraint$) modellieren.

**Definition 5.1.13 (Relation $containCons$)** Mit Hilfe der *Enthaltenseins-Relation*

$$containCons \subseteq \mathcal{P}(\mathcal{S})$$

können *obligatorische Elemente* definiert werden. Für Teilmengen $M, C \subseteq \mathcal{S}$ von $\mathcal{S}$ gilt:

$$M \models containCons(C) :\Longleftrightarrow C \subseteq M$$

$M$ erfüllt also genau dann das Constraint, wenn sämtliche Elemente der Constraintreferenzmenge $C$ auch in $M$ enthalten sind.

**Bemerkung 5.1.14 (Äquivalenz von Constraints)** Für Teilmengen $M, C \subseteq \mathcal{S}$ und $n \in \mathbb{N}_0$ gilt:

$$M \models cardinalityCons(C, n, n) \iff M \models cardinalityCons(C, n)$$
$$M \models cardinalityCons(C, \text{all}) \iff M \models containCons(C)$$

Es handelt sich hierbei also jeweils um semantisch äquivalente Constraints.

#### 5.1.2.3 Gewichtssummenconstraints

Gewichtssummenconstraints stellen eine Verallgemeinerung der Anzahlconstraints dar. Im Gegensatz zu den Anzahlconstraints spielen bei diesen nicht die Anzahl der Elemente, sondern deren Gesamtgewicht eine wichtige Rolle. Dies setzt eine Gewichtsfunktion voraus.

**Definition 5.1.15 (Gewichtsfunktion $weight$, Gewichtssummenfunktion $weightSum$)** Durch eine *Gewichtsfunktion* kann den Elementen aus $\mathcal{S}$ ein Gewicht zugeordnet werden.

$$weight : \mathcal{S} \longrightarrow \mathbb{N}_0, \quad s \mapsto weight(s)$$

In seltenen Fällen werden reellwertige Gewichte benötigt. Die entsprechende Definition kann analog erfolgen.

Die Gewichtsfunktion induziert die *Gewichtssummenfunktion*: Für eine Teilmenge $M \subseteq \mathcal{S}$ wird deren Gesamtgewicht $weightSum(M)$ definiert:

$$weightSum : \mathcal{P}(\mathcal{S}) \longrightarrow \mathbb{N}_0, \quad M \mapsto \sum_{s \in M} weight(s)$$

Die Einschränkungen der Funktionen $weight$ bzw. $weightSum$ auf eine Teilmenge von $\mathcal{S}$ bzw. $\mathcal{P}(\mathcal{S})$ können sinngemäß definiert werden und sollen dieselben Bezeichnungen wie die nicht-eingeschränkten Funktionen tragen.

**Bemerkung 5.1.16 (Defaultgewicht)** Das Defaultgewicht der Elemente aus $\mathcal{S}$ betrage 1: Ist die Gewichtsfunktion nicht explizit angegeben, so wird angenommen, dass $weight$ durch

$$weight : \mathcal{S} \longrightarrow \mathbb{N}_0, \quad s \mapsto weight(s) = 1.$$

gegeben ist.

Eine relationale Modellierung der Klasse $Weightsumconstraint$ geschieht in folgender Definition:

**Definition 5.1.17 (Relation $weightsumCons$)** Mit Hilfe der *Gewichtssummenconstraint-Relation*

$$weightsumCons \subseteq \mathcal{P}(\mathcal{S}) \times (\mathbb{N}_0 \times \mathbb{N}_0^\infty \cup \mathbb{N}_0)$$

können bei gegebener Gewichtsfunktion $weight$ und Gewichtssummenfunktion $weightSum$ (vgl. Definition 5.1.15) *Gewichtsconstraints* mit folgender Semantik definiert werden: Für eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ wird festgelegt:

a) Ist $(C, n_1, n_2) \in \mathcal{P}(\mathcal{S}) \times \mathbb{N}_0 \times \mathbb{N}_0^\infty$, so gilt:

$$M \models weightsumCons(C, n_1, n_2) : \Longleftrightarrow\ n_1 \leq weightSum(M \cap C) \leq n_2$$

$M$ erfüllt also genau dann das Constraint, wenn das Gesamtgewicht $weightSum(M \cap C)$ der in $M$ enthaltenen Elemente aus der Constraintreferenzmenge $C$ zwischen $n_1$ und $n_2$ beträgt.

b) Ist $(C, n) \in \mathcal{P}(\mathcal{S}) \times \mathbb{N}_0$, so gilt:

$$M \models weightsumCons(C, n) : \Longleftrightarrow\ weightSum(M \cap C) = n$$

$M$ erfüllt also genau dann das Constraint, wenn das Gesamtgewicht $weightSum(M \cap C)$ der Elemente von $C$ in $M$ genau $n$ beträgt.

Auch hier tritt Überladung auf (vgl. Abbildung 5.5).

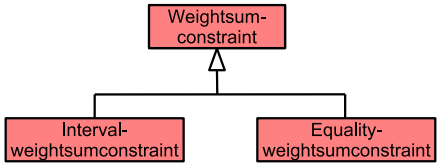



Abbildung 5.5: *Weightsumconstraint*

**Bemerkung 5.1.18 (Relation** $weightsumCons$**)** Sind mehrere, unterschiedliche Gewichtsfunktionen $(weight_1, \ldots, weight_r)$ relevant, so kann zur Unterscheidung die jeweilige Gewichtsfunktion auch als Parameter notiert werden: $weightSum_{weight_i}$ bzw. $weightsumCons_{weight_i}, 1 \leq i \leq r$. Ist aus dem Kontext klar, welche Gewichtsfunktion gerade gemeint ist, so wird deren Notation als Parameter in der Regel unterlassen.

**Bemerkung 5.1.19 (Äquivalenz von Constraints)** Für Teilmengen $M, C \subseteq \mathcal{S}$ von $\mathcal{S}$ und $n \in \mathbb{N}_0$ gilt:

$$M \models weightsumCons(C, n, n) \iff M \models weightsumCons(C, n)$$

Analog zu Bemerkung 5.1.14 handelt es sich bei $weightsumCons(C, n, n)$ und $weightsumCons(C, n)$ um semantisch äquivalente Constraints.

Darüber hinaus sind bei Verwendung der Defaultgewichtsfunktion (vgl. Bemerkung 5.1.16) die Constraints $weightsumCons(C, n_1, n_2)$ und $cardinalityCons(C, n_1, n_2)$ für $n_1 \in \mathbb{N}_0, n_2 \in \mathbb{N}_0^\infty$ semantisch äquivalent.

**Beispiel 5.1.20 (Relation $weightsumCons$)** Die Anforderung

„Das Gesamtgewicht der aus $C$ enthaltenen Elemente betrage mindestens 100."

wird bei vorhandener Gewichtsfunktion $weight$ durch

$$weightsumCons(C, 180, \infty)$$

modelliert.

#### 5.1.2.4 Ausschluss von Elementen

Vor allem in Kombination mit anderen Constraints benötigt man die Möglichkeit auszudrücken, dass gewisse Elemente aus $\mathcal{S}$ *nicht* im Modell enthalten sein dürfen. Hierzu dient folgende Relation (vgl. Klasse $Excludeconstraint$):

**Definition 5.1.21 (Relation $excludeCons$)** Mit Hilfe der *Elementausschluss-Relation*

$$excludeCons \subseteq \mathcal{P}(\mathcal{S})$$

können *nicht erlaubte* oder *auszuschließende* Elemente definiert werden. Für Teilmengen $M, C \subseteq \mathcal{S}$ von $\mathcal{S}$ gilt:

$$M \models excludeCons(C) :\Longleftrightarrow C \cap M = \emptyset \qquad (\text{bzw. } M \subseteq \mathcal{S} \backslash C)$$

$M$ erfüllt also genau dann das Constraint, wenn $M$ keine Elemente aus der Constraintreferenzmenge $C$ enthält.

Nach der Definition der vier Elementarconstraints folgt ein erster Einblick in relevante Komplexconstraints.

#### 5.1.2.5 Konjunktion und Disjunktion von Constraints

Der Vollständigkeit halber werden die wie üblich definierte Konjunktion und Disjunktion von Constraints (Klassen: $Conjuctionconstraint$, $Disjunctionconstraint$) angeführt:

**Definition 5.1.22 (Konjunktion und Disjunktion von Constraints)** Eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ *erfüllt* die *Konjunktion* $c_1 \wedge c_2$ *der Constraints* $c_1$ und $c_2$ genau dann, wenn $M$ das Constraint $c_1$ und das Constraint $c_2$ erfüllt[2]:

$$\begin{aligned} \wedge \quad &\subseteq \quad \mathcal{C} \times \mathcal{C} \\ &M \models c_1 \wedge c_2 :\Longleftrightarrow \; M \models c_1 \text{ und } M \models c_2 \end{aligned}$$

Die induktive Erweiterung dieser Definition auf eine endliche Indexmenge $I$ ergibt:

$$\begin{aligned} \bigwedge \quad &\subseteq \quad \mathcal{C}^I \\ &M \models \bigwedge_{i \in I} c_i :\Longleftrightarrow \text{ Für alle } i \in I \text{ gilt: } M \models c_i \end{aligned}$$

Eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ *erfüllt* die Disjunktion $c_1 \vee c_2$ der Constraints $c_1$ und $c_2$ genau dann, wenn $M$ das Constraint $c_1$ oder das Constraint $c_2$, also mindestens eines dieser Constraints erfüllt:

$$\begin{aligned} \vee \quad &\subseteq \quad \mathcal{C} \times \mathcal{C} \\ &M \models c_1 \vee c_2 :\Longleftrightarrow \; M \models c_1 \text{ oder } M \models c_2 \end{aligned}$$

Die induktive Erweiterung dieser Definition auf eine endliche Indexmenge $I$ ergibt:

$$\begin{aligned} \bigvee \quad &\subseteq \quad \mathcal{C}^I \\ &M \models \bigvee_{i \in I} c_i :\Longleftrightarrow \text{ Es gibt ein } i \in I \text{ mit: } M \models c_i \end{aligned}$$

**Beispiel 5.1.23 (Konjunktion und Disjunktion von Constraints)** Die Anforderung

> „Aus der Menge $C \subseteq \mathcal{S}$ sind genau $n \in \mathbb{N}$ Elemente zu wählen, wobei auf jeden Fall die Elemente $x_1, \ldots, x_m, m \in \mathbb{N}$, enthalten sein müssen."

kann durch die Konjunktion

$$\mathit{cardinalityCons}(C, n) \wedge \mathit{containCons}(\{x_1, \ldots, x_m\})$$

modelliert werden. Ist nur eine der beiden Teil-Anforderungen zu erfüllen, so lässt sich dies durch

$$\mathit{cardinalityCons}(C, n) \vee \mathit{containCons}(\{x_1, \ldots, x_m\})$$

ausdrücken.

[2] Vgl. Bemerkung 5.1.9: $\mathcal{C}$ ist die Menge der Instanzen vom Typ $\mathit{Constraint}$, also die Menge der Constraints.

#### 5.1.2.6 Auswahlconstraints

Gelegentlich möchte man auch ausdrücken, dass nur eine gewisse Anzahl aus einer bestimmten Menge von Constraints (Constraintmenge) gültig sein soll oder muss. Aus einer Menge von Constraints ist also eine Teilmenge zu wählen. Die Referenzmenge für das Constraint ist in diesem Fall daher nicht eine Teilmenge der wählbaren Elemente $\mathcal{S}$, sondern eine Teilmenge der Menge der Constraints $\mathcal{C}$.[3] Es handelt sich also um ein Metaconstraint.

In der Regel möchte man semantisch äquivalente Constraints miteinander identifizieren. Implizit wird dann davon ausgegangen, dass die Referenzmenge keine semantisch äquivalenten Constraints enthält bzw. dass semantisch äquivalente Constraints in der Art miteinander identifiziert werden, dass sie nur einmal gewertet werden.[4]

Folgende Relation wird eingeführt (siehe Klasse $Choiceconstraint$):

**Definition 5.1.24 (Relation $choiceCons$)** Mit Hilfe der *Auswahlconstraint-Relation*

$$choiceCons \subseteq \mathcal{P}(\mathcal{C}) \times (\mathbb{N}_0 \times \mathbb{N}_0^\infty \cup \mathbb{N}_0)$$

werden *Auswahlconstraints* mit folgender Semantik definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ wird festgelegt:

a) Ist $(C, n_1, n_2) \in \mathcal{P}(\mathcal{C}) \times \mathbb{N}_0 \times \mathbb{N}_0^\infty$, so gilt:

$$M \models choiceCons(C, n_1, n_2) :\Longleftrightarrow\ n_1 \leq |\{c \in C \mid M \models c\}| \leq n_2$$

$M$ erfüllt genau dann das Auswahlconstraint, wenn $M$ zwischen $n_1$ und $n_2$ Constraints aus der Constraintmenge $C$ erfüllt.

b) Ist $(C, n) \in \mathcal{P}(\mathcal{C}) \times \mathbb{N}_0$, so gilt:

$$M \models choiceCons(C, n) :\Longleftrightarrow\ |\{c \in C \mid M \models c\}| = n$$

$M$ erfüllt genau dann das Auswahlconstraint, wenn $M$ genau $n$ Constraints aus der Constraintmenge $C$ erfüllt.

**Beispiel 5.1.25 (Relation $choiceCons$)** Es sind folgende Anforderungen und die Constraintreferenzmengen $C_1, C_2, C_3 \subseteq \mathcal{S}$ gegeben:

[3] Vgl. Bemerkung 5.1.9: $\mathcal{C}$ ist die Menge der Instanzen vom Typ $Constraint$, also die Menge der Constraints.

[4] Formales Vorgehen: Auf $\mathcal{C}$ wird eine Äquivalenzrelation $\sim$ definiert, wobei $c_1 \sim c_2$ genau dann gelten soll, wenn $c_1$ und $c_2$ semantisch äquivalent sind. Statt $\mathcal{C}$ ist dann $\mathcal{C}/\sim$ und statt statt $C \subseteq \mathcal{C}$ ist $C/\sim$ relevant. Der Einfachheit der Darstellung halber wird hier aber $\mathcal{C}/\sim$ und $\mathcal{C}$ notationell nicht unterschieden.

- „Es sind genau drei Elemente aus $C_1$ zu wählen“:

  $$c_1 := \mathit{cardinalityCons}(C_1, 3)$$

- „Es sind ein Element aus $C_1$ und zwei Elemente aus $C_2$ zu wählen“:

  $$c_2 := \mathit{cardinalityCons}(C_1, 1) \wedge \mathit{cardinalityCons}(C_2, 2)$$

- „Es sind zwei Elemente aus $C_3$ zu wählen, wobei auf jeden Fall das Element $x \in C_3$ enthalten sein muss“:

  $$c_3 := \mathit{cardinalityCons}(C_3, 2) \wedge \mathit{containCons}(\{x\})$$

In einem gesuchten Modell müssen möglicherweise nicht alle Constraints erfüllt sein. Die Anforderung

„Es müssen genau zwei der drei Constraints $c_1$, $c_2$, $c_3$ erfüllt sein.“

lässt sich mit Hilfe des folgenden Auswahlconstraints modellieren:

$\mathit{choiceCons}(C, 2)$, wobei $C := \{c_1, c_2, c_3\}$.

Müssten alle drei Constraints erfüllt sein, so könnte dies sowohl durch die Konjunktion $c_1 \wedge c_2 \wedge c_3$ als auch durch das Auswahlconstraint $\mathit{choiceCons}(C, 3)$ ausgedrückt werden. Ist mindestens eines der Constraints zu erfüllen, so kann dies durch $c_1 \vee c_2 \vee c_3$ oder auch durch $\mathit{choiceCons}(C, 1, \infty)$ modelliert werden.

#### 5.1.2.7 Bedingte Constraints

Bestimmte Constraints müssen nur dann erfüllt sein, wenn ein anderes, vorausgesetztes Constraint erfüllt ist. Hierzu folgende Relation (siehe Klasse $\mathit{Conditionconstraint}$):

**Definition 5.1.26 (Relation $\mathit{conditionCons}$)** Mit Hilfe der *Bedingungsconstraint-Relation*

$$\mathit{conditionCons} \subseteq \mathcal{C} \times \mathcal{C}$$

werden *Bedingungsconstraints* mit folgender Semantik definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ und Constraints $c_1, c_2 \in \mathcal{C}$ gilt:

$$M \models \mathit{conditionCons}(c_1, c_2) : \Longleftrightarrow \text{ Aus } M \models c_2 \text{ folgt: } M \models c_1$$

$M$ erfüllt genau dann das Bedingungsconstraint, wenn gilt: Erfüllt die Menge $M$ das Constraint $c_2$, so erfüllt sie auch das Constraint $c_1$. Anstelle von $conditionCons(c_1, c_2)$ wird abkürzend folgende Notation zugelassen:

$$c_1 \longleftarrow c_2$$

**Beispiel 5.1.27 (Relation $conditionCons$)** Die Anforderung

> „In einem Modell muss das Element $s \in \mathcal{S}$ enthalten sein, wenn aus der Menge $C := \{s_1, \ldots, s_7\} \subseteq \mathcal{S}$ höchstens drei Elemente im Modell enthalten sind."

kann durch

$$conditionCons(containCons(\{s\}), cardinalityCons(C, 0, 3))$$

bzw.

$$containCons(\{s\}) \longleftarrow cardinalityCons(C, 0, 3)$$

modelliert werden.

Gerade im Anwendungsbereich „Studienberatung und Konsistenzprüfung von Studienordnungen" gibt es noch weitere komplexe Constraints. Natürlich könnte man diese ohne Weiteres und ohne größere Schwierigkeiten auch auf den Anwendungsbereich „Generierung von Tests" übertragen, was aber in der Praxis in der Regel nicht notwendig ist. Deshalb wird im Folgenden der Weg eingeschlagen, dass die zusätzlichen Komplexconstraints an relevanter Stelle definiert und erläutert werden. Dort lässt sich ihre Relevanz mit zahlreichen Beispielen untermauern.

#### 5.1.2.8 Syntaktische Variable für Constraints

Um Bezüge zu mehreren Constaints bzw. Constraintarten herstellen zu können und aus Gründen der Einfachheit und Kompaktheit wird folgende Notation eingeführt:

**Bemerkung 5.1.28 (Syntaktische Variable für Constraints)** Die syntaktische Variable $\mathfrak{C}$ dient als Variable für die verschiedenen Constraintarten/-typen oder eine Teilmenge hiervon. Die Verwendung von Parametern soll erlaubt sein, wenn aus dem Kontext deren mögliche Belegungen hervorgeht.

Beispielsweise würde für die Constraintarten $\mathfrak{C} \in \{cardinalityCons, weightsumCons, containCons\}$, die Constraintreferenzmenge $C \subseteq \mathcal{S}$ und $n \in \mathbb{N}_0$ die Formulierung

$$\mathfrak{C}(C, n)$$

erlaubt sein. Dieser Ausdruck steht für die konkreten Constraints

$cardinalityCons(C, n)$ und $weightsumCons(C, n)$.

Da keines der $contain$-Constraints eine natürliche Zahl als Parameter besitzt, kann der Ausdruck $\mathfrak{C}(C, n)$ kein derartiges Constraint repräsentieren. Im Gegensatz hierzu steht

$$\mathfrak{C}(C)$$

für die konkreten Constraints

$$cardinalityCons(C, n_1, n_2),\ cardinalityCons(C, n),\ cardinalityCons(C, \text{all}),$$
$$weightsumCons(C, n_1, n_2),\ weightsumCons(C, n),$$
$$containCons(C)$$

mit beliebigen $n, n_1 \in \mathbb{N}_0, n_2 \in N_0^\infty$.

### 5.1.3 Gemeinsamkeiten im Lösungsmodell

Eine Lösung oder ein Modell eines vorliegenden Problems ($Specification$) ist eine Teilmenge der Menge der wählbaren Elemente $\mathcal{S}$ ($Selectable$), die die Anforderungen (Bedingungen und Constraints) der Spezifikation erfüllt (vgl. Abbildung 5.6). Dies wird durch die Klasse $Model$ bzw. deren Objekte modelliert. Für eine Spezifikation kann es mehrere Lösungen geben.

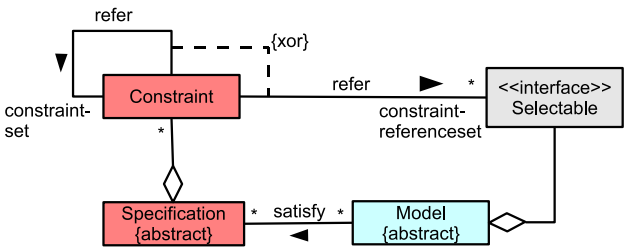



Abbildung 5.6: Gemeinsames Lösungsmodell

## 5.2 Generierung von Tests

Ausgehend von den Abschnitten 2.3 und 5.1 wird im Folgenden ein Modell für den Anwendungsbereich „Generierung von Tests“ definiert.

### 5.2.1 Strukturmodell

**Bemerkung 5.2.1 (Abstraktion von Fragetext und Antworten)** Es wird angenommen, dass eine Frage durch ihren Identifier $q$ eindeutig festgelegt ist. Von der konkreten Ausprägung der Frage wie dem Fragetext und den möglichen Antworten wird abstrahiert (siehe Bemerkung 2.3.3).

Die nachfolgende Abbildung 5.7 zeigt ein mögliches abstraktes Klassendiagramm des Strukturmodells[5], welches im Folgenden näher erläutert wird. Ein unter bestimmten Voraussetzungen alternatives abstraktes Modell wird in Bemerkung 5.2.17 angesprochen.

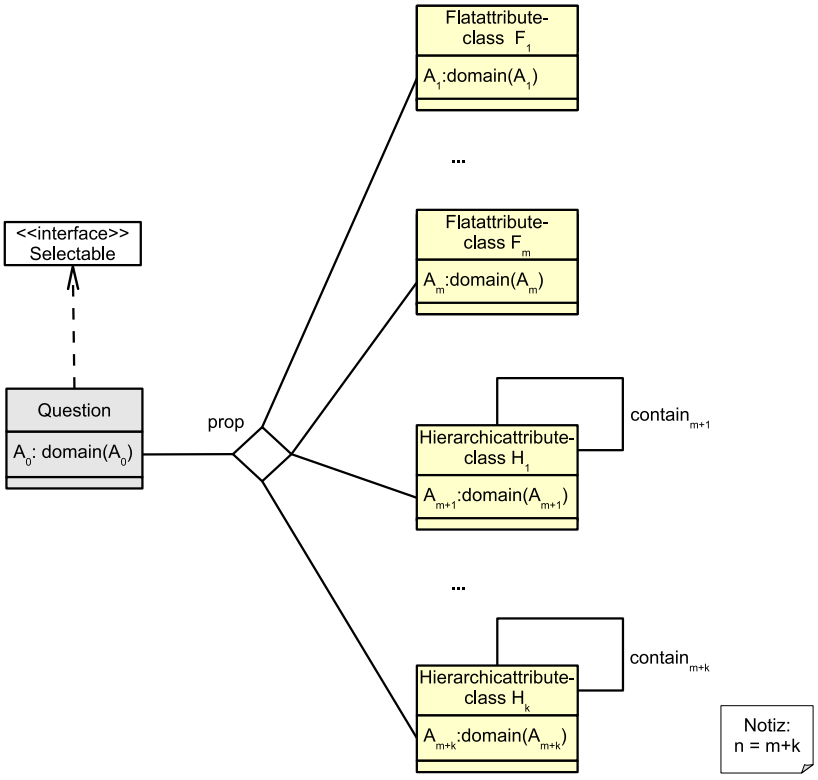



Abbildung 5.7: Abstraktes Strukturmodell, Klassendiagramm

Ein Kernelement des Strukturmodells bildet der Fragenpool mit den darin enthaltenen Fragen, welche durch Attribute bzw. Attributklassen näher spezifiziert sind.

[5] Notation in der Unified Modeling Language (UML) [RJB98]

**Definition 5.2.2 (Fragenkatalog und Attribute)** Ein *Fragenkatalog (Fragenpool, Questionpool)* ist eine endliche Menge

$$Question = \{q_i \mid 1 \leq i \leq k, k \in \mathbb{N}\}$$

von Fragen(objekten). Diesen ist eine endliche *Sequenz von Attributen*

$$\mathcal{A} = A_0, A_1, \ldots, A_n, n \in \mathbb{N}_0$$

zugeordnet, nämlich die $m + 1$ ($0 \leq m \leq n$) *flachen* Attribute $A_0, \ldots, A_m$ und die $k = n - m$ *hierarchischen* Attribute $A_{m+1}, \ldots, A_n$. Das Attribut $A_0$ ist das Identifier-Attribut der Fragen gemäß Bemerkung 5.2.1.

**Bemerkung 5.2.3 (Modellierung Fragenkatalog und Attribute)** Wie auch aus der Abbildung 5.7 ersichtlich wird, wird im Folgenden nicht die einfache Repräsentation einer Frage als Instanz einer Klasse mit Attributen gewählt, sondern eine *explizite* Assoziation von Objekt und Attributen vorgezogen (Klasse $Question$, Attributklassen $F_1, \ldots, F_m, H_1, \ldots, H_k$, Assoziation $prop$).

**Bemerkung 5.2.4 (Abkürzung Fragenpool)** Im Weiteren soll für die Menge $Question$ der Fragenobjekte abkürzend auch $Q$ stehen dürfen.

**Definition 5.2.5 (Domäne und Wertebereich)** Die *Domäne* eines Attributes $A_i$ ($0 \leq i \leq n, n \in \mathbb{N}$)

$$domain(A_i)$$

ist die endliche Menge von Werten, die einer Frage – direkt oder indirekt – zugeordnet werden können. Der *Wertebereich*

$$range(A_i)$$

eines Attributes $A_i$ ist die Teilmenge der Domäne $range(A_i) \subseteq domain(A_i)$, deren Elemente den Fragen direkt als Wert zugeordnet werden können.

**Bemerkung 5.2.6 (Domäne und Wertebereich)** Im Falle eines flachen Attributes $A_i, 0 \leq i \leq m$ sind Domäne und Wertebereich identisch, $domain(A_i) = range(A_i)$. Im Falle eines hierarchischen Attributes $A_i, m + 1 \leq i \leq n$ ist $domain(A_i) \supset range(A_i)$.

**Beispiel 5.2.7 (Fragenkatalog, Attribute und Attributwerte)** Die Fragen aus Abbildung 5.8 werden durch die flachen Attribute

`Id`, `Duration`, `Difficulty`

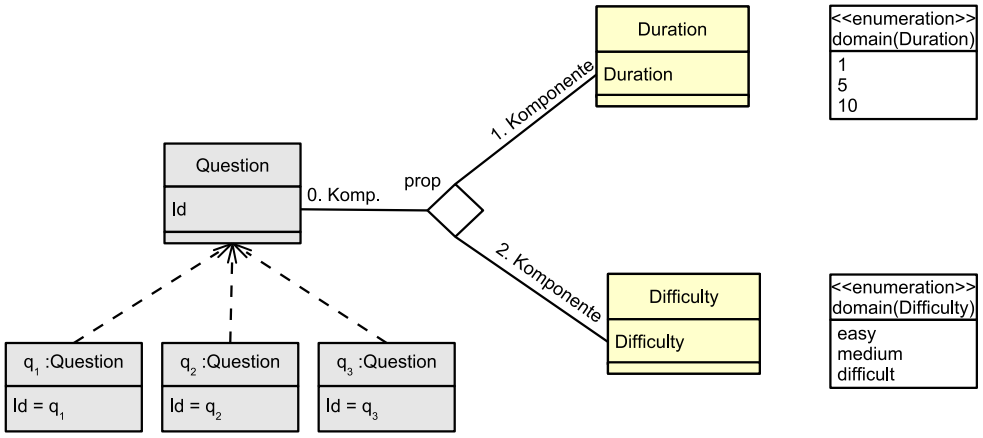



Abbildung 5.8: Beispiel: Fragen und Attribute

beschrieben. Für die Wertebereiche gilt:

$$domain(\texttt{Duration}) = range(\texttt{Duration}) = \{1, 5, 10\},$$
$$domain(\texttt{Difficulty}) = range(\texttt{Difficulty}) = \{\textit{easy}, \textit{medium}, \textit{difficult}\}$$

**Definition 5.2.8 (Assoziation $prop$)** Vermöge der mehrstelligen *Assoziation* $prop$ erfolgt eine direkte Zuordnung der Fragenobjekte zu den $Flat$- und $Hierarchicattributeclass$-Objekten (vgl. Abbildung 5.7):

$$prop \subseteq Q \times \prod_{i=1}^{m} F_i \times \prod_{i=1}^{k} H_i$$

Diese induziert eine Zuordung entsprechender direkter Attributwerte:

$$prop \subseteq \prod_{i=0}^{n} range(A_i)$$

Diese Zuordnungen können miteinander identifiziert werden. Zu beachten ist, dass sich hierbei $Q$ und $range(A_0)$ entsprechen (vgl. Bemerkung 5.2.1). Möchte man dies verdeutlichen, so soll auch die Notation

$$prop \subseteq Q \times \prod_{i=1}^{n} range(A_i)$$

erlaubt sein.

**Beispiel 5.2.9 (Assoziation $prop$)** Die Fragen aus Beispiel 5.2.7 sollen näher beschreiben werden: Frage $q_1$ ist eine *leichte (easy)* Frage mit *1* Minute Bearbeitungszeit. Für die Fragen $q_2$ und $q_3$ ist eine Bearbeitungsdauer von *5* Minuten vorgesehen, wobei $q_2$ eine *schwierige (difficult)* und $q_3$ eine *mittelschwere (medium)* Frage ist. Dieser Sachverhalt drückt sich durch die Tupel

$$(q_1, 1, \textit{easy}),\ (q_2, 5, \textit{difficult}),\ (q_3, 5, \textit{medium}) \in prop$$

aus (vgl. Abbildung 5.9).

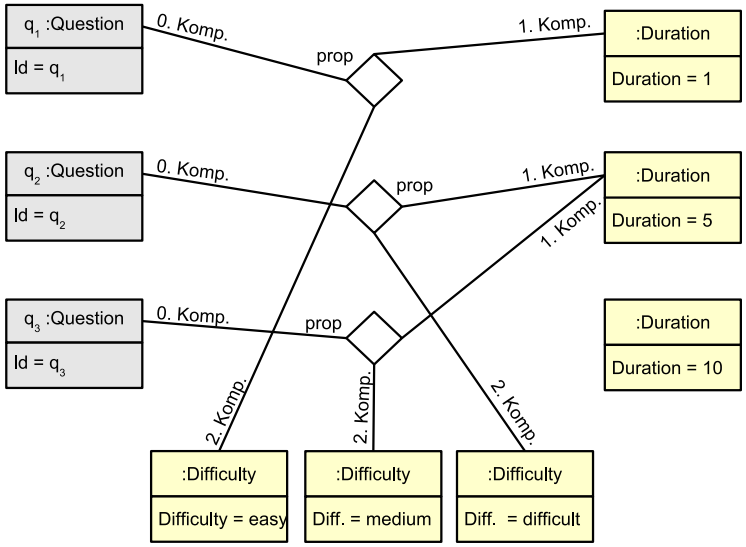



Abbildung 5.9: Beispiel: Assoziation $prop$, Objektdiagramm

**Definition 5.2.10 (Assoziation $contain$)** Vermöge von rekursiven *Enthaltenseins-Assoziationen* kann eine hierarchische Struktur auf den $Hierarchicattributeclass$-Objekten definiert werden:

$$contain_{m+i} \subseteq H_i \times H_i, \quad 0 \leq i \leq k \quad (n = m + k)$$

Diese induziert eine rekursive Zuordung der Attributwerte von $A_{m+i}$:

$$contain_{m+i} \subseteq domain(A_{m+i}) \times domain(A_{m+i})$$

Diese Zuordnungen können miteinander identifiziert werden.

**Beispiel 5.2.11 (Hierarchisches Attribut)** Die Fragen aus den Beispielen 5.2.7 und 5.2.9 sollen weiter spezifiziert werden. Dies kann durch das

hierarchische Attribut `topicId`

modelliert werden, welches die von einer Frage behandelten Themenbereiche repräsentieren möge. Hierbei ist auf den $Topic$-Objekten eine Hierarchie definiert: Themen enthalten Unterthemen und werden Oberthemen zugeordnet. Abbildung 5.10[6] verdeutlicht diese Struktur. Beispielsweise ist hier das Thema *topicC* ein Unterthema der Themen *topicX* und *topicY*, welche ihrerseits in *topicTop* enthalten sind.

Zu beachten ist, dass nicht notwendigerweise eine Baumstruktur vorliegen muss, so dass ein Thema Unterthema mehrerer Themen sein kann. Sowohl *topicX* als auch *topicY* enthält das Unterthema *topicC*, $contain$ ist somit eine $n : m$-Zuordnung. Der Wertebereich von `topicId` ist hier durch

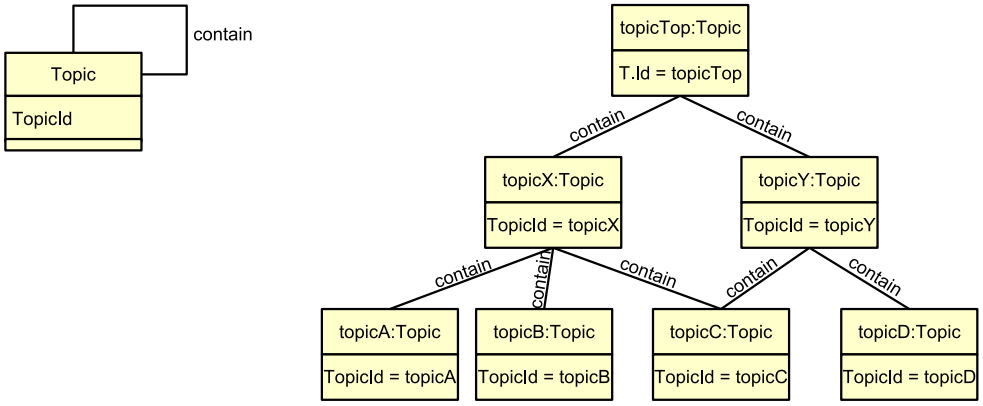



Abbildung 5.10: Beispiel: Hierarchisches Attribut, Klassen- und Objektdiagramm

$$range(\texttt{topicId}) = \{topicA, topicB, topicC, topicD\}$$

und die Domäne durch

$$domain(\texttt{topicId}) = range(\texttt{topicId}) \cup \{topicX, topicY, topicTop\}$$

gegeben.

**Bemerkung 5.2.12 (Graphsicht einer Hierarchie)** Bei einem hierarchischen Attribut $A_i, m \leq i \leq n$ wird mittels der rekursiven Assoziation $contain_i$ eine hierarchische Struktur auf der Domäne $domain(A_i)$ definiert. Das entsprechende Objektdiagramm kann als

Hierarchiegraph $H_i = (V_i, E_i)$

[6] Der Übersichtlichkeit halber wird in dieser Graphik auf die instance-of-Pfeile verzichtet.

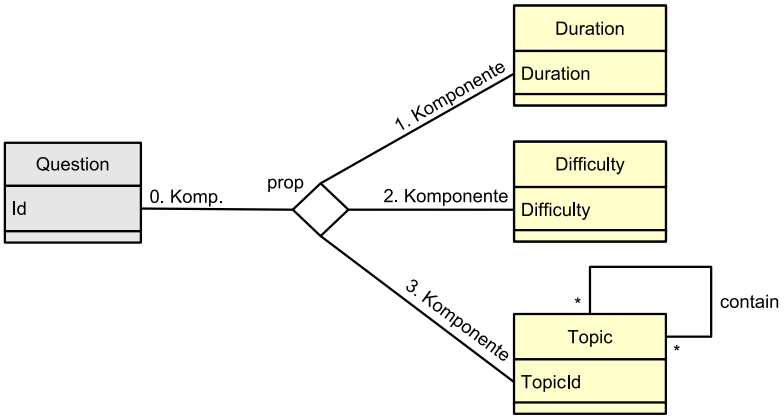



Abbildung 5.11: Beispiel: Strukturmodell-Klassendiagramm

mit der Knotenmenge $V_i = domain(A_i)$ und der Kantenmenge $E_i \subseteq domain(A_i) \times domain(A_i)$ aufgefasst werden (vgl. Definition 5.1.4). Hierbei gilt: $e = (v, w) \in E_i \iff contain_i(v, w)$.

**Beispiel 5.2.13 (Erweitertes Strukturmodell)** Aufbauend auf Beispiel 5.2.11 wird eine geeignete Erweiterung des Strukturmodells aus den Beispielen 5.2.7 und 5.2.9 vorgenommen: Das entsprechende Klassendiagramm enthält mit $Topic$ eine zusätzliche Attributklasse (vgl. Abbildung 5.11). Ein mögliches zugehöriges Objektdiagramm wird in Abbildung 5.12 gezeigt.

Hier ist die Frage $q_1$ dem Thema *topicA* und $q_3$ dem Thema *topicC* zugeordnet. Für Thema *topicB* ist $q_2$ eine *mittelschwere (medium)* Frage und für Thema *topicC* eine *schwierige (difficult)* Frage:

$(q_1, 1, \mathit{easy}, \mathit{topicA})$,
$(q_2, 5, \mathit{medium}, \mathit{topicB}), (q_2, 5, \mathit{difficult}, \mathit{topicC})$,
$(q_3, 5, \mathit{medium}, \mathit{topicC}) \in prop$

Zu beachten ist, dass eine Frage mehreren Themen zugeordnet sein kann. Wie man am Beispiel der Frage $q_2$ erkennen kann, kann sogar eine Art Kontextabhängigkeit der Attributwerte modelliert werden: Für *topicB* ist $q_2$ eine *mittelschwere (medium)* Frage, für *topicC* ist sie *schwierig (difficult)*. Die Frage $q_2$ taucht hier somit in mehreren $prop$-Tupeln auf.

**Definition 5.2.14 (Relation $propTrans$)** Die Relation $propTrans$ definiert eine Erweiterung („transitive Hülle“) der Assoziation $prop$ auf die Domänen der hierarchischen Attribute unter Berücksich-

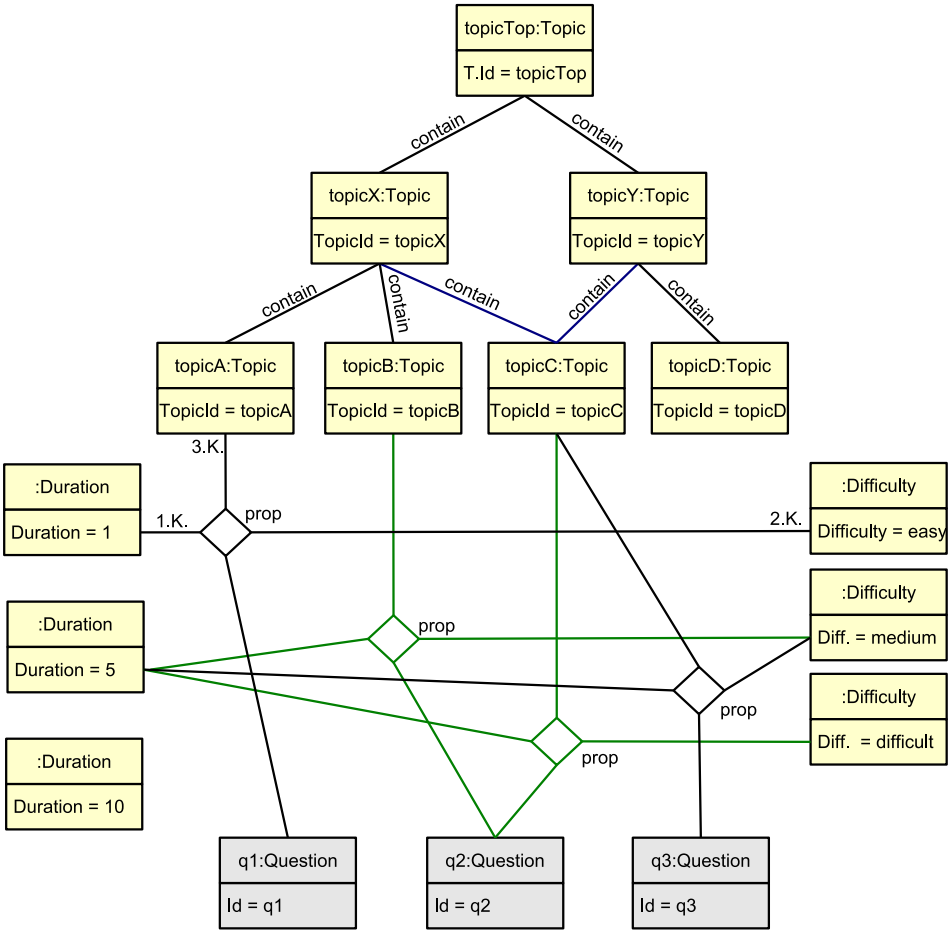



Abbildung 5.12: Beispiel: Strukturmodell-Objektdiagramm

tigung der $contain_i$-Assoziationen ($m+1 \leq i \leq n$):[7]

$$propTrans \subseteq Q \times \prod_{i=1}^{m} range(A_i) \times \prod_{j=1}^{k} domain(A_{m+j})$$

bzw. gleichbedeutend mit[8]

$$propTrans \subseteq Q \times \prod_{i=1}^{n} domain(A_i)$$

mit folgender rekursiven Definition: Es gilt

$$(q, v_1, \ldots, v_i, \ldots, v_n) \in propTrans$$

genau, wenn eine der folgenden Bedingungen erfüllt ist:

1. $(q, v_1, \ldots, v_i, \ldots, v_n) \in prop$
2. Es gibt ein $i \in [m+1, n]$ und ein $w_i \in domain(A_i)$, so dass $(v_i, w_i) \in contain_i$ und $(q, v_1, \ldots, v_{i-1}, w_i, v_{i+1} \ldots, v_n) \in propTrans$

**Beispiel 5.2.15 (Relation $propTrans$)** Auf der Grundlage der Assoziation $prop$ aus Beispiel 5.2.11 und der Assoziation $contain$ aus Beispiel 5.2.11 gilt unter anderem:

$(q_3, 5, medium, topicC)$,
$(q_3, 5, medium, topicX), (q_3, 5, medium, topicY)$,
$(q_3, 5, medium, topicTop) \in propTrans$

In der Regel werden in einem Test nicht alle Fragen des Fragenpools vertreten sein, sondern nur eine Auswahl hiervon. Ein Prüfer stellt gewisse Anforderungen an den Test. Nach diesen werden die Fragen für den Test „gewählt“. Bei der Problembeschreibung spielen daher „wählbare Elemente“ eine wichtige Rolle.

**Definition 5.2.16 (Wählbare Elemente)** Nach Definition 5.1.5 ist die Menge der *wählbaren Elemente* eine endliche Teilmenge $Selectable$ von Objekten des Anwendungs- bzw. Problembereichs.

Die Menge $Selectable$ stimmt in diesem Anwendungsbereich mit der Menge der Fragen $Question = Q$ überein:

[7] Beachte die Entsprechung von $Q$ und $range(A_0)$.
[8] Siehe Bemerkung 5.2.6: Für flache Attribute $A$ ist $range(A) = domain(A)$.

„$Selectable$ = $Question$“

Die Fragen des Fragenpools werden als *wählbar* bezeichnet.

**Bemerkung 5.2.17 (Alternatives Strukturmodell)** Möglicherweise kann in manchen Fällen die Assoziation $prop$ aus Abbildung 5.7 „verlustlos“ in $n$ Einzelassoziationen zerlegt werden. In diesem Fall kann das alternative Modell aus Abbildung 5.13 verwendet werden. Der „Join“ der Einzelassoziationen müsste die ursprüngliche Ausprägung von $prop$ ergeben. Geht man davon aus, dass im Beispiel 5.2.13 die Frage $q_2$ nur in den Tupeln

$$(q_2, 5, \textit{medium}, \textit{topicB}), (q_2, 5, \textit{difficult}, \textit{topicC})$$

vertreten ist, so wäre in diesem Fall eine „verlustlose“ Zerlegung in drei Einzelassoziationen nicht möglich, da sonst auch die Tupel

$$(q_2, 5, \textit{medium}, \textit{topicC}), (q_2, 5, \textit{difficult}, \textit{topicB})$$

Elemente in $prop$ (Kreuzprodukt!) sein müssten.

**Bemerkung 5.2.18 (Problematik um $prop$)** Für die weitere Modellierung spielt $prop$ (und $propTrans$) eine wichtige Rolle. Die Anforderungen an einen Test richten sich oftmals an Teilmengen von Fragen mit bestimmten Attributwertkombinationen in $prop$ (bzw. $propTrans$). Daher werden oft Informationen über sämtliche mögliche Attributwertkombinationen benötigt. Genaueres wird im Bedingungsmodell des Abschnitts 5.2.2 diskutiert. Da $prop$ sämtliche Attribute bzw. deren Wertkombinationen modelliert, treten möglicherweise Redundanzen auf. Im Sinne der Datenbanken ist hier keine Normalisierung durchgeführt. Darüber hinaus sind sämtliche Instanzen (Tupel) der Assoziation $prop$ anzupassen, wenn ein gegebenes konkretes Modell um ein zusätzliches Attribut erweitert wird (vgl. Beispiel 5.2.13).

Die angesprochenen „Problematiken“ werden hier aus eben genannten Gründen und aus Gründen der Allgemeingültigkeit und Einfachheit der Modellierung bewusst in Kauf genommen. Ein derartiges Vorgehen ist auch im Bereich der Data Warehouse Modellierung [BG04, Leh03] bekannt.

### 5.2.2 Constraintmodell

Ist ein Fragenpool $Question$ zusammen mit den beschreibenden Daten und Eigenschaften (vgl. Abschnitt 5.2.1) gegeben, so kann der Anwender Anforderungen an den zu erstellenden Test $T \subseteq Question$, also an die darin enthaltenen Fragen und deren Attributwerte spezifizieren. Ein gültiger Test $T$ muss diese dann erfüllen.

Im Folgenden werden die in diesem Anwendungsbereich typischerweise auftretenden Anforderungen (Bedingungen und Constraints) beschrieben. Natürlich sind darüber hinaus noch weitere denkbar. Die Gesamtheit der Fragen $Question$ wird gemäß Bemerkung 5.2.4 abkürzend mit $Q$ bezeichnet.

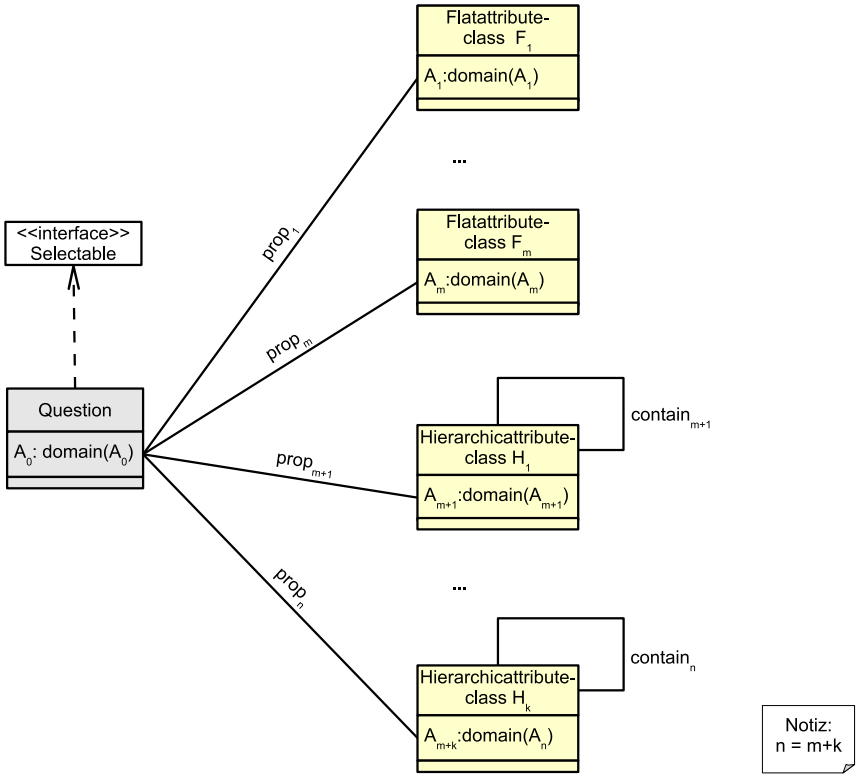



Abbildung 5.13: Alternatives abstraktes Strukturmodell bei bestimmten Randbedingungen

#### 5.2.2.1 Elementare Constraints

Wie schon in Abschnitt 5.1.2 und der Definition 5.1.7 beschrieben, treten folgende Elementarconstraints auf:

- Anzahlconstraints
- Vorgabe von obligatorischen Elementen
- Gewichtssummenconstraints
- Ausschluss von Elementen (Negation)

**Beispiel 5.2.19 (Elementares Constraint)** Bei einem Fragenpool wie in Beispiel 5.2.13 ist eine mögliche Anforderung durch

> „Im Test müssen mindestens drei *leichte* Fragen von *1*-minütiger Bearbeitungszeit enthalten sein."

gegeben. Eine naheliegende Formulierung ist

$$3 \leq |\{q \in Q \mid \text{ Es gibt ein } t \in range(\texttt{Topic}) \text{ mit } (q, 1, easy, t) \in prop\}| \leq \infty$$

Also handelt es sich gemäß 5.1.10 um das Anzahlconstraint

$$cardinalityCons(C, 3, \infty)$$

mit der Constraintreferenzmenge

$$C = \{q \in Q \mid \text{ Es gibt ein } t \in range(\texttt{Topic}) \text{ mit } (q, 1, easy, t) \in prop\}$$

Zur Spezifikation derartiger Constraints ist jeweils eine Constraintreferenzmenge $C \subseteq Q$ anzugeben. Diese ist eine Teilmenge des Fragenkatalogs. Sie kann explizit durch konkrete Angabe der Fragen oder implizit über bestimmte Eigenschaften (Attributwertkombinationen) spezifiziert sein.

#### 5.2.2.2 Constraintreferenzmengen

Zur impliziten Spezifikation von Constraintreferenzmengen $C$ in Constraints

$$\mathfrak{C}(C)$$

mit $\mathfrak{C} \in \{cardinalityCons, containCons, weightsumCons, excludeCons, \dots\}$ werden Fragenpool-Teilmengen definiert:

**Definition 5.2.20 (Notation für Fragenpool-Teilmengen)** Als Voraussetzung sind gegeben:

- Ein Fragenpool $Q$,
- eine beliebige Teilmenge des Fragenpools $D \subseteq Q$,
- die Zuordnungs-Assoziation direkter Attributwerte[9]

$$prop \subseteq Q \times \prod_{i=1}^{n} range(A_i),$$

- die Erweiterung auf die Domänen[10]

$$propTrans \subseteq Q \times \prod_{i=1}^{n} domain(A_i),$$

- eine beliebige Teilmenge hiervon: $prT \subseteq propTrans$,
- ein beliebiges Tupel mit Werten der Attribute $A_1, \dots, A_n$:

$$(v_1, \dots, v_n) \in \prod_{i=1}^{n} domain(A_i),$$

- ein entsprechendes beliebiges „Domänenbereichs-Tupel“:

$$(V_1, \dots, V_n) \in \prod_{i=1}^{n} \mathcal{P}(domain(A_i)).$$

[9] Zu beachten ist die Identifikation von $Q$ und $range(A_0)$, also $prop \subseteq \prod_{i=0}^{n} range(A_i)$

[10] Auch hier entspricht $Q$ der Menge $domain(A_0) = range(A_0)$

Die Menge der Fragen aus $D$, denen vermöge $prT$ das Attributwert-Tupel $(v_1, \ldots, v_n)$ zugeordnet ist, kann mit Hilfe der aus der relationalen Algebra der Datenbanken bekannten Projektion und Selektion in der Form

$$(\pi_{A_0} \circ \sigma_{A_1=v_1 \wedge \cdots \wedge A_n=v_n}(prT)) \cap D$$

ausgedrückt werden [KE06]. Der Kompaktheit halber wird hierfür die Abkürzung

$$D^{prT}_{(v_1,\ldots,v_n)}$$

eingeführt, also

$$D^{prT}_{(v_1,\ldots,v_n)} := \{q \in D \mid (q, v_1, \ldots, v_n) \in prT\}$$

definiert. $D^{prT}_{(v_1,\ldots,v_n)}$ enthält also die Fragenobjekte (Identifier) aus $D$, deren Attribute die Werte $v_1, \ldots, v_n$ besitzen. In Erweiterung auf „Domänen-Teilmengen-Tupel“ ergibt sich:

$$D^{prT}_{(V_1,\ldots,V_n)} := \bigcup_{(v_1,\ldots,v_n) \in \prod_{i=1}^{n} V_i} D^{prT}_{(v_1,\ldots,v_n)}$$

Ist für das Attribut $A_i$ ein beliebiger Wert möglich, d. h. gilt[11]

$$V_i \supseteq range(A_i) \cap (\pi_i \circ prT),$$

so kann als vereinfachende Notation $V_i$ in $D_{(V_1,\ldots,V_n)}$ auch durch das Wildcard-Zeichen $*$ – dieses deutet einen beliebigen Attributwert an – ersetzt werden:

$$D^{prT}_{(V_1,\ldots,V_{i-1},*,V_{i+1},\ldots,V_n)} = D^{prT}_{(V_1,\ldots,V_{i-1},V_i,V_{i+1},\ldots,V_n)}$$

**Bemerkung 5.2.21 (Notation für Fragenpool-Teilmengen)** Im Sinne der Definition 5.2.20 gilt insbesondere für ein Tupel $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} range(A_i)$

$$Q^{prop}_{(v_1,\ldots,v_n)} = \{q \in Q \mid (q, v_1, \ldots, v_n) \in prop\}$$

und für ein Tupel $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} domain(A_i)$

$$Q^{propTrans}_{(v_1,\ldots,v_n)} = \{q \in Q \mid (q, v_1, \ldots, v_n) \in propTrans\}$$

Derartige Teilmengen mit $D = Q$ spielen bei Testspezifikationen eine wichtige Rolle. Ferner kann festgestellt werden: Für Teilmengen-Tupeln mit direkten Attributwerten, also für $(V_1, \ldots, V_n) \in \prod_{i=1}^{n} \mathcal{P}(range(A_i))$ gilt:

$$D^{propTrans}_{(V_1,\ldots,V_n)} = D^{prop}_{(V_1,\ldots,V_n)}$$

[11] $\pi_i$: Projektion auf die $i$-te Komponente

Geht aus dem Zusammenhang hervor, welche Teilmenge $prT \subseteq propTrans$ gemeint ist oder ist sie für eine bestimmte Aussage gerade nicht bedeutsam, so kann deren Notation unterlassen werden. Im Sinne der Vereinbarungen der Definition 5.2.20 gilt für eine beliebige Teilmenge $D \subseteq Q$ und ein Tupel $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} domain(A_i)$:

$$D_{(\{v_1\},\ldots,\{v_n\})} = D_{(v_1,\ldots,v_n)}$$

Der Einfachheit halber kann $v_i \in domain(A_i)$ mit $\{v_i\} \in \mathcal{P}(domain(A_i))$ identifiziert werden. Ist eine der Mengen $V_i$ im Teilmengen-Tupel $(V_1, \ldots, V_n) \in \prod_{i=1}^{n} \mathcal{P}(domain(A_i))$ die leere Menge, $V_i = \emptyset$, so ist

$$D_{(V_1,\ldots,V_n)} = \emptyset.$$

Vor allem im Zusammenhang mit der im Abschnitt 5.2.2.3 diskutierten $prop$-Rollen-Semantik treten jedoch neben $prT = prop$ bzw. $prT = propTrans$ auch echte Teilmengen von $prop$ bzw. $propTrans$ als relevante Mengen $prT$ auf.

**Beispiel 5.2.22 (Fragenpool-Teilmengen)** Mit dem Fragenpool $Q = \{q_1, q_2, q_3\}$, den Attributen

$$A_1 = \texttt{Duration}, A_2 = \texttt{Difficulty} \text{ und } A_3 = \texttt{Topic}$$

und den schon bekannten $prop$-Tupeln

$(q_1, \mathit{1}, \mathit{easy}, \mathit{topicA})$,
$(q_2, \mathit{5}, \mathit{medium}, \mathit{topicB}), (q_2, \mathit{5}, \mathit{difficult}, \mathit{topicC})$,
$(q_3, \mathit{5}, \mathit{medium}, \mathit{topicC}) \in prop$

aus Beispiel 5.2.13 mit dem zugehörigen Objektdiagramm und der $Topic$-Hierarchie aus Abbildung 5.12 – incl. der zugehörigen Relation $propTrans$ – lassen sich beispielsweise folgende Teilmengen spezifizieren:

- Alle 5-minütigen Fragen:

$$Q^{prop}_{(\mathit{5},*,*)} = \{q_2, q_3\}$$

- Die leichten Fragen mit 1 Minute Bearbeitungszeit:

$$Q^{prop}_{(\mathit{1},\mathit{easy},*)} = \{q_1\}$$

- Fragen zum Thema *topicX*:

$$Q^{propTrans}_{(*,*,\mathit{topicX})} = \{q_1, q_2, q_3\}$$

- Alle mittelschweren Fragen mit 5 Minuten Bearbeitungszeit zum Thema *topicX*:

$$Q^{propTrans}_{(5,medium,topicX)} = \{q_2, q_3\}$$

- Schwierige Fragen mit 10-minütiger Bearbeitungszeit:

$$Q^{prop}_{(10,difficult,*)} = \emptyset$$

- Die Fragen zum Thema *topicB*:

$$Q^{prop}_{(*,*,topicB)} = \{q_2\}$$

- Die Fragen zum Thema *topicC*:

$$Q^{prop}_{(*,*,topicC)} = \{q_2, q_3\}$$

In den Beispielen, in denen sich die Anforderung auf direkte Attributwerte bezieht, wurde als entsprechende $propTrans$-Teilmenge $prop$ gewählt. Gleichbedeutend wäre hierbei die Verwendung von $propTrans$. Eine Unterscheidung der direkten und indirekten Attributwerte kann jedoch vor allem im Hinblick auf die Transformation in einen internen Berechnungs-Formalismus bedeutsam sein. Beziehen sich die zu betrachtenden Anforderungen nur auf direkte Attributwerte, so kann gegebenenfalls auf die Berechnung von $propTrans$ verzichtet werden. Zu beachten ist auch, dass die beiden letzten Beispiele Fragenpool-Teilmengen liefern, die einen nicht-leeren Schnitt besitzen, obwohl verschiedene Themen `topicB` $\neq$ `topicC` spezifiziert sind. Die einfache Ursache liegt darin, dass eine Frage mehreren Themen zugeordnet werden kann.

**Beispiel 5.2.23 (Modellierung von elementaren Anforderungen)** Mit Hilfe der Fragenpool-Teilmengen aus Beispiel 5.2.22 und den schon bekannten Elementarconstraints lassen sich folgende Constraints modellieren:

- Genau eine Frage zu Thema *topicB*:

$$c_1 = cardinalityCons(Q^{prop}_{(*,*,topicB)}, 1)$$

- Genau eine Frage zu Thema *topicC*:

$$c_2 = cardinalityCons(Q^{prop}_{(*,*,topicC)}, 1)$$

- Es müssen alle 5-minütigen Fragen enthalten sein:

$$c_3 = containCons(Q^{prop}_{(5,*,*)})$$

- Es dürfen keine leichten Fragen mit 1-minütiger Bearbeitungszeit vertreten sein:

  $$c_4 = excludeCons(Q^{prop}_{(1,easy,*)})$$

- Die Gesamtbearbeitungsdauer aller Fragen im Test soll mindestens 20 Minuten betragen:

  $$c_5 = weightsumCons(Q^{prop}_{(*,\ldots,*)}, 20, \infty) = weightsumCons(Q, 20, \infty)$$

  Die zugehörige Gewichtsfunktion ordnet den Fragen ihre (eindeutige) Bearbeitungsdauer als Gewicht zu.

Rein formal könnte auch hier $prop$ durch $propTrans$ ersetzt werden.

**Beispiel 5.2.24 (Zusammengesetzte Anforderung)** Im Kontext von Beipiel 5.2.23 wird die Anforderung

> „In dem Test muss eine Frage zu Thema *topicB* und eine Frage zu Thema *topicC* enthalten sein.“

gestellt. Werden keine weiteren Bedingungen an den Test gestellt, so könnte

$$T_1 = \{q_2\}$$

ein Modell (gültiger Test) dieser Anforderung darstellen: Der Frage $q_2$ ist sowohl das Thema *topicB* als auch das Thema *topicC* vermöge der Assoziation $prop$ zugeordnet. In dem Test $T_1$ sind durch diese eine Frage beide Themen vertreten – $q_2$ wird also für beide Teilanforderungen „gewertet“. Solche Mehrfachanrechnungen sind aber nicht immer gewünscht. Ein gültiger Test dieser Anforderung könnte in einem anderen Sinne auch durch

$$T_2 = \{q_2, q_3\}$$

gegeben sein: In diesem Fall wird $q_2$ für das Thema *topicB* und $q_3$ für das Thema *topicC* gewertet.

Dieses Beispiel 5.2.24 wirft die Frage nach der Modellierung der verschiedenen Zählmodi in einem Test auf. Dies wird im folgenden Abschnitt behandelt.

#### 5.2.2.3 Zählmodus und $prop$-Rollen-Semantik

Das Beispiel 5.2.24 zeigt zu einer Anforderung zwei mögliche, gleichermaßen sinnvolle Modelle. Eine Spezifikation für die erste Möglichkeit der Interpretation ist in folgendem Beispiel gegeben:

**Beispiel 5.2.25 (Spezifikation mit Mehrfachzählung einer zusammengesetzten Anforderung)** Im Kontext der Beispiele 5.2.23 und 5.2.24 wird die Anforderung

> „In dem Test muss eine Frage zu Thema *topicB* und eine Frage zu Thema *topicC* enthalten sein."

in das Constraint

$$c_1 \wedge c_2,$$

wobei

$$c_1 = cardinalityCons(Q^{prop}_{(*,*,topicB)}, 1) \text{ und } c_2 = cardinalityCons(Q^{prop}_{(*,*,topicC)}, 1)$$

gilt, übersetzt. Der Test $T_1 = \{q_2\}$ aus Beispiel 5.2.24 ist ein Modell dieser Spezifikation, denn wegen

$$|T_1 \cap Q^{prop}_{(*,*,topicB)}| = |\{q_2\} \cap \{q_2\}| = |\{q_2\}| = 1$$

wird das Constraint $c_1$ durch $T_1$ erfüllt, $T_1 \models c_1$; analog gilt

$$|T_1 \cap Q^{prop}_{(*,*,topicC)}| = |\{q_2\}| = 1,$$

also $T_1 \models c_2$. Im Gegensatz dazu ist die Menge $T_2 = \{q_2, q_3\}$ aus Beispiel 5.2.24 *kein* Modell dieser Spezifikation,

$$T_2 \not\models c_1 \wedge c_2,$$

denn $T_2 \not\models c_2$, was aus $|T_2 \cap Q^{prop}_{(*,*,topicC)}| = 2$ ersichtlich ist.

Vor der genauen Definition einer weiteren möglichen „Interpretation" ohne Mehrfachanrechnungen der Fragen in einem Test sollen zunächst noch einige Vereinbarungen zur Sprechweise und Teilmengen-Definitionen für $prop$ getroffen werden:

**Bemerkung 5.2.26 (Sprechweise: Wählen von Fragen)** Erfüllt ein Test $T \subseteq Q$ eine Testspezifikation, so sollen alle Fragen $q \in T$ bezüglich $T$ als „gewählt" gelten (vgl. Bemerkung 2.3.5, Definition 5.2.16). Diese sind also „gewählte" Fragen. In diesem Zusammenhang wird auch vom „Wählen" von Fragen gesprochen.

Die Mehrfachanrechnung der Fragen in einem Test wird durch mehrere $prop$-Tupel ermöglicht. Möchte man dies unterbinden, muss eine geeignete Teilmenge von $prop$ für die entsprechende Constraintreferenzmenge ausgewählt werden. Die prinzipielle Idee ist daher, zu einer Menge $pr \subseteq prop$ sämtliche Teilmengen zu finden, in denen jeweils eine Frage unter den Tupeln nicht mehrmals auftreten darf. Für eine derartige Teilmenge ist aus der Menge der Tupeln, die sich auf die gleiche Frage beziehen also jeweils eines auszuwählen. Ein formaler Weg wird im Folgenden beschritten:

**Definition 5.2.27 (Äquivalenzrelationen und $prop$-Auswahlfunktionen)** Ähnlich zu Definition 5.2.20 sind folgende Voraussetzungen gegeben:

- Ein Fragenpool $Q$,
- die Zuordnungs-Assoziation direkter Attributwerte[12]

$$prop \subseteq Q \times \prod_{i=1}^{n} range(A_i),$$

- eine beliebige Teilmenge $pr$ der $prop$-Assoziation: $pr \subseteq prop$.

Auf $pr$ wird folgende Äquivalenzrelation $R^{pr}$ definiert:

$$R^{pr} \subseteq pr \times pr$$

mit

$$(\underbrace{(q_1, v_{1,1}, \ldots, v_{1,n})}_{t_1}, \underbrace{(q_2, v_{2,1}, \ldots, v_{2,n})}_{t_2}) \in R^{pr} :\Longleftrightarrow\ q_1 = q_2.$$

Zwei Tupeln sind also genau dann äquivalent, wenn sie sich auf die gleiche Frage beziehen. Wie üblich soll auch die Notation $t_1 \sim_{R^{pr}} t_2$ bzw. einfach $t_1 \sim t_2$ anstelle von $(t_1, t_2) \in R^{pr}$ erlaubt sein. Bekanntlich ist die Äquivalenzklasse von $t$ durch

$$[t]_{R^{pr}} := \{\widetilde{t} \in pr \mid t \sim \widetilde{t}\}$$

– oft nur mit $[t]$ abgekürzt – gegeben. Auch die Schreibweise

$$pr_q := [(q, v_1, \ldots, v_n)]_{R^{pr}}$$

ist wohldefiniert und soll eine Abkürzung für die entprechende Äquivalenzklasse sein. Diese Notation lenkt den Blick auf die in den Tupeln $t \in [t]$ referenzierte bzw. enthaltene Frage $q \in Q$, welche nach Definition der Äquivalenzrelation eindeutig ist. Die Menge aller dieser Äquivalenzklassen ist durch

$$pr/_{\sim_{R^{pr}}} := \{[t]_{R^{pr}} \mid t \in pr\}$$

[12] Identifikation von $Q$ und $range(A_0)$

oder kurz $pr/_\sim$ gegeben. Um Mehrfachwertungen von Fragen bezüglich $pr$ zu verhindern ist aus jeder dieser Äquivalenzklassen $[t]_{R^{pr}}$ *ein* Repräsentant zu wählen. Jeder in $pr$ auftretenden Frage $q$ ist also ein Element aus $pr_q$ zuzuordnen. Formal lässt sich wie folgt argumentieren:

Die surjektive Abbildung[13]

$$\pi_0|_{pr} : pr \longrightarrow Q^{pr}_{(*,\ldots,*)}, \quad t \mapsto \pi_0(t)$$

welche einem Tupel $t = (q, v_1, \ldots, v_n)$ die zugehörige Frage $\pi_0(t) = q$ zuordnet, hat bekanntlich *mindestens einen* Schnitt, d. h. es gibt *mindestens eine* Abbildung[14]

$$c_{pr} : Q^{pr}_{(*,\ldots,*)} \longrightarrow pr$$

mit $\pi_0|_{pr} \circ c_{pr} = id_{Q^{pr}_{(*,\ldots,*)}}$ (identische Abbildung). Dies bedeutet: Ist $(\widetilde{q}, v_1, \ldots, v_n) = c_{pr}(q)$ der Funktionswert von $q$ unter $c_{pr}$, so ist $\widetilde{q} = q$. Durch die Funktion $c_{pr}$ wird den in $pr$ auftretenden Fragen jeweils *genau ein* Tupel aus $pr$ zugeordnet.[15] Hierdurch wird also ein mögliche „Auswahl" von $pr$-Tupeln vorgenommen. Gibt es mehrere Schnitte $c_{pr}$, so sind mehrere solcher „Auswahlen" möglich. Im Folgenden soll daher $c_{pr}$ auch als *$pr$-Auswahlfunktion* bezeichnet werden.

Im Besonderen gilt: Ist $pr = prop$, so gibt es gemäß obiger Beschreibung eine endliche Anzahl von $prop$-Auswahlfunktionen $c_{prop,k}(1 \leq k \leq N, N \in \mathbb{N})$, die die möglichen Auswahlen von jeweils genau einem $prop$-Tupel je Frage modellieren:[16]

$$c_{prop,k} : Q \longrightarrow prop$$

Für das Bild von $D \subseteq Q$ unter der Auswahlfunktion $c_{prop,k}$

$$c_{prop,k}(D) = \{c_{prop,k}(q) \mid q \in D\}$$

gilt:

$$(q, v_{1,1}, \ldots, v_{1,n}), (q, v_{2,1}, \ldots, v_{2,n}) \in c_{prop,k}(D) \Rightarrow v_{1,i} = v_{2,i}, 1 \leq i \leq n.$$

Zu beachten ist auch, dass $c_{prop,k}(D)$ eine Teilmenge von $prop$ ist

$$pr = c_{prop,k}(D) \subseteq prop$$

und daher bei der Spezifikation einer Constraintreferenzmenge auftreten kann. Zu bemerken ist ferner, dass die zu einer Frage $q \in Q$ gehörigen $c_{prop,k}(q)$-Funktionswerte $(1 \leq k \leq N)$ zueinander äquivalent sind:

$$c_{prop,k_1}(q) \sim c_{prop,k_2}(q) \quad \text{mit } 1 \leq k_1, k_2 \leq N$$

[13] $\pi_0$: Projektion auf die 0-te Komponente, also auf die Fragen-IDs; $\pi_0|_{pr}$: Einschränkung von $\pi_0$ auf $pr$

[14] $c_{pr}$: Buchstabe $c$ in Anlehnung an *c*hoice, Auswahl

[15] Zu beachten ist auch, dass die Mengen $[c_{pr}(q)]$ gerade die verschiedenen Äquivalenzklassen aus $pr/_{\sim_{R^{pr}}}$ darstellen.

[16] Erinnerung: $Q^{prop}_{(*,\ldots,*)} = Q$

Offensichtlich gilt sogar:

$$prop_q = \{c_{prop,k}(q) \mid 1 \leq k \leq N\}$$

Auch hier können entsprechende Indexe aus Gründen der Einfachheit bei klarem Kontext weggelassen werden.

**Beispiel 5.2.28 ($prop$-Auswahlfunktion)** Ist $prop$ wie in Beispiel 5.2.13 und der Abbildung 5.12 gegeben, also

$$\begin{aligned} prop = \{ & \\ & (q_1, \mathit{1}, \mathit{easy}, \mathit{topicA}), \\ & (q_2, \mathit{5}, \mathit{medium}, \mathit{topicB}), (q_2, \mathit{5}, \mathit{difficult}, \mathit{topicC}), \\ & (q_3, \mathit{5}, \mathit{medium}, \mathit{topicC}) \\ \}, & \end{aligned}$$

so sind folgende zwei $prop$-Auswahlfunktionen

$$\begin{aligned} c_{prop,1} : Q &\longrightarrow prop, \\ q_1 &\mapsto (q_1, \mathit{1}, \mathit{easy}, \mathit{topicA}), \\ q_2 &\mapsto (q_2, \mathit{5}, \mathit{medium}, \mathit{topicB}), \\ q_3 &\mapsto (q_3, \mathit{5}, \mathit{medium}, \mathit{topicC}) \end{aligned}$$

und

$$\begin{aligned} c_{prop,2} : Q &\longrightarrow prop, \\ q_1 &\mapsto (q_1, \mathit{1}, \mathit{easy}, \mathit{topicA}), \\ q_2 &\mapsto (q_2, \mathit{5}, \mathit{difficult}, \mathit{topicC}), \\ q_3 &\mapsto (q_3, \mathit{5}, \mathit{medium}, \mathit{topicC}) \end{aligned}$$

möglich.

**Beispiel 5.2.29 (Spezifikation ohne Mehrfachzählung einer zusammengesetzten Anforderung)** In Erinnerung an die Anforderung von Beispiel 5.2.24

> „In dem Test muss eine Frage zu Thema *topicB* und eine Frage zu Thema *topicC* enthalten sein.“

wird eine zweite Möglichkeit der Spezifikation – in Ergänzung zur ersten Spezifikation aus Beipiel 5.2.25 – angegeben: Zunächst werden mit Hilfe der in Beispiel 5.2.28 definierten Auswahlfunktion $c_{prop,1}$ folgende Constraints definiert:

$$c_1^* := cardinalityCons(Q^{pr}_{(*,*,\textit{topicB})}, 1),$$
$$c_2^* := cardinalityCons(Q^{pr}_{(*,*,\textit{topicC})}, 1)$$

wobei

$$pr = c_{prop,1}(Q)$$

Eine Vereinfachung der Constraintreferenzmengen

$$Q^{pr}_{(*,*,\textit{topicB})} = Q^{\{(q_1,1,\textit{easy},\textit{topicA}),(q_2,5,\textit{medium},\textit{topicB}),(q_3,5,\textit{medium},\textit{topicC})\}}_{(*,*,\textit{topicB})} = \{q_2\},$$
$$Q^{pr}_{(*,*,\textit{topicC})} = Q^{\{(q_1,1,\textit{easy},\textit{topicA}),(q_2,5,\textit{medium},\textit{topicB}),(q_3,5,\textit{medium},\textit{topicC})\}}_{(*,*,\textit{topicC})} = \{q_3\}$$

liefert:

$$c_1^* := cardinalityCons(\{q_2\}, 1),$$
$$c_2^* := cardinalityCons(\{q_3\}, 1)$$

Die Testspezifikation wird durch

$$c_1^* \wedge c_2^*$$

definiert. Der Test $T_2 = \{q_2, q_3\}$ aus Beispiel 5.2.24 ist ein Modell dieser Spezifikation, denn es ist

$$|T_2 \cap \{q_2\}| = |\{q_2\}| = 1,$$

was $T_2 \models c_1^*$ zeigt und es ist

$$|T_2 \cap \{q_3\}| = |\{q_3\}| = 1,$$

was $T_2 \models c_2^*$ zeigt. Wie leicht zu sehen ist, erfüllt die Menge $T_1 = \{q_2\}$ aus Beispiel 5.2.24 das Constraint $c_1^* \wedge c_2^*$ nicht:

$$T_1 \not\models c_1^* \wedge c_2^*.$$

**Beispiel 5.2.30 (Spezifikationen einer zusammengesetzten Anforderung)** In Zusammenfassung der Beispiele 5.2.25 und 5.2.29 und in Erinnerung an die Anforderung von Beispiel 5.2.24[17]

[17] Zu berücksichtigen ist insbesondere auch die gegebene $prop$-Funktion, vgl. Abbildung 5.12

> „In dem Test muss eine Frage zu Thema *topicB* und eine Frage zu Thema *topicC* enthalten sein.“

lässt sich festhalten: Prinzipiell gibt es zwei verschiedene Interpretationen dieser Anforderung. Die erste Interpretation lässt Mehranrechnungen von Fragen zu. Die zugehörige Spezifikation ist durch

$$c_1 \wedge c_2$$

mit

$$c_1 := cardinalityCons(Q^{pr}_{(*,*,\textit{topicB})}, 1),$$
$$c_2 := cardinalityCons(Q^{pr}_{(*,*,\textit{topicC})}, 1)$$

und

$$pr = prop$$

gegeben und liefert

$$T_1 = \{q_2\}$$

als gültigen Test. Die zweite Möglichkeit der Interpretation lässt Mehrfachanrechnung einer Frage im Test nicht zu. Die zugehörige Spezifikation ist durch

$$c_1^* \wedge c_2^*$$

mit

$$c_1^* := cardinalityCons(Q^{pr}_{(*,*,\textit{topicB})}, 1),$$
$$c_2^* := cardinalityCons(Q^{pr}_{(*,*,\textit{topicC})}, 1),$$

wobei (vgl. Beispiel 5.2.28)

$$pr = c_{prop,1}(Q)$$

gilt, gegeben und liefert

$$T_2 = \{q_2, q_3\}$$

als gültigen Test.

Zu beachten ist, dass sich die zur Spezifikation der Constraintreferenzmengen benötigten Attributwerttupel entsprechen: (*, *, *topicB*) tritt in $c_1$ und $c_1^*$ und (*, *, *topicC*) tritt in $c_2$ und $c_2^*$ auf.

Die Spezifikationen unterscheiden sich nur in der zu berücksichtigenden $prop$-Teilmenge. Im ersten Fall spielen zur Auswertung der Constraintreferenzmenge *alle* $prop$-Tupel eine Rolle, jede Frage tritt in allen ihren Rollen im Test auf. Im zweiten Fall wird pro Frage nur *ein* „gewähltes“ $prop$-Tupel gewertet, eine Frage tritt also nur in einer Rolle im Test auf.

**Definition 5.2.31 ($prop$-Rollen einer Frage im Test)** Für eine Frage $q \in Q$ definieren wir folgende Semantik der $prop$-Tupel in einem Test $T$:

Die Frage $q \in T$ erscheint im Test in *„allen Rollen“*, wenn für die Prüfung der Erfüllbarkeit einer gegebenen Spezifikation *alle* zugehörigen Tupel

$$prop_q = \{(q_0, v_1, \ldots, v_n) \in prop \mid q_0 = q\}$$

zum Tragen kommen und in gewissem Sinne *„gewählte $prop$-Tupel“* sind. Die Frage $q \in T$ erscheint in einem Test $T$ in *„einer Rolle“*, wenn für die Prüfung der Erfüllbarkeit einer gegebenen Spezifikation *genau ein* zugehöriges Tupel

$$m \in prop_q = \{(q_0, v_1, \ldots, v_n) \in prop \mid q_0 = q\}$$

zum Zuge kommt. Das entscheidende Tupel wird als *„gewähltes $prop$-Tupel“* bezeichnet.

**Beispiel 5.2.32 ($prop$-Rollen einer Frage im Test, Auswahlfunktionen)** In Anlehnung an die vorangegangenen Beispiele, wie auch Beispiel 5.2.30, kann festgestellt werden:

Die Frage $q_2$ erscheint in $T_1$ in der Semantik *alle Rollen*.

Dies wird durch Abbildung 5.14 veranschaulicht. Außerdem gilt:

Die Frage $q_2$ erscheint in $T_2$ in der Semantik *einer Rolle*.

Die „gewählte“ Rolle für $q_2$ ist in Abbildung 5.15 farblich markiert. Diese Wahl ist gemäß der Auswahlfunktion $c_{prop,1}$ aus Beispiel 5.2.28 getroffen.

Die zweite mögliche Auswahl eines $q_2$-$prop$-Tupels gemäß der Auswahlfunktion $c_{prop,2}$ aus Beispiel 5.2.28 wird in Abbildung 5.16 veranschaulicht. Für die ensprechende Spezifikation wäre diese Auswahlfunktion nicht geeignet, da hierbei Thema *topicB* nicht mehr vertreten wäre.[18]

Die gewünschte Auswahl der Interpretation, ob für eine Frage Mehrfachzählung erlaubt ist oder nicht, bzw. die Auswahl der $prop$-Rollen-Semantik „alle Rollen“ oder „eine Rolle“ kann als eine Art „Constraint 2. Stufe“ – hier werden Bedingungen an andere Constraints gestellt – aufgefasst werden. Hierzu folgende Definition:

[18] Ein entsprechender Algorithmus zur Berechnung geeigneter Auswahlen könnte Konzepte wie „looking ahead“ und Backtracking verwenden.

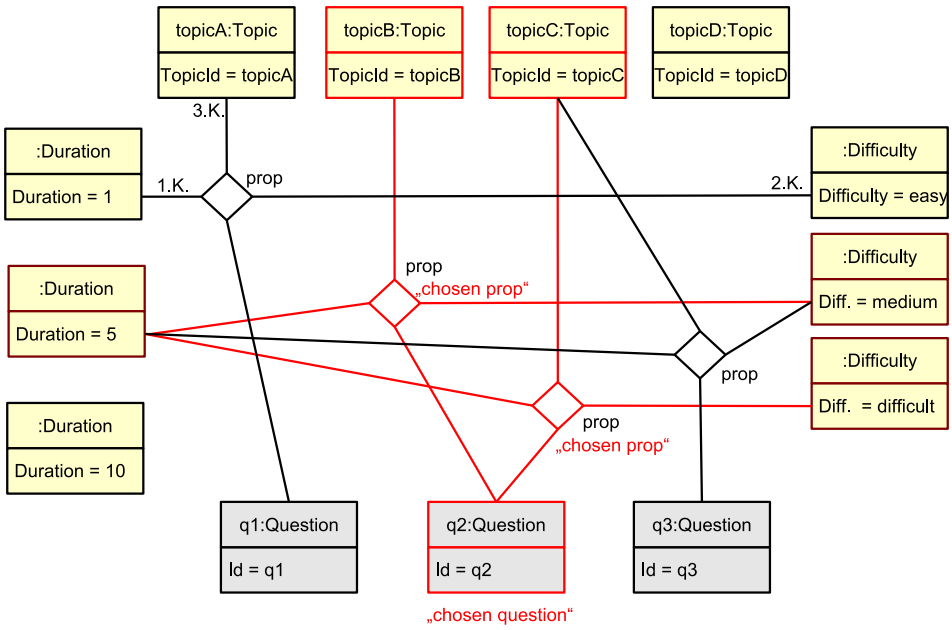



Abbildung 5.14: Mögliche *prop*-Semantik im Test: Wertung aller Rollen

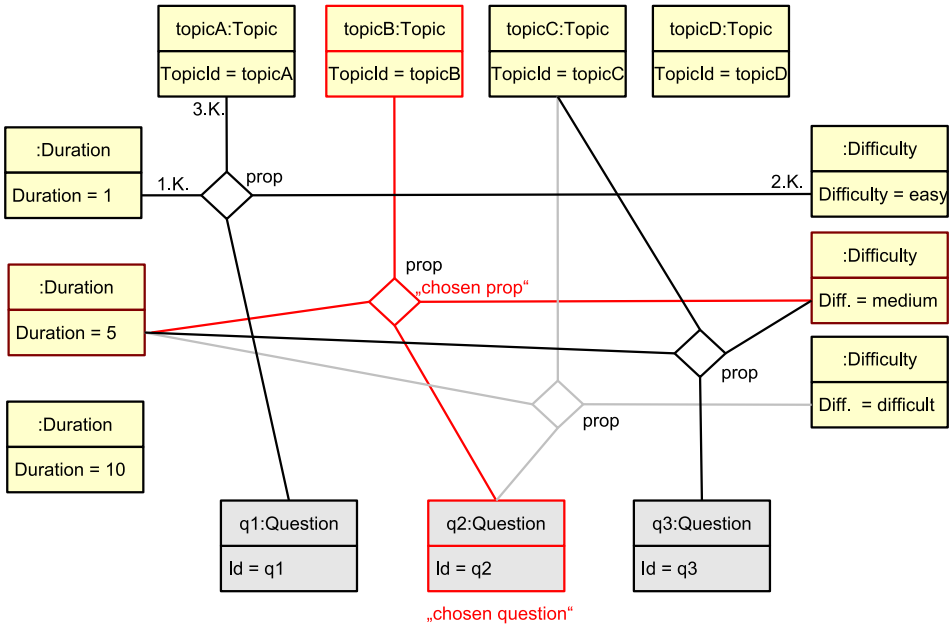



Abbildung 5.15: Mögliche *prop*-Semantik im Test: Wertung einer Rolle (1)

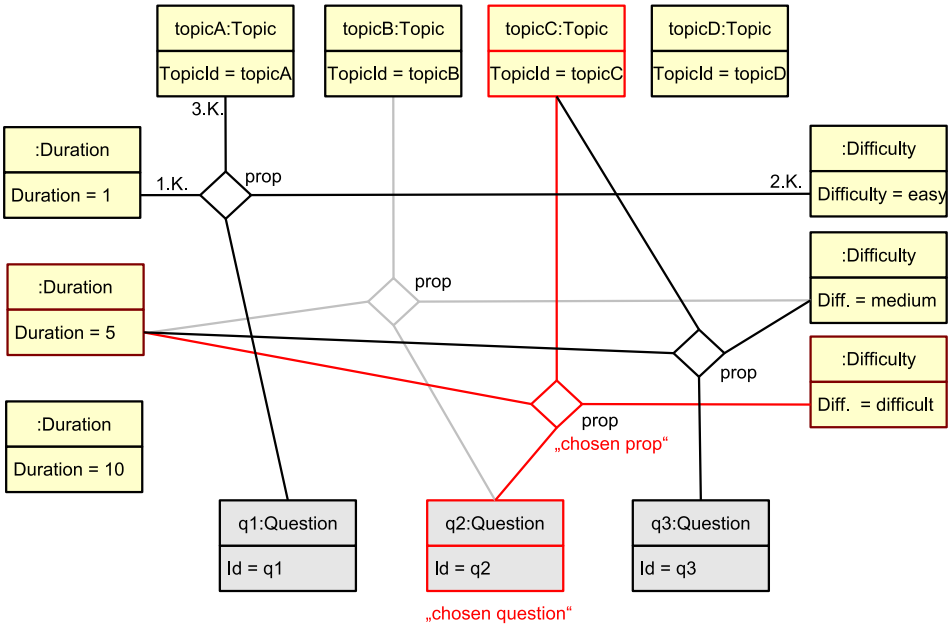



Abbildung 5.16: Mögliche *prop*-Semantik im Test: Wertung einer Rolle (2)

**Definition 5.2.33 (Rollenconstraint $roleCons$)** Mit Hilfe der *Rollenconstraint-Relation*

$$roleCons \subseteq Q \times \{\text{one}, \text{all}\}$$

kann die *prop*-Rollen-Semantik einer Frage festgelegt werden. Für $q \in Q$ soll gelten:

$$\begin{aligned} (q, \text{one}) \in roleCons \quad &:\Longleftrightarrow \quad \text{Frage } q \text{ erscheint in der Semantik „eine Rolle“} \\ &\qquad\quad \text{in einem möglichen Test} \\ (q, \text{all}) \in roleCons \quad &:\Longleftrightarrow \quad \text{Frage } q \text{ erscheint in der Semantik „alle Rollen“} \\ &\qquad\quad \text{in einem möglichen Test} \end{aligned}$$

Die Relation soll funktionale Eigenschaften besitzen, d. h. für eine Frage $q \in Q$ dürfen nicht beide Tupel $(q, \text{one})$, $(q, \text{all})$ in $roleCons$ auftreten.

**Definition 5.2.34 ($roleCons$-Fragenmengen)** Sind der Fragenpool $Q$, eine Teilmenge $D \subseteq Q$ des Fragenpools und die Relation $roleCons$ gegeben, so werden folgende Abkürzungen definiert[19]:

$$\begin{aligned} D_{\text{one}}^{roleCons} &= (\pi_1 \circ roleCons)\,(D \times \{\text{one}\}) = \{q \in D \mid roleCons(q, \text{one})\} \\ D_{\text{all}}^{roleCons} &= (\pi_1 \circ roleCons)\,(D \times \{\text{all}\}) = \{q \in D \mid roleCons(q, \text{all})\} \end{aligned}$$

[19] $\pi_1$: Projektion auf die 1. Komponente

**Bemerkung 5.2.35 (*prop*-Rollen einer Frage im Test)** Die Definitionen 5.2.31 und 5.2.33 lassen eine Entscheidung der $prop$-Rollen-Semantik *pro Frage* zu. Oftmals werden einfach entweder alle Fragen im Sinne der Semantik „alle Rollen“ oder im Sinne der Semantik „eine Rolle“ interpretiert.

Man könnte sich auch vorstellen, dass es neben den beiden extremen Möglichkeiten – *ein* bzw. *alle* $prop$-Tupel einer Frage $q$ in einer Spezifikation zählen zu lassen – es auch Semantiken gibt, die eine *bestimmte Anzahl* ($n \in \mathbb{N}$) an zu wertenden $prop$-Tupeln vorgeben. In den hier betrachteten Anwendungsdomänen werden derartige Spezifikationen jedoch nicht benötigt und daher nicht näher betrachtet.

**Definition 5.2.36 (Transitive Hülle für $prop$-Teilmengen)** In Analogie zur Definition 5.2.14 der $propTrans$-Relation wird definiert: Für eine Teilmenge $pr \subseteq prop$ ist die Relation

$$prTrans \subseteq Q \times \prod_{i=1}^{n} domain(A_i)$$

als die „transitive Hülle“ von $pr$ bezüglich der $contain$-Assoziationen definiert. Es gilt:

$$pr \subseteq prTrans \subseteq propTrans$$

**Bemerkung 5.2.37 ($propTrans$-Rollen einer Frage im Test)** Entsprechende Überlegungen zu den $prop$-Rollen einer Frage im Test können in einer Erweiterung auch auf hierarchische Attributwert-Tupel übertragen werden. Für eine Teilmenge $pr \subseteq prop$ spielen dann die „geerbten“ $prTrans$-Tupel eine Rolle. Vor allem, wenn die in einer Dimension zugrunde liegende Hierarchie kein Baum ist, stellt sich auch hier das Problem der Mehrfachwertung – auch wenn die $prop$-Semantik „eine Rolle“ gewählt wurde. Dies soll im Weiteren allerdings nicht mehr näher diskutiert werden.

Neben den schon besprochenen Anforderungen an einen Test können in Testspezifikationen auch weitere Constraintarten auftreten. In den nächsten beiden Abschnitten werden zwei neue Constraintarten definiert.

#### 5.2.2.4 Prozentualconstraints

Ein Prozentualconstraint schränkt den prozentualen Anteil von Elementen mit bestimmten Eigenschaften ein. Eine relationale Modellierung einer Klasse $Percentconstraint$ erfolgt in folgender Definition:

**Definition 5.2.38 (Relation $percentCons$)** Mit Hilfe der *Prozentualconstraint-Relation*

$$percentCons \subseteq \mathcal{P}(Q) \times [1,n] \times \left( \bigcup_{i=1}^{n} domain(A_i) \right) \times [0..1] \times [0..1]$$

werden *Prozentualconstraints* mit folgender Semantik definiert: Für eine Fragenmenge $T \subseteq Q$ und ein Tupel $(C, i, v_i, p_{min}, p_{max}) \in \mathcal{P}(Q) \times [1,n] \times (\bigcup_{i=1}^{n} domain(A_i)) \times [0..1] \times [0..1]$ und relevanter $propTrans$-Teilmenge $prTrans$ (vgl. Abschnitt 5.2.2.3) gilt:[20]

$$\begin{aligned} &T \models percentCons(C, i, v_i, p_{min}, p_{max}) \\ &\quad :\Longleftrightarrow \text{ Es gibt ein } p \in [p_{min}, p_{max}] \text{ mit: } |T \cap C^{prTrans}_{(*,\ldots,*,v_i,*,\ldots,*)}| = p \cdot |T \cap C|, \end{aligned}$$

$T$ erfüllt also genau dann das Constraint, wenn der Anteil der in $T \cap C$ enthaltenen Fragen mit dem $A_i$-Attributwert $v_i$ bezüglich $prTrans$ gerade im Bereich von $p_{min}$ und $p_{max}$ liegt.

**Beispiel 5.2.39 (Relation $percentCons$)** Die Anforderung

> „Die schwierigen Fragen müssen zu 25-30 Prozent von 5-minütiger Bearbeitungsdauer sein.“

kann unter den Gegebenheiten von Beispiel 5.2.13 und bei relevanter $propTrans$-Teilmenge $prTrans$ durch

$$percentCons(Q^{prTrans}_{(*,\mathit{difficult},*)}, \texttt{1}, 5, 0.25, 0.3)$$

modelliert werden: $A_1 = \texttt{Duration}$, also $i = \texttt{1}$; Attributwert: 5; Grundmenge $C = Q^{prTrans}_{(*,\mathit{difficult},*)}$; minimaler Prozentwert: 0.25, maximaler Prozentwert: 0.3.

**Bemerkung 5.2.40 (Rückführung von $percentCons$ auf $cardinalityCons$)** Ein prozentuales Constraint lässt sich auf Anzahlconstraints aufgrund folgender Feststellung

$$\begin{aligned} |T \cap C^{prTrans}_{(*,\ldots,*,v_i,*,\ldots,*)}| = p \cdot |T \cap C| \iff & T \models cardinalityCons(C^{prTrans}_{(*,\ldots,*,v_i,*,\ldots,*)}, n) \\ & \text{und } T \models cardinalityCons(C, N) \\ & \text{und } n = p \cdot N \end{aligned}$$

zurückführen.[21]

[20] In Erinnerung an Definition 5.2.20: $C^{prTrans}_{(*,\ldots,*,v_i,*,\ldots,*)} =$
$= \{q \in C \mid \text{Für alle } j \in [1,n] \backslash \{i\} \text{ gibt es } v_j \in domain(A_j) \text{ mit } (q, v_1, \ldots, v_n) \in prTrans\}$. Es wird auch davon ausgegangen, dass $prTrans$ so geeignet gewählt ist, dass $C = C^{prTrans}_{(*,\ldots,*)}$ gilt.

[21] Auch hier soll $C = C^{prTrans}_{(*,\ldots,*)}$ gelten.

#### 5.2.2.5 Optimierungsconstraints

Ein Optimierungsconstraint stellt eine Art „weiches“ Constraint dar, das den zu erstellenden Test in gewisser Hinsicht optimieren soll (Optimierung einer Zielfunktion). Unter den möglichen Lösungen sollen diejenigen relevant sein, die möglichst gut die Bedingung des Optimierungsconstraints erfüllen. Diese können Bezug zu einer Menge von Fragen oder auch zu einer Menge anderer Constraints haben.

**Definition 5.2.41 (Relation $optimizeQCons$)** Mit Hilfe der *Optimierungsconstraint-Relation mit Bezug zu Fragen*

$$optimizeQCons \subseteq \mathcal{P}(Q) \times \{\min, \max\} \times \mathcal{P}(\mathcal{C})$$

werden *Optimierungsconstraints* mit folgender Semantik definiert: Für eine Fragenmenge $T \subseteq Q$ und für eine Menge $C \in \mathcal{P}(Q)$ von Fragen, eine Menge $s \in \mathcal{P}(\mathcal{C})$ von Constraints und die Menge $\mathcal{M}_s := \{\widetilde{T} \subseteq Q \mid \widetilde{T} \models s\}$ aller Fragenmengen $\widetilde{T} \subseteq Q$, die $s$ erfüllen, gilt:

$$\begin{aligned} &T \models optimizeQCons(C, \min, s) \\ &\quad :\Longleftrightarrow\ T \in \mathcal{M}_s \quad \text{und} \quad |T \cap C| \leq |\widetilde{T} \cap C| \text{ für alle } \widetilde{T} \in \mathcal{M}_s \end{aligned}$$

und analog

$$\begin{aligned} &T \models optimizeQCons(C, \max, s) \\ &\quad :\Longleftrightarrow\ T \in \mathcal{M}_s \quad \text{und} \quad |T \cap C| \geq |\widetilde{T} \cap C| \text{ für alle } \widetilde{T} \in \mathcal{M}_s \end{aligned}$$

$M$ erfüllt also genau dann das Constraint, wenn $M$ unter allen Modellen der Spezifikation $s$ die Anzahl der aus $C$ enthaltenen Elemente „optimiert“.

**Beispiel 5.2.42 (Relation $optimizeQCons$)** Eine Spezifikation $s$ wird um folgendes weiche Constraint erweitert (vgl. auch Beispiel 5.2.13):

„Möglichst viele Fragen zu Thema *topicA*.“

Diese Anforderung kann durch

$$optimizeQCons(Q^{prTrans}_{(*,*,\mathit{topicA})}, \max, s)$$

bei relevanter $propTrans$-Teilmenge $prTrans$ spezifiziert werden.

**Definition 5.2.43 (Relation $optimizeCCons$)** Mit Hilfe der *Optimierungsconstraint-Relation mit Bezug zu Constraints*

$$optimizeCCons \subseteq \mathcal{P}(\mathcal{C}) \times \{\min, \max\} \times \mathcal{P}(\mathcal{C})$$

werden *Optimierungsconstraints* mit folgender Semantik definiert: Für eine Teilmenge $T \subseteq Q$ und Constraintmengen $C \in \mathcal{P}(\mathcal{C})$, $s \in \mathcal{P}(\mathcal{C})$ und die Menge $\mathcal{M}_s := \{\widetilde{T} \subseteq Q \mid \widetilde{T} \models s\}$ aller Fragenmengen $\widetilde{T} \subseteq Q$, die $s$ erfüllen, gilt:

$$T \models optimizeCCons(C, \min, s)$$
$$:\Longleftrightarrow\ T \in \mathcal{M}_s \quad \text{und} \quad |\{c \in C \mid T \models c\}| \leq |\{c \in C \mid \widetilde{T} \models C\}| \text{ für alle } \widetilde{T} \in \mathcal{M}_s$$

und analog

$$T \models optimizeQCons(C, \max, s)$$
$$:\Longleftrightarrow\ T \in \mathcal{M}_s \quad \text{und} \quad |\{c \in C \mid T \models c\}| \geq |\{c \in C \mid \widetilde{T} \models C\}| \text{ für alle } \widetilde{T} \in \mathcal{M}_s$$

$M$ erfüllt also genau dann das Constraint, wenn $M$ unter allen Modellen der Spezifikation $s$ die Anzahl der erfüllten Constraints aus $C$ „optimiert“.

Zusammen ergibt sich eine relationale Modellierung der Klasse $Optimizeconstraint$:

**Definition 5.2.44 (Relation $optimizeCons$)** Die *Optimierungsconstraint-Relation* fasst die Optimierungsconstraint-Relation mit Bezug zu Fragen und diejenige mit Bezug zu Constraints zu einer Relation zusammen:

$$optimizeCons \subseteq (\mathcal{P}(Q) \cup \mathcal{P}(\mathcal{C})) \times \{\min, \max\} \times \mathcal{P}(\mathcal{C})$$

wobei

$$optimizeCons = optimizeQCons \cup optimizeCCons$$

#### 5.2.2.6 Inkonsistenzen

Der Anwender erwartet mindestens einen gemäß seiner Spezifikation gültigen Test. Allerdings könnte es mehrere Ursachen dafür geben, dass es nicht möglich ist, eine Menge von Fragen zu finden, die alle benutzerdefinierten Anforderungen erfüllt. Prinzipiell können zwei Arten von Inkonsistenzen unterschieden werden:

- *Inkonsistente Menge von Anforderungen:* Die Anforderungen an sich sind inkonsistent, also widersprüchlich und mit keinem möglichen Fragenkatalog $Q$ erfüllbar.
- *Instanzbasierte Inkonsistenz:* Der gegebene Fragenkatalog erlaubt die Erfüllung der Anforderungen nicht, da nicht genügend Fragen mit den spezifizierten Attributwerten vorhanden sind.

**Beispiel 5.2.45 (Inkonsistente Menge von Anforderungen)** Mit den Gegebenheiten aus Beispiel 5.2.13 könnte ein Anwender folgende Anforderungen an einen Test stellen:

- Genau 2 Fragen
- 2 leichte und 1 schwierige Frage
- *prop*-Semantik der Fragen: „eine Rolle“

Wie leicht ersichtlich ist, ist diese Anforderung widersprüchlich und nicht erfüllbar.

**Beispiel 5.2.46 (Instanzbasierte Inkonsistenz)** Mit den Gegebenheiten aus Beispiel 5.2.13 könnte ein Anwender folgende Anforderungen an einen Test stellen:

- Mindestens 4 schwierige Fragen

Diese Anforderung ist bei vorhandenem Fragenpool nicht erfüllbar, da dieser im Beispiel nur 3 Fragen enthält. Die Spezifikation ist an sich nicht widersprüchlich. Durch Hinzunahme von zusätzlichen Fragen, also bei geeigneter Vergrößerung des Fragenpools kann diese Anforderung erfüllt werden.

Inkonsistenzen sollten erkannt und – im Sinne der Benutzerfreundlichkeit – möglichst genau lokalisiert werden.

Im Folgenden werden die in diesem Anwendungsbereich typischerweise auftretenden Anforderungen untersucht bzw. angegeben.

#### 5.2.2.7 Typische Constraintspezifikationen

Zur Definition eines einzelnen Constraints hat der Ersteller eines Tests Angaben über

- die Constraintart,
- die Constraintreferenzmenge (Teilmenge des Fragenpools) bzw. die Constraintmenge (Teilmenge aller Constraints) und
- spezielle Detail-Angaben im Constraint

vorzunehmen.

Bezüglich der *Constraintarten* sind folgende (siehe auch Abbildung 5.17) als die wichtigsten zu nennen:

1. Constraints, in denen eine Constraintreferenzmenge zu spezifizieren ist:
   - Anzahlconstraint
   - Vorgabe von obligatorischen Elementen
   - Gewichtssummenconstraint
   - Ausschluss von Elementen (Negation)
   - Prozentuales Constraint
   - Optimierungsconstraint
   - Rollenconstraint
2. Constraints mit Bezug zu einer Constraintmenge:
   - Konjunktion und Disjunktion von Constraints
   - Auswahlconstraint
   - Bedingtes Constraint
   - Optimierungsconstraint

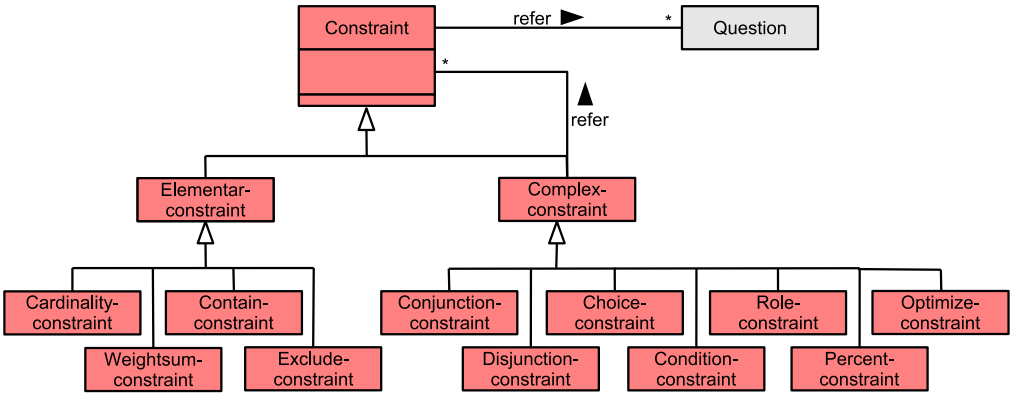



Abbildung 5.17: Constraintmodell-Klassendiagramm

Natürlich sind darüber hinaus auch noch weitere denkbar. Aus der Analyse des Anwendungbereiches ergeben sich bei den *Constraintreferenzmengen* folgende typische Möglichkeiten der Spezifikation:

**Definition 5.2.47 (Typische Spezifikationsarten für Constraintreferenzmengen)** Eine Constraintreferenzmenge $C$ ist *explizit spezifiziert*, wenn sie in aufzählender Weise angegeben ist:

$$C = \{q_i \in Q \mid 1 \leq i \leq n, n \in \mathbb{N}\} \subseteq Q.$$

Eine Constraintreferenzmenge $C$ ist *implizit spezifiziert*, wenn sie durch

- eine (implizit festgelegte) Teilmenge $D \subseteq Q$,
- die Angabe eines Domänenbereichs-Tupels

$$(V_1, \ldots, V_n) \in \prod_{i=1}^{n} \mathcal{P}(domain(A_i))$$

- und eine $propTrans$-Teilmenge $prT \subseteq propTrans$[22]

festgelegt wird:

$$C = D^{prT}_{(V_1,\ldots,V_n)}.$$

Bei einer Constraintreferenzmenge liegt eine *kombinierte Spezifikation* vor, wenn sie sich als endliche Vereinigung von – explizit oder implizit spezifizierten Mengen $C_1, \ldots, C_r, r \in \mathbb{N}$ – darstellen lässt:

$$C = \bigcup_{i=1}^{r} C_i$$

**Beispiel 5.2.48 (Explizite Spezifikation von Constraintreferenzmengen)** Eine explizite Spezifikation einer Constraintreferenzmenge $C$ tritt typischerweise in den $containCons$- und $excludeCons$-Constraints auf:

- Möchte der Anwender spezifizieren, dass auf jeden Fall die beiden Fragen $q_1$ und $q_2$ in einem Test enthalten sein sollen, so kann dies mit der Constraintreferenzmenge $C = \{q_1, q_2\}$ durch $containCons(C)$ modelliert werden.
- Möchte ein Prüfer die Fragen $q_1$, $q_2$ und $q_3$ aus der letzten Prüfung nicht nochmals verwenden, so lässt sich dies mit der Constraintreferenzmenge $C = \{q_1, q_2, q_3\}$ durch $excludeCons(C)$ ausdrücken.

Bei der impliziten Spezifikation einer Constraintreferenzmenge treten in den untersuchten Anwendungen nicht alle bisher vorgestellten Möglichkeiten auf. Im Folgenden werden die typischerweise auftretenden Constraintreferenzmengen angegeben:

**Bemerkung 5.2.49 (Typische Constraintreferenzmengen (implizit, kombiniert))** Bei einer impliziten Spezifikationen der Art

$$C = D^{prT}_{(V_1,\ldots,V_n)}$$

treten typischerweise folgende Rahmenbedingungen auf:

[22] In späteren Beispielen wird ersichtlich, dass auch $prT$ möglicherweise nicht explizit spezifiziert ist.

1. *Grundmenge $D$:* Zwei Hauptfälle sind zu unterscheiden:

   - Die Grundmenge besteht aus dem gesamten Fragenpool:

     $$D = Q$$

   - Die Grundmenge besteht aus einer Teilmenge des Fragenpools mit bestimmter $prop$-Rollen-Semantik (vgl. 5.2.34):

     $$D = Q_{one}^{roleCons} \quad \text{oder} \quad D = Q_{all}^{roleCons}.$$

2. *Attributwert-Tupel:* Die auftretenden Attributwert-Mengen im Attributwert-Tupel sind entweder einelementig oder der gesamte Wertebereich:

   $$(V_1, \ldots, V_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$$

   .

3. *$prop$-Teilmenge(n):* Die Angabe geschieht in der Regel implizit über die Definition der $prop$-Rollen-Semantik bei den Fragen (Relation $roleCons$):

   - Bei der $prop$-Semantik „alle Rollen“ wäre $prT = prop$ bzw. $prT = propTrans$.
   - Bei der $prop$-Semantik „eine Rolle“ wäre $prT = pr \subseteq prop$ die Bildmenge einer der möglichen $prop$-Auswahlfunktionen (siehe Definition 5.2.27) bzw. die „transitive“ Erweiterung auf die hierarchischen Attributwerte $prTrans$ (vgl. Definition 5.2.36).

Kombinierte Constraintreferenzmengen lassen sich als Vereinigung

$$C = \bigcup_{i=1}^{r} C_i$$

von explizit oder implizit spezifizierten Mengen $C_i$ darstellen.

**Beispiel 5.2.50 (Beispiel einer Testspezifikation I)** In Erweiterung des Beispiels 5.2.13 ist sei ein genügend großer Fragenpool $Q$ gegeben. Die Fragen sind durch Attributwerte der Attribute

$$A_1 = \texttt{Duration}, A_2 = \texttt{Difficulty}, A_3 = \texttt{Topic}$$

vermöge der Assozitation $prop$ bzw. vermöge $propTrans$, der Erweiterung auf die hierarchischen Attributwerte von `Topic`, beschrieben. Ein Prüfer macht folgende Angaben zur Spezifikation eines Tests:

1. Insgesamt zwischen 15 und 20 Fragen.

2. Mindestens 3 Fragen zum Thema *topicX*.
3. Genau 4 schwierige Fragen zu Thema *topicB*.
4. Gesamtbearbeitungsdauer: 60 Minuten.
5. Mehrfachzählung der Fragen erlaubt.
6. Auf jeden Fall soll die Frage $q$ vertreten sein.
7. Bei den *topicB*-Fragen soll ca. die Hälfte (45-55 Prozent) aus mittelschweren Fragen bestehen.
8. Möglichst wenig einminütige Fragen.

Da Mehrfachzählung (Anforderung 5) erlaubt sein soll, ist die Semantik „alle Rollen“ zu realisieren. Die zu verwendende $prop$- bzw. $propTrans$-Teilmenge ist also $prop$ bzw. $propTrans$ selbst.

Aufgrund der angegebenen Bearbeitungsdauer (Anforderung 4) ist eine Gewichtung der Fragen vorzunehmen. Das Gewicht einer Frage ist durch deren Bearbeitungsdauer bestimmt. Vorausgesetzt ist hier allerdings, dass diese Zuordnung eindeutig erfolgen kann, was nur dann möglich ist, wenn eine Frage nicht – kontextabhängig – mehrere Bearbeitungszeiten in der zugrunde liegenden $prop$-Teilmenge (hier: $prop$) besitzt.[23] Die einzelnen Constraints können der Reihe nach wie folgt übersetzt werden:

1. $cardinalityCons(Q^{prop}_{(*,*,*)}, 15, 20) =: c_1$
2. $cardinalityCons(Q^{propTrans}_{(*,*,\mathit{topicX})}, 3, \infty) =: c_2$
3. $cardinalityCons(Q^{prop}_{(*,\mathit{difficult},\mathit{topicB})}, 4) =: c_3$
4. $weightsumCons(Q^{prop}_{(*,*,*)}, 60) =: c_4$ mit $weight(q) = v_1$ für $(q, v_1, v_2, v_3) \in prop$
5. $roleCons(q, \text{all}),\ q \in Q$
6. $containCons(\{q\}) =: c_5$
7. $percentCons(Q^{prop}_{(*,*,\mathit{topicB})}, 2, \mathit{medium}, 0.45, 0.55) =: c_6$
8. $optimizeCons(Q^{prop}_{(1,*,*)}, \text{min}, \{c_i \mid 1 \leq i \leq 6\})$

In der Anforderung 2 ist ein hierarchischer Attributwert (*topicX*) angegeben, daher ist hier $propTrans$ für die Modellierung notwendig.

[23] Sollte einer Frage $q$ mehrere Bearbeitungszeiten zugeordnet sein, so könnte man dieser – entsprechend den verschiedenen Bearbeitungszeiten – mehrere ID's $q_1, \ldots, q_k, k \in \mathbb{N}$ in $prop$ zuordnen. Dadurch kann jeder dieser ID's eindeutig eine Bearbeitungszeit zugeordnet werden. Zusätzlich muss aber dann noch gefordert werden, dass in einem Test höchstens eine dieser ID's $q_1, \ldots, q_k, k \in \mathbb{N}$ vertreten ist, da diese ja nur eine Frage repräsentieren.

**Beispiel 5.2.51 (Beispiel einer Testspezifikation II)** Mit den gleichen Voraussetzungen wie in Beispiel 5.2.50 macht ein Prüfer folgende Angaben zur Spezifikation eines Tests:

1. Die Fragen $q_1$, $q_{25}$ und $q_{67}$ aus der letzten Prüfung dürfen nicht vorkommen.
2. Zwischen 15 und 18 Fragen zu Thema *topicB*.
3. 10 Fragen zu *topicC*.
4. Bei den Fragen $q_i, 1 \leq i \leq 50$ ist keine Mehrfachanrechnung erlaubt, bei den restlichen Fragen des Fragenkataloges schon.

Die Anforderung 4 erfordert eine differenzierte Spezifikation der Constraintreferenzmengen. Die Fragen aus $Q_{one} := \{q_i \mid 1 \leq i \leq 50\} \subseteq Q$ erlauben keine Mehrfachzählung, die Fragen $Q_{all} := Q \backslash Q_{one}$ erlauben die Berücksichtigung sämtlicher zugeordneter $prop$-Tupel.

Seien nun durch

$$pr_i \subseteq prop \; 1 \leq i \leq r, r \in \mathbb{N}$$

die verschiedenen Bildmengen sämtlicher $prop$-Auswahlfunktionen (siehe Definition 5.2.27) gegeben. Die einzelnen Anforderungen der Testspezifikation $s$ lassen sich wie folgt modellieren:

1. $c_1 = excludeCons(\{q_1, q_{25}, q_{67}\})$
2. $c_2 = \bigvee_{i=1}^{r} cardinalityCons\left((Q_{all})^{prop}_{(*,*,topicB)} \cup (Q_{one})^{pr_i}_{(*,*,topicB)}, 15, 18\right)$
3. $c_3 = \bigvee_{i=1}^{r} cardinalityCons\left((Q_{all})^{prop}_{(*,*,topicC)} \cup (Q_{one})^{pr_i}_{(*,*,topicC)}, 10\right)$
4. $c_4 :$ $roleCons(q, \text{all})$ für $q \in Q_{all}$, $roleCons(q, \text{one})$ für $q \in Q_{one}$[24]

Da die Teilmengen $pr_i, 1 \leq i \leq r$ die Bilder von $Q$ unter den Auswahlfunktionen sind, ist es möglich, dass einige Einschränkungen von $pr_i$ auf $Q_{one}$ identisch – und im Sinne der Spezifikation redundant – sind. Dies ist hier jedoch nicht bedeutsam. Da nicht vorgegeben ist, welche der möglichen $pr_i$-Teilmengen für die Auswertung eines Constraints Verwendung finden soll, muss jeweils nur gefordert werden, dass die Disjunktion der möglichen Einzelconstraints – in Variation der verwendeten $pr_i$-Teilmenge – erfüllt ist.

Damit allerdings die Gesamtspezifikation erfüllt ist, muss *zusätzlich* gefordert sein, dass bei *allen* relevanten Einzelconstraints die *gleiche* $pr_i$-Teilmenge zum Tragen kommt. Eine Teilmenge $T \subseteq Q$ erfüllt damit obige Spezifikation $s$, genau wenn es ein $i \in [1, r]$ gibt, so dass folgende Bedingungen erfüllt sind:

[24] Durch $Q_{all}$ und $Q_{one}$ ist implizit $roleCons$ vorgegeben: $roleCons$ wird so definiert, dass $Q_{all}^{roleCons} = Q_{all}$ bzw. $Q_{one}^{roleCons} = Q_{one}$ gilt.

1. $T \models c_1 = excludeCons(\{q_1, q_{25}, q_{67}\})$
2. $T \models (c_2)_i := cardinalityCons\left(\left(Q_{all}^{roleCons}\right)_{(*,*,topicB)}^{prop} \cup \left(Q_{one}^{roleCons}\right)_{(*,*,topicB)}^{pr_i}, 15, 18\right)$
3. $T \models (c_3)_i := cardinalityCons\left(\left(Q_{all}^{roleCons}\right)_{(*,*,topicC)}^{prop} \cup \left(Q_{one}^{roleCons}\right)_{(*,*,topicC)}^{pr_i}, 10\right)$
4. $roleCons$ ist so definiert, dass $Q_{one}^{roleCons} = Q_{one}$ und $Q_{all}^{roleCons} = Q_{all}$ erfüllt ist

gilt.

**Definition 5.2.52 (Abkürzende Constraintspezifikation)** Als Voraussetzungen sind gegeben:

- Ein Fragenpool $Q$,
- die Attribute $A_i, 1 \leq i \leq n$ mit ihren Wertebereichen und Domänen,
- die Zuordnungsassoziation direkter Attributwerte $prop$ und deren Erweiterung auf indirekte Attributwerte $propTrans$,
- die verschiedenen Bildmengen aller $prop$-Auswahlfunktionen

  $$pr_i \subseteq prop,\ 1 \leq i \leq r, r \in \mathbb{N},$$

- und deren „transitive Hüllen“ $pr_i\,Trans$ in Erweiterung um die hierarchischen Attributwerte,
- die Rollenconstraint-Relation

  $$roleCons \subseteq Q \times \{\text{one}, \text{all}\},$$

- ein Constraint-Spezifikationstupel

  $$t \in \prod_{i=1}^{n} \left(domain(A_i) \cup \{*\}\right),$$

- eine Constraintart $\mathfrak{C}$, in der eine Constraintreferenzmenge zu spezifizieren ist.

Es werden folgende Abkürzungen eingeführt[25]:

$$\mathfrak{C}^u_{pr_i\,Trans}(t) := \mathfrak{C}\left(\left(Q_{all}^{roleCons}\right)_t^{propTrans} \cup \left(Q_{one}^{roleCons}\right)_t^{pr_i\,Trans}\right)$$

[25] Der Index im Constraint $\mathfrak{C}^u$ ist an „*u*ser specification“ angelehnt und dient zur Unterscheidung von den Constraints $\mathfrak{C}$, welche sich auf Fragenmengen (Constraintreferenzmengen) beziehen.

und

$$\mathfrak{C}^u(t) := \bigvee_{i=1}^{r} \mathfrak{C}^u_{pr_i\,Trans}(t)$$

Ist $t \in \prod_{i=1}^{n} (range(A_i) \cup \{*\})$ ein Attributwerttupel direkter Attributwerte, so gilt

$$\mathfrak{C}^u_{pr_i\,Trans}(t) = \bigvee_{i=1}^{r} \mathfrak{C}\left(\left(Q^{roleCons}_{\textit{all}}\right)^{prop}_{t} \cup \left(Q^{roleCons}_{\textit{one}}\right)^{pr_i}_{t}\right).$$

**Beispiel 5.2.53 (Abkürzende Constraintspezifikation)** Die Anforderung

„Zwischen 15 und 18 Fragen zu Thema *topicB*."

aus der Testspezifikation des Beispiels 5.2.51 kann mit der Notations-Definition 5.2.52 kurz durch

$$cardinalityCons^u((*, *, \textit{topicB}), 15, 18)$$

bei gegebener Rollenconstraint-Relation $roleCons$ beschrieben werden.

#### 5.2.2.8 Bestandteile einer typischen Testspezifikation

In Anlehnung an die Definitionen und Beispiele vorheriger Abschnitte wird zusammenfassend eine kurze Übersicht über eine typische Testspezifikation angegeben:

**Definition 5.2.54 (Typische Testspezifikation)** Elemente einer Testspezifikation zu einem gegebenen Fragenpool (siehe Abschnitt 5.2.1) sind:

1. Explizite oder implizite Definition der Rollenconstraint-Relation

   $$roleCons \subseteq Q \times \{\text{one,all}\}$$

2. Angabe der Constraints $\mathfrak{C}$ ($cardinalityCons$, $containCons$, $weightsumCons$, $excludeCons$, $percentCons$, $optimizeCons$), in denen eine Constraintreferenzmenge $C$ zu spezifizieren ist.

   - Explizite Spezifikation:

     $$\mathfrak{C}(C)$$

- Implizite Spezifikation, $t \in \prod(domain(A_i) \cup \{*\})$ :[26]

  $\mathfrak{C}^u(t)$

3. Angabe der Constraints $\mathfrak{C}$ ($choiceCons$, $optimizeCons$, $conditionCons$), in denen eine Constraintmenge $C$ zu spezifizieren ist.

   - (Explizite) Spezifikation:

     $\mathfrak{C}(C)$

   - Werden die Constraints, welche in Constraintmengen des Constraints $choiceCons$ auftreten, getrennt modelliert, so sind sie als *nicht hart* anzusehen. Dies muss in der Regel nicht eigens angegeben werden.
   - Üblicherweise besteht die Constraintmenge von $optimizeCons$ aus allen anderen harten Constraints der Testspezifikation.
   - In diesem Anwendungsbereich spielen $choiceCons$ und $conditionCons$ nur eine nebensächliche Rolle.

In Abschnitt 5.2.4 wird ein weiteres Beispiel einer typischen Testspezifikation – mit Blick auf eine mögliche graphische Bedienoberfläche und einer internen Repräsentation – gezeigt.

#### 5.2.2.9 Weitere Anforderungen

In diesem Anwendungsbereich sind natürlich noch weitere benutzerdefinierte Anforderungen denkbar. Einige hiervon sind angedacht:

- *Ausschluss von ähnlichen Fragen:* Ein Anwender möchte ähnliche Fragen innerhalb eines Tests ausschließen.
  - Die Ähnlichkeit der Fragen – hierfür hat der Anwender Kriterien einzuführen – kann mit einer zusätzlichen Relation $similar$ modelliert werden: Sind zwei Fragen $q_1$ und $q_2$ ähnlich, so kann dies durch $similar(q_1, q_2)$ ausgedrückt werden.
  - Es ist zu fordern, dass die Relation $similar$ reflexiv und transitiv ist.
  - Durch ein Constraint $excludeSimilar$ können Fragenmengen spezifiziert werden, von denen im Test keine ähnlichen Fragen auftreten dürfen.

[26] Siehe Definition 5.2.52

- *„Vermeidung des gleichen oder ähnlichen Tests wie bestimmte vorliegende Tests“:* In gewisser Weise ist dies eine Anforderung an die Gesamtheit der Tests. Es müssten hierbei Kriterien definiert werden, unter welchen Bedingungen zwei Tests als ähnlich anzusehen sind.
- *Einschränkung des Fragenkataloges über „erlaubte Themenbereiche“:*
  - UND-Semantik: Eine Frage ist dann erlaubt, wenn ALLE ihr zugeordneten Themen erlaubt sind.
  - ODER-Semantik: Eine Frage ist dann erlaubt, wenn mindestens eines ihrer zugeordneten Themen erlaubt ist.

### 5.2.3 Lösungsmodell

An einen Test oder eine Prüfung werden gewisse Anforderungen (vgl. Abschnitt 5.2.2) gestellt. Ein gültiger Test muss diese erfüllen.

**Definition 5.2.55 (Testspezifikation)** Eine *Testspezifikation* besteht aus einer endlichen Menge von Constraints ($Constraint$, siehe Abschnitt 5.2.2). Eine bestimmte Spezifikationen kann als Objekt der Klasse $Testspecification$ modelliert werden.

**Definition 5.2.56 (Gültiger Test)** Ein bezüglich einer bestimmten Testspezifikation $s \in Testspecification$ *gültiger Test* $T$ ist eine Teilmenge des Fragenkataloges (siehe Definition 5.2.2), die die Spezifikation erfüllt, kurz[27]:

$$T \subseteq Q \quad \text{und} \quad T \models s$$

Die Bedingungen hierzu sind: Sind $c_j \in s$, $1 \leq j \leq l, l \in \mathbb{N}$ die neben dem Rollenconstraint definierten (harten) Constraints und $pr_i Trans$ sämtliche relevante $propTrans$-Teilmengen , so muss gelten (vgl. auch Definition 5.2.52):

- $T \models c_j$, $1 \leq j \leq l$ – die Constraints in einer Spezifikation sind somit implizit konjunktiv verknüpft.
- Treten in verschiedenen Constraints implizit oder explizit die $propTrans$-Teilmengen $pr_i Trans$ auf, so muss für alle Constraints *dieselbe* $pr_i Trans$-Teilmenge die für die Erfüllung des Constraints relevante sein! Ist also $c_{j_1} = \mathfrak{C}_1{}^u(t_1)$ und $c_{j_2} = \mathfrak{C}_2{}^u(t_2)$, so muss es ein $i \in [1, r]$ geben mit $T \models \mathfrak{C}_1{}^u_{pr_i\,Trans}(t_1)$ und $T \models \mathfrak{C}_2{}^u_{pr_i\,Trans}(t_2)$.

Die Klasse $Test$ modelliert gültige Tests. Möglicherweise kann es zu einer Spezifikation mehrere gültige Tests $T \in Test$ geben (vgl. Abbildung 5.18).

[27] Abkürzung für „erfüllt (satisfy)“: $\models$

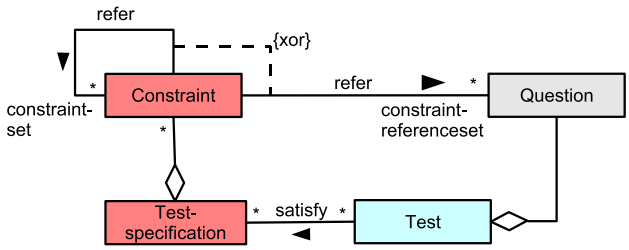



Abbildung 5.18: Lösungsmodell

### 5.2.4 Verschiedene Modellebenen einer Testspezifikation am Beispiel

In Anlehnung an Abbildung 3.4 werden für ein konkretes Beispiel die verschiedenen Modellebenen bei einer Testspezifikation aufgezeigt.

**Beispiel 5.2.57 (Verschiedene Modellebenen bei der Erstellung eines Tests)** Die folgenden Ebenen sind zu unterscheiden:

1. *Informelle Spezifikation:*

   - Die Fragen dürfen im Test mehrfach gezählt werden.
   - Insgesamt soll der Test zwischen 25 und 30 Fragen enthalten.
   - Die Fragen $q_1, q_{27}$ und $q_{40}$ dürfen nicht vertreten sein.
   - Genau 2 der 3 nachfolgenden Anforderungen sind zu erfüllen:
     - Zwischen 10 und 12 Fragen zu Thema *topicB* sind enthalten.
     - Die Gesamtbearbeitungsdauer liegt zwischen 55 und 60 (Minuten).
     - Alle Fragen zu *topicA*, Bearbeitungsdauer *5* und Schwierigkeitsgrad *medium* sind vertreten.

2. *Anwendernahe Spezifikation (GUI):*

   Abbildung 5.19 zeigt einen ersten einfachen Ansatz zur Definition einer graphischen Bedienoberfläche (GUI) für die Testkonfiguration. Der Benutzer kann hier folgende Einstellungen vornehmen:

   - Angabe über die Wertung der Fragen: Die *prop*-Rollen-Semantik sämtlicher Fragen ist entweder „eine Rolle“ oder „alle Rollen“.

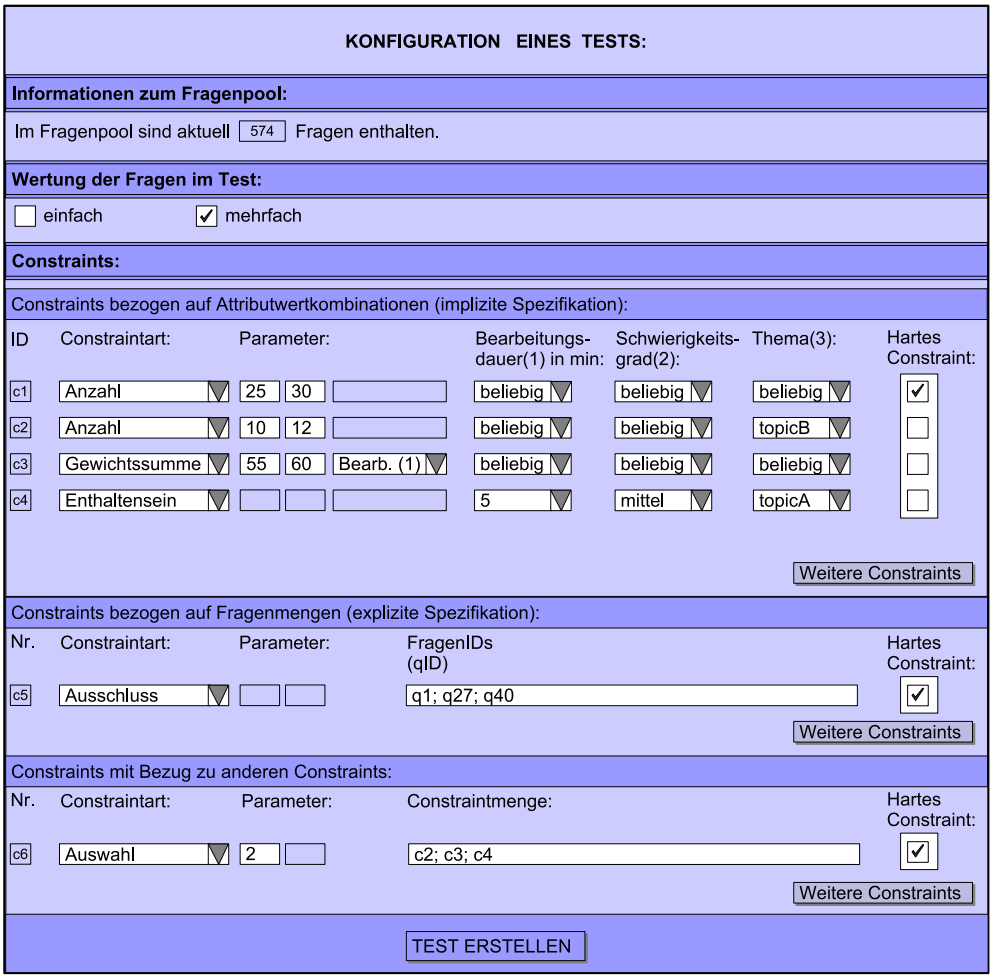



Abbildung 5.19: Beispiel einer benutzernahen Testkonfiguration

- Festlegung der Constraints: Die Constraintart ist hier jeweils mit Hilfe einer Dropdown-Liste anzugeben. Zugehörige Attributwertkombinationen sind ebenfalls einstellbar. Bei expliziter Spezifikation sind auch direkte Eingaben möglich. Zu beachten ist, dass hier auch die Constraints, die in Auswahlconstraints auftreten, eigens angegeben werden. Zur Unterscheidung von diesen werden die harten Constraints extra markiert.

3. *Spezifikation im Modell:*

   - Wertung der Fragen im Test:

     $$roleCons(q, \text{all}),\ q \in Q$$

- Modellierung der Anforderungen:
  - Constraints mit implizit spezifizierter Constraintreferenzmenge:

$$
\begin{aligned}
c_1 &= cardinalityCons^u((*, *, *), 25, 30), \text{ hartes Constraint} \\
c_2 &= cardinalityCons^u((*, *, \textit{topicB}), 10, 12) \\
c_3 &= weightsumCons^u((*, *, *), \texttt{Duration}, 55, 60) \\
c_4 &= containCons^u((5, \textit{medium}, \textit{topicA}))
\end{aligned}
$$

    bzw.

$$
\begin{aligned}
c_1 &= cardinalityCons(Q, 25, 30), \text{ hartes Constraint} \\
c_2 &= cardinalityCons(Q^{prop}_{(*,*,\textit{topicB})}, 10, 12) \\
c_3 &= weightsumCons(Q, \texttt{Duration}, 55, 60) \\
c_4 &= containCons(Q^{prop}_{(5,\textit{medium},\textit{topicA})})
\end{aligned}
$$

  - Constraint mit explizit spezifizierter Constraintreferenzmenge:

$$c_5 = excludeCons(\{q_1, q_{27}, q_{40}\}), \text{ hartes Constraint}$$

  - Constraint mit explizit definierter Constraintmenge:

$$c_6 = choiceCons(\{c_2, c_3, c_4\}, 2), \text{ hartes Constraint}$$

    bzw. in kompakter Darstellung:

$$
\begin{aligned}
s = \{ & \\
& roleCons(q, \text{all}),\ q \in Q; \\
& cardinalityCons^u((*, *, *), 25, 30); \\
& excludeCons(\{q_1, q_{27}, q_{40}\}); \\
& choiceCons(C, 2), \\
& \quad \text{wobei } C = \{cardinalityCons^u((*, *, \textit{topicB}), 10, 12), \\
& \qquad weightsumCons^u((*, *, *), \texttt{Duration}, 55, 60), \\
& \qquad containCons^u((5, \textit{medium}, \textit{topicA})) \\
& \qquad \} \\
\}
\end{aligned}
$$

4. *Modellierung im internen Formalismus (Answer Set Programming mit Gewichten):*

   Dies wird in Abschnitt 6.1.1.5 nach der allgemeinen Beschreibung der Transformation des Modells in ASP gezeigt.

## 5.3 Studienberatung und Konsistenzprüfung von Studienordnungen

Ausgehend von den Abschnitten 2.4 (vor allem 2.4.4) und 5.1 wird im Folgenden ein Modell für den Anwendungsbereich „Studienberatung und Konsistenzprüfung von Studienordnungen“ definiert. Hierbei wird zunächst die Kernstruktur modelliert. Hier geht es vor allem darum, die in ihr repräsentierten Veranstaltungen zusammen mit deren Struktur (Strukturmodell) und die Anforderungen an die zu belegenden Veranstaltungen (Constraintmodell) abzubilden. Es geht hier nicht um die Modellierung einer Prüfungsordnung, so dass beispielsweise Aspekte der Form und Dauer von Prüfungen keine Rolle spielen. Erweiterungen der Modellierung (zeitliche Aspekte, konkrete Lehrveranstaltungen) erfolgen im Abschnitt 5.3.4.

### 5.3.1 Strukturmodell

Die Rahmenvorgaben für modularisierte Studienordnungen sind sehr vage und offen. Es gibt kein sinnvolles, einheitliches Metamodell für die Studienordnungen [Eurf, Kul], so dass es im Prinzip große Freiheiten bei der Erstellung von Studienordnungen gibt. Trotzdem ist eine gewisse typische Grundstruktur festzustellen.[28]

Wichtige strukturelle Bestandteile dieses Anwendungsbereichs sind die in der Studienordnung vorgesehenen Veranstaltungen und der Modulkatalog (vgl. Definition 2.4.1). Abbildung 5.20 zeigt ein abstraktes Strukturmodell (Klassendiagramm) für modularisierte Studiengänge. Dieses ist im Beispiel der Abbildung 5.25 konkretisiert. Im Folgenden werden das Modell und dessen Elemente erläutert.

Da die Formulierungen der Studienordnungen eine gewisse Allgemeingültigkeit zum Ziele haben, stellen die darin beschriebenen Veranstaltungen nur einen Platzhalter (Templates) für ganz konkrete Veranstaltungen, die in einem bestimmten Semester an einer bestimmten Universität angeboten werden, dar. Wir bezeichnen diese daher als *Veranstaltungstemplates (Coursetemplates)*. Sie sind von den tatsächlich abgehaltenen, konkreten Lehrveranstaltungen zu unterscheiden.

**Definition 5.3.1 (Veranstaltungstemplates-Katalog und Attribute)** Ein *Veranstaltungstemplates-Katalog* ist eine endliche Menge

$$Coursetemplate = \{c_i \mid 1 \leq i \leq k, k \in \mathbb{N}\}$$

von Veranstaltungstemplate-Objekten (Veranstaltungstemplates). Diesen ist eine endliche *Sequenz von (flachen) Attributen*

$$\mathcal{A} = A_0, \ldots, A_n, n \in \mathbb{N}$$

zugeordnet. Hierbei soll $A_0$ das Identifier-Attribut darstellen.

[28] Ein entsprechendes Metamodell hätte wenig Aussagekraft; eine Modellierung in Form eines abstrakten Modells scheint hier der adäquatere Weg zu sein, da die Grundstruktur sichtbar gemacht werden kann.

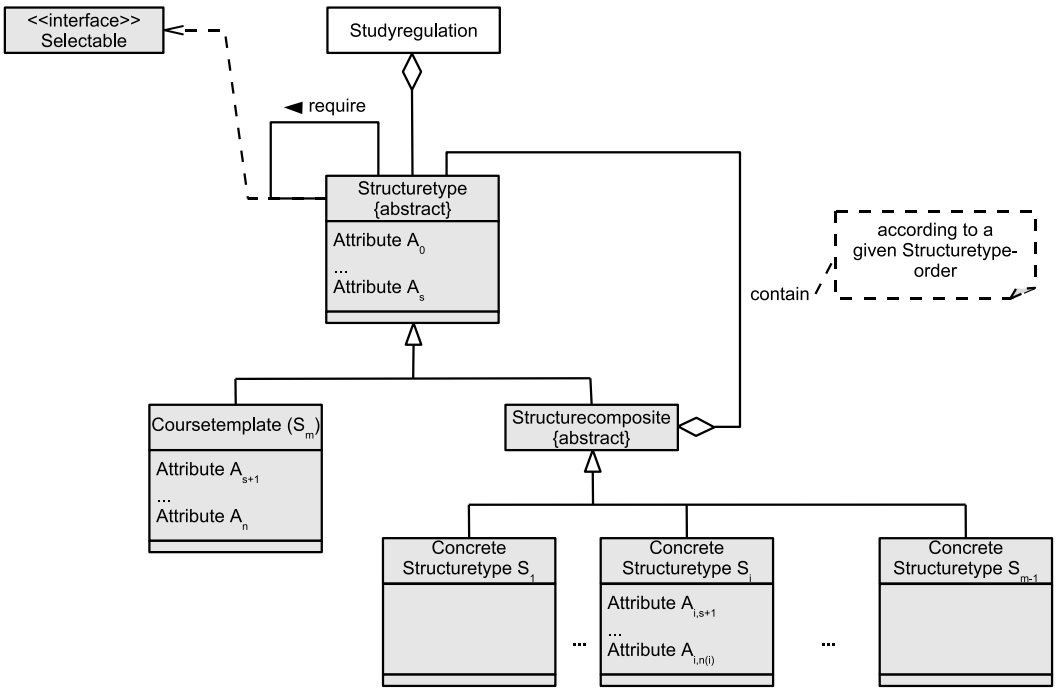



Abbildung 5.20: Abstraktes Strukturmodell, Klassendiagramm

In Analogie zu Definition 5.2.5 des Wertebereiches eines Attributes aus dem Anwendungsbereich Generierung von Tests wird hier festgelegt:

**Definition 5.3.2 (Wertebereich von Attributen)** Der *Wertebereich* $range(A)$ eines Attributes $A$ ist die endliche Menge von Attributwerten, die dem entprechenden Element bzw. Objekt direkt zugeordnet werden kann.

**Bemerkung 5.3.3 (Relationale Sicht des Veranstaltungstemplates-Katalogs)** Die relationale Sicht auf die Klasse $Coursetemplate$ bzw. deren Objektmenge ist durch

$$Coursetemplate \subseteq \prod_{i=0}^{n} range(A_i)$$

gegeben, wobei ein Veranstaltungstemplate-Objekt $c \in Coursetemplate$ durch das zugehörige Attributwert-Tupel $(c_0, \ldots, c_n) \in \prod_{i=0}^{n} range(A_i)$ eindeutig identifiziert wird:

$$c = (c_0, \ldots, c_n)$$

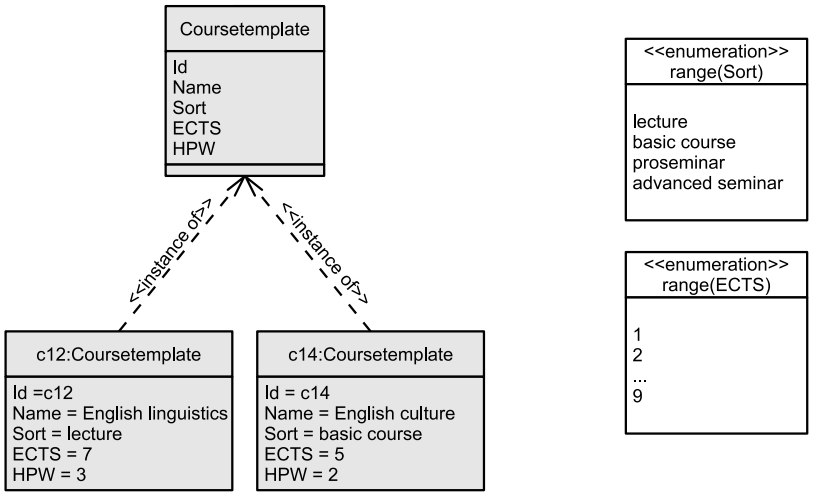



Abbildung 5.21: Beispiel: $Coursetemplate$-Klasse und -Objekte

**Beispiel 5.3.4 (Veranstaltungstemplates)** Nehmen wir an, dass die Vorlesung *Englische Linguistik* (*English linguistics*) die ID *c12* hat, pro Woche *3* Stunden umfasst und *7* ECTS-Punkte besitzt. Außerdem soll der Grundkurs *Englische Kultur* (*English culture*) mit der ID *c14* die SWS-Zahl (`HPW`) *2*

und *5* ECTS-Punkte haben. Die Veranstaltungstemplates können dann durch folgende Tupel

$(c12, \textit{English linguistic}, \textit{lecture}, 7, 3),$
$(c14, \textit{English culture}, \textit{basic course}, 5, 2)$

mit dem Schema[29]

$$\mathit{Coursetemplate}\,(\texttt{ID}, \texttt{Name}, \texttt{Sort}, \texttt{ECTS}, \texttt{HPW})$$

beschrieben werden. Die entsprechende Klassen- und Objektsicht wird in Abbildung 5.21 gezeigt. Als Wertebereiche sind beispielsweise

$\mathit{range}(\texttt{Sort}) = \{\textit{lecture}, \textit{basic course}, \textit{proseminar}, \textit{advanced seminar}\},$
$\mathit{range}(\texttt{ECTS}) = \{1, 2, \ldots, 9\}$

denkbar.

Um dem Modularisierungsgedanken – der Zusammenfassung von Stoffgebieten zu thematisch und zeitlich abgerundeten, in sich abgeschlossenen Einheiten – gerecht zu werden, werden die zahlreichen Veranstaltungstemplates einer Studienordnung noch gruppiert und ihnen dadurch eine Struktur verliehen. Die verschiedenen Gruppierungseinheiten erhalten einen Namen.

**Definition 5.3.5 (Strukturtypen)** Die *Gesamtheit der Strukturierungseinheiten (Struktureinheiten)* der Veranstaltungstemplates ist eine endliche Menge

$$\mathit{Structuretype} = \{S_i \mid 1 \leq i \leq m, m \in \mathbb{N}\}$$

von Typen (Klassen), den *Strukturtypen*. Der *Strukturtyp* $S_i$ $(1 \leq i \leq m)$, auch $S_i$*-Strukturtyp* genannt, ist eine endliche Menge

$$S_i = \{s_{i,j} \mid 1 \leq j \leq m_i, m_i \in \mathbb{N}\}$$

von $S_i$*-Strukturobjekten (*$S_i$*-Strukturelementen)*. Diesem ist eine endliche Sequenz von (flachen) Attributen mit endlichen Wertebereichen, wobei das erste Attribut das $S_i$-Identifikator-Attribut sein soll, zugeordnet.

Darüber hinaus ist auf den Strukturtypen eine Ordnungsrelation – in der Regel eine Totalordnung –

$$R_{>} \subseteq \mathit{Structuretype} \times \mathit{Structuretype}$$

definiert. Hierbei ist $\mathit{Coursetemplate}$ stets das kleinste Element.

[29] Zu beachten ist, dass die hier modellierten Veranstaltungstemplates keine konkreten Veranstaltungen sind, sondern eben nur Templates hierfür. Die `ID` ist daher unabhängig von einem konkreten Semester ein „globaler“ Schlüssel. Sie repräsentiert nicht konkrete Vorlesungsnummern.

**Beispiel 5.3.6 (Strukturtypen)** Das Beispiel aus Abbildung 5.25 enthält

$$Structuretype = \{\mathsf{Modulecatalogue}, \mathsf{Modulegroup}, \mathsf{Subjectgroup}, \mathsf{Subject}, \mathsf{Module}, \mathsf{Coursetemplategroup}, \mathsf{Coursetemplate}\}$$

als Menge der Strukturierungseinheiten. Durch

$$(\mathsf{Modulecatalogue}, \mathsf{Modulegroup}), (\mathsf{Modulegroup}, \mathsf{Subjectgroup}), (\mathsf{Subjectgroup}, \mathsf{Subject}),$$
$$(\mathsf{Subject}, \mathsf{Module}), (\mathsf{Module}, \mathsf{Coursetemplategroup}),$$
$$(\mathsf{Coursetemplategroup}, \mathsf{Coursetemplate}) \in R_{>}$$

wird auf den Strukturtypen eine Totalordnung definiert.

**Bemerkung 5.3.7 (Verschiedene Sichten der Strukturierungseinheiten)** In Definition 5.3.5 wird $Structuretype$ als endliche Menge von Typen (Klassen) aufgefasst. Dies entspricht der Klassensicht von $Structuretype$. Darüber hinaus kann – in Anlehnung an Bemerkung 5.1.1 und an das ODMG-Datenmodell [JBB+00] – eine Art Objektsicht eingenommen werden. $Structuretype$ kann als Vereinigung der Mengen der Objekte der Klassen $S_i, 1 \leq i \leq m$ aufgefasst werden. Geht aus dem Zusammenhang die gerade verwendete Sicht nicht hervor bzw. soll diese hervorgehoben werden, so kann die Notation

$$Structuretype = Structuretype^{cl} = \{S_i \mid 1 \leq i \leq m, m \in \mathbb{N}\}$$

für die Typ- und Klassensicht bzw.

$$Structuretype = Structuretype^{ob} = \bigcup_{i=1}^{m} \{o \mid o : S_i\}$$

für die Objektsicht Verwendung finden. Für ein Objekt $o$ vom Typ $S_i$ soll die Notation

$$Type(o) = S_i$$

bzw. $o \in S_i$ anstelle von $o : S_i$ erlaubt sein. Derartige Objekte sollen allgemein auch als *Strukturobjekte* oder *Strukturelemente* bezeichnet werden.

**Definition 5.3.8 (Wählbare Elemente)** Nach Definition 5.1.5 ist die Menge der *wählbaren Elemente* eine endliche Teilmenge $Selectable = \mathcal{S}$ von Objekten des Anwendungs- bzw. Problembereichs.

In dem gerade betrachteten Anwendungsbereich stimmt diese Menge mit der Menge der Objekte sämtlicher Strukturtypen überein:

„$Selectable = Structuretype^{ob}$ “

Diese Strukturelemente werden als *wählbar* bezeichnet.

**Definition 5.3.9 (Assoziation $contain$)** Vermöge der *Assoziation* $contain$ wird unter Berücksichtigung der Relation $R_>$ (vgl. Definition 5.3.5) eine Enthaltenseins-Beziehung auf den Strukturelementen definiert:

$$contain \subseteq Structuretype^{ob} \times Structuretype^{ob}$$

Diese induziert eine Zuordung der ID-Attributwerte der Strukturierungseinheiten:

$$contain \subseteq \left(\bigcup_{i=1}^{m} range(A_{i,0})\right) \times \left(\bigcup_{i=1}^{m} range(A_{i,0})\right)$$

Hierbei muss für jedes $(id_i, id_j) \in contain$ gelten, dass es der gegebenen Ordnungsrelation genügt, d. h. dass

$$(id_i, id_j) \in contain \Longrightarrow (S_i, S_j) \in R_>$$

gilt. Die Ordnungsrelation $R_>$ gibt also für einen $S_i$-Strukturtypen vor, von welchen anderen $S_j$-Strukturtypen ein $S_i$-Strukturelement $S_j$-Strukturelemente enthalten darf.

**Beispiel 5.3.10 (Assoziation $contain$)** Die Abbildung 5.22 zeigt graphisch einige Elemente einer mit der Ordnungsrelation aus Beispiel 5.3.6 kompatiblen $contain$-Assoziation: Die $contain$-Instanzen ($mc$,$mg_A$), ($mc$,$mg_B$), ($mg_B$,$sg_1$), ($sg_1$,$s_c$) genügen der gegebenen Ordnungsrelation, da $(\mathsf{Modulecatalogue}, \mathsf{Modulegroup})$, $(\mathsf{Modulegroup}, \mathsf{Subjectgroup})$, $(\mathsf{Subjectgroup}, \mathsf{Subject})$ Elemente der Ordnungsrelation sind. Aber auch ($mg_A$,$s_A$) und ($mg_A$,$s_B$) sind mit $R_>$ verträglich, da auch $(\mathsf{Modulegroup}, \mathsf{Subject})$ ein Element von $R_>$ ist.

Abbildung 5.23 zeigt eine etwas abgewandelte Situation: Wegen $(\mathsf{Module}, \mathsf{Subject}) \notin R_>$ verstößt das Element ($m_1$,$s_c$)$\in$ $contain$ gegen die gegebene Ordnungsrelation $R_>$. Dies spiegelt also eine unerlaubte Situation wieder.

Manche Strukturelemente setzen andere Strukturelemente voraus. Ein Studierender kann ein bestimmtes Strukturelement nur dann „wählen“, wenn er die vorausgesetzten Strukturelemente „bestanden“ hat. Zu beachten ist, dass es eventuell mehrere Möglichkeiten für das Bestehen eines Strukturelements gibt.

**Definition 5.3.11 (Assoziation $require$)** Die *Assoziation* $require$ modelliert direkte Voraussetzungen zwischen den Strukturelementen:

$$require \subseteq Structuretype^{ob} \times Structuretype^{ob}$$

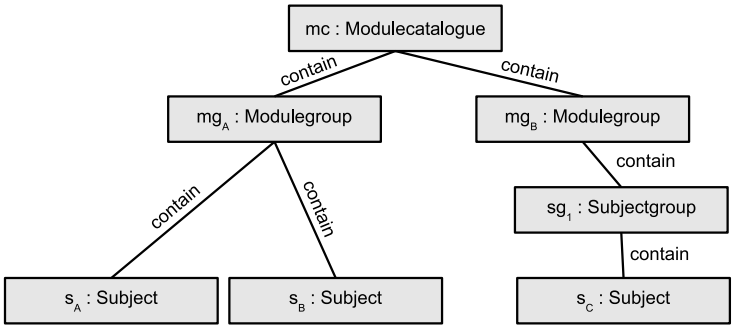



Abbildung 5.22: Beispiel: *contain*-Assoziation – verträglich mit der gegebenen Ordnungsrelation

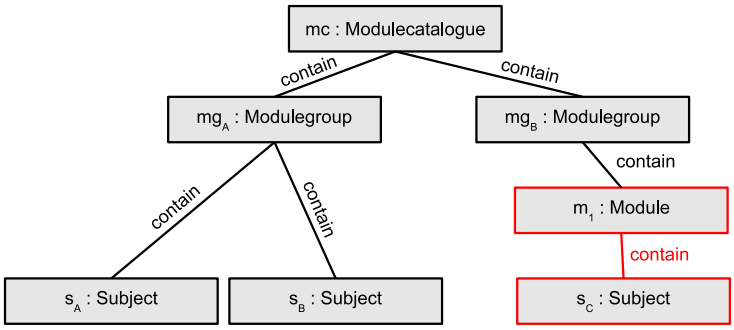



Abbildung 5.23: Beispiel: *contain*-Assoziation mit Verstoß gegen die Ordnungsrelation

Hierbei soll $(s_1, s_2) \in \mathit{require}$ für Strukturobjekte $s_1, s_2$ ausdrücken, dass $s_2$ eine direkte Voraussetzung für $s_1$ ist.[30] Hierdurch wird eine Zuordnung der ID-Attributwerte der Strukturelemente induziert:

$$\mathit{require} \subseteq \left( \bigcup_{i=1}^{m} \mathit{range}(A_{S_i,0}) \right) \times \left( \bigcup_{i=1}^{m} \mathit{range}(A_{S_i,0}) \right)$$

Ein wesentliches Element der Studienordnung ist der sogenannte Modulkatalog, welcher die eben genannten Modellierungsaspekte zusammenfasst.

**Definition 5.3.12 (Modulkatalog)** Ein *Modulkatalog* $mc$ ist eine Datenstruktur bestehend aus folgenden Modellierungselementen:

[30] In einem gültigen Studium muss also $s_2$ enthalten sein, wenn $s_1$ enthalten ist. Bei Berücksichtigung einer Reihenfolge bzw. eines zeitlichen Aspekts muss $s_2$ also schon „bestanden“ sein, um $s_1$ „belegen“ zu können. Genaueres hierzu ist im Abschnitt 5.3.4 zu finden.

1. einem Veranstaltungstemplates-Katalog mit den zugehörigen Attributen (Definition 5.3.1) und Wertebereichen (Definition 5.3.2),
2. einer Gesamtheit von Strukturierungseinheiten, den Strukturtypen und deren Elemente, den Strukturelementen zusammen mit den zugehörigen Attributen und Wertebereichen und einer Ordnungsrelation auf den Strukturtypen (Definition 5.3.5, Bemerkung 5.3.7),
3. den wählbaren Elementen (Definition 5.3.8),
4. der Assoziation $contain$, die die Strukturelemente in eine Enthaltenseins-Beziehung bringt (Definition 5.3.9) und
5. der (fakultativen) Assoziation $require$ (Definition 5.3.11).

**Beispiel 5.3.13 (Studienordnung und Modulkatalog)** Die Gegebenheiten eines Modulkatalogs (vgl. Definition 5.3.12) sind zu Beginn dieses Abschnitts 5.3 in Abbildung 5.20 veranschaulicht. Hier wird das Klassendiagramm des abstrakten Strukturmodells einer Studienordnung gezeigt.

Betrachtete konkrete Studienordnungen genügen diesem abstrakten Modell. So zeigt Abbildung 5.25 ein mögliches konkretes Klassendiagramm, das gemäß dem abstrakten Modell der Abbildung 5.20 aufgebaut ist. Hier sind die Strukturtypen schon angegeben (vgl. Beispiel 5.3.6). Die Module - Strukturelemente, oder kürzer die Module, werden nach ihrer Art, Basis- oder Prüfungsmodul (`M-Sort`: *basic*, *examination*), unterschieden. Auch bei den Coursetemplategroup - Strukturelementen, kurz Veranstaltungsgruppen, gibt es zwei Arten, nämlich die Wahl- und die Pflichtgruppen (`Ctg-Sort`: *choice*, *compulsory*). Anzumerken ist ferner, dass die bei der Modellierung von Structurecomposite auftretende Aggregation im allgemeinen Fall keine Komposition ist: Ein Strukturelement kann in mehreren Strukturelementen enthalten sein.

Ein weiteres mögliches, noch detaillierteres Klassenmodell, das dem abstrakten Modell genügt, ist in Abbildung 5.26 gezeigt. In diesem ist die Ordnung auf den Strukturtypen $Structuretype^{cl}$ schon explizit angegeben. Darüber hinaus wird hier angedeutet, dass es nur Voraussetzungen innerhalb von Module - und Coursetemplate - Strukturelementen geben kann.

Um die strukturellen Gegebenheiten einer konkreten Studienordnung (Modulkatalog) im Detail darzustellen, wird neben dem Klassendiagramm, welches ja noch nicht die konkreten Strukturelemente modelliert, auch das entsprechende konkrete Objektdiagramm benötigt. Ein solches Objektdiagramm zum Klassendiagramm aus Abbildung 5.26 findet sich in Abbildung 5.27. Dieses stellt die strukturellen Gegebenheiten auf der Ebene der Strukturelemente dar und spiegelt so die Struktur einer konkreten Studienordnung (Modulkatalog) wider.

Diese hier auftretenden verschiedenen Ebenen der Modellierung der Struktur einer Studienordnung (Modulkatalog) sind in Abbildung 5.24 aufgezeigt: Die konkreten Studienordnungen werden mit Hilfe der Objektdiagramme der konkreten Klassendiagramme, welche sich an die Vorgaben des abstrakten Modells halten, modelliert. An dieser Stelle sei nochmals betont, dass hier die Studienordnung

mit ihren Veranstaltungstemplates modelliert wird und daher auch keine „semesterspezifischen“ Bestandteile enthalten sind. Diese Modellierung ist unabhängig von einem konkreten Lehrangebot einer bestimmten Universität in einem bestimmten Semester. Veränderungen im Modell sind nur bei Änderungen der (strukturellen Gegebenheiten der) Studienordnung notwendig.

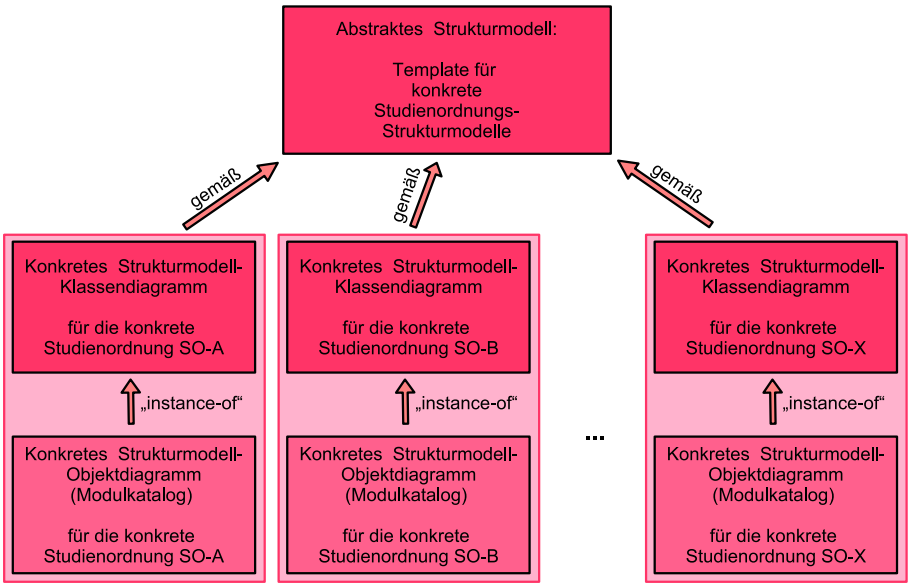



Abbildung 5.24: Strukturmodell: Verschiedene Ebenen der Modellierung

Eine vereinfachende Notation des Objektdiagrammes der Abbildung 5.27 sieht man in Abbildung 5.28. Die Klassenbezeichnungen befinden sich in dieser Darstellung am Rand. Manche Attributwerte werden nicht notiert, wenn sie entweder aus dem Kontext heraus ersichtlich sind oder gerade nicht relevant sind. Die ID-Attributwerte sind beispielsweise nicht angegeben, da sie mit den Objektbezeichnungen übereinstimmen. Die oberste Ebene ( Studyregulation ) lässt man der Einfachheit halber ebenso oftmals weg.

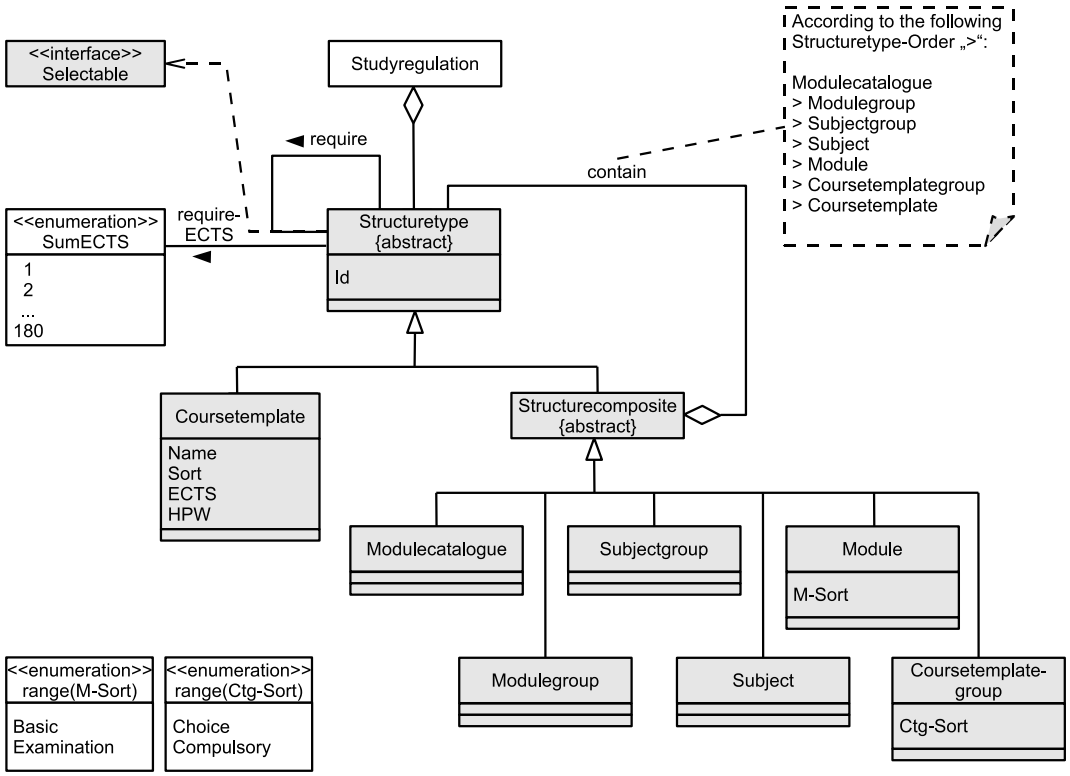



Abbildung 5.25: Beispiel: konkretes Strukturmodell-Klassendiagramm

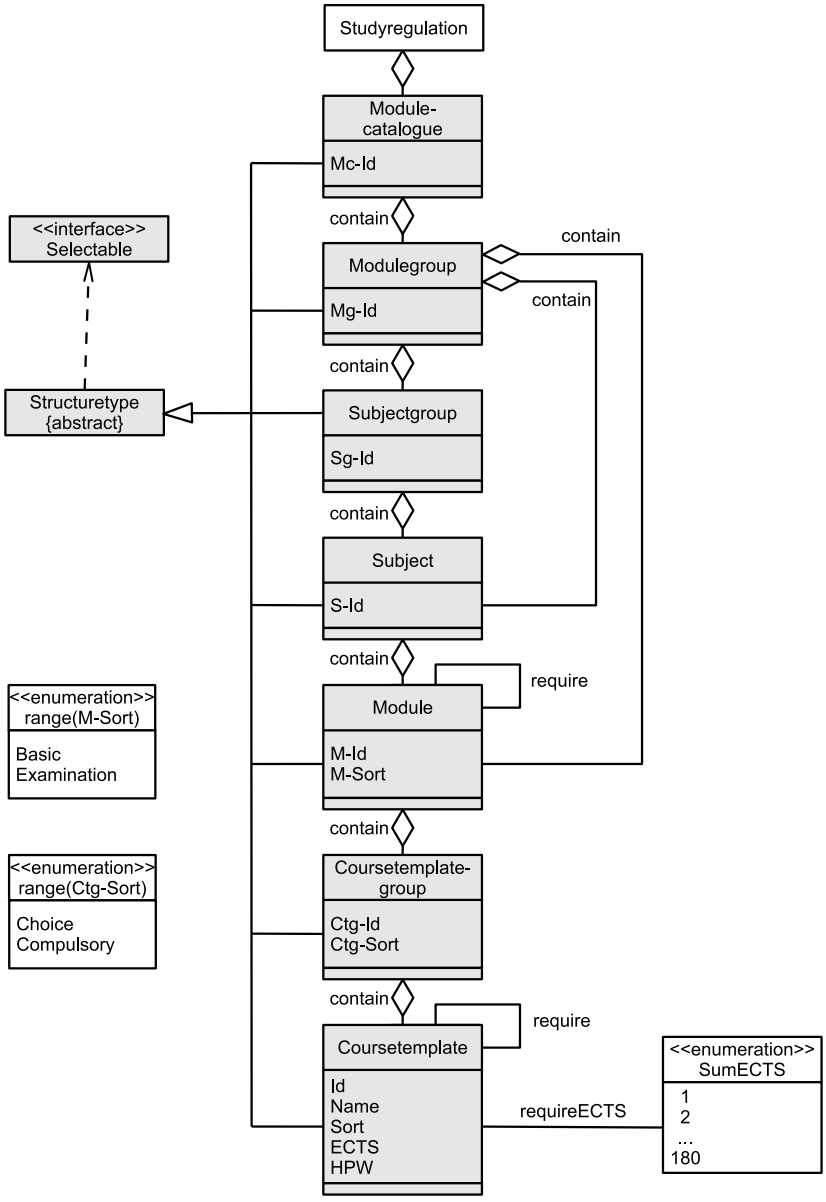



Abbildung 5.26: Beispiel: Konkretes Strukturmodell, „feineres“ Klassendiagramm

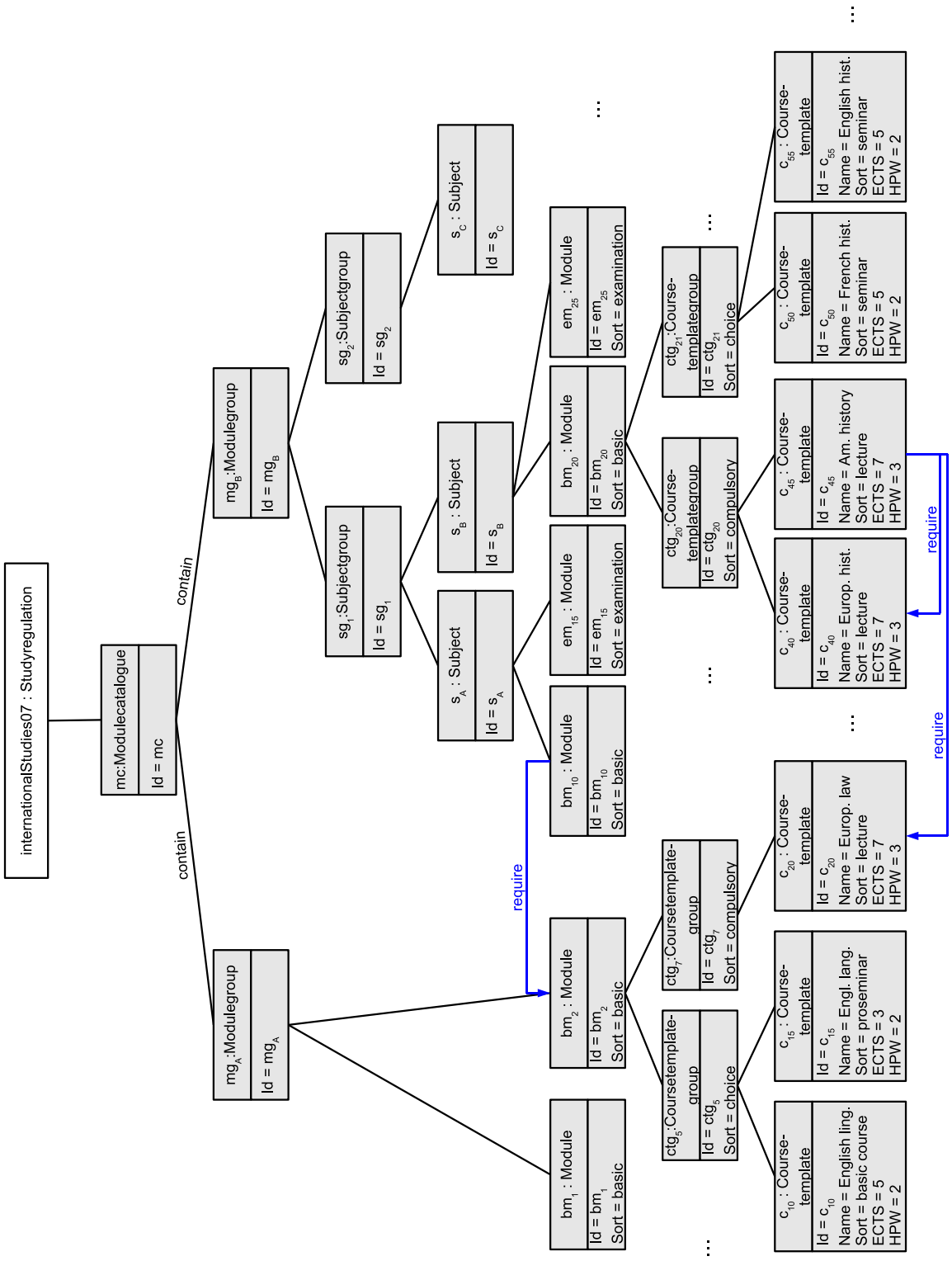



Abbildung 5.27: Konkretes Strukturmodell, Ausschnitt aus einem Objektdiagramm

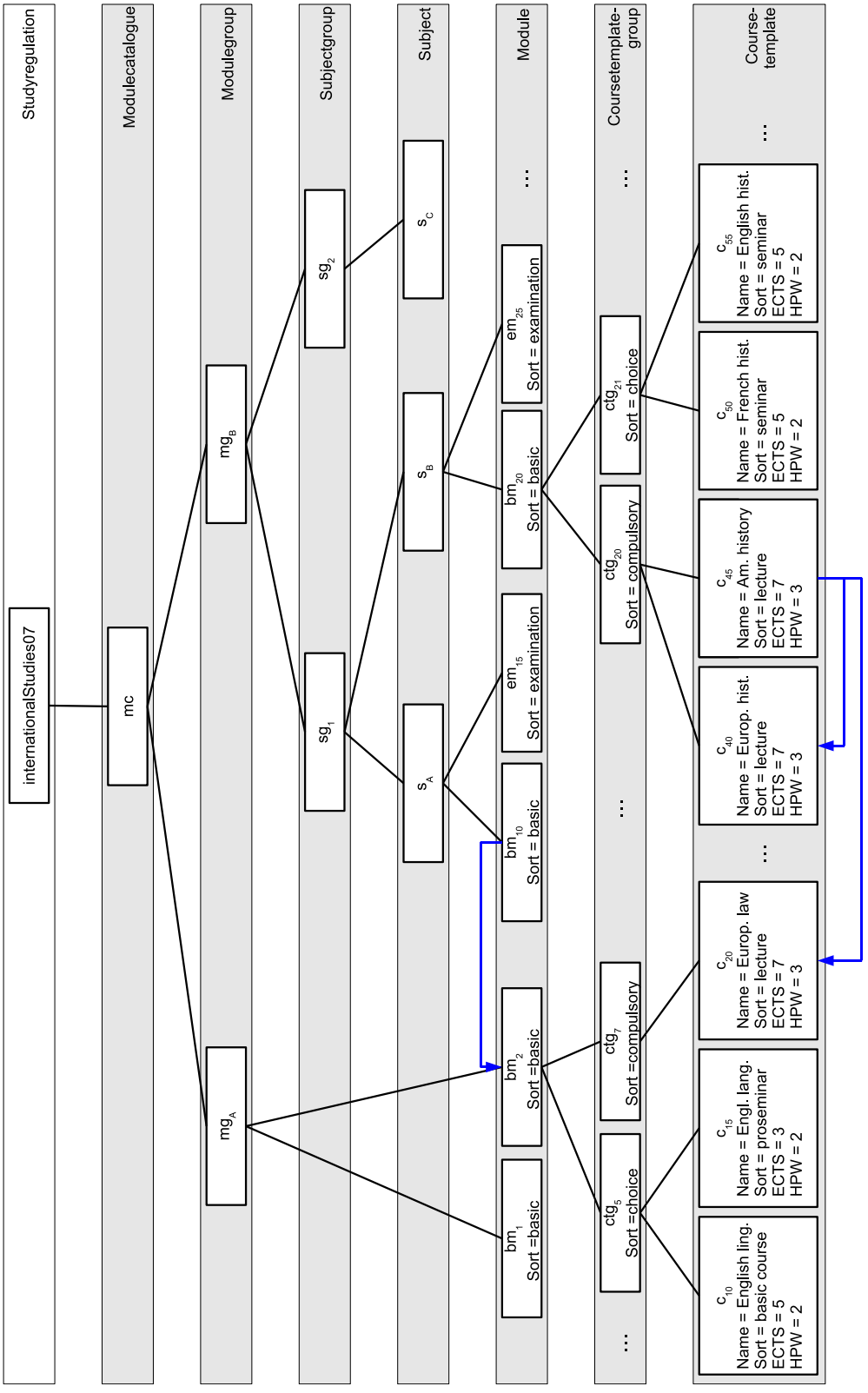



Abbildung 5.28: Konkretes Strukturmodell, Ausschnitt aus einem Objektdiagramm in vereinfachter Notation

Man kann einen Modulkatalog auch als Graphen betrachten. In diesem Zusammenhang sei an die Begrifflichkeiten von Definition 5.1.4 erinnert.

**Bemerkung 5.3.14 (Graphsicht des Modulkatalogs)** Das Objektdiagramm eines Modulkataloges $mc$ (vgl. Definition 5.3.12 und Abbildungen 5.27 und 5.28) lässt sich als markierter Graph $G_{mc}$, oder kurz $G$, $G = (V, E)$ mit Knoten $V$ und Kanten $E$ interpretieren. Hierbei wird jedes Objekt $o$ des Objektdiagrammes bzw. dessen Identifikator mit genau einem Knoten $v \in V$ identifiziert. Die Kanten $e \in E$ repräsentieren die Instanzen der Assoziation $a$, markiert mit deren Namen: Ist $f_E$ die Kantenmarkierungsfunktion, so ist $f_E(e) = a$. Wir kürzen die Menge der Kanten, die mit $a$ markiert sind, mit $E_a \subseteq E$ ab. Auf Objektebene wird hierbei also eine Assoziation $a$ mit $E_a$ identifiziert.

Für einige Betrachtungen sind nur die $contain$-Beziehungen zwischen den Knoten von Bedeutung. Der ensprechende Graph, also etwa

$$G = (V, E), \;\; E = contain$$

ohne die $require$-Beziehungen wird ebenso als Modulkatalog-Objektgraph bezeichnet. Aus dem Zusammenhang sollte die verwendete Interpretation hervorgehen.

**Beispiel 5.3.15 (Graphsicht des Modulkatalogs)** Das Objektdiagramm des Modulkataloges aus den Abbildungen 5.27 bzw. 5.28 stellt einen Graphen

$$G = (V, E), \;\; E = contain \cup require$$

dar. Beispielsweise sind

$$(mc, mgA), (mc, mgB) \in contain,$$
$$(c_{45}, c_{40}), (c_{45}, c_{20}) \in require$$

Kanten im Objektgraphen. Der Einfachheit halber ist oftmals bei den Kanten aus $E_{contain}$ die Richtung nicht explizit eingezeichnet. Diese geht aus der zeichnerischen Repräsentation implizit hervor.

### 5.3.2 Constraintmodell

In der Regel sind nicht alle Veranstaltungstemplates einer Studienordnung zu belegen, sondern es ist eine gewisse Auswahl zu treffen. Die Möglichkeiten hierfür sind durch bestimmte Bedingungen in der Studienordnung geregelt. Verschiedene Studienordnungen unterscheiden sich hier stark in den Wahlmöglichkeiten. Einige lassen den Studierenden große Freiheiten für eigene Entscheidungen (z. B. die derzeitige Fassung der Studienordnung B.A. European Studies an der Universität Passau, vgl. [SoP]), während andere hingegen nur wenig Wahlmöglichkeiten (z. B. derzeitige Fassung der Studienordnung B.Sc. Informatik an der Universität Passau, vgl. [SoP]) lassen. Für ein gültiges Studium ist nur eine Teilmenge der Menge der Strukturelemente zu wählen.

#### 5.3.2.1 Teilmengen wählbarer Elemente

Im Folgenden werden abkürzende Bezeichnungen für bestimmte Klassen von Teilmengen wählbarer Elemente definiert.

**Definition 5.3.16 (Teilmengen wählbarer Elemente)** Sei $G = (V, E)$ der Objektgraph eines Modulkataloges, $\mathcal{S}$ die Menge der wählbaren Elemente, also

$$\mathcal{S} = Selectable = Structuretype^{ob}$$

und $\mathcal{T}$ die Menge der Strukturtypen, also

$$\mathcal{T} = Structuretype^{cl}.$$

Seien außerdem $T \in \mathcal{T}$ ein Strukturtyp, $s \in T$ ein Strukturelement, $A_{T,0}, \ldots, A_{T,r(T)}, r(T) \in \mathbb{N}$ Attribute dieses Typs und $v_{T,j} \in range(A_{T,j}), 0 \leq j \leq r(T)$ zugehörige Attributwerte.

Hat $s$ im Attribut $A_{T,j}$ den Attributwert $v_{T,j}$, so wird dies wie üblich in der Form

$$s.A_{T,j} = v_{T,j}$$

notiert. Dass das Strukturelement $s$ gerade Element des $T$-Strukturtyps ist, d. h. dass es ein $T$-Strukturelement ist, kann in der Form

$$Type(s) = T$$

ausgedrückt werden (Definition 5.3.5, Bemerkung 5.3.7).

Als typische *elementare Bedingungen* an Teilmengen von $\mathcal{S}$ sind zu nennen:[31]

- $Type = T$
- $A_{T,j} = v_{T,j}$
- $\mathfrak{P} = b$, wobei $\mathfrak{P} \in \{anc, parent, child, desc, selfAnc, selfParent, selfChild, selfDesc\}$

*(Allgemeine) Bedingungen* an Teilmengen von $\mathcal{S}$ sind wie folgt induktiv definiert:

- Eine elementare Bedingung ist eine Bedingung
- Sind $cond_1$ und $cond_2$ Bedingungen, so auch $cond_1 \wedge cond_2$ und $cond_1 \vee cond_2$

[31] Die Notation der Bedingungen erfolgt in Anlehnung an die in der relationalen Algebra üblichen Darstellung von Bedingungen.

Für eine Teilmenge $D \subseteq \mathcal{S}$ und eine Bedingung $cond$ wird

$$D_{cond} := \{s \in D \mid s \text{ erfüllt die Bedingung } cond\}$$

als die Menge der Elemente in $D$, die die Bedingung $cond$ erfüllen, definiert.[32] Es gilt:

$$D_{cond_1 \wedge cond_2} = D_{cond_1} \cap D_{cond_2}$$

und

$$D_{cond_1 \vee cond_2} = D_{cond_1} \cup D_{cond_2}$$

Speziell ist

$$D_{Type=T} = \{s \in D \mid Type(s) = T\}$$

die Menge der $T$-Strukturelemente aus $D$,

$$D_{A_{T,0}=v_{T,0} \wedge \cdots \wedge A_{T,r}=v_{T,r}} = \{s \in D \mid s.A_{T,j} = v_{T,j}, 0 \leq j \leq r\}$$

die Menge der Strukturelemente aus $D$, die in den Attributen $A_{T,j}$ die Werte $v_{T,j}$ besitzen und ferner

$$D_{\mathfrak{P}=b} = \{s \in D \mid b \in \mathfrak{P}(s)\},$$

wobei $\mathfrak{P} \in \{child, parent, selfChild, selfParent, desc, anc, \dots\}$ und $b \in \mathcal{S}$ gilt, die Menge der Strukturelemente aus $D$, die $b$ in der entsprechenden „Pfadmenge“ enthalten. Zu beachten ist, dass hierbei stets nur die $contain$-Pfade und keine $require$-Pfade betrachtet werden. Beispielsweise ist

$$D_{anc=b} = \{s \in D \mid b \in anc(s)\}.$$

die Menge der Strukturelemente aus $D$, für die das Element $b$ ein Vorfahre ist. Anders ausgedrückt sind es gerade die Elemente aus $D$, die Nachfahren von $b$ sind.

In Erweiterung hierzu soll für eine (endliche) Teilmenge $B \subseteq \mathcal{S}$

$$D_{\mathfrak{P}=B} = D_{\bigvee_{b \in B} \mathfrak{P}=b}$$

definiert sein. Es gilt also:

$$D_{\mathfrak{P}=B} = D_{\bigvee_{b \in B} \mathfrak{P}=b} = \{s \in D \mid \text{ Es gibt ein } b \in B \text{ mit } b \in \mathfrak{P}(s)\} = \bigcup_{b \in B} D_{\mathfrak{P}=b}$$

Beispielsweise ist

$$D_{anc=B} = \{s \in D \mid \text{ Es gibt ein } b \in B \text{ mit } b \in anc(s)\}$$

Neben $D_{cond_1 \wedge cond_2}$ soll auch die gleichbedeutende Notation

$$D_{cond_1, cond_2}$$

erlaubt sein.

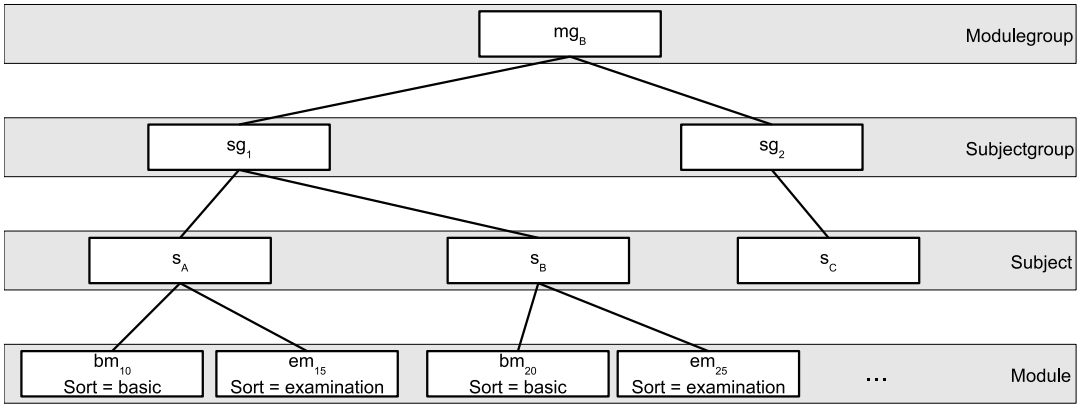



Abbildung 5.29: Ausschnitt aus dem Strukturmodell-Objektdiagramm von Abbildung 5.28

**Beispiel 5.3.17 (Teilmengen wählbarer Elemente)** Folgende Beispiele beziehen sich auf Abbildung 5.29, dem hier relevanten Ausschnitt aus Abbildung 5.28.

Die Menge der Subjectgroup-Strukturelemente, kurz Fächergruppen, ist durch

$$\mathcal{S}_{\mathit{Type}=\mathsf{Subjectgroup}} = \{sg_1, sg_2\}$$

und die Menge der Elemente, die die Fächergruppe $sg_1$ als Elternknoten haben, durch

$$\mathcal{S}_{\mathit{parent}=sg_1} = \{s_A, s_B\}$$

gegeben. Dies sind die Subject-Strukturelemente, kurz Fächer, der Fächergruppe $sg_1$. Die Prüfungsmodule (Module-Strukturelemente mit `Sort`-Attributwert *examination*) der Fächergruppe $sg_1$ werden durch

$$\mathcal{S}_{\mathit{Type}=\mathsf{Module},\mathtt{Sort}=\mathit{examination},\mathit{anc}=sg_1} = \{em_{15}, em_{25}\}$$

spezifiziert. Diese Situation wird in Abbildung 5.30 veranschaulicht. Hierbei sind die in diesem Beispiel relevanten $contain$-Pfade farblich markiert.

In diesem Beispiel ist klar, dass Module ein Strukturtyp, *examination* ein Attributwert zum Attribut `Sort` und $sg_1$ ein Strukturelement ist. Die Spezifikation der Prüfungsmodule im Fach $sg_1$ kann daher auch durch

$$\mathcal{S}_{\mathsf{Module},\mathit{examination},sg_1}$$

geschehen, da aus dem Zusammenhang die Art der Bedingungen hervorgehen. Dies soll in künftigen Beispielen allgemein erlaubt sein, wenn keine Verwechslung zu befürchten ist.

[32] Mit der bekannten Selektion $\sigma$ aus der relationalen Algebra kann $D_{cond}$ als eine abkürzende Schreibweise für $\sigma_{cond}(D)$ betrachtet werden. Soll außerdem betont werden, dass $G$ der zugrunde liegende Modulkataloggraph ist, so wird dieser in die Notation mit aufgenommen: $D^G_{cond}$.

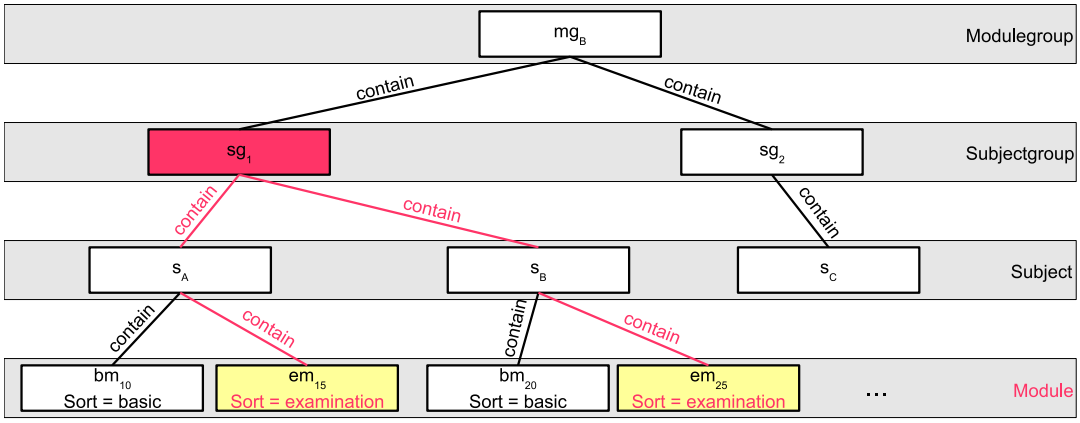



Abbildung 5.30: Teilmenge wählbarer Elemente: $\mathcal{S}_{Type=\mathsf{Module},\mathtt{Sort}=examination,anc=sg_1}$

**Definition 5.3.18 (Abkürzende Notation für Teilmengen wählbarer Elemente)** Mit den Gegebenheiten der Definition 5.3.16 und in Ergänzung hierzu soll für die Teilmenge

$$D_{Type=\mathsf{T},\mathtt{Sort}=v,anc=B}$$

auch die abkürzende Schreibweise

$$D_{\mathsf{T},v,B}$$

erlaubt sein, falls keine Verwechslung zu befürchten ist. Entsprechendes soll möglich sein, falls in der Spezifikation weniger als diese drei Bedingungen auftreten.

**Bemerkung 5.3.19 (Teilmengen wählbarer Elemente)** Die Mengen $\mathcal{S}_T, T \in \mathcal{T}$ bilden eine Partition von $\mathcal{S}$, d. h. eine überschneidungsfreie Vereinigung.

#### 5.3.2.2 Elementare Constraints

Wie schon in Abschnitt 5.1.2 und der Definition 5.1.7 beschrieben, treten folgende Elementarconstraints auf:

- Anzahlconstraints
- Vorgabe von obligatorischen Elementen
- Gewichtssummenconstraints

- Ausschluss von Elementen (Negation)

**Beispiel 5.3.20 (Elementares Constraint)** Bei einer Studienordnung und einem Modulkatalog wie in Beispiel 5.3.13 bzw. Abbildung 5.28 könnte

„Die Modulgruppe ( Modulegroup - Strukturelement) $mg_B$ ist zu wählen."

gefordert sein. Es handelt sich um ein Constraint über obligatorische Elemente (vgl. Definition 5.1.13), das daher durch

$containCons(\{mg_B\})$

modelliert werden kann. Die Anforderung

„In der Modulgruppe $mg_A$ sind alle Veranstaltungen zu wählen."[33]

kann in der Form[34]

$cardinalityCons(\{\mathcal{S}_{\mathsf{Coursetemplate},mg_A}\}, \text{all})$

wobei $\mathcal{S}$ wieder die Menge der wählbaren Elemente darstellt, notiert werden.

#### 5.3.2.3 Successorconstraints mit explizitem Kontext

In Studienordnungen treten noch weitere Arten von Constraints auf, die einen engen Bezug zum zugehörigen Objektgraphen haben. Typischerweise müssen in „gewählten" Strukturelementen des Modulkatalog-Objektgraphen (siehe Bemerkung 5.3.14) weitere nachfolgende Strukturelemente „gewählt" werden. Zur Veranschaulichung soll zunächst ein Beispiel betrachtet werden.

**Beispiel 5.3.21 (Motivation von Successorconstraints)** Abbildung 5.31 zeigt einen Ausschnitt eines Modulkatalog-Objektdiagramms mit den Strukturtypen Modulegroup, Subjectgroup und Subject. Folgende (umgangssprachlich formulierte) alternative Anforderungen werden gestellt:

„In der Fächergruppenmenge $\{sg_2, sg_3\}$ sind

(Lo) *je* Fächergruppe zwei Fächer zu wählen."

(Gl) *insgesamt* zwei Fächer zu wählen."

[33] Die exakte Beschreibung dieser Anforderung lautet: „Im Modulegroup-Strukturelement $mg_A$ sind alle Coursetemplate-Strukturelemente zu wählen." Der Einfachheit halber soll hier und in ähnlichen Beispielen auch die umgangssprachliche Beschreibung in dem Wissen um die exakte Bedeutung erlaubt sein.

[34] Zur Erinnerung: $\mathcal{S}_{\mathsf{Coursetemplate},mg_A} = \{s \in \mathcal{S} \mid Type(s) = \mathsf{Coursetemplate} \wedge mg_A \in anc(s)\}$ (Definition 5.3.18)

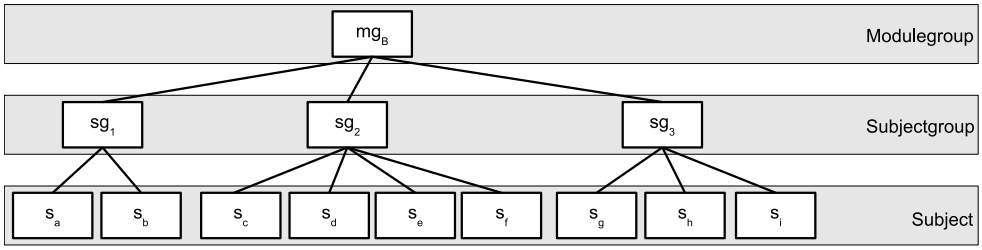



Abbildung 5.31: Ausschnitt aus einem Modulkatalog-Objektdiagramm

Hierbei handelt es sich um Anforderungen an Anzahlen von Fächern mit bestimmten vorgegebenen Vorfahrenknoten, welche eine Kontextmenge $B = \{sg_1, sg_2\}$ darstellen. Bei der ersten Anforderung (Lo) besteht ein Bezug zu jedem einzelnen Element aus dieser Kontextmenge. Die zweite Anforderung (Gl) bezieht sich auf sämtliche Nachfahren von Elementen aus der gesamten Kontextmenge. Nachfolgende Definition trägt diesem Sachverhalt Rechnung.

**Definition 5.3.22 (Successorconstraints mit explizitem Kontext)** Gegeben sei das Strukturmodell einer Studienordnung (Abschnitt 5.3.1) inklusive des Modulkatalog-Objektgraphen $G$ (vgl. Bemerkung 5.3.14). Sei $\mathcal{S}$ wieder die Menge der wählbaren Elemente. Mit Hilfe einer *Kontextmenge*

$$B \subseteq \mathcal{S},$$

den bekannten elementaren Constraints

$$\mathfrak{C} \in \{cardinalityCons, containCons, weightsumCons, excludeCons\}$$

und einer Constraintreferenzmenge

$$C \subseteq \mathcal{S}$$

werden *Successorconstraints* (*local-Successorconstraints*, *global-Successorconstraints*) eingeführt. Implizit besteht hier ein enger Bezug zum Graphen $G$.

Definition der Semantik des *local-Successorconstraints*: Für eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ gilt:

$$M \models {}^{local}_{B}\mathfrak{C}(C) : \Longleftrightarrow \text{ Für alle } b \in B \text{ gilt: } M \models \mathfrak{C}(C_{anc=b})$$

Eine Teilmenge $M$ *erfüllt bezüglich der Kontextmenge* $B$ das *local-Successorconstraint* ${}^{local}_{B}\mathfrak{C}(C)$ genau dann, wenn $M$ für jedes Element $b \in B$ das Constraint $\mathfrak{C}(C_{anc=b})$ erfüllt.

Definition der Semantik des *global-Successorconstraints*: Für eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ gilt:

$$M \models {}^{global}_{B}\mathfrak{C}(C) : \Longleftrightarrow \; M \models \mathfrak{C}(C_{anc=B})$$

Eine Teilmenge $M$ *erfüllt bezüglich der Kontextmenge* $B$ das *global-Successorconstraint* ${}^{global}_{B}\mathfrak{C}(C)$ genau dann, wenn $M$ das Constraint $\mathfrak{C}(C_{anc=B})$ erfüllt.

**Beispiel 5.3.23 (Successorcontraints)** Mit Hilfe der Successorconstraints lassen sich die Anforderungen aus Beispiel 5.3.21 wie folgt modellieren:

Die erste Anforderung (Lo) lässt sich bei der Kontextmenge $B = \{sg_2, sg_3\}$ durch

$$\overset{local}{B} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2)$$

wobei $\mathcal{S}_{\mathsf{Subject}} = \{s \in \mathcal{S} \mid Type(s) = \mathsf{Subject}\}$, ausdrücken. Ein mögliches Modell ist

$$M = \{s_c, s_e, s_g, s_h.\}$$

Die zweite Anforderung (Gl) lässt sich bei gleicher Basismenge $B$ durch

$$\overset{global}{B} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2)$$

modellieren. Mögliche Lösungen sind unter anderem

$$M_1 = \{s_c, s_h\} \quad \text{oder} \quad M_2 = \{s_c, s_e\}.$$

Die Abbildungen 5.32 und 5.33 veranschaulichen die gegebene Situation und zeigen je eine mögliche Lösung der entsprechenden Anforderung.

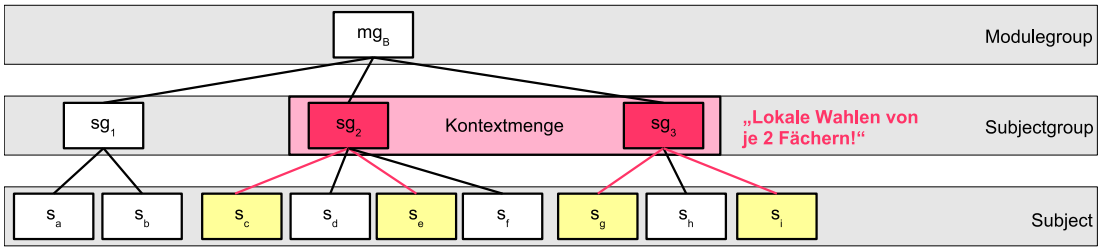



Abbildung 5.32: Beispiel: local-Successorconstraint

**Bemerkung 5.3.24 (Erste Beobachtungen zu Successorcontraints im Anwendungsbereich)** Typischerweise gilt für die im betrachteten Anwendungsbereich erforderlichen Successorconstraints:

- Es ist ein enger Bezug zum Modulkatalog-Objektgraphen gegeben.
- Hat ein Knoten mehrere Elternknoten, so muss möglicherweise eine Einschränkung auf einen Teilgraphen von $G$ vorgenommen werden.[35]

[35] Näheres hierzu im Abschnitt zum Lösungsmodell 5.3.3.

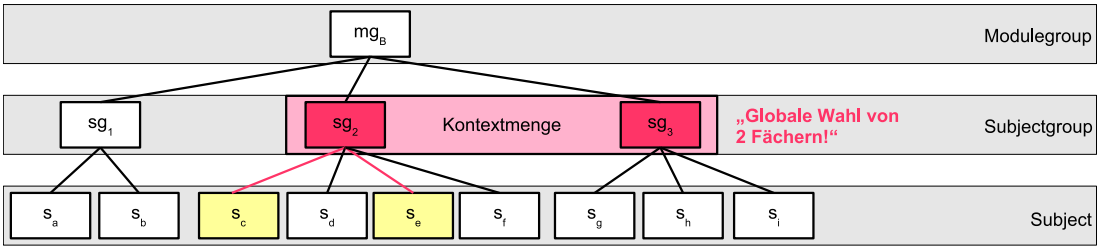



Abbildung 5.33: Beispiel: global-Successorconstraint

- Die Anforderungen beziehen sich meist auf Elemente eines bestimmten Strukturtyps, gegebenenfalls eingeschränkt durch einen zusätzlichen Attributwert des Attributs `Sort`.
- Die Kontextmenge selbst besteht ebenfalls meist aus Elementen eines bestimmten Typs und evtl. einer bestimmten Art (`Sort`).
- Häufig sind es einfach Kindelemente der Kontextmenge, auf die sich die Anforderung bezieht.

#### 5.3.2.4 Successorconstraints mit implizitem Kontext

Oftmals ist die Kontextmenge nicht explizit bekannt. Hierzu ein einführendes Beispiel:

**Beispiel 5.3.25 (Successorconstraints mit implizitem Kontext)** Die Basisanforderung lautet:

(Ba) „In der Modulgruppe $mg_B$ sind zwei Fächergruppen zu wählen.“

Weitere alternative nachfolgende Bedingungen (Successorconstraints) sind:

(Lo) „*Je gewählter* Fächergruppe der Modulgruppe $mg_B$ sind zwei Fächer zu wählen.“

(Gl) „In den *gewählten* Fächergruppen der Modulgruppe $mg_B$ sind *insgesamt* zwei Fächer zu wählen.“

Die Basisanforderung dieses Beispiels kann mit unseren bisherigen Mitteln leicht modelliert werden:

(Ba) $cardinalityCons(\mathcal{S}_{Type=\mathsf{Subjectgroup},anc=mg_B}, 2) =$
$cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2)$

Der wesentliche Unterschied zu Beispiel 5.3.21 besteht darin, dass nicht von vornherein klar ist, welche Fächergruppenmenge den Successor-Anforderungen zugrunde liegt. Hierzu gibt es verschiedene Wahlmöglichkeiten. In diesem Fall sind

$B_1 = \{sg_1, sg_2\}$, $B_2 = \{sg_1, sg_3\}$ und $B_3 = \{sg_2, sg_3\}$

die verschiedenen Lösungen der Basisanforderung. Die Anforderung $(\mathsf{Ba}, \mathsf{Lo})$ kann nun als Disjunktion von local-Successorconstraints mit den expliziten Kontexten $B_1$, $B_2$ und $B_3$ modelliert werden:

$$\begin{aligned} & cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local}_{B_1} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2) \\ & \quad \vee cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local}_{B_2} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2) \\ & \quad \vee cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local}_{B_3} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2) \end{aligned}$$

Analog hierzu kann $(\mathsf{Ba}, \mathsf{Gl})$ modelliert werden. Wie man sieht, können derartige Anforderungen mit Hilfe sämtlicher Lösungen der Basisanforderung und Successorconstraints mit explizitem Kontext modelliert werden. Dies ist jedoch sehr umständlich und wird deshalb als nicht adäquat beurteilt. Man möchte die Möglichkeit haben, Constraints zu definieren, in denen der Kontext implizit definiert ist – so wie man es auch bei XPath-Anfragen auf semistrukturierte Daten kennt (vgl. Abschnitt 4.3).

In Anlehnung an XPath-Pfadanfragen könnte eine entsprechende Notation der Anforderungen dieses Beispiels 5.3.25 folgende Gestalt haben – eine exakte Definition der Syntax und Semantik erfolgt jedoch an späterer Stelle:

1. $(\mathsf{Ba}, \mathsf{Lo})$
   $cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2)$
2. $(\mathsf{Ba}, \mathsf{Gl})$
   $cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{global} cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2)$

Die Kontextmenge wird in diesem Fall nicht notiert, sie soll sich aus der „aktuell geltenden Lösung“ der Basisanforderung $cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2)$ ergeben.

Für die Anforderungen $(\mathsf{Ba}, \mathsf{Lo})$ bzw. $(\mathsf{Ba}, \mathsf{Gl})$ wünscht man sich folgende Semantik: Eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ soll das Constraint $(\mathsf{Ba}, \mathsf{Lo})$ genau dann erfüllen, wenn

1. $M$ das Basisconstraint $(\mathsf{Ba})$ erfüllt, also zwei Fächergruppen aus der Modulgruppe $mg_B$ enthält, und wenn
2. $M$ für jede „gewählte Fächergruppe“

   $$sg \in \mathcal{S}_{\mathsf{Subjectgroup},mg_B} \cap M$$

   das Constraint

   $$cardinalityCons(\mathcal{S}_{\mathsf{Subject},sg}, 2)$$

   erfüllt, also je gewählter Fächergruppe $sg$ zwei Fächer enthält.

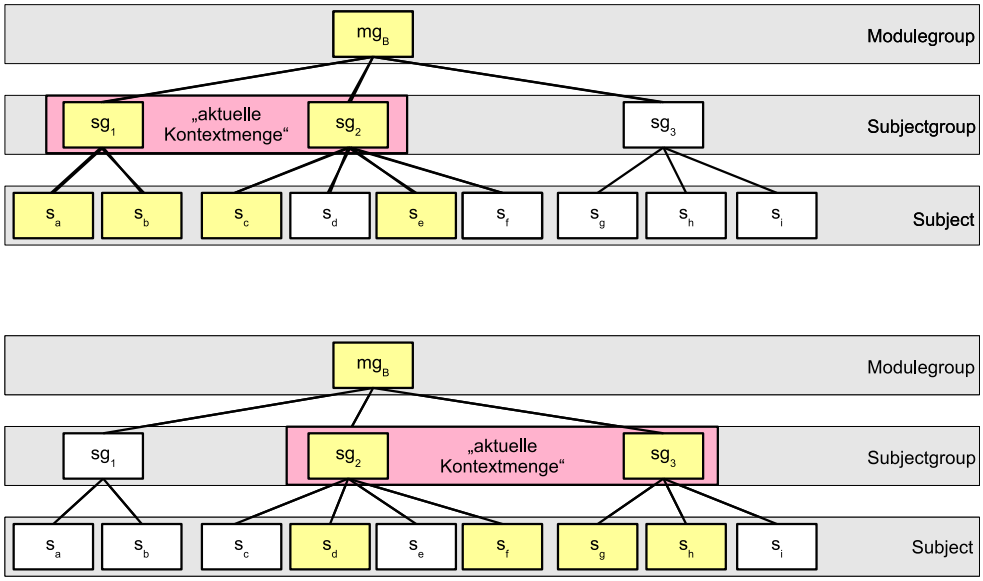



Abbildung 5.34: Zwei verschiedene mögliche Lösungen zur Anforderung $(\mathsf{Ba}, \mathsf{Lo})$

Beispiele möglicher Modelle dieser Anforderung $(\mathsf{Ba}, \mathsf{Lo})$ werden in Abbildung 5.34 durch Markierung gezeigt.

Eine Teilmenge $M \subseteq \mathcal{S}$ von $\mathcal{S}$ soll das Constraint $(\mathsf{Ba}, \mathsf{Gl})$ genau dann erfüllen, wenn

1. $M$ das Basisconstraint $(\mathsf{Ba})$ erfüllt, also zwei Fächergruppen aus der Modulgruppe $mg_B$ enthält, und wenn
2. $M$ in den „gewählten Fächergruppen“

   $$M \cap \mathcal{S}_{\mathsf{Subjectgroup}, mg_B}$$

   der Modulgruppe $mg_B$ zwei Fächer enthält, also das Constraint

   $$cardinalityCons(\mathcal{S}_{\mathsf{Subject}, M \cap \mathcal{S}_{\mathsf{Subjectgroup}, mg_B}}, 2)$$

   erfüllt.

Beispiele möglicher Modelle dieser Anforderung sind in Abbildung 5.35 markiert.

**Bemerkung 5.3.26 (Sinnvolle Modelle und implizite Anforderungen)** Die Abbildung 5.35 zeigt zwei verschiedene mögliche Modelle der Anforderung $(\mathsf{Ba}, \mathsf{Gl})$ des Beispiels 5.3.25. Im zweiten Modell wurden die Fächer in einer Fächergruppe, nämlich $sg_2$, gewählt. Die Fächer der anderen gewählten Fächergruppe $sg_3$ sind nicht im Modell vertreten. Im Rahmen der Entscheidungen für ein gültiges

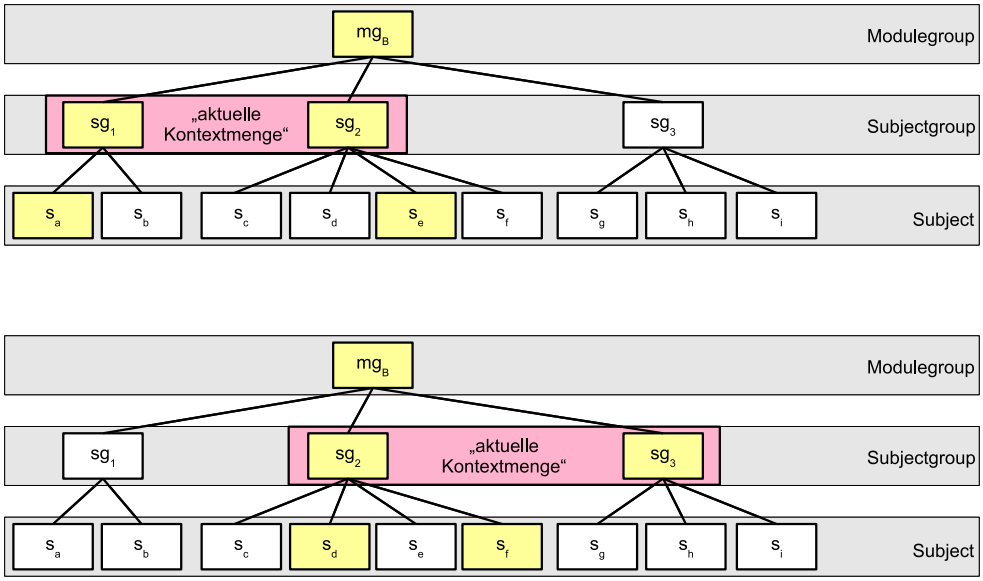



Abbildung 5.35: Zwei verschiedene mögliche Lösungen zur Anforderung $(\mathsf{Ba}, \mathsf{Gl})$

Studium ist dies in gewisser Weise nicht sinnvoll. Dies würde bedeuten, dass eine Fächergruppe gewählt wird, in der keine Fächer und damit auch keine Veranstaltungen gewählt werden. Derzeit kann man derartige Modelle noch nicht ausschließen.

Man möchte als implizite Anforderung, dass für innere Knoten $e$ gilt: Ist $e$ in einem Gesamtmodell gewählt, so gibt es mindestens einen Kindknoten, der ebenso im Gesamtmodell vertreten ist. Umgekehrt soll auch jeder gewählte Knoten $e$ mit Ausnahme der Wurzel mindestens einen gewählten Elternknoten besitzen. Näheres hierzu findet sich im Lösungsmodell (vgl. Abschnitt 5.3.3).

Zur formalen Definition der Syntax und Semantik von Constraints der Art $(\mathsf{Ba}, \mathsf{Lo})$ und $(\mathsf{Ba}, \mathsf{Lo})$ des Beispiels 5.3.25 wird zunächst ein Hilfsconstraint definiert, welches in weiteren komplexen Constraints, den Sequenz- und Baumconstraints (Abschnitte 5.3.2.5 und 5.3.2.6), Verwendung findet:

**Definition 5.3.27 (Successorconstraint mit implizitem Kontext)** Ein *Successorconstraint mit implizitem Kontext* ist ein local-Successorconstraint oder global-Successorconstraint wie in Definition 5.3.22 mit dem Unterschied, dass die Kontextmenge nicht notiert wird und aus dem Zusammenhang, wie den Lösungen einer „Basisanforderung“, hervorgeht. Ein Successorconstraint mit implizitem Kontext wird in der Form

$$^{local}\mathfrak{C}(C) \quad \text{bzw.} \quad ^{global}\mathfrak{C}(C)$$

notiert und kann nicht alleine stehen. Es handelt sich also *nicht* um ein eigenständiges Constraint. Es hat stets einen Bezug zu einem anderen Constraint und kann in der Definition weiterer komplexer Constraints auftreten.

#### 5.3.2.5 Sequenzconstraints

Successorconstraints mit implizitem Kontext (Definition 5.3.27) finden Verwendung in einer neuen Art von komplexen Constraints, den sogenannten Sequenzconstraints. Hierbei werden einige Überlegungen aus Beispiel 5.3.25 formalisiert.

**Definition 5.3.28 (Sequenzconstraint, Grundversion)** Gegeben sei ein Modulkatalog $mc$ (Definition 5.3.12); $G$ sei der zugehörige Modulkatalog-Objektgraph und $\mathcal{S}$ die Menge der wählbaren Elemente. Ein *Sequenzconstraint (Grundversion)* ist eine endliche Folge von Constraints

$$c = c_1, c_2, \ldots c_n,\ n \geq 1$$

mit folgenden Eigenschaften:

1. Das erste Constraint (*Basisconstraint*, *Wurzelconstraint*) $c_1$ der Folge ist ein elementares Constraint

   $$\mathfrak{C}_1(C_1),$$

   wobei $\mathfrak{C}_1 \in \{cardinalityCons, weightsumCons, containCons\}$ und $C_1 \subseteq \mathcal{S}$.

2. Ist $n \geq 2$, so sind die nachfolgenden Constraints $c_i, 2 \leq i \leq n$ Successorconstraints mit implizitem Kontext

   $$^{local}\mathfrak{C}_i(C_i) \quad \text{oder} \quad ^{global}\mathfrak{C}_i(C_i),$$

   wobei die elementaren Constraintarten $\mathfrak{C}_i \in \{cardinalityCons, weightsumCons, containCons\}$ Verwendung finden und $C_i \subseteq \mathcal{S}$ Constraintreferenzmengen darstellen.

   Das letzte Constraint der Folge $c_n$ wird auch als *Blattconstraint* bezeichnet. Ein Constraint der Folge ist ein sogenanntes *inneres Constraint*, wenn es weder ein Basis- noch ein Blattconstraint ist.

Die Semantik wird wie folgt definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ wählbarer Elemente gilt:[36]

$M \models c : \Longleftrightarrow$ Folgende Bedingungen sind erfüllt:

$$(1)\ M \models c_1 = \mathfrak{C}_1(C_1) \text{ und}$$

$$(2)\ \left\{ \begin{array}{ll} M \models {}^{local}_{B_1}\mathfrak{C}_2(C_2), & \text{falls } c_2 = {}^{local}\mathfrak{C}_2(C_2), \\ M \models {}^{global}_{B_1}\mathfrak{C}_2(C_2), & \text{falls } c_2 = {}^{global}\mathfrak{C}_2(C_2), \end{array} \right\}$$

wobei $B_1 = M \cap C_1$ und

$$(3)\ \left\{ \begin{array}{ll} M \models {}^{local}_{B_{i-1}}\mathfrak{C}_i(C_i), & \text{falls } c_i = {}^{local}\mathfrak{C}_i(C_i), \\ M \models {}^{global}_{B_{i-1}}\mathfrak{C}_i(C_i), & \text{falls } c_i = {}^{global}\mathfrak{C}_i(C_i), \end{array} \right\}$$

wobei $B_{i-1} = M \cap C_{i-1} \cap desc(B_{i-2}),\ 3 \leq i \leq n$

Informell gesprochen ist das Sequenzconstraint also genau dann erfüllt, wenn sämtliche Einzelcons-

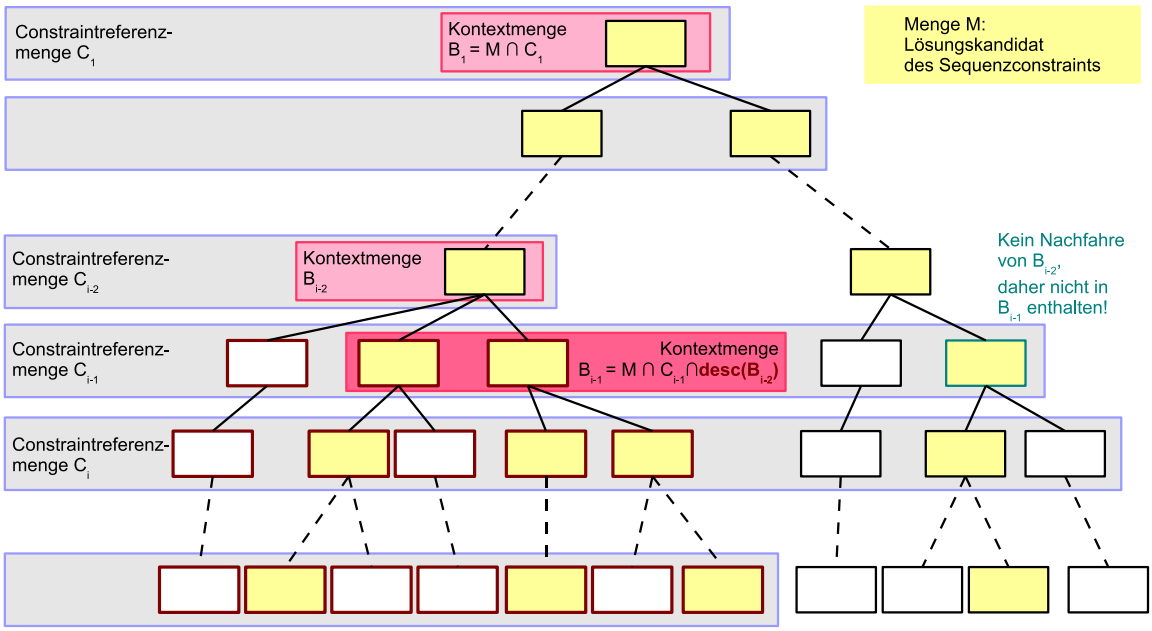



Abbildung 5.36: Veranschaulichung zu Sequenzconstraint

traints „bei jeweils aktuell geltendem Kontext“ erfüllt sind. Die Mengen $B_i, 1 \leq i \leq n$ entsprechen

[36] Implizit wird hiermit auch ausgedrückt, dass die Bedingung $(1)$ für alle $n \geq 1$, die Bedingung $(2)$ nur für $n \geq 2$ und die Bedingung $(3)$ nur für $n \geq 3$ erfüllt sein muss.

im Prinzip den „Wahlen“ bezüglich des Constraints $c_i$. Elemente einer „nachfolgenden Wahl“ sind stets auch Nachfahren der „schon gewählten Elemente“. Dies bedeutet, dass die Kontextmenge $B_{i-1}$ des $i$-ten Constraints sowohl in der Menge der bezüglich $c_{i-1}$ gewählten Elemente als auch in der Nachfahrenmenge $desc(B_{i-2})$ der vorherigen Kontextmenge enthalten sein muss. Abbildung 5.36 verdeutlicht beispielhaft die gegebene Situation.

**Beispiel 5.3.29 (Semantik von Sequenzconstraint)** Die angegebene gewünschte Semantik zu Beispiel 5.3.25 entspricht gerade dieser eben angegebenen Semantik in der Definition 5.3.28.

**Beispiel 5.3.30 (Sequenzconstraint)** Abbildung 5.37 zeigt einen gegenüber Abbildung 5.31 erweiterten Ausschnitt eines Modulkatalog-Objektdiagramms.

Die aus Beispiel 5.3.25 bekannten – hier teilweise umbenannten – Anforderungen $(\mathsf{Ba})$ und $(\mathsf{Lo})$

- $(\mathsf{Ba})$ „In der Modulgruppe $mg_B$ sind zwei Fächergruppen zu wählen.“
- $(\mathsf{Su}_1)$ „*Je gewählter* Fächergruppe der Modulgruppe $mg_B$ sind zwei Fächer zu wählen.“ (Anforderung $(\mathsf{Lo})$ aus Beispiel 5.3.25)

werden durch zusätzliche alternative Bedingungen ergänzt:

- $(\mathsf{Su}_{2a})$ „In den gewählten Fächern sind *insgesamt* drei Basismodule zu wählen.“
- $(\mathsf{Su}_{2b})$ „In den gewählten Fächern sind *alle* Basismodule zu wählen.“

Die zugehörigen Sequenzconstraints lauten:

1. $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2a})$

   $$cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2),$$
   $$^{local}cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2), {}^{global}cardinalityCons(\mathcal{S}_{\mathsf{Module},basic}, 3)$$

2. $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2b})$

   $$cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2),$$
   $$^{local}cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2), {}^{global}cardinalityCons(\mathcal{S}_{\mathsf{Module},basic}, \text{all})$$

   Offensichtlich erhält man hierzu semantisch äquivalente Constraints, wenn man in diesem Sequenzconstraint das Blattconstraint

   $$^{global}cardinalityCons(\mathcal{S}_{\mathsf{Module},basic}, \text{all})$$

   durch

   $$^{local}cardinalityCons(\mathcal{S}_{\mathsf{Module},basic}, \text{all}) \quad \text{oder} \quad {}^{global}containCons(\mathcal{S}_{\mathsf{Module},basic})$$

   ersetzt.

Die Menge der in der Abbildung 5.38 farblich markierten Elemente bildet sowohl für die Anforderung $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2a})$ als auch für die Anforderung $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2b})$ ein Modell.

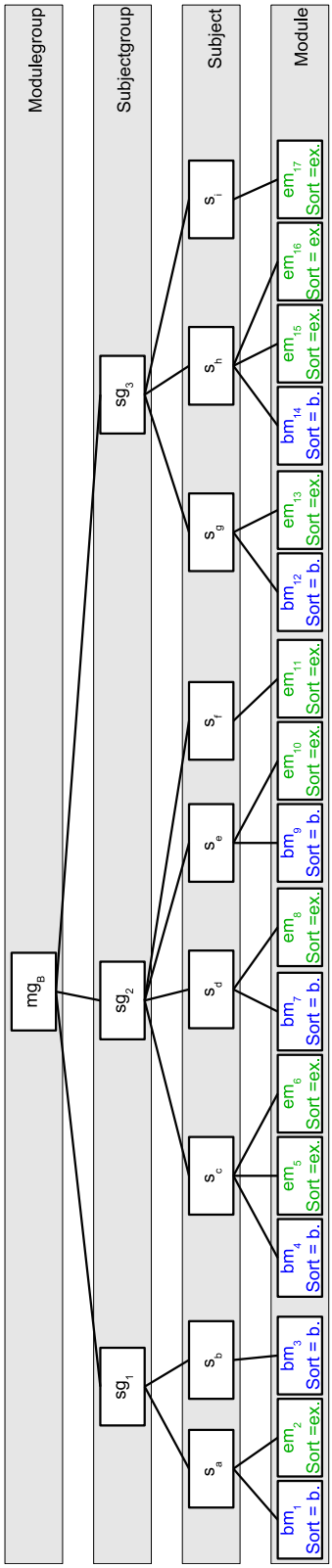



Abbildung 5.37: Ausschnitt eines Modulkatalog-Objektdiagramms

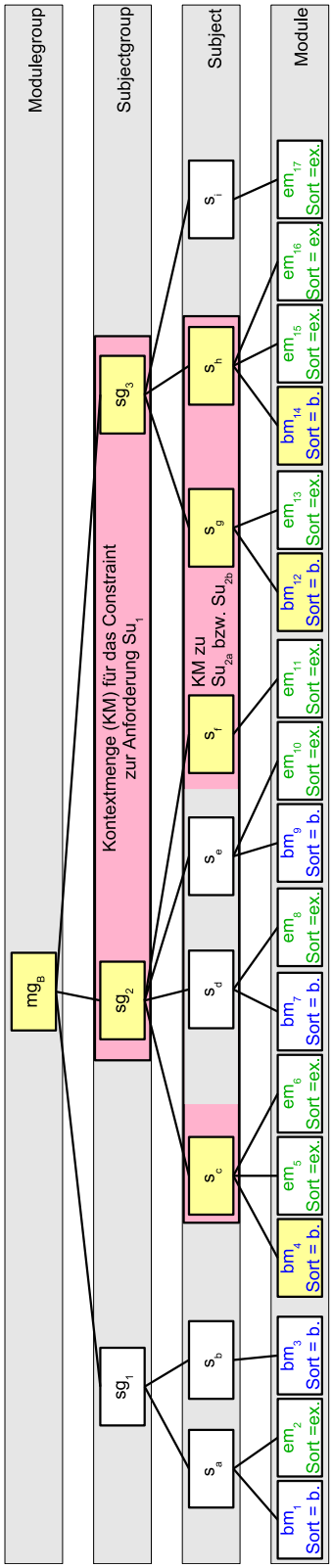



Abbildung 5.38: Gemeinsames Modell der Sequenzconstraints zu $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2a})$ und $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2b})$

**Bemerkung 5.3.31 (Navigationsrichtungen bei Sequenzconstraints)** Bei „steporientierter“ Betrachtungsweise erfolgt im Rahmen der Sequenzconstraints (Definition 5.3.28) eine Navigation von den „Kontextknoten“ in Richtung der zugehörigen *Nachfahren*.[37] In den hier betrachteten Anwendungsdomänen werden Spezifikationen, die sich auf *andere Navigationsrichtungen* beziehen, nicht benötigt und werden daher nicht näher betrachtet. Eine Erweiterung der Sequenzconstraints um diesen Aspekt ist jedoch sehr einfach möglich: Bezüglich der Syntax werden die Sequenzconstraints um die Navigationsrichtung *direction* erweitert. Entsprechende *direction*-Mengen (Verallgemeinerung von Definition 5.1.4) beeinflussen im Rahmen der Definition der Semantik der Constraints die zugehörigen Kontextmengen. In der ursprünglichen Definition sind die *direction*-Mengen $desc$-Mengen. Möchte man nun beispielsweise beim $i$-ten Einzelschritt in Richtung der Vorfahren marschieren, so könnte dies in der entsprechenden Kontextmenge $B_i$ („Wahl“ bzgl. des Constraints $c_i$) mit Hilfe der $anc$-Menge (*direction*-Menge) $anc(B_{i-1})$ in der Form $B_i = M \cap C_i \cap anc(B_{i-1})$ ausgedrückt werden.

**Bemerkung 5.3.32 (Erste Beobachtungen zu Sequenzconstraints)** Im Zusammenhang mit Sequenzconstraints $c = c_1, \ldots, c_n, n \in \mathbb{N}$ (Definition 5.3.28) lässt sich Folgendes beobachten:

1. Die Constraintreferenzmengen $C_i (1 \leq i \leq n)$ der Einzelconstraints $c_i$ sind Teilmengen der Menge der wählbaren Elemente $\mathcal{S}$, also Mengen von Strukturelementen. Oftmals handelt es sich um Mengen wählbarer Elemente wie sie in Abschnitt 5.3.2.1 dargestellt sind. Im besonderen sind hierbei Mengen der Art $D_{Type=\mathsf{T},\mathtt{Sort}=v,anc=B}$ (kurz: $D_{\mathsf{T},v,B}$ Definition 5.3.18) mit $D = \mathcal{S}$ zu nennen, deren Spezifikation über einen Strukturtypen, einem Attributwert zum Attribut `Sort` und (mehreren) konkreten Vorfahrens-Strukturelementen vorgenommen werden kann. Gegebenenfalls ist weniger zu spezifizieren. In jedem Fall ist hierdurch ein enger Bezug zum Modulkatalog-Objektgraphen gegeben.

2. Analoges gilt für die Kontextmengen.

3. Möchte man ein Modell *finden*, so kann dies in der Regel dadurch geschehen, dass schrittweise für jedes Einzelconstraint gemäß dessen Anforderung und gemäß der gegebenen Kontextmenge eine Auswahl von Strukturelementen vorgenommen wird, die dann in das zu bildende Modell „übernommen“ wird. So kann etwa ein Modell von $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2a})$ aus Beispiel 5.3.30 wie folgt gefunden werden:

   - Um das Basisconstraint $(\mathsf{Ba})$ zu erfüllen werden in der Modulgruppe $mg_B$ zwei Fächergruppen gewählt.
   - Die gewählten Fächergruppen bilden die Kontextmenge für die nächste Anforderung $(\mathsf{Su}_1)$. Aus jeder der gewählten Fächergruppen werden zwei Fächer gewählt.
   - Die gewählten Fächer bilden die neue Kontextmenge für die Anforderung $(\mathsf{Su}_{2a})$. Aus diesen wird die notwendige Auswahl von drei Basismodulen vorgenommen.

[37] Vergleiche Definition 5.3.28, Definition des Kontextes $B_{i-1} = M \cap C_{i-1} \cap desc(B_{i-2}),\ 3 \leq i \leq n$

- Sämtliche gewählte Elemente bilden schließlich ein Modell des betrachteten Sequenzconstraints zur Anforderung $(\mathsf{Ba}, \mathsf{Su}_1, \mathsf{Su}_{2a})$.

In gewisser Weise und bei „steporientierter " Sichtweise wird hierdurch entlang von Pfaden durch den Modulkatalog-Objektgraphen gemäß der gegebenen Anforderungen navigiert, wobei die Einzelschritte kontextbezogen sind.

4. Aufgrund der Möglichkeit der Navigation durch den Graphen und der Kontextbezogenheit der Einzelschritte erinnern die Sequenzconstraints an XPath-Pfadanfragen (vgl. Abschnitt 4.3). Eine nähere Beleuchtung dieses Aspekts erfolgt in späteren Ausführungen.

Gelegentlich kommt es vor, dass bei der „Navigation" durch den Modulkatalog-Objektgraphen ein Einzelconstraint auftritt, das keinen Bezug zu in „vorherigen" Schritten gewählten Elementen hat. Dieses kann dann als ein „neues" Basisconstraint (Elementarconstraint) definiert werden. Aus diesem Grund und für die Einfachheit späterer Definitionen ist folgende allgemeine Auffassung eines Sequenzconstraints hilfreich:

**Definition 5.3.33 (Sequenzconstraint, allgemeine Version)** Ein *Sequenzconstraint (allgemeine Version)* ist eine endliche Folge von Constraints

$$c = cs_1, cs_2, \ldots cs_n,\ n \in \mathbb{N}$$

mit der Eigenschaft, dass $cs_i, 1 \leq i \leq n$ Sequenzconstraints der Grundversion (vgl. Definiton 5.3.28) sind. Für diese soll die konjunktive Semantik gelten: Für eine Teilmenge $M \subseteq \mathcal{S}$ der Menge wählbarer Elemente $\mathcal{S}$ gilt:

$$M \models c : \Longleftrightarrow \text{ Für alle } 1 \leq i \leq n \text{ gilt: } M \models cs_i$$

Insbesondere ist ein Sequenzconstraint (Grundversion) auch ein Sequenzconstraint (allgemeine Version). Der Einfachheit halber spricht man oftmals im Falle beider Versionen einfach nur von *Sequenzconstraints*.

#### 5.3.2.6 Baumconstraints

Typischerweise treten die Constraints nicht nur als Folge auf: Es kann auch „Verzweigungen" geben. Diese kommen unter anderem dann vor, wenn sich die Bedingungen auf Objekte eines Strukturtyps mit in einem Attribut verschiedenen Attributwerten beziehen. Dies führt zu einer neuen Constraintart, den Baumconstraints. Eine Modifikation des Beispiels 5.3.30 macht dies deutlich:

**Beispiel 5.3.34 (Motivation von Baumconstraints)** Zusätzlich zu den schon vorhandenen Einschränkungen aus Beispiel 5.3.30 (vgl. auch Abbildung 5.37)

(Ba) „In der Modulgruppe $mg_B$ sind zwei Fächergruppen zu wählen."

($\mathsf{Su}_1$) „*Je gewählter* Fächergruppe der Modulgruppe $mg_B$ sind zwei Fächer zu wählen."

wird folgende Anforderung an ein gültiges Studium gestellt:

($\mathsf{Su}_2$) „In den gewählten Fächern sind insgesamt 3 Basis- und 4 Prüfungsmodule zu belegen."

Hier treten nach den Anforderungen an die Fächer unterschiedliche Bedingungen an die Anzahl zu wählender Module auf – unterschieden nach der Art der Module (Attribut `Sort` mit Werten *basic*, *examination*). Im Prinzip könnte man die Anforderung $(\mathsf{Ba}, Su_1, Su_2)$ als Konjunktion von zwei Sequenzconstraints modellieren:

$$cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local}cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2),$$
$${}^{global}cardinalityCons(\mathcal{S}_{\mathsf{Module},basic}, 3)$$
$$\wedge$$
$$cardinalityCons(\mathcal{S}_{\mathsf{Subjectgroup},mg_B}, 2), {}^{local}cardinalityCons(\mathcal{S}_{\mathsf{Subject}}, 2),$$
$${}^{global}cardinalityCons(\mathcal{S}_{\mathsf{Module},examination}, 4)$$

Eine vereinfachende Schreibweise führt zu einer Baumstruktur wie in Abbildung 5.39. Ein Modell wird in Abbildung 5.40 gezeigt.

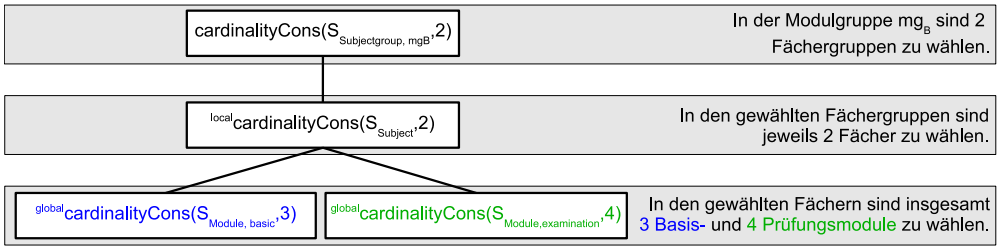



Abbildung 5.39: Beispiel: Baumconstraint

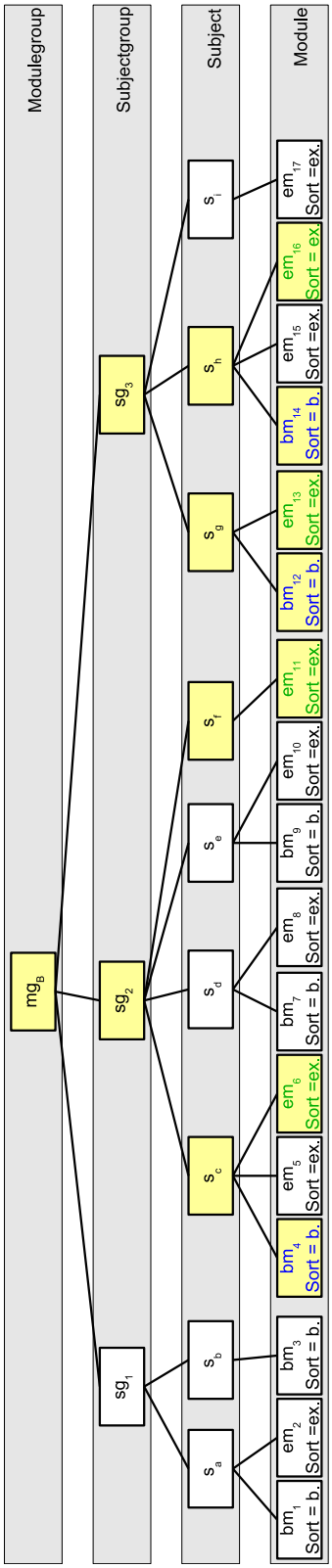



Abbildung 5.40: Beispiel: Modell des Baumconstraints

**Definition 5.3.35 (Baumconstraint)** Ein *Baumconstraint*

$$T = (C_T, E_T)$$

ist ein Baum von Constraints mit der Knotenmenge $C_T$ und der Kantenmenge $E_T$, mit der Eigenschaft, dass jeder Pfad von der Wurzel zu einem Blatt ein Sequenzconstraint (vgl. Definitionen 5.3.28 und 5.3.33) darstellt.

Es bezeichne $\mathcal{P}_T$ die Menge aller Pfade in $T$, die im Wurzelconstraint, d. h. der Wurzel von $T$, beginnen und in einem Blattconstraint, d. h. einem Blatt von $T$, enden. Es wird folgende Semantik für $T$ definiert: Ist $M \subseteq \mathcal{S}$ eine Teilmenge der Menge $\mathcal{S}$ wählbarer Elemente, so gilt:

$$M \models T :\iff \text{ Für alle } c_p \in \mathcal{P}_T \text{ gilt: } M \models c_p$$

**Bemerkung 5.3.36 (Baumconstraint)** Die „Erfüllbarkeit“ eines Baumconstraints ist eine lokale Eigenschaft. Sämtliche „lokale“ Bedingungen müssen erfüllt sein, damit das Baumconstraint erfüllt ist.

**Bemerkung 5.3.37 (Mehrfachzählung bei Strukturelementen)** In manchen Studienordnungen gibt es mehrfach zählende Strukturelemente. Wird ein solches Strukturelement gewählt, so gilt es genau so viel wie die entsprechende Anzahl anderer Strukturelemente.

Um diesen Sachverhalt zu modellieren, kann wie folgt vorgegangen werden:

1. Die Gewichtung der Strukturelemente erfolgt gemäß ihrer „Wertigkeit“. Entsprechende Constraints werden als Gewichtssummenconstraints anstelle von Anzahlconstraints formuliert.
2. Gibt es bezüglich der „Wertigkeit“ nur wenige Ausnahmen unter den Strukturelementen, d. h. gibt es nur wenige mehrfach zählende Strukturelemente, so können die Ausnahmen auch in der Form bedingter Constraints modelliert werden: Die im allgemeinen Fall zu wählende Anzahl an Strukturelementen verringert sich im Fall der Wahl mehrfach zählender Elemente um eine gemäß der Gewichtung entsprechende Anzahl (jeweils: Gewicht$-1$).

#### 5.3.2.7 Auswahl-Baumconstraints

Studienordnungen lassen teilweise auch individuelle Gewichtungen zu. In der Regel bedeutet dies, dass die Studierenden unter möglichen „gleichwertigen“ Bedingungen eine Auswahl treffen können. Die gewählten Anforderungen sind dann jedoch zu erfüllen. Dies ist beispielsweise bei Schwerpunktbildungen der Fall. Mit Hilfe der in Abschnitt 5.1.2.6 definierten $choiceCons$-Constraints kann eine Auswahl von gewissen Constraints aus einer gegebenen Constraintmenge modelliert werden. Jedoch tritt auch der Fall ein, dass derartige Anforderungen in Sequenz- oder Baumconstraints integriert und nicht unabhängig davon modellierbar sind.

Hierzu folgende Abwandlung des Beispiels 5.3.34:

**Beispiel 5.3.38 (Individuelle Gewichtung durch Auswahlmöglichkeiten)** Zusätzlich zu den schon vorhandenen Einschränkungen aus Beispiel 5.3.34

- (Ba) „In der Modulgruppe $mg_B$ sind zwei Fächergruppen zu wählen."
- ($\mathsf{Su}_1$) „*Je gewählter* Fächergruppe der Modulgruppe $mg_B$ sind zwei Fächer zu wählen." (Anforderung (Lo) aus Beispiel 5.3.25)

wird gefordert:

- ($\mathsf{Su}_2$) „In den gewählten Fächern sind insgesamt entweder
  - ($\mathsf{Su}_{2A}$) 4 Basis- und 1 Prüfungsmodul oder
  - ($\mathsf{Su}_{2B}$) 2 Basis- und 2 Prüfungsmodule

  zu belegen."

Zur allgemeinen Modellierung solcher Auswahlmöglichkeiten wird eine erweiterte Version von Baumconstraints definiert. Die grundlegende Idee ist dabei, mit Hilfe eines „Auswahlknotens", der die Anzahl der zu erfüllenden Nachfolger bzw. der entsprechenden Teilbäume oder Teilwälder festlegt, eine Art Auswahlconstraint in Baumconstraints zu integrieren. Die zu erfüllenden „Nachfolgewälder" werden dann an den Elternknoten des „Auswahlknotens" gehängt. Dieses „Baumconstraint" muss dann erfüllt werden.

Zur formalen Definition werden hierzu erst noch weitere Begriffe und Grundlagen im Zusammenhang mit Graphen – in Ergänzung zur Definition 5.1.4 – benötigt.

**Definition 5.3.39 (Konkatenation von Wäldern und weitere Begriffe)** Ist $G = (V, E)$ ein Graph, so kann die Menge $V$ der Knoten von $G$ auch als $nodes(G)$ und die Menge $E$ der Kanten von $G$ auch als $edges(G)$ bezeichnet werden. Ein Teilgraph $G' \subseteq G$ von $G$ entsteht aus $G$ durch Entfernen von Knoten und den damit verbundenen Kanten.

Ist $\mathcal{G}$ eine endliche Menge von Graphen, so bezeichne $nodes(\mathcal{G})$ die Vereinigung der Knotenmengen der Graphen aus $\mathcal{G}$, $\bigcup_{G\in\mathcal{G}} nodes(G)$, und $edges(\mathcal{G})$ die Vereinigung der Kantenmengen der Graphen aus $\mathcal{G}$, $\bigcup_{G\in\mathcal{G}} edges(G)$. Die *Vereinigung dieser Graphen*

$$G' = \bigcup_{G\in\mathcal{G}} G$$

stelle den Graphen mit der Knotenmenge $nodes(G') = nodes(\mathcal{G})$ und der Kantenmenge $edges(G') = edges(\mathcal{G})$ dar.

Eine endliche Vereinigung von Wäldern ist wieder ein Wald. Ist $\mathcal{T}$ eine Menge von Bäumen, so bildet die Vereinigung dieser Bäume einen Wald $F = \bigcup_{T \in \mathcal{T}} T$. Die Wurzel des Baums $T$ kann durch $root(T)$ notiert werden. Die Menge der Wurzeln von $F$ sei mit $roots(F)$ abgekürzt und besteht aus den $|\mathcal{T}|$ Wurzeln der den *Wald bildenden Bäume*, $trees(F) = \mathcal{T}$. Insbesondere ist jeder Baum $T$ ein Wald. In diesem Fall bestehen die Mengen $roots(T) = \{root(T)\}$ und $trees(T) = \{T\}$ aus je genau einem Element.

Sind nun $F_1$ und $F_2$ Wälder und $v \in nodes(F_1)$ ein beliebiger Knoten von $F_1$, so wird die *Konkatenation von $F_1$ mit $F_2$ an $v$*,

$$F = F_1 \star_v F_2,$$

durch das Anhängen der den Wald $F_2$ bildenden Bäume an den Knoten $v$ definiert: $F$ ist ein Wald mit

$$\begin{aligned} nodes(F) =& nodes(F_1) \cup nodes(F_2) \\ edges(F) =& edges(F_1) \cup edges(F_2) \cup \bigcup_{r \in roots(F_2)} \{(v, r)\} \end{aligned}$$

Die Wurzeln von $F$ sind gerade die Wurzeln von $F_1$. Darüber enthält $F$ genau so viele Bäume wie $F_1$: $|trees(F)| = |trees(F_1)|$. Der verbindende Knoten $v$ wird auch *Konkatenationsknoten* genannt. Die beschriebene Situation wird in Abbildung 5.41 veranschaulicht. Ist $T_0 \in trees(F_2)$, so gilt offen-

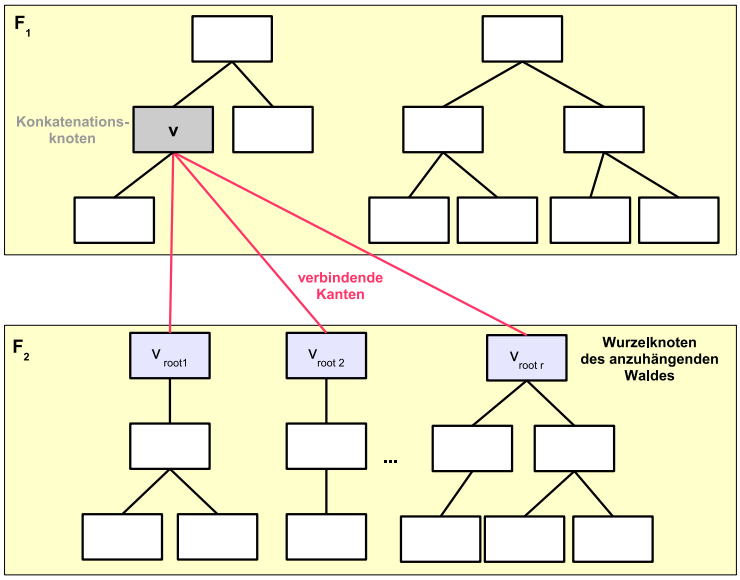



Abbildung 5.41: Konkatenation von Wäldern

sichtlich:

$$F_1 \star_v F_2 = (F_1 \star_v T_0) \star_v \left( \bigcup_{T \in trees(F_2) \setminus \{T_0\}} T \right)$$

Das Anhängen eines Waldes $F_2$ kann also durch sukzessives Anhängen der in $F_2$ enthaltenen Bäume realisiert werden. Ist $F_2 = \emptyset$, so soll $F_1 \star_v F_2 = F_1$ bedeuten.

Ist $F_1$ ein Wald, $v \in nodes(F_1)$ ein Knoten von $F_1$ und $\mathcal{F}$ eine endliche Menge von Wäldern, so wird die *Konkatenation von $F_1$ mit $\mathcal{F}$ an $v$* als Konkatenation von $F_1$ mit der Vereinigung der Wälder aus $\mathcal{F}$ an $v$ definiert:

$$F_1 \star_v \mathcal{F} = F_1 \star_v \left( \bigcup_{F \in \mathcal{F}} F \right)$$

Im Folgenden werden nun Auswahl-Baumconstraints definiert, was im Wesentlichen durch sukzessives Anhängen weiterer „Auswahlknoten“ im entsprechenden „Constraintbaum“ geschieht. Formal werden hierbei $n$-Auswahl-Baumconstraints $(n \geq 0)$ definiert.

**Definition 5.3.40 ($0$-Auswahl-Baumconstraint)** Ein $0$-*Auswahl-Baumconstraint* ist ein Baumconstraint $T = (C, E)$ mit der Knotenmenge $C$ und der Kantenmenge $E$ gemäß Definition 5.3.35. Bezeichne $\mathcal{C}$ die Menge der (im Anwendungsbereich auftretenden) Constraints, dann definiere

$$\mathcal{C}^{tree} \subseteq \mathcal{C}$$

die Menge der $0$-Auswahl-Baumconstraints (Baumconstraints).

**Bemerkung 5.3.41 (Teilwälder von Baumconstraints)** Ist $C \in \mathcal{C}^{tree}$ ein Baumconstraint, so sei

$$subforest(C)$$

die Menge der aus $C$ gebildeten Teilwälder. Darüber hinaus repräsentiere

$$subforest(\mathcal{C}^{tree}) = \bigcup_{C \in \mathcal{C}^{tree}} subforest(C)$$

die Menge aller möglichen Teilwälder von Baumconstraints. Zu beachten ist hierbei, dass ein Element $F$ von $subforest(\mathcal{C}^{tree})$ in der Regel kein Constraint im Sinne unserer bisherigen Definitionen ist.[38] Gemäß Definition 5.3.39 stellt $nodes(\mathcal{C}^{tree})$ die Menge der in $\mathcal{C}^{tree}$ auftretenden Knoten dar.

[38] Der Begriff „Teilwald“ soll hier nämlich zwei Sachverhalte beschreiben: Zum einen ist $F$ eine *endliche Vereinigung* von Teilbäumen eines möglichen Baumconstraints, d. h. $T_i$ kann mehrere Wurzeln besitzen. Zum anderen handelt es sich bei $F$ um eine endliche Menge von *Teilbäumen* von Baumconstraints, was bedeutet, dass die Wurzeln von $F$ nicht notwendigerweise elementare Constraints darstellen, sondern es sich hierbei auch um local-Successorconstraints oder global-Successorconstraints mit implizitem Kontext (vgl. Abschnitt 5.3.2.4) handeln kann. $F$ kann also in der Regel nicht für sich alleine stehen.

Ist nun $T \in \mathcal{C}^{tree}$ ein Baumconstraint, $c \in nodes(T)$ ein darin enthaltener Knoten und $\mathcal{F} \subseteq subforest(\mathcal{C}^{tree})$ eine Menge von Teilwäldern von Baumconstraints , so stellt offensichtlich

$$T \star_c \mathcal{F}$$

ebenso ein Baumconstraint (Definition 5.3.35) dar.

**Definition 5.3.42** (**1-Auswahl-Baumconstraint**) Mit Hilfe der 1-*Auswahl-Baumconstraint-Relation*

$$1-choicetreeCons \subseteq \mathcal{C}^{tree} \times nodes(\mathcal{C}^{tree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times (\mathbb{N}_0 \times \mathbb{N}_0^\infty \cup \mathbb{N}_0)$$

werden 1-*Auswahl-Baumconstraints* mit folgender Semantik definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ der Menge wählbarer Elemente wird festgelegt:

a) Ist $(T, c, \mathcal{F}, n_1, n_2) \in \mathcal{C}^{tree} \times nodes(\mathcal{C}^{tree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times \mathbb{N}_0 \times \mathbb{N}_0^\infty$, so gilt:

$$M \models 1-choicetreeCons(T, c, \mathcal{F}, n_1, n_2) : \Longleftrightarrow$$

1. Der Knoten $c$ befindet sich im Baumconstraint $T$:

   $$c \in nodes(T)$$

2. Es gibt eine Teilmenge $\mathcal{F}^{choice} \subseteq \mathcal{F}$, so dass gilt:
   i. $M$ erfüllt das Baumconstraint, das durch Konkatenation von $T$ mit $\mathcal{F}^{choice}$ an $c$ entsteht:

      $$M \models T \star_c \mathcal{F}^{choice}$$

      im Sinne von Baumconstraints (Definition 5.3.35).
   ii. Die Anzahl der in $\mathcal{F}^{choice}$ enthaltenen Teilwälder liegt im Bereich von $n_1$ und $n_2$:

      $$n_1 \leq |\mathcal{F}^{choice}| \leq n_2$$

3. Für jede Teilmenge $\mathcal{F}^{choice'} \subseteq \mathcal{F}$ der Menge der Teilwälder $\mathcal{F}$, für das $M$ ein Modell von $T \star_c \mathcal{F}^{choice'}$ ist, also $M \models T \star_c \mathcal{F}^{choice'}$, folgt:

   $$n_1 \leq |\mathcal{F}^{choice'}| \leq n_2$$

b) Ist $(T, c, \mathcal{F}, n) \in \mathcal{C}^{tree} \times nodes(\mathcal{C}^{tree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times \mathbb{N}_0$, so wird in analoger Weise zu a)

$$M \models 1-choicetreeCons(T, c, \mathcal{F}, n)$$

definiert – mit der Abwandlung, dass die in a) auftretenden Grenzen $n_1$ und $n_2$ in diesem Fall übereinstimmen und zu $n$ verändert werden.

**Bemerkung 5.3.43 (Graphische Repräsentation von 1-Auswahl-Baumconstraints)** Man kann ein 1-Auswahl-Baumconstraint, etwa $1-choicetreeCons(T, c, \mathcal{F}, n_1, n_2)$ mit einem Baumconstraint $T \in \mathcal{C}^{tree}$, einem Knoten $c \in nodes(T)$, einer Menge $\mathcal{F}$ von Teilwäldern von Baumconstraints und den Grenzen $n_1 \in \mathbb{N}_0$ und $n_2 \in \mathbb{N}_0^\infty$ auch als eine Art Baum auffassen. Zu beachten ist jedoch, dass es hierbei verschiedene Arten von Knoten und Kanten gibt. Neben den eigentlichen Constraint-Knoten treten nämlich Hilfsknoten auf: Die einen dienen zum Markieren der Teilwälder von $\mathcal{F}$, um sie so als eigenständige Einheiten erkennen zu lassen, ein anderer repräsentiert eine Art „markierten Auswahlknoten“, in dem die Grenzen $n_1$ und $n_2$ notiert sind. Abbildung 5.42 verdeutlicht dies. Das Constraint $1-choicetreeCons(T, c, \mathcal{F}, n_1, n_2)$ kann also als Baum $T^{1-ch} = (V, E)$ mit der

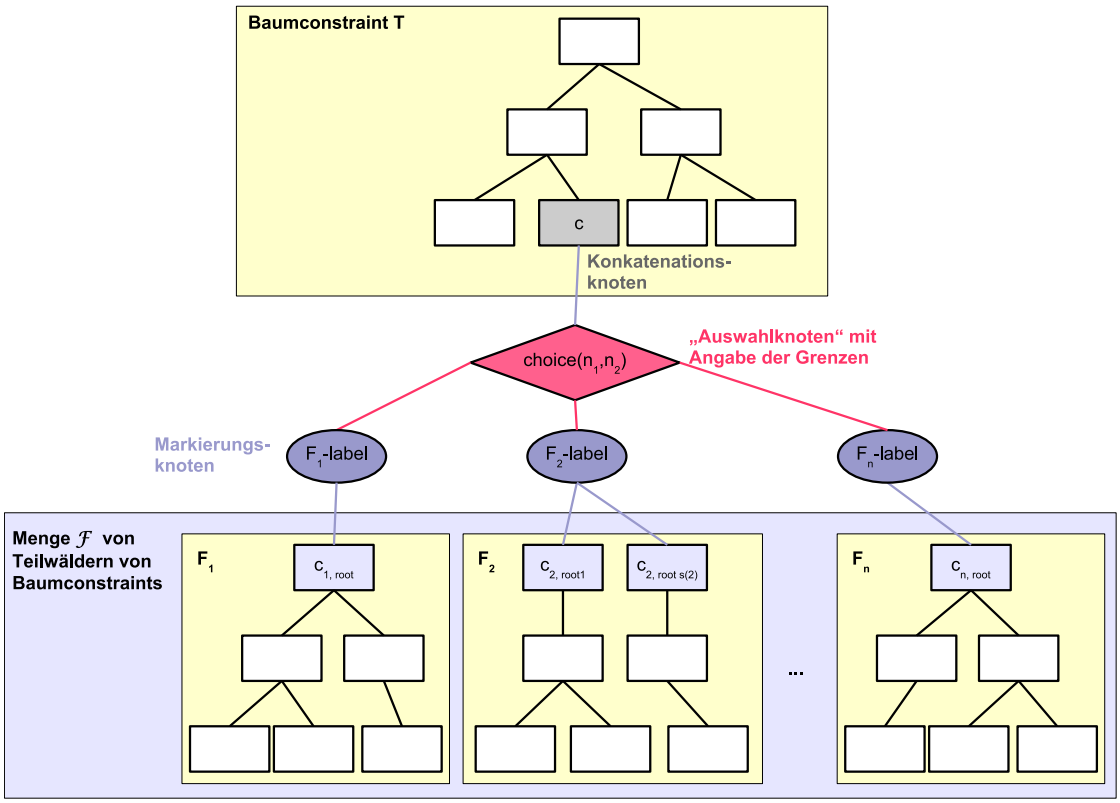



Abbildung 5.42: Graphische Repräsentation eines 1-Auswahl-Baumconstraints

Knotenmenge $V$ und der Kantenmenge $E$ betrachtet betrachtet werden, wobei gilt:

1. Die Knotenmenge $V$ besteht aus drei verschiedenen Knotenarten, $V = Co \cup L \cup Ch$:

   (a) Die Menge der *Constraintknoten* von $T^{1-ch}$ setzt sich aus der Knotenmenge von $T$ und

der von $\mathcal{F}$ zusammen:

$$Co = nodes(T) \cup nodes(\mathcal{F})$$

Sie wird auch als $constraintnodes(T^{1-ch})$ bezeichnet.

(b) Für jeden Teilwald $F$ von $\mathcal{F}$ gibt es einen Markierungsknoten $label(F)$. Die Menge dieser *Labelknoten* von $T^{1-ch}$ bildet $L$

$$L = \bigcup_{F \in \mathcal{F}} label(F)$$

und wird auch als $labelnodes(T^{1-ch})$ notiert.

(c) Schließlich gibt es noch einen weiteren Knoten, den *Auswahlknoten* $choice(n_1, n_2)$, der die „Entscheidungsmöglichkeiten“ repräsentieren soll und mit den Grenzen $n_1$ und $n_2$ markiert ist:

$$Ch = \{choice(n_1, n_2)\}$$

Diese Menge heißt auch $choicenodes(T^{1-ch})$.

2. Die Kantenmenge $E$ besteht ebenso aus drei verschiedenen Arten entsprechend den Knotenarten: $E = E_{Co} \cup E_L \cup E_{Ch}$:

(a) Die Menge der *Constraintkanten* von $T^{1-ch}$ setzt sich aus der Kantenmenge von $T$ und der von $\mathcal{F}$ zusammen:

$$E_{Co} = edges(T) \cup edges(\mathcal{F})$$

Sie wird auch als $constraintedges(T^{1-ch})$ bezeichnet.

(b) Die Kanten von den Markierungsknoten zu den Wurzeln der Teilwälder von $\mathcal{F}$ und die Kante des Konkatenationsknotens zum Auswahlknoten sind *Markierungskanten*:

$$E_L = \{(c, choice(n_1, n_2))\} \cup \bigcup_{F \in \mathcal{F}} \bigcup_{r \in roots(F)} \{(choice(n_1, n_2), label(F))\}$$

Diese Kantenmenge wird auch als $labeledges(T^{1-ch})$ notiert.

(c) Schließlich sind die Kanten, die vom Auswahlknoten ausgehen, sogenannte *Auswahlkanten*:

$$E_{Ch} = \bigcup_{F \in \mathcal{F}} \{(choice(n_1, n_2), label(F))\}$$

Diese Menge heißt auch $choiceedges(T^{1-ch})$.

**Bemerkung 5.3.44 (Relevantes Baumconstraint)** Ist nun eine Situation wie in Definition 5.3.42 und Bemerkung 5.3.43 gegeben, so ist das für die Prüfung der Modelleigenschaft von $M$ relevante Baumconstraint durch

$$T \star_c \mathcal{F}^{choice}$$

gegeben, welches durch Konkatenation von $T$ mit $\mathcal{F}^{choice}$ an $c$ entsteht. Im Prinzip kann man sich vorstellen, dass im Baum der Abbildung 5.42 zwischen $n_1$ und $n_2$ Auswahlkanten gewählt werden und der Konkatenationsknoten mit den entsprechenden Wäldern verbunden wird. Abbildung 5.43 veranschaulicht dies.

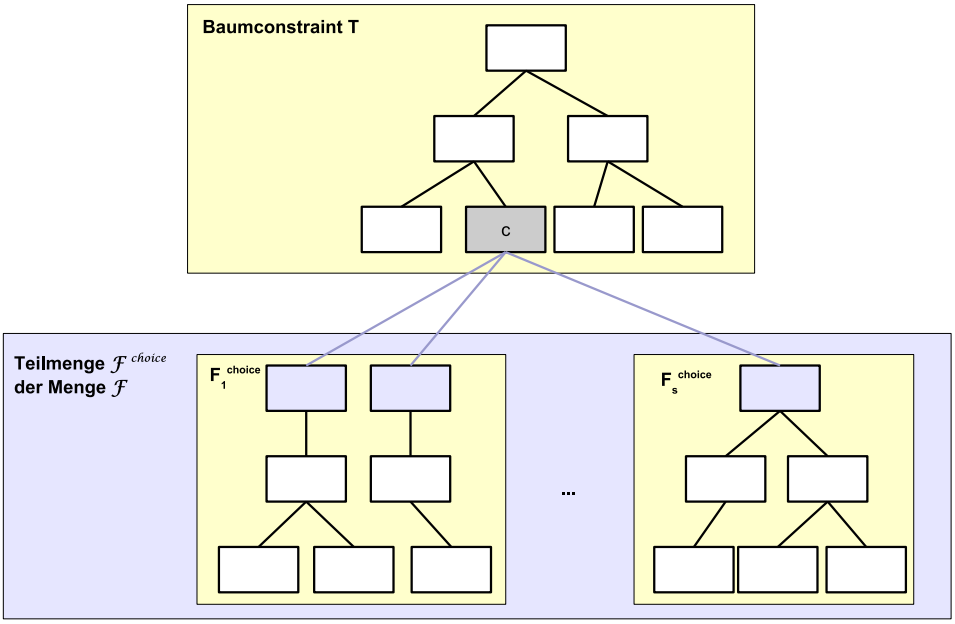



Abbildung 5.43: Relevantes Baumconstraint zu einem 1-Auswahl-Baumconstraint: $T \star_c \mathcal{F}^{choice}$

**Beispiel 5.3.45 (1-Auswahl-Baumconstraint)** Die Situation in Beispiel 5.3.38 lässt sich offensichtlich durch einen 1-Auswahl-Baumconstraint wie in Abbildung 5.44 modellieren. Abbildung 5.45 zeigt die zugehörigen relevanten Baumconstraints.

Um die allgemeine Situation zu modellieren, dass es mehrere Entscheidungsmöglichkeiten zwischen verschiedenen Constraints gibt, werden im Folgenden induktiv $n$-Auswahl-Baumconstraints ($n \geq 1$) definiert.

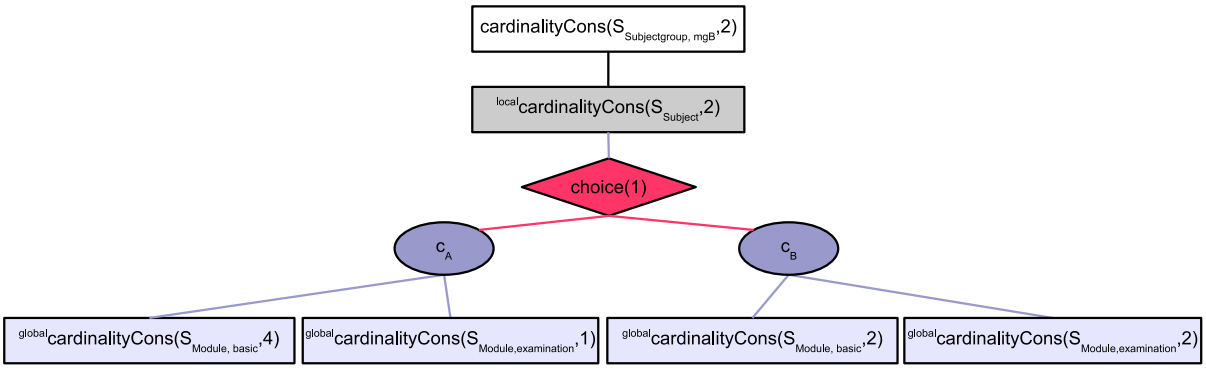



Abbildung 5.44: Beispiel: 1-Auswahl-Baumconstraint

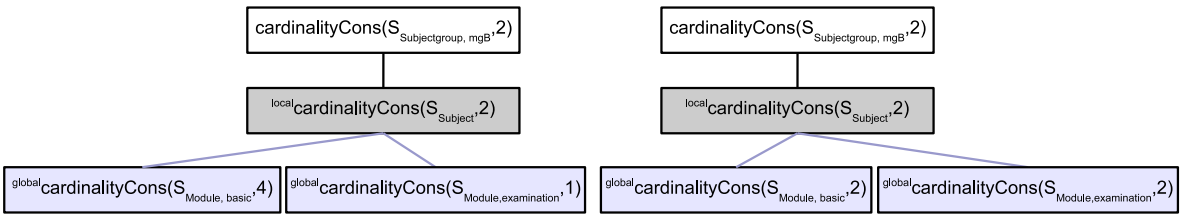



Abbildung 5.45: Beispiel: Relevante Baumconstraints zum 1-Auswahl-Baumconstraint

**Definition 5.3.46** ($n$**-Auswahl-Baumconstraint)** Für ein $n \geq 1$ seien Syntax und Semantik der $n$-Auswahl-Baumconstraints definiert.[39] Es bezeichne $\mathcal{C}^{0-ctree} = \mathcal{C}^{tree}$ die Menge der Baumconstraints und allgemein $\mathcal{C}^{n-ctree}$ die Menge der $n$-Auswahl-Baumconstraints. Mit Hilfe der $(n+1)$-*Auswahl-Baumconstraint-Relation*

$$(n+1)-choicetreeCons \subseteq \mathcal{C}^{n-ctree} \times nodes(\mathcal{C}^{n-ctree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times (\mathbb{N}_0 \times \mathbb{N}_0^{\infty} \cup \mathbb{N}_0)$$

werden $(n+1)$-*Auswahl-Baumconstraints* mit folgender Semantik definiert: Für eine Teilmenge $M \subseteq \mathcal{S}$ der Menge wählbarer Elemente wird festgelegt:

a) Ist $(T, c, \mathcal{F}, n_1, n_2) \in \mathcal{C}^{n-ctree} \times nodes(\mathcal{C}^{n-ctree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times \mathbb{N}_0 \times \mathbb{N}_0^{\infty}$, so gilt:

$$M \models (n+1)-choicetreeCons(T, c, \mathcal{F}, n_1, n_2) :\Longleftrightarrow$$

[39] Die Definition 5.3.42 wird hier verallgemeinert. Da die Baumconstraints gerade die 0-Auswahl-Baumconstraints darstellen und deren Syntax und Semantik in Definition 5.3.35 schon festgelegt ist, wäre die eigenständige Definition von 1-Auswahl-Baumconstraints nicht notwendig gewesen. Das induktive Schließen für ein $n \geq 0$ nach $n+1$ mit dem Anfang $n = 0$ hätte genügt. Zur Verdeutlichung des Vorgehens und der besseren Lesbarkeit wegen wurde die eigenständige Definition von 1-Auswahl-Baumconstraint eingeführt.

1. Der Knoten $c$ befindet sich unter den Constraintknoten des $n$-Auswahl-Baumconstraint $T$:

   $$c \in constraintnodes(T)$$

2. Es gibt eine Teilmenge $\mathcal{F}^{choice} \subseteq \mathcal{F}$, so dass gilt:
   i. $M$ erfüllt das $n$-Auswahl-Baumconstraint, das durch Konkatenation von $T$ mit $\mathcal{F}^{choice}$ an $c$ entsteht:

      $$M \models T \star_c \mathcal{F}^{choice}$$

      im Sinne von $n$-Auswahl-Baumconstraints.[40]
   ii. Die Anzahl der in $\mathcal{F}^{choice}$ enthaltenen Teilwälder liegt im Bereich von $n_1$ und $n_2$:

      $$n_1 \leq |\mathcal{F}^{choice}| \leq n_2$$

3. Für jede Teilmenge $\mathcal{F}^{choice'} \subseteq \mathcal{F}$ der Menge der Teilwälder $\mathcal{F}$, für das $M$ ein Modell von $T \star_c \mathcal{F}^{choice'}$ ist, $M \models T \star_c \mathcal{F}^{choice'}$, folgt:

   $$n_1 \leq |\mathcal{F}^{choice'}| \leq n_2$$

b) Ist $(T, c, \mathcal{F}, n) \in \mathcal{C}^{n-ctree} \times nodes(\mathcal{C}^{n-ctree}) \times \mathcal{P}(subforest(\mathcal{C}^{tree})) \times \mathbb{N}_0$, so wird in analoger Weise zu a)

$$M \models (n+1) - choicetreeCons(T, c, \mathcal{F}, n)$$

definiert – mit der Abwandlung dass, die in a) auftretenden Grenzen $n_1$ und $n_2$ übereinstimmen und in diesem Fall zu $n$ verändert werden.

**Definition 5.3.47 (Auswahl-Baumconstraint)** Ein $n$-Auswahl-Baumconstraint $(n \geq 0)$ wird auch einfach als *Auswahl-Baumconstraint* bezeichnet. Die Menge dieser Constraints ist durch die Vereinigung der $n$-Auswahl-Baumconstraints $(n \geq 0)$ gegeben:

$$\mathcal{C}^{ctree} = \bigcup_{n=0}^{\infty} \mathcal{C}^{n-ctree}$$

**Bemerkung 5.3.48 (Graphische Repräsentation von $n$-Auswahl-Baumconstraints)** Auch die allgemeinen $n$-Auswahl-Baumconstraints besitzen eine graphische Veranschaulichung. Das in Bemerkung 5.3.43 beschriebene Verfahren kann sehr einfach für den allgemeinen Fall $n \in \mathbb{N}$ übertragen werden. Zu beachten ist hierbei, dass es nun mehrere Auswahlknoten geben kann. Abbildung 5.46 skizziert die Situation.

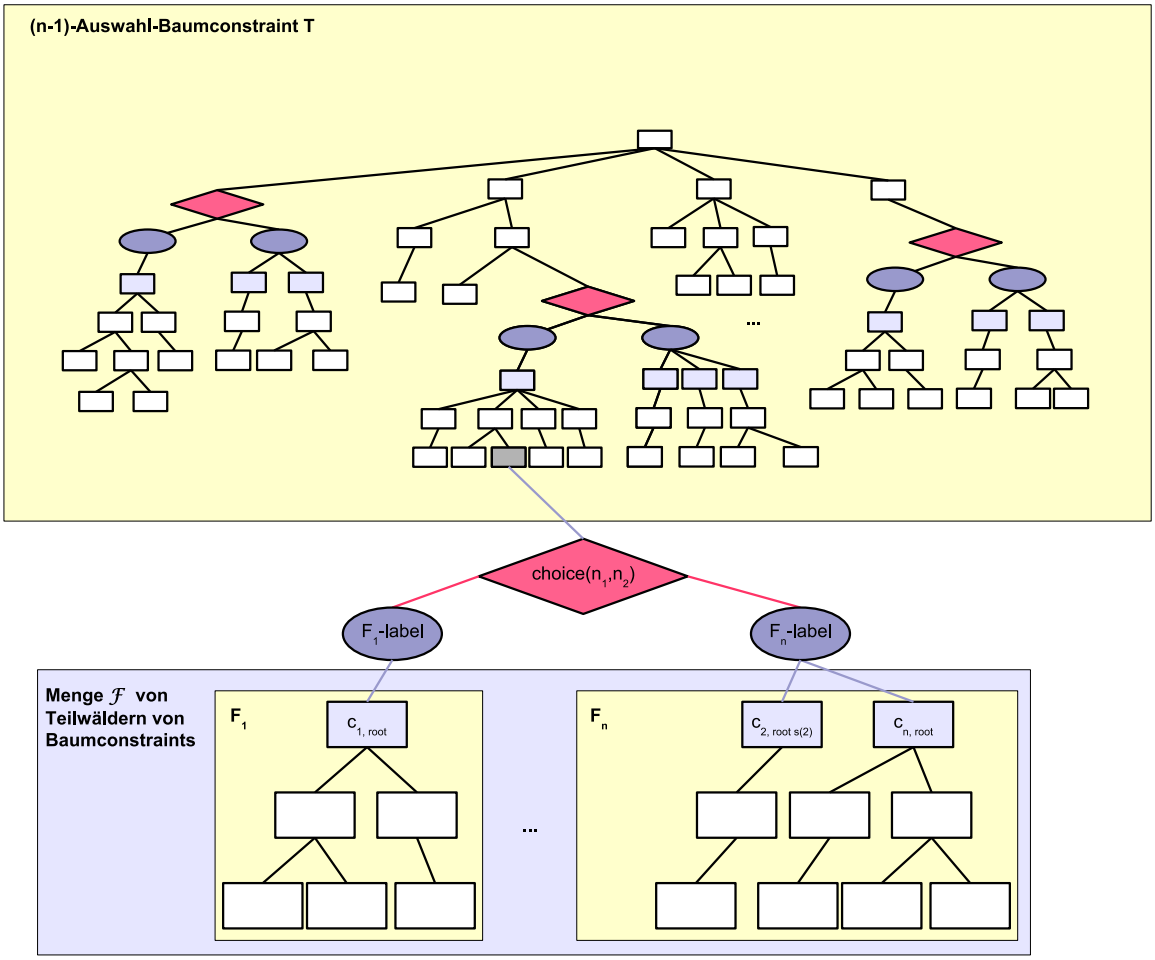



Abbildung 5.46: $n$-Auswahl-Baumconstraint

**Bemerkung 5.3.49 (Beobachtungen)** Zu den Beobachtungen im Rahmen der Sequenzconstraints (Bemerkung 5.3.32) sei Folgendes ergänzt:

1. Mit Hilfe der im Constraintmodell definierten elementaren und komplexen Constraints, insbesondere der Sequenz-, Baum- und Auswahl-Baumconstraints, ist eine sehr kompakte implementierungsunabhängige Beschreibungsmöglichkeit gegeben, mit deren Hilfe man komplexe Bedingungen, welche eine Verzahnung verschiedenartiger Aspekte struktureller Gegebenheiten einschließen, ausdrücken kann.

[40] Hinweis: Offensichtlich ist die Konkatenation eines $n$-Auswahl-Baumconstraints mit einer Menge von Teilwäldern von Baumconstraints an einen Knoten ein $n$-Auswahl-Baumconstraint.

2. Die Sequenz-, Baum- und Auswahl-Baumconstraints stellen eine logische Formulierung und Formalisierung der Pfadanfragen dar, welche in Anfragesprachen für baum- oder netzartige Datenmodelle wie etwa XML+XPath auftreten. Vor allem die Kontextbezogenheit der Einzelschritte der Anfragen ist zu beachten (vgl. auch Abschnitte 4.3 und 6.2).

3. Im Vergleich mit bekannten pfadbasierten Ansätzen (vgl. Abschnitte 4.3 und 6.2) zeigt sich die Ausdrucksmächtigkeit der definierten Constraints, insbesondere der Baum- und Auswahl-Baumconstraints. Das hier definierte Constraintmodell verallgemeinert wesentliche Aspekte der bekannten Pfadanfragesprache XPath.

Weitere Diskussionen hierzu finden sich in den Abschnitten 7.4.1 und 7.6.

**Bemerkung 5.3.50 (Namensähnliche Konzepte in der Literatur)** In der Literatur sind zwar zu Baumconstraints namensähnliche Konzepte zu finden, diese besitzen jedoch eine andere Semantik. So repräsentiert beispielsweise im Constraintprogramming [RvBW06] der Constraintgraph die zwischen Variablen bestehenden Constraints, notiert in der Gestalt eines Graphen. Dieser spielt im Rahmen der sogenannten „Constraintpropagation“ und der Optimierung eine wichtige Rolle. Im Zusammenhang mit strukturierten und semistrukturierten Daten stellen „path constraints“ Integritätsbedingungen dar, welche mit der Semantik der Daten assoziiert sind (vgl. [AV97, BFW00]). Auch weitere ähnliche Begriffe spiegeln jeweils eine andere Problematik bzw. Semantik wider.

Offenbar existiert also kein vergleichbarer Ansatz zur Formulierung von Constraints auf Objektbäumen, der mit dem im Rahmen dieser Arbeit entwickelten Constraintbegriff vergleichbar ist.

### 5.3.3 Lösungsmodell

Die Studienordnungen bestimmen die Möglichkeiten für ein gültiges Studium. Sie enthalten eine Menge von elementaren und komplexen Constraints, die Constraintspezifikation, und bestehen aus einer Menge von Strukturelementen bzw. aus einem Modulkatalog-Objektgraphen. Es ist noch zu klären, was genau unter einem „gültigen Studium“ zu verstehen ist. Grob gesprochen kann es als Teilmenge der Menge der wählbaren Elemente aufgefasst werden, die die Studienordnung bzw. deren Constraintspezifikation erfüllt. Einem gültigen Studium kann vermöge der Graphstruktur des Modulkatalog-Objektdiagramms eine Baumstruktur zugeordnet werden. Abbildung 5.47 zeigt die Grobstruktur der Zusammenhänge.

**Definition 5.3.51 (Gültiges Studium)** Sei $mc$ ein Modulkatalog, $\mathcal{S}$ die zugehörige Menge der wählbaren Elemente und $G = (V, E)$ mit $V = \mathcal{S}$, $E = contain$ der $mc$-Objektgraph und ferner $spec$ eine Constraint-Spezifikation, also eine Menge von elementaren und komplexen Constraints.

Ein *gültiges Studium* kann sowohl als Teilmenge von $\mathcal{S}$ als auch als Teilgraph von $G$ aufgefasst werden.

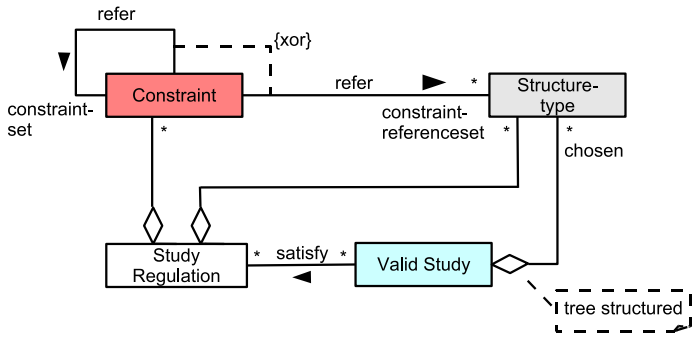



Abbildung 5.47: Lösungsmodell

Es stellt $Stud = (M_{Stud}, E_{Stud}) \subseteq G$ bzw. $M_{Stud} \subseteq \mathcal{S}$ genau dann ein *gültiges Studium* bezüglich der Gegebenheiten dar, falls folgende Bedingungen erfüllt sind:

1. $Stud$ ist ein *Teilbaum* von $G$ mit folgenden zusätzlichen Eigenschaften:
   - $Stud$ enthält die Wurzel von $G$
   - Jeder Blattknoten von $Stud$ ist auch ein Blattknoten von $G$
2. $Stud \models s$ bzw. $M_{Stud} \models s$ in folgendem Sinn: Für alle Constraints $c$ der Spezifikation $spec$, $c \in spec$ gilt: $M_{Stud} \models c$
3. Ist außerdem eine $require$-Assoziation gegeben, so ist ferner zu fordern: Für jedes Tupel $(e_1, e_2) \in require$ gilt:

   Aus $e_1 \in M_{Stud}$ folgt: $e_2 \in M_{Stud}$,

   das zu einem Element aus $M_{Stud}$ vorausgesetzte Element muss also auch im Modell enthalten sein.

Wie in Bemerkung 5.3.26 schon erwähnt, ist es nicht sinnvoll, einen inneren Knoten des Strukturgraphen zu wählen, ohne mindestens einen Nachfolger ebenso zu wählen. Daher dürfen keine inneren Knoten von $G$ als Blattknoten des „Modellgraphen“ $Stud$ auftreten.

Wichtig in der Definition ist ferner, dass $Stud$ ein *Baum* ist, also ein Knoten nicht mehrere Elternknoten besitzen kann. Dies verhindert die „mehrfache Wahl“ von Elementknoten (vgl. auch Problematik „eine Rolle – mehrere Rollen“ im Anwendungsbereich „Studienberatung und Konsistenzprüfung von Studienordnungen“, Abschnitt 5.2.2.3). Sind beispielsweise im Graphen aus Abbildung 5.48 zwei Fächergruppen und je Fächergruppe zwei Fächer zu wählen, so würde der markierte Teilgraph kein gültiges Studium darstellen, da $s_c$ zweifach gewählt wird. Durch die Forderung der Baumeigenschaft der Lösung wird dies verhindert.

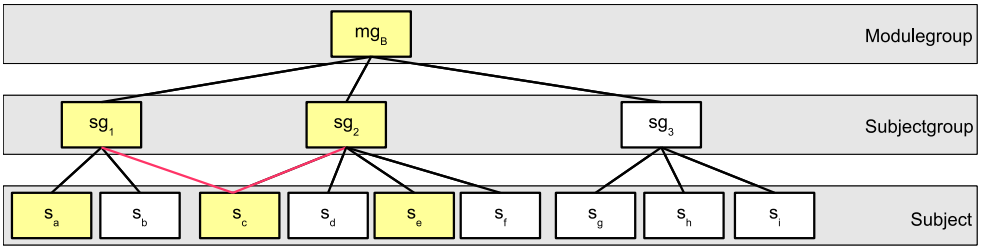



Abbildung 5.48: Ungültiges Studium

### 5.3.4 Erweiterungen

#### 5.3.4.1 Zeitliche Aspekte und Studienverlauf

Modularisierte Studienordnungen lassen eine mehr oder weniger große Freiheit in der Wahl von Veranstaltungen. In der Regel wird auch kein Studienplan verpflichtend vorgegeben. Um den Studierenden eine Orientierungshilfe zu geben, werden Empfehlungen (*Recommendation*) ausgesprochen, bis zu welchem Semester ein Strukturelement spätestens belegt oder bestanden werden sollte. Gelegentlich wird auch eine entsprechende untere Grenze angegeben. Die Regelstudienzeit ist ebenso eine derartige Empfehlung. Damit die Studierenden ihr Studium in dieser absolvieren können, sollten sie sich an sämtliche Empfehlungen halten.

Wie im Lösungsmodell schon besprochen, besteht ein gültiges Studium aus einer endlichen Menge von Strukturelementen, welche dann bezüglich dieses Studiums als „gewählt“ bezeichnet werden.

Zu Beginn seines Studiums darf ein Student nur solche Veranstaltungen bzw. allgemein Strukturelemente belegen, die keinerlei Voraussetzungen besitzen. Diese sind dann für ihn erlaubt (Assoziation *allowed*). Durch das Absolvieren von Strukturelementen (Assoziation *passed*) erfüllt der Student gewisse Voraussetzungen und erwirbt damit die Erlaubnis für das Belegen weiterer Strukturelemente. Die schon bestandenen Elemente sind dann nicht mehr erlaubt, da sie nicht nochmals belegt werden dürfen.

Ein Strukturelement kann ein anderes Strukturelement voraussetzen (Assoziation *require*) oder sie erfordert eine bestimmte Anzahl an „bestandenen ECTS-Punkten“ (Assoziation *requireEcts*).

Die Veranstaltungen des Modulkatalogs (*Coursetemplate*) werden oftmals nur in einem gewissen Turnus (Wintersemester, Sommersemester oder jedes Semester, *Cycle*) angeboten. Für die Planung des individuellen Studienverlaufs spielt daher neben dem aktuellen Semester (Attribut `CurrentSemester`) auch der Studienbeginn (Attribut `StartCycle`) eine Rolle. Abbildung 5.49 skizziert die besprochenen Sachverhalte.

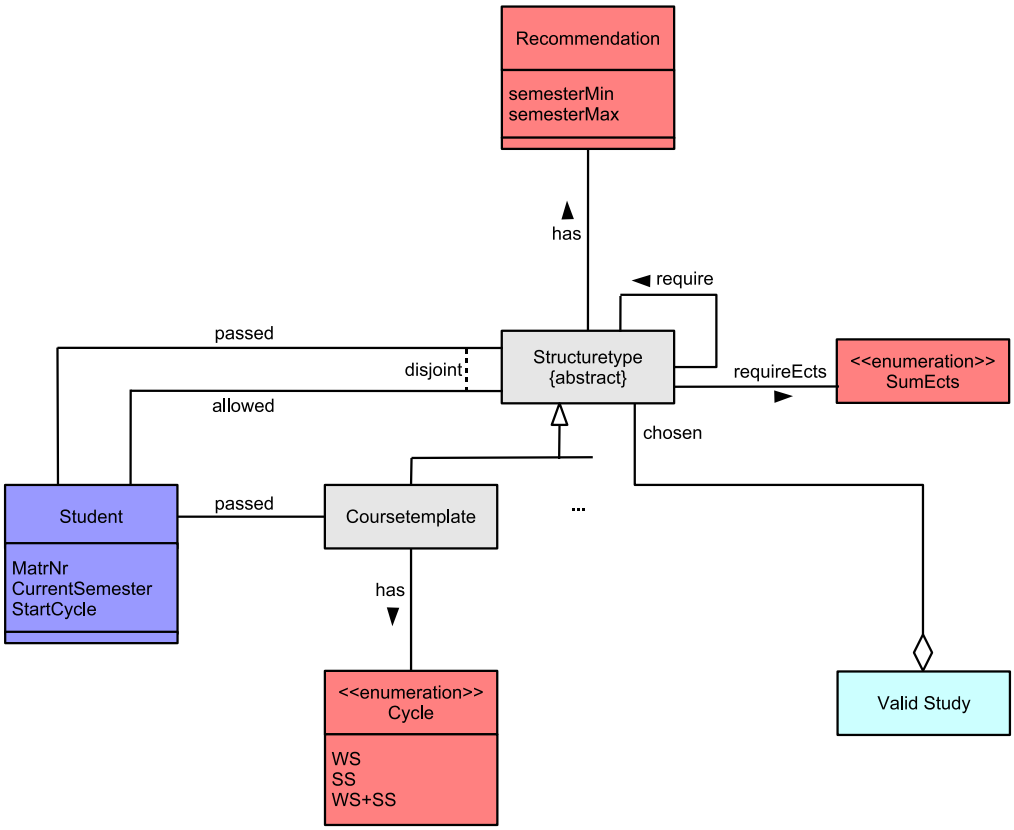



Abbildung 5.49: Erweiterungen im Modell

**Erweiterungen im Strukturmodell** Folgende Ergänzungen zum Strukturmodell aus Abschnitt 5.3.1 können vorgenommen werden:

**Definition 5.3.52 (Assoziation $recommendation$)** Die *Assoziation* $recommendation$ modelliert die zeitlichen Empfehlungen für die Strukturelemente. Sie geben an, zwischen welchen Semestern diese absolviert werden sollten:

$$recommendation \subseteq Structuretype^{ob} \times \mathbb{N}_0 \times \mathbb{N}$$

Ein Tupel $(s, n_1, n_2) \in recommendation$ drückt aus, dass das Strukturelement $s$ frühestens im $n_1$-ten und spätestens im $n_2$-ten Semester absolviert werden soll.

Vermöge der Identifikation eines Strukturelements mit seiner ID (Wert des 0-ten Attributs des entsprechenden Strukturtyps) erhält man folgende induzierte Zuordnung:

$$recommendation \subseteq \left( \bigcup_{i=1}^{m} range(A_{i,0}) \right) \times \mathbb{N}_0 \times \mathbb{N}$$

Falls laut der Studienordnung die Veranstaltungstemplates nicht jedes Semester angeboten werden, so ist eine weitere Assoziation zur Modellierung des Turnus notwendig:

**Definition 5.3.53 (Assoziation $ctCycle$)** Die *Assoziation* $ctCycle$ modelliert den Turnus der Veranstaltungstemplates:

$$ctCycle \subseteq Coursetemplate \times \{\textit{WS}, \textit{SS}, \textit{WS+SS}\}$$

Hierbei bedeutet beispielsweise $ctCycle(c, \textit{WS})$, dass das Veranstaltungstemplate $c$ nur im *WS* angeboten wird. Die Identifikation von $Coursetemplate$ mit $range(A_{m,0})$ liefert:

$$ctCycle \subseteq range(A_{m,0}) \times \{\textit{WS}, \textit{SS}, \textit{WS+SS}\}$$

Außerdem sei noch erwähnt, dass auch eine gewisse Anzahl an „ECTS-Punkten“ als Voraussetzung angesetzt sein kann:

**Definition 5.3.54 (Assoziation $requireEcts$)** Die *Assoziation* $requireEcts$ modelliert Voraussetzungen von ECTS-Punkten:[41]

$$requireEcts \subseteq Structuretype^{ob} \times \mathbb{N}$$

Ist $s$ ein Strukturelement und $n \in \mathbb{N}$, so modelliert das Tupel $(s, n) \in requireEcts$, dass die Summe der ECTS-Punkte der bestandenen Veranstaltungstemplates mindestens $n$ betragen muss, um das Strukturelement $s$ belegen zu können.[42]

Mit der üblichen Identifikation eines Strukturelements mit seinem Identifikator, dem Attributwert des 0-ten Attributs, erhält man:

$$requireEcts \subseteq \left( \bigcup_{i=1}^{m} range(A_{i,0}) \right) \times \mathbb{N}$$

Diese Art der Voraussetzungen sollen allerdings in den nachfolgenden Ausführungen keine Berücksichtigung finden.

**Modellierung der Studierenden** Im Rahmen einer möglichen Studienberatung sind als relevante Daten das aktuelle Semester, der Startturnus und die bestandenen Veranstaltungen zu nennen. Weitere persönliche Daten wie der Name des Studierenden spielen hierfür keine Rolle. Sie sollen daher auch nicht in Betracht gezogen werden.

**Definition 5.3.55 (Relation $student$)** Die *Relation* $student$ modelliert die Grunddaten eines Studierenden bezüglich seines Studienverlaufs:

$$student \subseteq range(\texttt{MatrNr}) \times range(\texttt{CurrentSemester}) \times range(\texttt{StartCycle}),$$

wobei das Attribut `MatrNr` die den Studierenden identifizierende Matrikelnummer darstellt. `CurrentSemester` modelliert das aktuelle Semester und `StartCycle` den Startturnus des Studierenden.

**Beispiel 5.3.56 (Relation $student$)** Ist ein Student mit der Matrikelnummer *12345* im *7.* Semester und hat in einem Sommersemester sein Studium begonnen, so kann dies durch

$$student(12345, 7, SS)$$

modelliert werden.

[41] Die vorausgesetzten ECTS-Punkte sind in einem gewissen vorgegebenen endlichen Bereich, z. B. bis 180 ECTS-Punkte. Zur Betonung dieses Aspekts kann anstelle $\mathbb{N}$ auch $SumECTS$ notiert werden.

[42] Ist $s \notin Coursetemplate$, d. h. ist $s$ ein innerer Knoten im Modulkatalog-Objektdiagramm, so überträgt sich die vorausgesetzte ECTS-Punktezahl an die zu bestehenden Nachfolger von $s$.

**Definition 5.3.57 (Assoziation $passedCt$)** Die *Assoziation* $passedCt$ modelliert die bestandenen Veranstaltungstemplates eines Studierenden:

$$passedCt \subseteq student \times Coursetemplate$$

Vermöge der Identifikation eines Studierenden mit seiner Matrikelnummer und der eines Veranstaltungstemplates mit seinem Identifikator (Attributwert zum 0-ten Attribut) erhält man:

$$passedCt \subseteq range(\texttt{MatrNr}) \times range(A_{m,0}).$$

Hierbei muss gefordert werden, dass $passedCt$ mit der $require$-Assoziation kompatibel ist. Dies bedeutet, dass für jedes Element $e \in passedCt$ auch die explizit oder implizit vorausgesetzten Veranstaltungstemplates bestanden und damit in der Relation $passedCt$ repräsentiert sein müssen. Gegebenenfalls ist auch die $requireEcts$-Assoziation zu berücksichtigen.

**Beispiel 5.3.58 (Bestandene Veranstaltungstemplates)** Abbildung 5.50 zeigt einen einfachen Modulkatalog-Objektgraphen einer Studienordnung. Das Veranstaltungstemplate $c$ soll schon bestanden sein. Damit die Relation $passedCt$ mit der Assoziation $require$ kompatibel ist, muss auch das Veranstaltungstemplate $c_r$ bestanden sein, $c_r \in passedCt$. Da aber $c$ auch das Modul $m_r$ voraussetzt, muss die geforderte Anzahl – hier zwei – von Veranstaltungstemplates aus $m_r$ bestanden sein. Dies wäre beispielsweise durch $c_1, c_2 \in passedCt$ erfüllt. In einem erweiterten Sinne wäre damit auch $m_r$ bestanden.

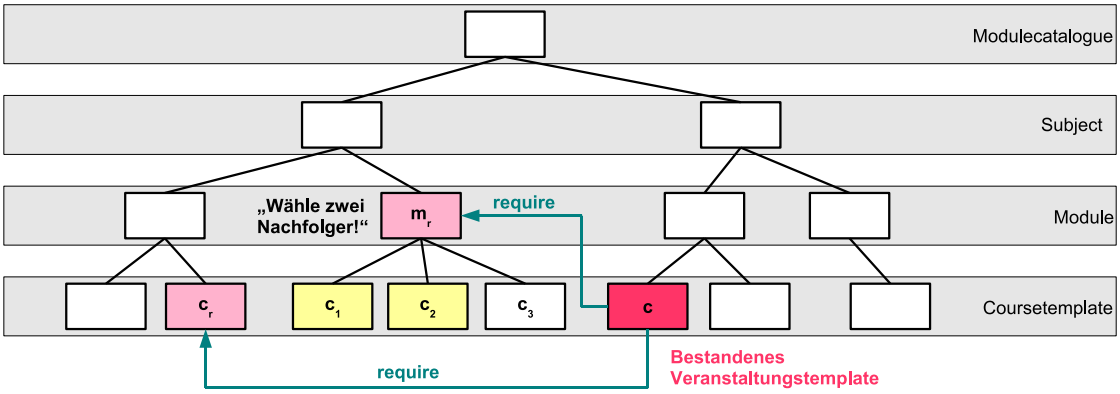



Abbildung 5.50: Bestandene Veranstaltungstemplates

Ist die erforderliche Anzahl an Kindelementen eines Strukturelements $e$ bestanden, so soll auch $e$ als bestanden gelten. Mit dieser erweiterten Auffassung können ausgehend von den bestandenen Veranstaltungstemplates sukzessive sämtliche bestandenen Strukturelemente bestimmt werden. Diesem Sachverhalt wird in der Relation $passed$ Rechnung getragen. Zu beachten ist die Bedeutung der Bedingung, dass die „erforderliche Anzahl“ an Kindelementen bestanden sein muss.

**Definition 5.3.59 (Assoziation $passed$)** Sei $G = (V, E)$ der Modulkatalog-Objektgraph der betrachteten Studienordnung. Ist $M \subset V$ ein gültiges Studium, das die bestandenen Veranstaltungstemplates umfasst, $M \supseteq passedCt$, so modelliert die *Assoziation* $passed_M$ den Studienstand eines Studierenden bezüglich $M$ im Hinblick auf bestandene Strukturelemente:[43]

$$passed_M \subseteq student \times Structuretype^{ob}$$

bzw. vermöge der Identifikation der Studierenden mit ihrer Matrikelnummer bzw. der Strukturelemente mit ihrer ID

$$passed_M \subseteq range(\texttt{MatrNr}) \times \left( \bigcup_{i=1}^{m} range(A_{i,0}) \right),$$

wobei wie folgt induktiv definiert wird:

1. Die bestandenen Veranstaltungstemplates sind bestandene Strukturelemente bezüglich $M$:

   $$passedCt \subseteq passed_M$$

2. Ein Strukturelement $e \in Structuretype^{ob} \setminus Coursetemplate$ ist bestanden bezüglich $M$, wenn

   (a) alle Nachfahren von $e$ aus $M$ bestanden sind bezüglich $M$ und

   (b) alle vorausgesetzten Elemente von $e$ bestanden sind bezüglich $M$.

Ist das konkrete Modell $M$ nicht von Bedeutung, so kann statt $passed_M$ kann auch einfach nur $passed$ notiert werden. Ein Strukturelement ist ein Element von $passed$, wenn es ein Modell $M$ gibt, so dass es ein Element von $passed_M$ ist.

**Beispiel 5.3.60 (Bestandene Strukturelemente)** Abbildung 5.51 zeigt einen einfachen Modulkatalog-Objektgraphen einer Studienordnung. Die Veranstaltungstemplates $c_1, c_2, c_4$ und $c_6$ seien schon bestanden, also $passedCt = \{c_1, c_2, c_4, c_6\}$. Wie man leicht sieht, ist $passedCt$ mit der gegebenen $require$-Assoziation verträglich.

Ein gültiges Studium wird durch das Modell $M$ repräsentiert. Nun werden sukzessive sämtliche bestandenen Strukturelemente, also die Elemente von $passed_M$, gesucht. Das Modul $m_1$ ist bestanden, weil der einzige in $M$ liegende Kindknoten $c_1$ schon bestanden und $m_1$ keine weiteren Strukturelemente voraussetzt. Ähnliches gilt für das Modul $m_2$. Da die Module $m_1$ und $m_2$ als einzige Kindknoten des Faches $s_1$ bestanden sind und $s_1$ keine weiteren Voraussetzungen hat, ist auch $s_1$ bestanden. Hingegen ist das Modul $m_3$ noch nicht bestanden, weil das Veranstaltungstemplate $c_7$ noch nicht bestanden ist. Damit ist natürlich auch das Fach $s_2$ und der Modulkatalog $mc$ noch nicht bestanden.

[43] Auch hier müsste gegebenenfalls die Assoziation $requireEcts$ berücksichtigt werden.

Ein ähnliche Situation ist in Abbildung 5.52 gegeben. Zu beachten ist hierbei, dass statt $c_4$ das Veranstaltungstemplate $c_5$ im betrachteten Modell $M$ enthalten ist. Obwohl im Modul $m_2$ nun zwei Veranstaltungstemplates bestanden sind, ist das Modul $m_2$ *bezüglich* $M$ noch nicht bestanden. Hierzu fehlt $c_5$ unter den bestandenen Veranstaltungstemplates.

In Abbildung 5.53 liegt eine weitere ähnliche, leicht modifizierte Situation vor. Zu beachten ist hierbei, dass das Modul $m_2$ im Sinne von Definition 5.3.59 noch nicht bestanden ist, obwohl alle in $M$ liegenden Kindknoten von $m_2$ bestanden sind. Dies liegt daran, dass das vorausgesetzte Strukturelement, nämlich $m_1$, noch nicht bestanden ist. Die in Abbildung 5.53 skizzierte Situation sollte eigentlich gar nicht auftreten: Das Modul $m_2$ setzt das Modul $m_1$ voraus und trotzdem sind im Modul $m_2$ schon Veranstaltungstemplates bestanden. Ein Studierender sollte die Veranstaltungstemplates von $m_2$ erst dann belegen können, wenn er das Modul $m_1$ schon bestanden hat.

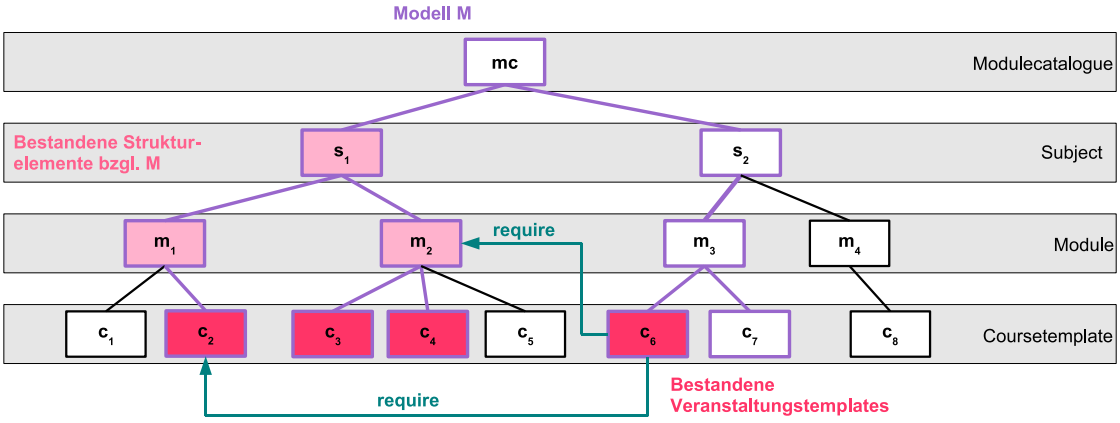



Abbildung 5.51: Beispiel 1a: Bestandene Strukturelemente bezüglich eines Modells

**Bemerkung 5.3.61 (Bestandenes Studium)** Ist $e_{mc}$ der Wurzelknoten des Modulkatalog-Objektgraphen, so hat der Studierende das Studium bestanden, wenn $e_{mc}$ bestanden ist, also $e_{mc} \in passed$ ein Element der Relation $passed$ ist.

Dadurch, dass manche Strukturelemente Voraussetzungen besitzen, können sie für einen Studierenden abhängig von seinem Studienstand noch nicht erlaubt sein, so dass er diese noch nicht besuchen und belegen darf. Die anderen Strukturelemente sind hingegen erlaubt.

Vor der Einführung der entsprechenden Assoziation $allowed$ ist noch ein genauerer Blick auf die Strukturelemente zu werfen, die Voraussetzungen, direkte oder „geerbte“, besitzen.

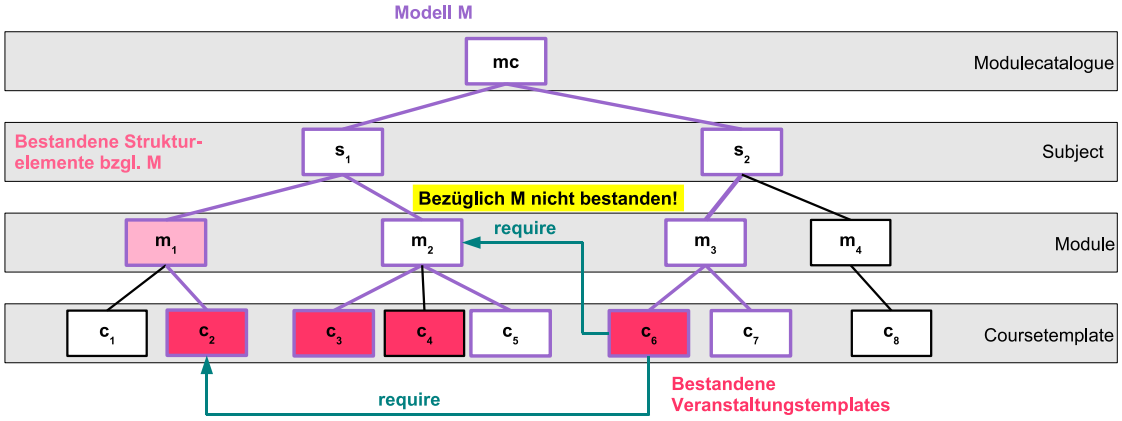



Abbildung 5.52: Beispiel 1b: Bestandene Strukturelemente bezüglich eines Modells

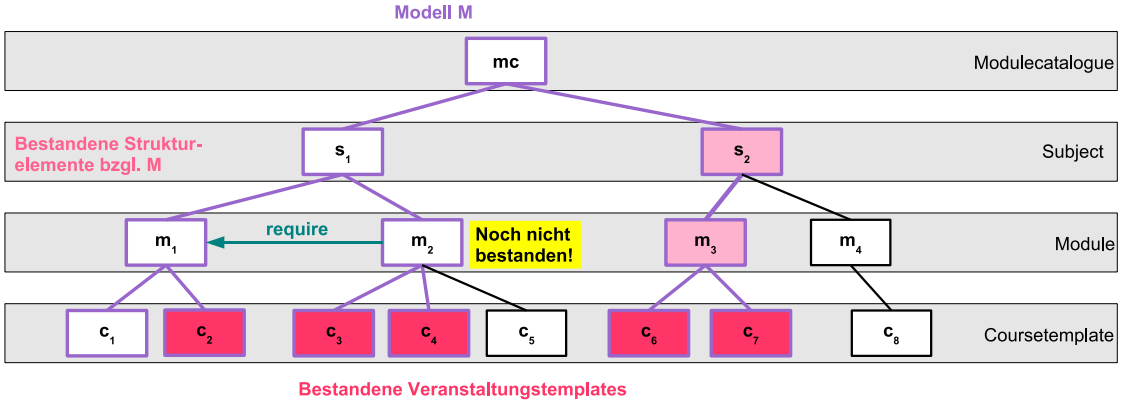



Abbildung 5.53: Beispiel 2: Bestandene Strukturelemente bezüglich eines Modells

**Definition 5.3.62 („Geerbte“ Voraussetzungen)** Die Relation $inheritedRequire$ modelliert die direkten und geerbten Voraussetzungen von Strukturelementen und ist eine Erweiterung der Assoziation $require$ (Definition 5.3.11):

$$inheritedRequire \subseteq Structuretype^{ob} \times Structuretype^{ob}$$

bzw. in der induzierten Form:

$$inheritedRequire \subseteq \left( \bigcup_{i=1}^{m} range(A_{S_i,0}) \right) \times \left( \bigcup_{i=1}^{m} range(A_{S_i,0}) \right)$$

wobei wie folgt definiert wird: Ein Strukturelement $e$ setzt das Strukturelement $r$ voraus, $inheritedRequire(e, r)$, wenn entweder

1. das Element $e$ das Element $r$ direkt voraussetzt: $require(e, r)$ oder
2. $e$ ein Vorfahre eines Elements $e_d$ ist, welches $r$ als Voraussetzung hat: $e \in anc(e_d)$ und $require(e_d, r)$

Darauf aufbauend modelliert die Relation $hasRequirement$ die Strukturelemente, welche eine Voraussetzung besitzen:

$$hasRequirement \subseteq Structuretype^{ob}$$

bzw. in der induzierten Form:

$$hasRequirement \subseteq \bigcup_{i=1}^{m} range(A_{S_i,0})$$

Hierbei gilt: Ein Strukturelement $e$ hat eine Voraussetzung, $e \in hasRequirement$, genau dann, wenn es ein Strukturelement $r$ gibt, das es voraussetzt: $inheritedRequire(e, r)$. Die Relation $hasRequirement$ ist also die Projektion auf die 1. Komponente von $inheritedRequire$.

**Beispiel 5.3.63 („Geerbte" Voraussetzungen)** Abbildung 5.54 zeigt einen Modulkatalog-Objektgraphen. Hierbei soll gelten, dass das Veranstaltungstemplate $c_3$ das Modul $m_1$ und außerdem das Module $m_3$ das Modul $m_2$ (direkt) voraussetzt. Die Kindknoten von $m_3$ haben daher auch $m_2$ zur Voraussetzung. In diesem Fall sind also $c_3$, $m_3$, $c_6$ und $c_7$ die Strukturelemente, die eine Voraussetzung haben.

**Definition 5.3.64 (Assoziation $allowed$)** Die *Assoziation* $allowed$ modelliert die Strukturelemente, die für den Studierenden bezüglich seines Studienstandes (bestandene Strukturelemente) und den Voraussetzungsbeziehungen ($require$, $inheritedRequire$) erlaubt sind, d. h. die er prinzipiell „besuchen" oder „belegen" kann – ohne Berücksichtigung weiterer Anforderungen in der Studienordnung:

$$allowed \subseteq student \times Structuretype^{ob}$$

bzw. die induzierte Relation:

$$allowed \subseteq range(\texttt{MatrNr}) \times \left( \bigcup_{i=1}^{m} range(A_{i,0}) \right)$$

Hierbei gilt: Ein Strukturelement ist erlaubt, $e \in allowed$, wenn Folgendes zutrifft:

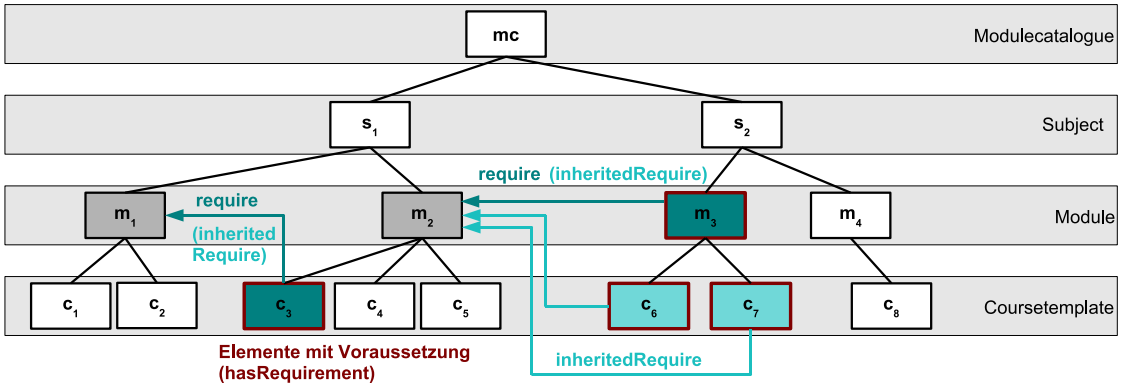



Abbildung 5.54: Voraussetzungen

1. Das Strukturelement $e$ ist noch nicht bestanden: $e \notin passed$

2. Das Element $e$ hat entweder keine Voraussetzungen, weder direkte noch „geerbte“, oder alle vorausgesetzten Strukturelemente sind schon bestanden:

   - $e \notin hasRequirement$ oder
   - für alle Strukturelemente $r$ mit $inheritedRequire(e, r)$ gilt auch: $r \in passed$.

**Beispiel 5.3.65 (Erlaubte Strukturelemente)** Ausgehend von dem schon bekannten Modulkatalog-Objektgraphen aus Abbildung 5.54 soll zunächst die Situation betrachtet werden, dass noch keine Strukturelemente bestanden sind. Abbildung 5.55 zeigt die „erlaubten Strukturelemente“, also diejenigen, die keine Voraussetzung besitzen.

Besteht nun der Studierende die Veranstaltungstemplates $c_2$ und $c_4$, so sind diese nicht mehr erlaubt. Ist mit $c_2$ auch das Modul $m_1$ bestanden, so wird das Veranstaltungstemplate $c_3$ zu einem erlaubten Strukturelement (vgl. Abbildung 5.56).

Nach dem Bestehen von $c_3$ soll auch das Modul $m_2$ bestanden sein, wodurch auch das Fach $s_1$ als bestanden gilt. Da nun das vorausgesetzte Strukturelement (Modul $m_2$) des Moduls $m_3$ und der Veranstaltungstemplates $c_6$ und $c_7$ bestanden ist, sind schließlich auch letztere Elemente erlaubt (vgl. Abbildung 5.57).

Durch weiteres Bestehen der Elemente $c_6$ und $c_7$ könnte $m_3$ ebenso bestanden und daher letztendlich $s_2$ und $mc$, also das ganze Studium, bestanden sein (vgl. Abbildung 5.58). Es gibt im Sinne der Definition 5.3.64 zwar noch erlaubte Strukturelemente, diese sind aber nicht mehr zu belegen, da ja das Studium schon bestanden ist.

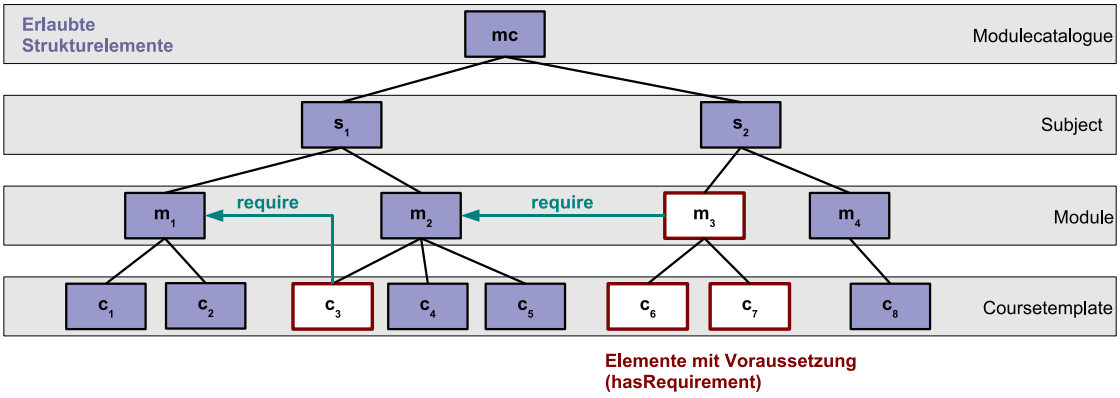



Abbildung 5.55: Beispiel a: Erlaubte Strukturelemente

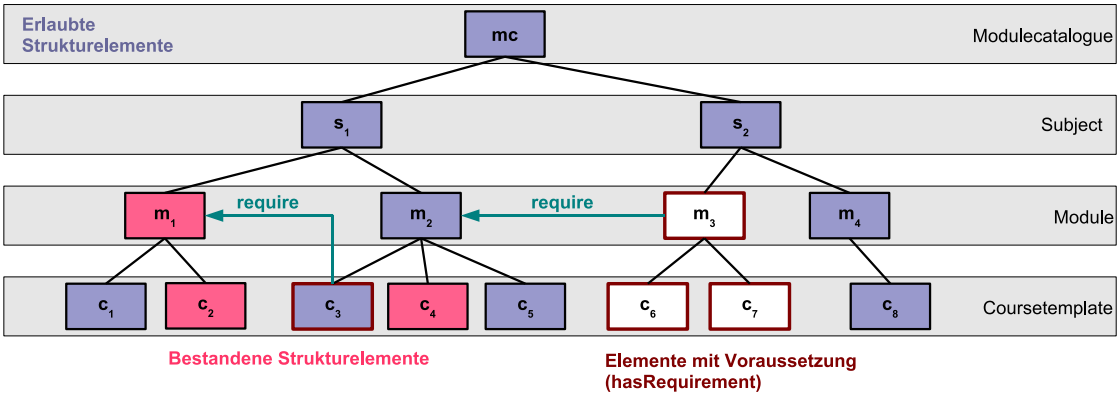



Abbildung 5.56: Beispiel b: Erlaubte Strukturelemente

**Zeitliche Empfehlungen** Die Assoziation *allowed* gibt dem Studierenden an, welche Strukturelemente er unter Berücksichtigung der bestandenen Veranstaltungen und der Voraussetzungen zwischen Strukturelementen belegen kann. Der zeitliche Rahmen für das Belegen von Strukturelementen wird durch zeitliche Empfehlungen ($recommendation$) vorgegeben. Möchte ein Studierender in der Regelstudienzeit sein Studium absolvieren, so sollte er sich an derartige Empfehlungen halten.

Im Folgenden wird kurz umrissen, welche Aspekte hier zusammenspielen.

1. Für die Veranstaltungen $e_{ct} \in Coursetemplate$ geben die Tupel $(e_{ct}, \min, \max) \in recommendation$ einen ersten zeitlichen Rahmen für das Belegen vor.

2. Die zeitlichen Empfehlungen für die inneren Knoten $e \in Structuretype^{ob} \setminus Coursetemplate$

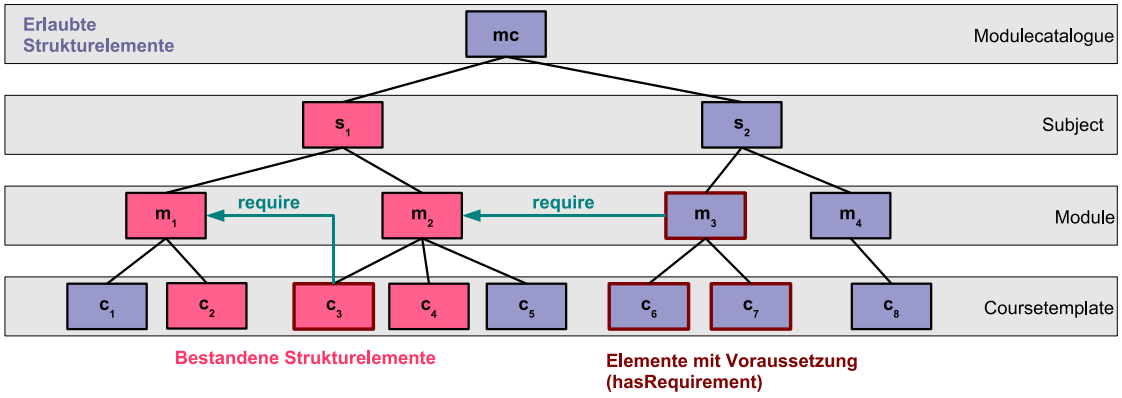



Abbildung 5.57: Beispiel c: Erlaubte Strukturelemente

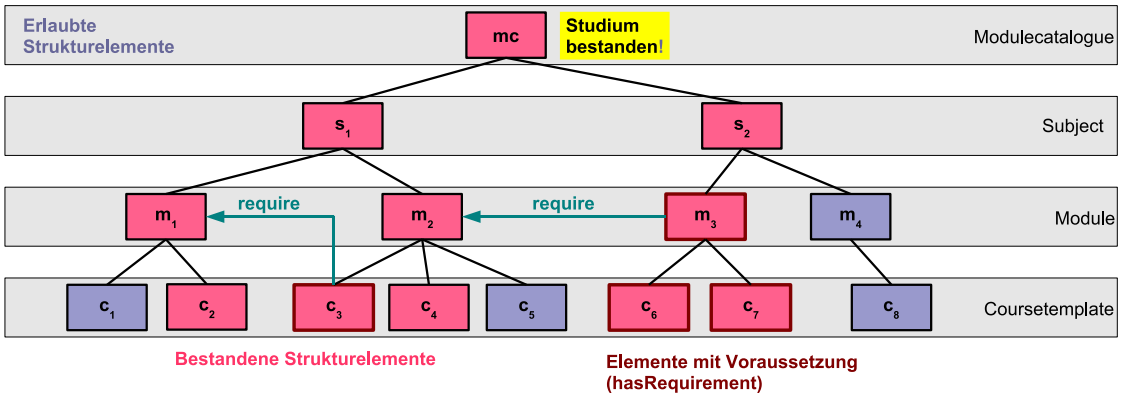



Abbildung 5.58: Beispiel d: Erlaubte Strukturelemente

des Modulkatalog-Objektgraphen geben einen zeitlichen Rahmen für das Belegen dieser Elemente vor. Diese sind auch auf die Nachfolger zu übertragen.

3. Darüber hinaus ist der Turnus ($ctCycle$) der Veranstaltungen zu berücksichtigen. Obere und untere Grenzen für Empfehlungen sind gegebenenfalls anzupassen.

4. Schließlich sind auch noch Voraussetzungen ($require$, $inheritedRequire$) einzuberechnen. Die Empfehlungen für die vorausgesetzten Strukturelemente sind im Einklang mit den Empfehlungen des Strukturelements zu bringen, das Voraussetzungen hat.

Im Hinblick auf den Studierenden und die tatsächlichen Empfehlungen, welche Veranstaltungen im nächsten Semester gehört werden sollen, sind neben den obigen Betrachtungen auch das aktuelle Se-

mester (`CurrentSemester`) des Studierenden und sein Studienbeginn (`StartCycle`) relevant. Entsprechende Berechnungen können systematisch durchgeführt werden, sollen hier aber nicht gezeigt werden.

#### 5.3.4.2 Konkrete Lehrveranstaltungen

Die in den Studienordnungen auftretenden Veranstaltungen (Veranstaltungstemplates) sind Templates für konkrete Veranstaltungen, welche an einer bestimmten Hochschule in einem bestimmten Semester angeboten werden. Um eine umfangreiche Beratung unter Berücksichtigung des konkreten Studienangebots gewährleisten zu können und um Fragen des Studiendekans, welche Veranstaltungen denn nun angeboten werden müssen, damit der Studiengang studierbar ist, beantworten zu können, ist eine Modellierung der konkreten Veranstaltungen unabdingbar.

Die Modellierung der Studienordnung bzw. des Modulkatalogs in Abschnitt 5.3 des Kapitels 5 ist jedoch so variabel und offen gehalten, dass eine Hinzunahme einer zusätzlichen Ebene nicht wirklich eine echte Erweiterung darstellt:

Die konkreten Veranstaltungen können durch einen zusätzlichen Strukturtypen ( ConcreteCourse ) modelliert werden. Bezüglich einer Lösung (gültiges Studium) ist die Anforderung zu realisieren, dass pro gewähltem Veranstaltungstemplate $c \in Coursetemplate$ genau eine konkrete Veranstaltung $cc \in ConcreteCourse$ zu wählen ist (bedingtes Constraint verknüpft mit local-Successorconstraint). Natürlich müssen die geforderten Veranstaltungen außerdem auch tatsächlich konkret angeboten werden.

In diesem Zusammenhang kann auch das Thema der Anrechenbarkeit behandelt werden. Sollen externe Leistungen eines Studierenden für ein bestimmtes Veranstaltungstemplate anerkannt werden, so kann die externe Leistung im Prinzip wie eine konkrete Veranstaltung behandelt werden, indem eine Verbindung der anrechenbaren Veranstaltung zum entsprechenden Veranstaltungstemplate modelliert wird.

# Kapitel 6

# Transformation

Ausgehend von der Modellierung in Kapitel 5 erfolgt die Beschreibung der Transformation in interne Formalismen. Eine ausführliche Darstellung wird für den gewählten logikbasierten Ansatz mit Constraints, genauer für Answer Set Programming mit Gewichtsconstraints, vorgenommen. Als Alternative hierzu werden Überlegungen zur Transformation in einen pfadbasierten Ansatz, nämlich XML mit XPath/XQuery angestellt.

## 6.1 Logikbasierter Ansatz mit Constraints

Die methodischen Grundlagen hierzu sind im Abschnitt 4.2 dargelegt. Erste Überlegungen und Ansätze zur Transformation sind schon in einigen Beispielen des eben zitierten Abschnitts sowie in [Sch05, SF06] zu finden.

Zu bemerken ist ferner, dass die in nachfolgenden Ausführungen gezeigten Regeln im Wesentlichen der Syntax des ASP-Systems LPARSE + SMODELS entsprechen (vgl. Abschnitt 4.2.2.6). Bedeutsam ist hierbei außerdem, dass die Programme *domänenbeschränkt* sein müssen. Dies wird im Wesentlichen dadurch gewährleistet, dass jede Variable in einer Regel in einem positiven Rupfliteral vorkommen muss, das sich in einem echt niedrigeren Stratum als der Kopf der Regel – bezüglich der Statifizierung der Domänenregeln bzw. deren Prädikate – befindet (Abschnitt 4.2.2.4, [Syr01, SNS02, Syr]). Informell gesprochen sind Domänenprädikate die Prädikate, die nicht über eine negative Rekursion definiert sind. Üblicherweise ist man an den Nicht-Domänenprädikaten interessiert, wohingegen die Domänenprädikate „nur“ die positiven Variablenbindungen vorgeben. Der Teil des Programms, der die Domänenprädikate definiert, besitzt ein einziges (endliches) stabiles Modell, welches relativ effizient bestimmt wird. Das Front-end LPARSE übersetzt das Programm mit Variablen in die Kernsprache des Systems SMODELS, also in (grundierte) Basisconstraint-Regeln (Abschnitt 4.2.2.5) unter Berücksichtigung der Variablenbindungen der Domänenprädikat-Literale. Die hierbei erzeugte (endliche) Grundinstanziierung des Programms, in der für jede grundierte Regel die Instanzen der

Domänenprädikat-Literale erfüllt sind, besitzt dieselben stabilen Modelle (Antwortmengen) wie das ursprüngliche Programm mit Variablen (Abschnitt 4.2.2.4).

Die nachfolgend dargestellte Transformation ist allgemein gehalten, um sämtliche möglichen auftretenden Fälle zu berücksichtigen. Nicht alle Regeln werden in allen konkreten Anwendungsbereichen benötigt, was bei einer entsprechenden konkreten Transformation zu bedenken ist. Auch die Vorberechnung bestimmter Teile des Programms, die Tabellierung und die Verwendung zusätzlicher Wächterprädikate kann der Optimierung dienen und sollte gegebenenfalls auch angewandt werden. Auch wenn dies nicht an jeder möglichen Stelle der folgenden Ausführungen diskutiert wird, ist dies trotzdem zu bedenken.

### 6.1.1 Generierung von Tests

#### 6.1.1.1 Kernkomponenten

Das logische ASP-Programm umfasst folgende Bestandteile:

1. *Fragenpool (Fragen und Eigenschaften)* (realisierbar als extensionale Datenbank, EDB)
2. *Strukturelle Ableitungsregeln* (realisierbar als intensionale Datenbank, IDB)
3. *Allgemeine Regeln zur Modellierung des Auswählens von Fragen und teilweise der Realisierung der benutzerdefinierten Anforderungen* (realisierbar als IDB)
4. *Regeln zur Repräsentation der benutzerdefinierten Anforderungen und deren Realisierung* (realisierbar als EDB und teilweise als IDB)

Der SMODELS-basierte Systemkern implementiert folgende Kernfunktionalitäten:

- Bereitstellung der Daten zur Beschreibung der Fragen
- Auswahl der erforderlichen Anzahl von Fragen
- Auswahl und Bestimmung der direkten und hierarchischen Attributwerte gewählter Fragen
- Formalisierung und Speicherung der benutzerdefinierten Anforderungen
- Berechnung der Antwortmengen

Im Folgenden ist ein Fragenpool wie im Modell aus Abschnitt 5.1 und 5.2 gegeben.

#### 6.1.1.2 Strukturmodell: Fragenpool und Eigenschaften

Die Attributfolge $\mathcal{A}_{ord} = A_1, \ldots, A_n$ mit ihren $m$ flachen und $n - m$ hierarchischen Attributen und ihren Domänen und Wertebereichen wird nicht explizit übersetzt. Diesbezüglich notwendige Informationen stecken unter anderem in den transformierten $contain$-Hierarchien und der Relation $prop$ und können durch Regeln abgeleitet werden.

**Zuordnung der Fragen zu ihren direkten Attributwerten und Hierarchien** Jedes Tupel $(q, v_1, \ldots, v_n) \in prop$ wird durch ein Fakt

$$\mathsf{prop}(\mathsf{q}, \mathsf{v_1}, \ldots, \mathsf{v_n}).$$

repräsentiert. Die die Enthaltenseins-Beziehung vertretenden Elemente $(w_1, w_2) \in contain_i$ der Attribute $A_i, m + 1 \leq i \leq n$ werden durch Fakten

$$\mathsf{contain}(\mathsf{i}, \mathsf{w_1}, \mathsf{w_2}).$$

dargestellt.

**Wertebereiche und Domänen** Die Fragen bzw. die Attributwerte der Fragen-IDs können durch

$$\mathsf{question}(\mathsf{Q}) \leftarrow \mathsf{prop}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}).$$

abgeleitet werden. Im Falle eines flachen Attributs $A_i, 1 \leq i \leq m$ genügt zur Ableitung der Domäne die Regel

$$\mathsf{domain}(\mathsf{i}, \mathsf{V_i}) \leftarrow \mathsf{prop}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_{i-1}}, \mathsf{V_i}, \mathsf{V_{i+1}}, \ldots, \mathsf{V_n}).$$

Im Falle eines hierarchischen Attributs $A_i, m + 1 \leq i \leq n$ werden zur obigen Regel die folgenden

$$\mathsf{domain}(\mathsf{i}, \mathsf{V_i}) \leftarrow \mathsf{contain}(\mathsf{i}, \mathsf{W}, \mathsf{V_i}).$$
$$\mathsf{domain}(\mathsf{i}, \mathsf{V_i}) \leftarrow \mathsf{contain}(\mathsf{i}, \mathsf{V_i}, \mathsf{W}).$$

noch hinzugefügt.

**Erweiterung der Zuordnungsrelation um die hierarchischen Attributwerte** Die Daten aus $propTrans$ werden mit Hilfe von rekursiven Regeln aus den $prop$- und $contain$-Relationen errechnet: „Alle $prop$-Fakten sind auch in $propTrans$ enthalten“:

$$\mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}) \leftarrow \mathsf{prop}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}).$$

Die Erweiterung auf die hierarchischen Attribute erfolgt durch die $n - m$ Regeln

$$\begin{aligned}
&\mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_m}, \mathsf{W_{m+1}}, \mathsf{V_{m+2}}, \ldots, \mathsf{V_n}) \leftarrow \mathsf{contain}(\mathsf{m}+1, \mathsf{W_{m+1}}, \mathsf{V_{m+1}}),\\
&\qquad \mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}), \mathsf{question}(\mathsf{Q}), \mathsf{domain}(1, \mathsf{V_1}), \ldots, \mathsf{domain}(\mathsf{n}, \mathsf{V_n}).\\
&\ldots\\
&\mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_{n-1}}, \mathsf{W_n}) \leftarrow \mathsf{contain}(\mathsf{n}, \mathsf{W_n}, \mathsf{V_n}),\\
&\qquad \mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}), \mathsf{question}(\mathsf{Q}), \mathsf{domain}(1, \mathsf{V_1}), \ldots, \mathsf{domain}(\mathsf{n}, \mathsf{V_n}).
\end{aligned}$$

**Bemerkung 6.1.1 (Stratifizierung und Domänenprädikate)** Wie zu Beginn des Abschnitts 6.1 bzw. in Abschnitt 4.2.2.4 beschrieben, werden in den *smodels*-Programmen stratifizierte Regeln verwendet. Ohne die formalen Grundlagen in Ausführlichkeit darzustellen –, der interessierte Leser sei hier auf weiterführende Literatur verwiesen (z. B. [Syr01, Syr04]) – soll hier für die eben definierten Regeln die Stratifizierung in Abbildung 6.1 gezeigt werden. Hierbei geht man wie folgt vor:

1. *Erstellen des Abhängigkeitsgraphen:*
   - Das Prädikat question ist vom Prädikat prop vermöge der Regel
     $$\mathsf{question}(\mathsf{Q}) \leftarrow \mathsf{prop}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}).$$
     positiv abhängig – das question-Literal erscheint im Kopf und das prop-Literal im Rumpf der Regel. In analoger Weise ist domain von prop und contain positiv abhängig.
   - Vermöge der zuletzt gezeigten Regeln ist das Prädikat propTrans von prop und sich selbst positiv abhängig.
2. *Konstruktion der Prädikathierarchie/Stratifizierung* unter Berücksichtigung folgender Regeln [Syr01]:
   - Hängt das Prädikat $P$ vom Prädikat $Q$, jedoch $Q$ nicht von $P$ ab, so ist $P$ in einer echt höheren Schicht als $Q$.
   - Hängt das Prädikat $P$ positiv vom Prädikat $Q$ und $Q$ positiv von $P$ ab, so befinden sich $P$ und $Q$ in derselben Schicht.
   - Wenn die Prädikate $P$ und $Q$ gegenseitig abhängig sind und eine negative Kante im Abhängigkeitszyklus enthalten ist, so befinden sich $P$ und $Q$ in der obersten Schicht ($\omega$-Schicht).

Wie schon erwähnt, besteht die Einschränkung der hier betrachteten Programme im Wesentlichen darin, dass jede Variable in einer Regel in einem positiven Rumpfliteral vorkommen muss, das sich in einem echt niedrigeren Stratum als der Kopf der Regel – bezüglich einer Stratifizierung der Domänenregeln bzw. deren Prädikate – befindet. Wie man leicht sieht, ist diese Bedingung für die zuletzt

gezeigte rekursive Regel erfüllt. Der Grund hierfür liegt daran, dass die Domänenprädikate question, domain und contain in echt niederen Schichten als das Domänenprädikat propTrans liegen. Daher stellt die rekursive Berechnung von propTrans kein Problem hinsichtlich der Terminierung des Programms dar.

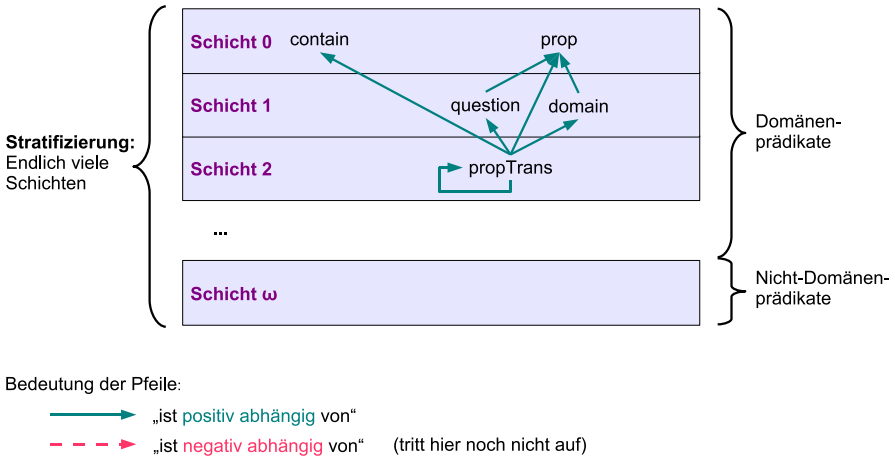



Abbildung 6.1: Stratifizierung und Domänenprädikate

**Projektionen von *propTrans* auf bestimmte Attribute** Zur allgemeinen Modellierung und aufgrund möglicher *prop*-Rollen-Semantik „alle Rollen“ werden zusätzliche Hilfsprädikate benötigt. Für jede nicht leere Teilmenge $proj \subset \{1, \ldots, n\}$ wird ein Prädikat $\mathsf{propTransProj}_{\mathsf{proj}}$ definiert, welches die Projektion von propTrans auf $Q$ und die $proj$-Attribute repräsentieren soll:

$$\mathsf{propTransProj}_{\mathsf{i_1,\ldots,i_s}}(\mathsf{Q}, \mathsf{V_{i_1}}, \ldots, \mathsf{V_{i_s}}) \leftarrow \mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}).$$

wobei ohne Einschränkung $i_1 < \cdots < i_s$ gilt. Im allgemeinen Fall sind dies also $2^n - 1$ Regeln. Wie einführend schon erwähnt, ist in manchen Fällen eine Vorberechnung und Tabellierung von Prädikaten sinnvoll. Dies könnte hier auch geschehen.

Tauchen in einem Anwendungsfall keine hierarchischen Attributwerte auf oder werden keinerlei Anforderungen an diese gestellt, so können entsprechende Projektionen auch für *prop* definiert werden. Die Berechnung von *propTransProj* ist dann nicht erforderlich. Ist außerdem in einem Anwendungsbereich nur die Semantik „eine Rolle“ gewünscht, so sind diese Hilfsregeln nicht notwendig.

#### 6.1.1.3 Constraintmodell: Benutzerdefinierte Anforderungen

Ausgehend vom Modell wird im Folgenden schrittweise die Repräsentation der benutzerdefinierten Anforderungen in ASP aufgezeigt. Durch

$$\mathsf{number(0..N).}$$

sind die $N + 1$ Fakten „number(i).“ $(0 \leq i \leq N)$ definiert. Dieses Hilfsprädikat number kann dazu benutzt werden, um einen endlichen Wertebereich für zahlenwertige Variablen vorzugeben und damit entsprechende Regeln sicher („domain-restricted“) zu machen (vgl. [SNS02]). Es wird davon ausgegangen, dass $N$ geeignet (groß) gewählt wird.

**Auswahl der Fragen** Hier geht es um die Gesamtanzahl der zu wählenden Fragen, also um die Transformation folgender Anforderung:

$$cardinalityCons(Q_{(*,\ldots,*)}, n_{min}, n_{max}) \text{ bzw. } cardinalityCons^{u}((*,\ldots,*), n_{min}, n_{max})$$

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^{\infty}$

TEMPLATE:

1. „Grundierte Modellierung“: Für $n_{max} \in \mathbb{N}_0$ :

   $$\mathsf{n_{min}\ \{chosenQuestion(Q) : question(Q)\}\ n_{max}\ \leftarrow}$$

   bzw. für $n_{max} = \infty$:

   $$\mathsf{n_{min}\ \{chosenQuestion(Q) : question(Q)\}\ \leftarrow}$$

2. „Variable Modellierung“: Neben dem Fakt zur Spezifikation der Anzahlen

   $$\mathsf{cardConsAll(n_{min}, n_{max}).}$$

   werden die allgemeinen Berechnungsregeln

   $$\mathsf{N_{min}\{chosenQuestion(Q) : question(Q)\}N_{max} \leftarrow cardConsAll(N_{min}, N_{max}), number(N_{min}), number(N_{max}).}$$

   $$\mathsf{N_{min}\{chosenQuestion(Q) : question(Q)\}\ \leftarrow cardConsAll(N_{min}, \infty), number(N_{min}).}$$

   benötigt.

Anzumerken ist, dass durch diese Regeln chosenQuestion-Atome erzeugt werden.

**Bemerkung 6.1.2 (Auswahl der Fragen: Vereinfachung der variablen Modellierung)** Um einen endlichen Wertebereich für die Variablen $\mathsf{N_{min}}$ und $\mathsf{N_{max}}$ vorzugeben wird in obiger Regel das Domänenprädikat number verwendet. Damit ist diese Regel auch „sicher“, auch wenn keine konkrete Bedingung $\mathsf{cardConsAll}(\mathsf{n_{min}}, \mathsf{n_{max}})$ spezifiziert ist. Da jedoch nur mit den entsprechenden Berechnungsregeln die chosenQuestion-Atome erzeugt, also hiermit die Fragen gewählt werden, ist eine Spezifikation von cardConsAll unabdingbar. Ist hier keine konkrete Einschränkung vorgegeben, so kann von einer belieben Anzahl an zu wählenden Fragen ausgegangen werden. Dies kann durch das Fakt $\mathsf{cardConsAll}(0, \infty)$ ausgedrückt werden.

Alternativ zu obiger Darstellung könnte man auch einfach zur IDB das Spezifikationsfakt

$$\mathsf{cardConsAll}(0, \infty)$$

hinzufügen und die Berechnungsregeln zu

$$\mathsf{N_{min}}\{\mathsf{chosenQuestion(Q)} : \mathsf{question(Q)}\}\mathsf{N_{max}} \leftarrow \mathsf{cardConsAll}(\mathsf{N_{min}}, \mathsf{N_{max}}), \mathsf{N_{max}} \neq \infty.$$

$$\mathsf{N_{min}}\{\mathsf{chosenQuestion(Q)} : \mathsf{question(Q)}\} \leftarrow \mathsf{cardConsAll}(\mathsf{N_{min}}, \infty).$$

abwandeln. Eine zusätzliche Spezifiktation durch den Benutzer kann in der EDB durch ein weiteres Spezifikationsfakt $\mathsf{cardConsAll}(\mathsf{n_{min}}, \mathsf{n_{max}})$ realisiert werden. Die schränkt die Anzahl der zu wählenden Fragen einfach nur noch weiter ein. In diesem Fall kann also auf das number-Prädikat verzichtet werden.

**Bemerkung 6.1.3 (Verschiedene Transformationsansätze)** Zur ASP-Modellierung von benutzerdefinierten Anforderungen der Art $\mathfrak{C}(c_1, \ldots, c_n)$ mit Konstanten $c_1, \ldots, c_n$ gibt es prinzipiell zwei verschiedene Ansätze:

1. *Variante 1 („Grundierte Modellierung“[1]):* Jede Instanz dieser Constraints $\mathfrak{C}(c_1, \ldots, c_r)$ wird eigens modelliert. Die transformierte Regel enthält die konkreten Konstanten $c_1, \ldots, c_r$ der Anforderung. Die Grundstruktur der entsprechenden Regel ist:

   $$\mathsf{head}(\mathsf{c_1}, \ldots, \mathsf{c_r}) \leftarrow \mathsf{body}(\mathsf{c_1}, \ldots, \mathsf{c_r}).$$

2. *Variante 2 („Variable Modellierung“):* Die Modellierung erfolgt für die Gesamtheit der Constraints $\mathfrak{C}(C_1, \ldots, C_r)$, wobei $C_i, 1 \leq i \leq r, r \in \mathbb{N}$ als Variablen für gültige Belegungen $c_i$ auftreten. Die Grundstruktur der entsprechenden Regeln ist:

   $$\mathsf{constraintSpecification}(\mathsf{c_1}, \ldots, \mathsf{c_r}).$$
   $$\mathsf{head}(\mathsf{C_1}, \ldots, \mathsf{C_r}) \leftarrow \mathsf{body}(\mathsf{C_1}, \ldots, \mathsf{C_r}), \mathsf{constraintSpecification}(\mathsf{C_1}, \ldots, \mathsf{C_r}).$$

[1] Der Begriff „grundiert“ bezieht sich auf die konkrete Belegung mit den Konstanten $c_1, \ldots, c_r$ an den entsprechenden Stellen. Die Regel muss allerdings noch nicht vollständig grundiert sein, es können noch weitere Variablen auftreten.

Vor der Bearbeitung mit SMODELS muss noch eine Grundierung der Regeln bezüglich der Variablen $C_i$ erfolgen.

Variante 2 hat den Vorteil, dass die invarianten Kernregeln für viele konkrete Anforderungen $\mathfrak{C}(c_1, \ldots, c_r)$ verwendet werden können. Dieser Ansatz ist also in gewisser Weise allgemeingültiger. Bei Variante 1 sind diese für jede Anforderung eigens zu erstellen. Nachteil von Variante 2 ist jedoch, dass meist für den konkreten Anwendungsfall auch unnötige Regeln auszuwerten sind.

Im Folgenden wird oftmals die „variable Modellierung“ gezeigt, in dem Wissen, dass auch die einfachere, „grundierte Modellierung“ möglich wäre. Eine Entscheidung für die eine oder andere Variante kann im konkreten Anwendungsfall erfolgen.

**Implementierung der *prop*-Rollen-Semantik** Je nachdem, welche *prop*-Rollen-Semantik für eine Frage implementiert werden soll, müssen für eine gewählte Frage unterschiedlich viele Attributwerte gewählt werden.

CONSTRAINTART: Rollenconstraint *roleCons*

PARAMETER: Frage $q \in Q$, Rolle $r \in \{\text{all}, \text{one}\}$

TEMPLATE: Die Fakten zur Spezifikation der gewünschten *prop*-Rollen-Semantik

$$\mathsf{roleCons}(\mathsf{q}, \mathsf{r}).$$

werden eins zu eins auf Grundlage der *roleCons*-Relation transformiert. Die allgemeine Berechnungsregel zur Implementierung der Semantik „alle Rollen“ – alle *prop*-Tupel (chosenProp) der gewählten Fragen werden gewählt – lautet

$$\mathsf{chosenProp}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n) \leftarrow \mathsf{chosenQuestion}(\mathsf{Q}), \mathsf{prop}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n), \mathsf{roleCons}(\mathsf{Q}, \mathsf{all}).$$

und die zur Implementierung der Semantik „eine Rolle“ – ein *prop*-Tupel wird pro gewählter Fragen gewählt – ist

$$1\ \{\mathsf{chosenProp}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n) : \mathsf{prop}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n)\}\ 1 \leftarrow \mathsf{chosenQuestion}(\mathsf{Q}), \mathsf{question}(\mathsf{Q}), \mathsf{roleCons}(\mathsf{Q}, \mathsf{one}).$$

Anzumerken ist, dass es in manchen Fällen sinnvoll ist, diese *roleCons*-Atome mit Hilfe von Regeln zu „erzeugen“. Möchte man beispielsweise für alle Fragen die Semantik „eine Rolle“ realisieren, so kann dies durch

$$\mathsf{roleCons}(\mathsf{Q}, \mathsf{one}) \leftarrow \mathsf{question}(\mathsf{Q}).$$

geschehen. Analoges gilt für die Implementierung der Semantik „alle Rollen“. Gemischte Varianten sind in ähnlicher Weise zu realisieren.

**Berechnung der zugehörigen hierarchischen Attributwerte gewählter direkter Attributwerte** In Analogie zur Definition von $propTrans$ als Erweiterung von $prop$ um die entsprechenden hierarchischen Attributwerte wird $chosenPropTrans$ als Erweiterung von $chosenProp$ definiert. Die „gewählten Attributwerte“ setzen sich neben den gewählten direkten Attributwerten

$$\mathsf{chosenPropTrans}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{n}) \leftarrow \mathsf{chosenProp}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{n}),\mathsf{prop}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{n}).$$

auch aus den zugehörigen indirekten zusammen:[2]

$$\begin{aligned}
\mathsf{chosenPropTrans}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{m},\mathsf{W}_{\mathsf{m}+1},\mathsf{V}_{\mathsf{m}+2}\ldots,\mathsf{V}_\mathsf{n}) \leftarrow\ & \mathsf{contain}(\mathsf{m}+1,\mathsf{W}_{\mathsf{m}+1},\mathsf{V}_{\mathsf{m}+1}),\\
& \mathsf{chosenPropTrans}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{n}),\\
& \mathsf{question}(\mathsf{Q}),\mathsf{domain}(1,\mathsf{V}_1),\ldots,\mathsf{domain}(\mathsf{n},\mathsf{V}_\mathsf{n}).\\
\ldots\\
\mathsf{chosenPropTrans}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_{\mathsf{n}-1},\mathsf{W}_\mathsf{n}) \leftarrow\ & \mathsf{contain}(\mathsf{n},\mathsf{W}_\mathsf{n},\mathsf{V}_\mathsf{n}),\\
& \mathsf{chosenPropTrans}(\mathsf{Q},\mathsf{V}_1,\ldots,\mathsf{V}_\mathsf{n}),\\
& \mathsf{question}(\mathsf{Q}),\mathsf{domain}(1,\mathsf{V}_1),\ldots,\mathsf{domain}(\mathsf{n},\mathsf{V}_\mathsf{n}).
\end{aligned}$$

**Bemerkung 6.1.4 (Fragen und gewählte Fragen)** Eine im Fragenpool befindliche Frage $q$ wird durch das Fakt $\mathsf{question}(\mathsf{q})$ repräsentiert. Wird diese hingegen durch einen Prüfer *gewählt*, so drückt sich dies im Modell durch das Atom $\mathsf{chosenQuestion}(\mathsf{q})$ aus. Man benötigt also zwei verschiedene Prädikate, um die Fragen von den gewählten Fragen zu unterscheiden.

Dies wirkt sich auf sämtliche Prädikate, welche der Beschreibung der Fragen dienen, aus. Neben jedem solchen Prädikat ist ein entsprechendes chosen-Prädikat erforderlich. Man benötigt ja die Möglichkeit, Bedingungen an die gewählten Eigenschaften gewählter Fragen auszudrücken. Im allgemeinen Fall treten daher im Programm neben den Prädikaten prop, propTrans, propTransProj zur Beschreibung der Fragen auch die Prädikate chosenProp, chosenPropTrans und chosenPropTransProj zur Beschreibung der gewählten Fragen auf. Entsprechende Berechnungsregeln sind hierzu natürlich notwendig.

In manchen Anwendungen sind nicht alle dieser Prädikate erforderlich. Aus Optimierungsgründen sollten in diesem Fall dann die entsprechenden Regeln nicht verwendet werden. Werden beispielsweise keinerlei Bedingungen an hierarchische Attributwerte gestellt, so ist weder propTrans noch chosenPropTrans notwendig. Eine Analyse des konkreten Anwendungsfalles kann hierüber entscheiden.

[2] Auch bei diesen rekursiven Regel sind keine Terminierungsprobleme aufgrund der gewährleisteten Domänenbeschränktheit zu erwarten.

**Projektionen gezogener Attributwerte auf bestimmte Attribute** Berechnung der „gewählten $\mathsf{propTransProj}_{\mathsf{proj}}$-Tupel“ für beliebige nicht leere Teilmengen $proj \subset \{1, \ldots, n\}$ :

$$\mathsf{chosenPropTransProj}_{\mathsf{i_1,\ldots,i_s}}(\mathsf{Q}, \mathsf{V_{i_1}}, \ldots, \mathsf{V_{i_s}}) \leftarrow \mathsf{chosenPropTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}), \mathsf{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}).$$

wobei ohne Einschränkung $i_1 < \cdots < i_s$ gilt. Dies sind $2^n - 1$ Regeln. Für die allgemeine Modellierung und aufgrund möglicher $prop$-Rollen-Semantik „alle Rollen“ sind diese Hilfsregeln notwendig. Wie oben schon erwähnt sind diese allerdings nicht in allen Anwendungsfällen erforderlich.

**Anzahlconstraints bei impliziter Spezifikation mit numerischen Grenzen** Im Gegensatz zu dem oben besprochenen Fall der „Auswahl von Fragen“ geht es hier nicht um die Gesamtzahl der Fragen im Test, sondern um die Anzahl an Fragen mit gewissen Eigenschaften (Attributwertkombinationen):

$$cardinalityCons^u((v_1, \ldots, v_n), n_{min}, n_{max}),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\}) \setminus \{(*, \ldots, *)\}$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^{\infty}$. Dies stellt eine Bedingung an die gewählten Attributwerte dar.

**Bemerkung 6.1.5 (Grundidee der Transformation der Anforderungen)** Viele Anforderungen können gemäß folgender Grundstruktur umgesetzt werden (vgl. auch Bemerkung 6.1.3):

1. „Grundierte Modellierung:“
   (a) Bildung der Hauptregel zur Behandlung des Constraints:

   $$\mathsf{check}(\mathsf{id}) \leftarrow \text{< REQUIREMENT >}.$$

   Hierbei wird die eigentliche Anforderung im Rumpf der Regel durch < REQUIREMENT > repräsentiert. Ist diese erfüllt, so kann auch das Atom $\mathsf{check}(\mathsf{id})$ abgeleitet werden. Das Atom $\mathsf{check}(\mathsf{id})$ ist nur dann in einem Modell enthalten, wenn die Anforderung erfüllt ist.
   (b) Forderung, dass $\mathsf{check}(\mathsf{id})$ im Modell enthalten sein muss:

   $$\bot \leftarrow \mathsf{not\ check}(\mathsf{id}).$$

   Dies wird hier mit Hilfe einer Integritätsconstraint-Regel umgesetzt: Die rechte Seite der Regel darf *nicht* erfüllt sein („Denial“)!
2. In Analogie: „Variable Modellierung“:
   (a) Spezifikation der Anforderung unter Verwendung eines Constraint-Identifikators:

   $$\mathsf{constraintSpecification}(\mathsf{id}, \ldots).$$

(b) Bildung der Hauptregel zur Behandlung des Constraints:

$$\mathsf{check}(\mathsf{Id}) \leftarrow \text{< REQUIREMENT >}, \mathsf{constraintSpecification}(\mathsf{Id}, \dots).$$

Auch hierbei wird die eigentliche Anforderung im Rumpf der Regel durch < REQUIREMENT > repräsentiert. Ist der Rumpf der Regel für die gegebene Constraintspezifikation mit dem Identifikator id, also die entsprechende Anforderung, erfüllt, so auch das Atom check(id).

(c) Forderung, dass check(id) im Modell enthalten sein muss:

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{ConstraintSpecification}(\mathsf{Id}, \dots).$$

Gelegentlich ist die Negation der Anforderung einfacher umzusetzen. Dann ist folgendes Vorgehen oft adäquater – hier skizziert für den Fall „grundierter Modellierung“:

1. Umsetzung der Negation der Anforderung:

$$\mathsf{checkFalse}(\mathsf{id}) \leftarrow \text{< NOT REQUIREMENT >}.$$

2. Übertragung auf die „positive Formulierung“:

$$\mathsf{check}(\mathsf{id}) \leftarrow \mathsf{not\ checkFalse}(\mathsf{id}).$$

3. Formulierung als hartes Constraint:

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{id}).$$

Die jeweils letzte Regel macht das betrachtete Constraint zu einem sogenannten „harten“ Constraint. In einem Modell muss es auf jeden Fall erfüllt sein. Eine leichte Modifikation des aufgezeigten Ansatzes lässt auch eine entsprechende Modellierung als „weiches“ Constraint zu. Mehr hierzu im Abschnitt 6.1.1.4.

Nun zur konkreten Transformation:

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $(v_1, \dots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\}) \setminus \{(*, \dots, *)\}$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^{\infty}$

TEMPLATE: Spezifikation:[3]

$$\mathsf{cardCons}(\mathsf{id}, \mathsf{v}_1, \dots, \mathsf{v}_n, \mathsf{n}_{\mathsf{min}}, \mathsf{n}_{\mathsf{max}}).$$

[3] Um Inkonsistenzhinweise geben zu können, wird hier ein numerischer Constraint-Identifikator id noch eingeführt zur Identifikation des entsprechenden Constraints. Dieser muss eindeutig sein. Der Platzhalter * für beliebige Attributwerte wird hier durch die Konstante *x* repräsentiert. Es kann auch ein anderes Symbol gewählt werden, das als Attributwert nicht vorkommt.

Kernregeln für den Fall direkter oder indirekter Attributwerte ohne den Platzhalter für beliebige Attributwerte * und für $n_{max} \neq \infty$:

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}\{\mathsf{chosenPropTrans(Q, V_1, \ldots, V_n)} : \mathsf{propTrans(Q, V_1, \ldots, V_n)}\}\mathsf{N_{max}},\\ & \mathsf{cardCons(Id, V_1, \ldots, V_n, N_{min}, N_{max})}, \mathsf{V_1 \neq x, \ldots, V_n \neq x, N_{max} \neq \infty}.\end{aligned}$$

bzw. für $n_{max} = \infty$:

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}\{\mathsf{chosenPropTrans(Q, V_1, \ldots, V_n)} : \mathsf{propTrans(Q, V_1, \ldots, V_n)}\},\\ & \mathsf{cardCons(Id, V_1, \ldots, V_n, N_{min}, N_{max})}, \mathsf{V_1 \neq x, \ldots, V_n \neq x, N_{max} = \infty}.\end{aligned}$$

Behandlung des Falls, dass genau bei den $i_1, \ldots, i_s$-ten Attributwerten direkte oder hierarchische Attributwerte und bei den übrigen $j_1, \ldots, j_{n-s}$-ten Attributwerten gerade die Platzhalter * bzw. *x* stehen und $n_{max} \neq \infty$:[4]

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}\{\mathsf{chosenPropTransProj}_{\mathsf{i_1,\ldots,i_s}}\mathsf{(Q, V_{i_1}, \ldots, V_{i_s})} :\\ & \qquad \mathsf{propTransProj}_{\mathsf{i_1,\ldots,i_s}}\mathsf{(Q, V_{i_1}, \ldots, V_{i_s})}\}\mathsf{N_{max}},\\ & \mathsf{cardCons(Id, V_1, \ldots, V_n, N_{min}, N_{max})},\\ & \mathsf{V_{i_1} \neq x, \ldots, V_{i_s} \neq x, V_{j_1} = x, \ldots, V_{j_{n-s}} = x, N_{max} \neq \infty}.\end{aligned}$$

Für $n_{max} = \infty$ analog.

Die Deklaration der Anzahlconstraints als „harte Constraints“ erfolgt mit Hilfe einer Integritätsconstraint-Regel:

$$\bot \leftarrow \mathsf{not\ check(Id)}, \mathsf{cardCons(Id, V_1, \ldots, V_n, N_{min}, N_{max})}.$$

Aus Performanzgründen kann eine eigenständige Behandlung direkter Attributwerte sinnvoll sein. Möchte man die Angaben bei direkten Attributwerten $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (range(A_i) \cup \{*\}) \setminus \{(*, \ldots, *)\}$ eigens behandeln und sich eventuell dann die Berechnungen für propTrans und chosenPropTrans sparen, so kann man auch folgende Regel für den Fall direkter Attributwerte und $n_{max} \neq \infty$ – und für die anderen Fälle analoge Regeln – aufstellen:[5]

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}\{\mathsf{chosenProp(Q, V_1, \ldots, V_n)} : \mathsf{prop(Q, V_1, \ldots, V_n)}\}\mathsf{N_{max}},\\ & \mathsf{cardCons(Id, V_1, \ldots, V_n, N_{min}, N_{max})}, \mathsf{V_1 \neq x, \ldots, V_n \neq x},\\ & \mathsf{N_{max} \neq \infty}.\end{aligned}$$

[4] Zu beachten ist, dass in Berücksichtigung der Bemerkung 6.1.3 abhängig vom konkreten Anwendungsfall möglicherweise die „grundierte Version“ zu bevorzugen ist. Diese Regeln sind im allgemeinen Fall notwendig, da eine Frage mehreren Werten einer Domäne zugeordnet werden kann. Dadurch dürfen in diesen Anzahlconstraints bis auf die Fragen-ID keine freien Variablen auftreten, was andernfalls zu falschem „Zählen“ führen kann.

[5] Weiß man, dass in einem konkreten Anwendungsbereich die Fragen stets in der Semantik „einer Rolle“ auftreten soll, so kann die Modellierung mit den zusätzlichen Regeln mit chosenPropProj unterbleiben, da jeder gewählten Frage nur ein gewähltes *prop*-Tupel zugeordnet ist.

Andere Fälle sind in entsprechender Weise zu behandeln.

Wegen der semantischen Äquivalenz der Constraints $cardinalityCons(C, m, m)$ und $cardinalityCons(C, m)$ (vgl. Bemerkung 5.1.14) ist eine eigene Behandlung von $cardinalityCons(C, m)$ bzw. $cardinalityCons^u((v_1, \ldots, v_n), m), m \in \mathbb{N}_0$ nicht erforderlich.

**Anzahlconstraints bei expliziter Spezifikation mit numerischen Grenzen** Als Nächstes wird der Fall explizit spezifizierter Constraintreferenzmengen und einer unteren und oberen Schranke für die Anzahl der zu wählenden Elemente betrachtet.

$$cardinalityCons(C, n_{min}, n_{max})$$

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ und $C = \{q_1, \ldots, q_N \mid N \in \mathbb{N}\} \subseteq Q$

TEMPLATE: Für $n_{max} \neq \infty$:

$$\mathsf{check(id)} \leftarrow \mathsf{n_{min}}\{\mathsf{chosenQuestion(q_1)}, \ldots, \mathsf{chosenQuestion(q_N)}\}\ \mathsf{n_{max}}.$$

Für $n_{max} = \infty$:

$$\mathsf{check(id)} \leftarrow \mathsf{n_{min}}\{\mathsf{chosenQuestion(q_1)}, \ldots, \mathsf{chosenQuestion(q_N)}\}.$$

Erzwingung der Erfüllung der entsprechenden Anforderung:

$$\bot \leftarrow \mathsf{not\ check(id)}.$$

Nun zu den Anzahlconstraints mit extensionalen Constraintreferenzmengen und dem Schlüsselwort „all“:

$$cardinalityCons(C, \text{all})$$

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $C = \{q_1, \ldots, q_N \mid N \in \mathbb{N}\} \subseteq Q$

TEMPLATE: Spezifikation: Für alle $1 \leq i \leq N$:

$$\mathsf{cardCons(id, q_i, all)}.$$

Eine einfache Umsetzung der Anforderung erfolgt durch:

$$\mathsf{chosenQuestion(Q)} \leftarrow \mathsf{question(Q), cardCons(Id, Q, all)}.$$

Möchte man bei der ASP-Umsetzung die entsprechenden Regeln für die Anforderung wie in obigen Fällen auch durch eine ID ausstatten, so wäre alternativ auch

$$\mathsf{checkFalse(Id)} \leftarrow \mathsf{not\ chosenQuestion(Q)}, \mathsf{question(Q)}, \mathsf{cardCons(Id, Q, all)}.$$
$$\mathsf{check(Id)} \leftarrow \mathsf{not\ checkFalse(Id)}, \mathsf{cardCons(Id, Q, all)}.$$

möglich – zusammen mit:

$$\bot \leftarrow \mathsf{not\ check(Id)}, \mathsf{cardCons(Id, Q, all)}.$$

Das checkFalse-Atom ist erfüllt, wenn es mindestens eine Frage unter den spezifizierten gibt, die nicht gewählt ist. Ist dies nicht der Fall, so ist das zugehörige check-Atom und damit die Anforderung erfüllt. Würde man das checkFalse-Atom durch $\bot$ ersetzen, so wäre dies ebenso eine korrekte Umsetzung der Anforderung. Allerdings wäre hiermit auch ein Verlust der Identifikation über das check-Atom gegeben.

**Anzahlconstraints mit nichtnumerischer Spezifikation „all“** Als Letztes soll hier der Fall betrachtet werden, dass aus einer Menge von Fragen mit bestimmten Attributwertkombinationen alle Elemente zu wählen sind:

$$cardinalityCons^u((v_1, \ldots, v_n), \text{all}),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\}) \setminus \{(*, \ldots, *)\}$.

Obwohl dieser Fall in diesem Anwendungsbereich oftmals nur eine nebensächliche Rolle spielt, soll er der Vollständigkeit halber erläutert werden. Allerdings werden zur Umsetzung der korrekten Semantik dieser Constraints zusätzliche Hilfsregeln erforderlich, welche gemäß der $prop$-Rollen-Semantik gewisse Teilmengen von $prop$ repräsentieren. Die Bedingung an eine derartige Teilmenge ist, dass jede *Frage* mit der passenden Anzahl von Tupel vertreten ist. Das Prädikat chosenProp genügt hier nicht, da hierdurch nur die gewählten Attributwerte *gewählter* Fragen ausgedrückt werden, also nur ein Teil der Fragen vertreten ist. Aus diesem Grunde wird ein neues Prädikat roleProp eingeführt, das entprechend der $prop$-Rollen-Semantik zu *jeder* Frage bestimmte Attributwerte auszeichnet. Es muss ja irgendwie geprüft werden, ob auch *alle* Fragen mit bestimmten Attributwertkombination bezüglich dieser $prop$-Teilmengen im potentiellen Test enthalten sind. Als Nebenbedingung ist dann zu fordern, dass die gewählten Attributwerte *gewählter* Fragen in diesen enthalten sind.

Ähnlich zur Errechnung von chosenProp wird definiert:

$$\mathsf{roleProp(Q, V_1, \ldots, V_n)} \leftarrow \mathsf{prop(Q, V_1, \ldots, V_n)}, \mathsf{role(Q, all)}.$$
$$1\{\mathsf{roleProp(Q, V_1, \ldots, V_n)} : \mathsf{prop(Q, V_1, \ldots, V_n)}\}1 \leftarrow \mathsf{question(Q)}, \mathsf{role(Q, one)}.$$

Die Erweiterung auf die hierarchischen Attributwerte (rolePropTrans) geschieht mit Hilfe rekursiver Regeln analog zur Errechnung von propTrans aus prop. Die Nebenbedingung, dass roleProp für die

gewählten Fragen auch die gewählten Attributwerte darstellt, wird durch folgende Regel umgesetzt:

$$\begin{aligned}\mathsf{chosenProp}(\mathsf{Q},\mathsf{V_1},\ldots,\mathsf{V_n}) \leftarrow\ & \mathsf{chosenQuestion}(\mathsf{Q}),\mathsf{roleProp}(\mathsf{Q},\mathsf{V_1},\ldots,\mathsf{V_n}),\\ & \mathsf{prop}(\mathsf{Q},\mathsf{V_1},\ldots,\mathsf{V_n}).\end{aligned}$$

Nun zum eigentlichen Constraint:

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $(v_1,\ldots,v_n) \in \prod_{i=1}^{n}(domain(A_i)\cup\{*\})\setminus\{(*,\ldots,*)\}$

TEMPLATE: Spezifikation:

$$\mathsf{cardCons}(\mathsf{id},\mathsf{v_1},\ldots,\mathsf{v_n},\mathsf{all}).$$

Behandlung dieser Anforderung:

$$\begin{aligned}\mathsf{checkFalse}(\mathsf{Id}) \leftarrow\ & \mathsf{not\ chosenQuestion}(\mathsf{Q}),\mathsf{rolePropTrans}(\mathsf{Q},\mathsf{V_1},\ldots,\mathsf{V_n}),\\ & \mathsf{propTrans}(\mathsf{Q},\mathsf{V_1},\ldots,\mathsf{V_n}),\mathsf{cardCons}(\mathsf{Id},\mathsf{V_1},\ldots,\mathsf{V_n},\mathsf{all}).\\ \mathsf{check}(\mathsf{Id}) \leftarrow\ & \mathsf{not\ checkFalse}(\mathsf{Id}),\mathsf{cardCons}(\mathsf{Id},\mathsf{V_1},\ldots,\mathsf{V_n},\mathsf{all}).\end{aligned}$$

Integritätsconstraint-Regel, welche die Anforderung als hartes Constraint umsetzt:

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}),\mathsf{cardCons}(\mathsf{Id},\mathsf{V_1},\ldots,\mathsf{V_n},\mathsf{all}).$$

**Obligatorische Elemente** Hier geht es um die Anforderung, dass gewisse Fragen obligatorisch im Test enthalten sein müssen:

$$containCons^{u}(v_1,\ldots,v_n) \text{ bzw. } containCons(C),$$

wobei $(v_1,\ldots,v_n) \in \prod_{i=1}^{n}(domain(A_i)\cup\{*\})\setminus\{(*,\ldots,*)\}$ bzw. $C = \{q_i \mid 1 \leq i \leq N, N \in \mathbb{N}\} \subseteq Q$.

Wegen der semantischen Äquivalenz der Constraints $cardinalityCons(C,\text{all})$ und $containCons(C)$ (vgl. Bemerkung 5.1.14) ist eine eigene Behandlung von $containCons(C)$ eigentlich nicht erforderlich. Der Deutlichkeit halber und um das Schlüsselwort „all“ in den Regeln zu vermeiden, kann die Umsetzung von $containCons$ bevorzugt werden. Zur Spezifikation wird das einstellige Prädikat

$$\mathsf{containCons}$$

eingeführt. Ansonsten erfolgt die Umsetzung wie oben.

**Gewichtssummenconstraints bei impliziter Spezifikation und einer Gewichtsfunktion** Anforderungen an die Gewichtssumme stellen eine Verallgemeinerung von Anzahlconstraints dar. Implizit spezifiziert sind sie von der Gestalt

$$weightsumCons^u((v_1, \ldots, v_n), n_{min}, n_{max}),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$. Dies stellt ebenso wie die Anzahlconstraints eine Bedingung an die gezogenen Attributwerte dar.

Die Gewichtsfunktion $weight : Q \longrightarrow \mathbb{N}_0$ sei durch die Attributwerte eines numerischen Attributs $A_w, w \in \{1, \ldots, n\}$ festgelegt.

CONSTRAINTART: Gewichtssummenconstraint *weightsumCons*

PARAMETER: $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$ und $w \in [1, n]$

TEMPLATE: Transformation der Gewichtsfunktion:

$$\#\text{weight chosenPropTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}) = \mathsf{V_w}.$$

Und für $w \in \{i_1, \ldots, i_s\} \subset \{1, \ldots, n\}$:

$$\#\text{weight chosenPropTransProj}_{\mathsf{i_1},\ldots,\mathsf{i_s}}(\mathsf{Q}, \mathsf{V_{i_1}}, \ldots, \mathsf{V_{i_s}}) = \mathsf{V_w}.$$

Spezifikation der Anforderung:

$$\text{weightsumCons}(\mathsf{id}, \mathsf{v_1}, \ldots, \mathsf{v_n}, \mathsf{n_{min}}, \mathsf{n_{max}}).$$

Behandlungsregel für den Fall $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} domain(A_i)$ und $n_{max} \neq \infty$

$$\begin{aligned}\text{check}(\mathsf{Id}) \leftarrow\ & \mathsf{N_{min}}[\text{chosenPropTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n}) : \text{propTrans}(\mathsf{Q}, \mathsf{V_1}, \ldots, \mathsf{V_n})]\mathsf{N_{max}}, \\ & \text{weightsumCons}(\mathsf{Id}, \mathsf{V_1}, \ldots, \mathsf{V_n}, \mathsf{N_{min}}, \mathsf{N_{max}}), \mathsf{V_1} \neq \mathsf{x}, \ldots, \mathsf{V_n} \neq \mathsf{x}, \mathsf{N_{max}} \neq \infty.\end{aligned}$$

und für $n_{max} = \infty$ analog. Tritt in der Spezifikation mindestens ein Platzhalter * auf, so werden wie bei den Anzahlconstraints weitere Regeln benötigt. Sind genau an den Stellen $i_1, \ldots, i_s$ keine Platzhalter * und an den Stellen $j_1, \ldots, j_{n-s}$ schon und ist außerdem $w \in \{i_1, \ldots, i_s\}$ und ferner $n_{max} \neq \infty$, so ist folgende Regel relevant:

$$\begin{aligned}\text{check}(\mathsf{Id}) \leftarrow\ & \mathsf{N_{min}}[\text{chosenPropTransProj}_{\mathsf{i_1},\ldots,\mathsf{i_s}}(\mathsf{Q}, \mathsf{V_{i_1}}, \ldots, \mathsf{V_{i_s}}) : \\ & \qquad \text{propTransProj}_{\mathsf{i_1},\ldots,\mathsf{i_s}}(\mathsf{Q}, \mathsf{V_{i_1}}, \ldots, \mathsf{V_{i_s}})]\mathsf{N_{max}}, \\ & \text{weightsumCons}(\mathsf{Id}, \mathsf{V_1}, \ldots, \mathsf{V_n}, \mathsf{N_{min}}, \mathsf{N_{max}}), \\ & \mathsf{V_{i_1}} \neq \mathsf{x}, \ldots, \mathsf{V_{i_s}} \neq \mathsf{x}, \mathsf{V_{j_1}} = \mathsf{x}, \ldots, \mathsf{V_{j_{n-s}}} = \mathsf{x}, \mathsf{N_{max}} \neq \infty.\end{aligned}$$

Taucht hingegen das Gewichtsattribut $A_w$ nicht unter den Attributen $A_{i_1}, \ldots, A_{i_s}$ auf, so muss dieses zu den projizierten Attributen hinzugenommen werden. Damit korrekte Gewichtssummen bestimmt

werden, ist gefordert, dass zu einer Frage nicht mehrere Tupel in $\mathsf{chosenPropTransProj}_{i_1,\ldots,i_s,w}$[6] vertreten sind. Die Umsetzung ist:

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}[\mathsf{chosenPropTransProj}_{i_1,\ldots,i_s}(\mathsf{Q},\mathsf{V_{i_1}},\ldots,\mathsf{V_{i_s}},\mathsf{V_w}) : \\ & \qquad \mathsf{propTransProj}_{i_1,\ldots,i_s,w}(\mathsf{Q},\mathsf{V_{i_1}},\ldots,\mathsf{V_{i_s}},\mathsf{V_w})]\mathsf{N_{max}}, \\ & \mathsf{weightsumCons(Id,V_1},\ldots,\mathsf{V_n},\mathsf{N_{min}},\mathsf{N_{max}}), \\ & \mathsf{V_{i_1}} \neq \mathsf{x},\ldots,\mathsf{V_{i_s}} \neq \mathsf{x},\mathsf{V_{j_1}} = \mathsf{x},\ldots,\mathsf{V_{j_{n-s}}} = \mathsf{x},\mathsf{N_{max}} \neq \infty.\end{aligned}$$

Andere Fälle sind analog zu behandeln. Deklaration als hartes Constraint:

$$\bot \leftarrow \mathsf{not\ check(Id)},\mathsf{weightsumCons(Id,V_1},\ldots,\mathsf{V_n},\mathsf{N_{min}},\mathsf{N_{max}}).$$

**Gewichtssummenconstraints bei impliziter Spezifikation und mehreren Gewichtsfunktionen** Ist mehr als eine Gewichtungsfunktion $weight_t, 1 \leq t \leq k, k \in \mathbb{N}$ – also ein Attribut über dessen Summe eine Anforderung definiert ist – gegeben, so werden weitere Regeln zur Unterscheidung benötigt.

ZUSÄTZLICHER PARAMETER: $t \in [1,k], k \in \mathbb{N}$ (als Nummer der Gewichtsfunktion)

TEMPLATE: Dies geschieht analog zu oben! Die Constraint-Spezifikation für ein Gewichtsconstraint zum $t$-ten Gewicht kann durch

$$\mathsf{weightsumCons(id,t,v_1},\ldots,\mathsf{v_n},\mathsf{n_{min}},\mathsf{n_{max}}).$$

modelliert werden. Neben den Regeln

$$\begin{aligned}\mathsf{propTrans(t,Q,V_1},\ldots,\mathsf{V_n}) &\leftarrow \mathsf{propTrans(Q,V_1},\ldots,\mathsf{V_n}). \\ \mathsf{chosenProp(t,Q,V_1},\ldots,\mathsf{V_n}) &\leftarrow \mathsf{propTrans(Q,V_1},\ldots,\mathsf{V_n}),\mathsf{chosenPropTrans(Q,V_1},\ldots,\mathsf{V_n}).\end{aligned}$$

und den Anweisungen zur Gewichtung

$$\#\mathsf{weight\ chosenProp(t,V_1},\ldots,\mathsf{V_n}) = \mathsf{V_{w_t}}.$$

werden Regeln zur Behandlung der Anforderungen benötigt, hier für $(v_1,\ldots,v_n) \in \prod_{i=1}^{n} domain(A_i), n_{max} \neq \infty$:

$$\begin{aligned}\mathsf{check(Id)} \leftarrow\ & \mathsf{N_{min}}[\mathsf{chosenPropTrans(T,Q,V_1},\ldots,\mathsf{V_n}) : \mathsf{propTrans(T,Q,V_1},\ldots,\mathsf{V_n})]\mathsf{N_{max}}, \\ & \mathsf{weightsumCons(Id,T,V_1},\ldots,\mathsf{V_n},\mathsf{N_{min}},\mathsf{N_{max}}),\mathsf{V_1} \neq \mathsf{x},\ldots,\mathsf{V_n} \neq \mathsf{x},\mathsf{N_{max}} \neq \infty.\end{aligned}$$

Auch hier sind weitere Fälle in analoger Weise und unter Berücksichtigung obiger Ausführungen zu formulieren.

[6] Der Einfachheit halber wird $w$ an letzter Stelle notiert, ist aber an entsprechend richtiger Stelle bei aufsteigender Reihenfolge zu denken.

**Ausschluss von Elementen** Das Gegenstück zu obligatorischen Elementen ist der Ausschluss von Fragen. Bei expliziter Spezifikation geht es um das Constraint

$$excludeCons(C),$$

wobei $C \subseteq Q$ eine Teilmenge des Fragenkatalogs ist.

Zunächst für den Fall expliziter Spezifikation:

CONSTRAINTART: Ausschluss von Elementen *excludeCons*

PARAMETER: $C = \{q_1, \ldots, q_N \mid N \in \mathbb{N}\} \subseteq Q$

TEMPLATE: Spezifikation: Für alle $1 \leq i \leq N$:

excludeCons(id, $q_i$).

Eine einfache Umsetzung der Anforderung erfolgt durch die Regel

$\bot \leftarrow$ chosenQuestion(Q), question(Q), excludeCons(Id, Q).

und eine alternative – unter Verwendung eines Identifikators für die Constraints – durch die Regeln

checkFalse(Id) $\leftarrow$ not chosenQuestion(Q),question(Q), excludeCons(Id, Q).
check(Id) $\leftarrow$ not checkFalse(Id), excludeCons(Id, Q).

zusammen mit der Integritätsconstraint-Regel für die Behandlung als harte Anforderungen:

$\bot \leftarrow$ not check(Id),excludeCons(Id, Q).

Bei impliziter Spezifikation, also bei

$$excludeCons^u(v_1, \ldots, v_n),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$, kann in entsprechender Weise verfahren werden.

CONSTRAINTART: Ausschluss von Elementen *excludeCons*

PARAMETER: $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$

TEMPLATE: Spezifikation:

excludeCons(id, $v_1, \ldots, v_n$).

Berechnungsregeln:

checkFalse(Id) $\leftarrow$ chosenPropTrans(Q, $V_1, \ldots, V_n$), question(Q),
excludeCons(Id, $V_1, \ldots, V_n$).
check(Id) $\leftarrow$ not checkFalse(Id), excludeCons(Id, $V_1, \ldots, V_n$).

Deklaration als hartes Constraint:

$\bot \leftarrow$ not check(Id), excludeCons(Id, $V_1, \ldots, V_n$).

**Prozentuale Anforderungen** Prozentuale Constraints sind in der Regel implizit spezifiziert:

$$percentCons^u((v_1, \ldots, v_n), i, w_i, p_{min}, p_{max}),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$. Sie können durch die „Überführung“ in ein Anzahlconstraint (siehe Bemerkung 5.2.40) in ASP transformiert werden.

CONSTRAINTART: Prozentualconstraint *percentCons*

PARAMETER: $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$ mit der zusätzlichen Einschränkung $v_i = *$, $i \in [1, n]$, $w_i \in domain(A_i)$, $p_{min}$, $p_{max} \in \mathbb{R}_0^+$

TEMPLATE: Spezifikation:

$$\mathsf{percentCons}(\mathsf{id}, \mathsf{v_1}, \ldots, \mathsf{v_n}, \mathsf{i}, \mathsf{w_i}, \mathsf{p_{min}}, \mathsf{p_{max}}).$$

Ähnlich zu den Anzahlconstraints sind hier einige Fälle bezüglich der Belegung der Parameter zu unterscheiden. Dies soll hier allerdings nicht durchgeführt werden. Exemplarisch wird die „grundierte Version“ für den Fall gezeigt, dass außer $v_i$ keines der Attribute durch * belegt ist. Hierbei ist $v_i$ in der ASP-Regel durch eine neue Variable $V_i$ zu ersetzen:[7]

$$\begin{aligned}\mathsf{check(id)} \leftarrow\ & \mathsf{T}\{\mathsf{chosenPropTrans(Q, v_1, \ldots, v_n)} : \mathsf{propTrans(Q, v_1, \ldots, v_n)}\}\mathsf{T},\\ & \mathsf{N_{min}}\{\mathsf{chosenPropTrans(Q, v_1, \ldots, v_{i-1}, w_i, v_{i+1}, \ldots, v_n)} :\\ & \qquad \mathsf{propTrans(Q, v_1, \ldots, v_{i-1}, w_i, v_{i+1}, \ldots, v_n)}\}\mathsf{N_{max}},\\ & \mathsf{number(T), N_{min} = T * p_{min}, N_{max} = T * p_{max}}).\end{aligned}$$

Und wieder allgemein die Deklaration als hartes Constraint:

$$\bot \leftarrow \mathsf{not\ check(Id), percentCons(Id, V_1, \ldots, V_n, I, W_I, P_{min}, P_{max})}$$

**Konjunktion und Disjunktion von Constraints** Da in einem Modell sämtliche Anforderungen gelten müssen, ist dies durch die Aufschreibung der einzelnen zu erfüllenden Regeln automatisch in ASP realisiert. Diese sind implizit konjunktiv verknüpft.

Die Disjunktion spielt hier eine untergeordnete Rolle und kann als Sonderfall des Auswahlconstraints betrachtet werden, dessen Umsetzung in ASP weiter unten besprochen wird.

[7] Aus „technischen Gründen“ (Rechnen mit Dezimalzahlen) ist statt dem Prozentwert $p$ eigentlich $p * 100 = \widetilde{p}$ anzugeben und in der ASP-Regel ist $N_{min} = T * p_{min}$ durch $N_{min} = (T * \widetilde{p})/100$ zu ersetzen; $N_{max}$ analog.

**Bedingte Constraints** Bestimmte Anforderungen sind möglicherweise nur dann zu erfüllen, wenn bestimmte Voraussetzungen gelten. Im Modell haben diese Constraints die Gestalt

$$conditionCons(c_1, c_2),$$

wobei $c_1, c_2$ (möglicherweise auch komplexe) Constraints sind.

CONSTRAINTART: Bedingungsconstraint *conditionCons*

PARAMETER: $c_1, c_2 \in Constraint$ bzw. deren IDs in den Regeln $\mathsf{id}_1, \mathsf{id}_2$

TEMPLATE: Spezifikation:

$$\mathsf{conditionCons}(\mathsf{id}, \mathsf{id}_1, \mathsf{id}_2).$$

Behandlung:

$$\mathsf{checkFalse}(\mathsf{Id}) \leftarrow \mathsf{not\ check}(\mathsf{Id}_1), \mathsf{check}(\mathsf{Id}_2), \mathsf{conditionCons}(\mathsf{Id}, \mathsf{Id}_1, \mathsf{Id}_2).$$
$$\mathsf{check}(\mathsf{Id}) \leftarrow \mathsf{not\ checkFalse}(\mathsf{Id}), \mathsf{conditionCons}(\mathsf{Id}, \mathsf{Id}_1, \mathsf{Id}_2).$$

Deklaration als hartes Constraint:

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{conditionCons}(\mathsf{Id}, \mathsf{Id}_1, \mathsf{Id}_2).$$

Hierbei sind die Constraints mit den Identifikatoren $\mathsf{id}_1$ und $\mathsf{id}_2$ durch Regeln der Art

$$\mathsf{check}(\mathsf{id}_1) \leftarrow < \text{REQUIREMENT1} >.$$
$$\mathsf{check}(\mathsf{id}_2) \leftarrow < \text{REQUIREMENT2} >.$$

modelliert. Zu beachten ist, dass die Integritätsregeln zur Deklaration als harte Constraints

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{id}_1).$$
$$\bot \leftarrow \mathsf{not\ check}(\mathsf{id}_2).$$

*nicht* enthalten sein sollen.

Gelegentlich genügt auch eine einfachere Umsetzung. Muss beispielsweise die Frage $q_2$ im Test enthalten sein, wenn dies für $q_1$ der Fall ist, so kann dies einfach durch

$$\mathsf{chosenQuestion}(\mathsf{q}_2) \leftarrow \mathsf{chosenQuestion}(\mathsf{q}_1).$$

modelliert werden.

**Auswahlconstraints** Ist unter einer vorgegebenen Menge an Constraints nur eine bestimmte Anzahl zu erfüllen, so kann dies mit Hilfe eines Auswahlconstraints modelliert werden:

$$choiceCons(C, n_1, n_2),$$

wobei $C \subseteq \mathcal{C}$ eine Constraintmenge und $n_1, n_2 \in \mathbb{N}_0$ eine untere und eine obere Grenze darstellen.

CONSTRAINTART: Auswahlconstraint $choiceCons$

PARAMETER: Constraintmenge $C = \{c_1, \ldots c_N \mid N \in \mathbb{N}\} \subseteq \mathcal{C}$ bzw. deren IDs in den Regeln $\mathsf{id}_1, \ldots, \mathsf{id}_N$, Grenzen $n_{min}, n_{max} \in \mathbb{N}_0$

TEMPLATE: Umsetzung des Constraints:

$$\mathsf{check(id)} \leftarrow \mathsf{n_{min}}\{\mathsf{check}(\mathsf{id}_1), \ldots, \mathsf{check}(\mathsf{id}_N)\}\mathsf{n_{max}}.$$

wobei id eine noch nicht verwendete Constraint-ID darstellt. Die Deklaration als hartes Constraint kann wie gewohnt geschehen:

$$\bot \leftarrow \mathsf{not\ check(id)}.$$

Hierbei sind die Constraints $c_i$ durch die Regeln

$$\mathsf{check}(\mathsf{id_i}) \leftarrow \text{< REQUIREMENT I >}.$$

repräsentiert. Wie bei den Bedingungsconstraints dürfen hier die Regeln zur Deklaration als harte Constraints *nicht* enthalten sein.

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{id_i}).$$

**Bemerkung 6.1.6 (Behandlung der Metaconstraints)** Zu bemerken ist, dass die Constraints, welche sich auf eine Constraintmenge beziehen (Metaconstraints) im Prinzip auch nach der in Bemerkung 6.1.5 dargestellten Grundidee transformieren lassen – hier in grundierter Form:

$$\begin{aligned} \mathsf{check(id)} &\leftarrow \text{<REQUIREMENT>}. \\ \bot &\leftarrow \mathsf{not\ check(id)}. \end{aligned}$$

Beachtenswert ist hier ferner, dass die einzelnen Constraints in der transformierten Anforderung <REQUIREMENT> durch die check-Atome identifiziert werden. Es gilt ja, dass die Anforderung $c_i$ erfüllt ist, wenn das Atom $\mathsf{check}(\mathsf{id_i})$ im Modell enthalten ist.

**Optimierung einer Zielfunktion** Mit Hilfe von Optimierungsconstraints kann eine Art weicher Constraints modelliert werden. Zunächst zu den Optimierungsconstraints, die sich auf eine Fragenmenge beziehen. Bei impliziter Spezifikation handelt es sich um ein Constraint der Art

$$optimizeQCons^{u}((v_1, \ldots, v_n), o, s),$$

wobei $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$, $o \in \{\text{min, max}\}$, $s \in \mathcal{P}(\mathcal{C})$.

CONSTRAINTART: Optimierungsconstraint *optimizeQCons*

PARAMETER: $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} (domain(A_i) \cup \{*\})$, $o \in \{\text{min, max}\}$[8]

TEMPLATE: Für den Fall $(v_1, \ldots, v_n) \in \prod_{i=1}^{n} domain(A_i)$ und $o = \text{max}$ ergibt sich folgende Transformation:

$$\#\text{maximize}\,\{\text{chosenPropTrans}(\text{Q}, \text{v}_1, \ldots, \text{v}_\text{n}) : \text{propTrans}(\text{Q}, \text{v}_1, \ldots, \text{v}_\text{n})\}.$$

Treten im Tupel $(v_1, \ldots, v_n)$ an einer oder mehreren Stellen $j$ auch die Platzhalter $v_j = *$ auf, so werden diese in der ASP-Regel jeweils durch eine neue Variable $V_j$ ersetzt. Im Fall $o = \text{min}$ wird das #minimize-Statement verwendet.

Bei expliziter Spezifikation einer Fragenmenge

$$optimizeQCons(C, o, s),$$

wobei $C \subseteq Q$, $o \in \{\text{min, max}\}$, $s \in \mathcal{P}(\mathcal{C})$, wird wie folgt transformiert:

CONSTRAINTART: Optimierungsconstraint *optimizeQCons*

PARAMETER: $C = \{q_1, \ldots q_N \mid N \in \mathbb{N}\} \subseteq Q$, $o \in \{\text{min, max}\}$

TEMPLATE: Falls $o = \text{max}$:

$$\#\text{maximize}\,\{\text{chosenQuestion}(\text{q}_1), \ldots, \text{chosenQuestion}(\text{q}_\text{N})\}.$$

Andere Fälle mit Bezug zu Fragenmengen sind analog zu behandeln. Nun zu den Optimierungsconstraints, welche sich auf Constraints beziehen.

$$optimizeCCons(C, o, s),$$

wobei $C \subseteq \mathcal{C}$, $o \in \{\text{min, max}\}$, $s \in \mathcal{P}(\mathcal{C})$.

CONSTRAINTART: Optimierungsconstraint *optimizeCCons*

PARAMETER: $C = \{c_1, \ldots c_N \mid N \in \mathbb{N}\} \subseteq \mathcal{C}$ bzw. deren IDs in den ASP-Regeln $id_1, \ldots, id_N$, $o \in$

[8] Es wird hier davon ausgegangen, dass $s$ durch die restlichen Regeln des Programms repräsentiert wird. Ein Parameter wird für $s$ also nicht benötigt.

{min, max}

TEMPLATE: Falls $o = \max$:

$$\#\mathsf{maximize}\,\{\mathsf{check}(\mathsf{c_1}), \ldots, \mathsf{check}(\mathsf{c_N})\}.$$

Analog für $o = \min$. Es wird wieder davon ausgegangen, dass die Constraints $c_i$ durch die Regeln

$$\mathsf{check}(\mathsf{id_i}) \leftarrow <\text{ REQUIREMENT I }>.$$

repräsentiert sind und diese Constraints nicht hart sind, also

$$\bot \leftarrow \mathsf{not}\ \mathsf{check}(\mathsf{id_i}).$$

*nicht* im Programm enthalten sind.

Es sei weiterhin bemerkt, dass es möglich ist, mehrere Optimierungs-Statements zu formulieren, welche eine Priorität gemäß ihrer Reihenfolge besitzen.

#### 6.1.1.4 Lösungsmodell: Generierung der Tests

Die Antwortmengen einer Spezifikation stellen eine Codierung der gültigen Prüfungen bezüglich der gegebenen Testspezifikation dar.

**Berechnung der Antwortmengen** Die Berechnung aller Antwortmengen geschieht durch die Anweisung

$$\#\ \mathsf{compute}\ 0\{\}.$$

Oftmals möchte man nicht alle, sondern nur eine gewisse maximale Anzahl $n$ berechnen. Dies kann durch die Anweisung

$$\#\ \mathsf{compute}\ \mathsf{n}\{\}.$$

geschehen. Die Ausgabe kann durch hide- und show-Deklarationen gesteuert werden, so dass nur Atome mit bestimmten Prädikatbezeichnungen ausgegeben werden.

**Erweiterung um den Modus „harte“ Constraints** Um die Deklaration von harten Constraints zu erleichtern bzw. um auch diesbezüglich einfache Veränderungen vornehmen zu können, wird ein neues Prädikat hard eingeführt, welches dazu dient, Constraints als hart zu kennzeichnen.

EINSTELLUNG: Constraintmodus

PARAMETER: Constraint $c$ bzw. deren ASP-Regel-ID id

UMSETZUNG: Spezifikation eines Constraints als hart:

$$\mathsf{hard}(\mathsf{id}).$$

Behandlung: Die in vorherigen Abschnitten eingeführten Regeln der Art[9]

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{constraintSpecification}(\mathsf{Id}).$$

können einfach durch diese eine Regel

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{hard}(\mathsf{Id}).$$

ersetzt werden.

**Behandlung von Inkonsistenzen** Möglicherweise gibt es keine gemäß der Testspezifikation gültige Prüfung (siehe Abschnitt 5.2.2.6). In diesem Falle möchte man Hinweise auf Inkonsistenzen oder zumindest eine Prüfung, die „fast“ eine gültige Prüfung darstellt. Hierzu muss zumindest bei einem Teil der ursprünglich harten Constraints eine Umstellung auf „weich“ erfolgen.

EINSTELLUNG: Inkonsistenzcheck

PARAMETER: Constraint $c$ bzw. deren ASP-Regel-ID id

UMSETZUNG: Spezifikation eines Constraints als „gewünscht“ (weich, aber gewünscht als hart)

$$\mathsf{wanted}(\mathsf{id}).$$

Zur Realisierung wird die Regel

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{hard}(\mathsf{Id}).$$

wie folgt erweitert:[10]

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{hard}(\mathsf{Id}), \mathsf{not\ wanted}(\mathsf{Id}).$$

[9] Das Prädikat constraintSpecification enthält neben der Constraint-ID abhängig von der Constraintart noch weitere Parameter.

[10] Ist das Constraint mit dem Identifikator id „weich“ (wanted), so folgt auch dann kein Widerspruch ($\bot$), wenn check(id) nicht erfüllt ist.

Weiterhin wird das Programm um die Hilfsregel

$$\mathsf{wantedFalse(Id)} \leftarrow \mathsf{not\ check(Id), wanted(Id).}$$

ergänzt. Zur Maximierung der Anzahl der check(id)-Atome wird die Optimierungs-Anweisung

$$\#\,\mathsf{maximize\{check(Id) : wanted(Id)\}.}$$

hinzugefügt und die Berechnung durch den Befehl

$$\#\,\mathsf{compute\ 0\{\}.}$$

angestoßen. Die in einer Antwortmenge enthaltenen wantedFalse-Atome geben schließlich Hinweis auf die gewünschten, jedoch nicht erfüllten Constraints.

Die Möglichkeiten der Variation verschiedener Einstellungs-Modi und der Behandlung von Inkonsistenzen leisten einen wichtigen Beitrag zur Benutzerfreundlichkeit des Programms.

**Beispiel 6.1.7 (Programm und Antwortmengen)** Es soll in diesem Beispiel von den Gegebenheiten des Beispiels 5.2.13 ausgegangen werden. In einem konkreten Anwendungsbereich sollen keine Anforderungen an hierarchische Attribute gestellt werden. Das in ASP transformierte Strukturmodell sieht daher gemäß Abschnitt 6.1.1.2 wie folgt aus:[11]

% Zuordnung der Fragen zu ihren direkten Attributwerten und Hierarchien:

$\mathsf{prop}(q_1, 1, \mathit{easy}, \mathit{topicA})$.

$\mathsf{prop}(q_2, 5, \mathit{medium}, \mathit{topicB})$. $\mathsf{prop}(q_2, 5, \mathit{difficult}, \mathit{topicC})$.

$\mathsf{prop}(q_3, 1, \mathit{medium}, \mathit{topicC})$.

$\mathsf{contain}(3, \mathit{topicTop}, \mathit{topicX})$. $\mathsf{contain}(3, \mathit{topicTop}, \mathit{topicY})$.

$\mathsf{contain}(3, \mathit{topicX}, \mathit{topicA})$. $\mathsf{contain}(3, \mathit{topicX}, \mathit{topicB})$. $\mathsf{contain}(3, \mathit{topicX}, \mathit{topicC})$.

$\mathsf{contain}(3, \mathit{topicY}, \mathit{topicC})$. $\mathsf{contain}(3, \mathit{topicY}, \mathit{topicD})$.

% Wertebereiche und Domänen:

$$\begin{aligned}
\mathsf{question(Q)} &\leftarrow \mathsf{prop(Q, V_1, V_2, V_3).}\\
\mathsf{domain(1, V_1)} &\leftarrow \mathsf{prop(Q, V_1, V_2, V_3).}\\
\mathsf{domain(2, V_2)} &\leftarrow \mathsf{prop(Q, V_1, V_2, V_3).}\\
\mathsf{domain(3, V_3)} &\leftarrow \mathsf{prop(Q, V_1, V_2, V_3).}\\
\mathsf{domain(3, V_3)} &\leftarrow \mathsf{contain(3, V_3, W).}\\
\mathsf{domain(3, V_3)} &\leftarrow \mathsf{contain(3, W, V_3).}
\end{aligned}$$

[11] Die nach dem Zeichen „%“ eingefügten Texte sind Kommentare.

Im Constraintmodell wird die Auswahl einer beliebigen Anzahl von Fragen durch folgende Regeln realisiert:

$$\mathsf{cardConsAll}(0,\infty).$$
$$\mathsf{N_{min}}\{\mathsf{chosenQuestion}(\mathsf{Q}) : \mathsf{question}(\mathsf{Q})\}\mathsf{N_{max}} \leftarrow \mathsf{cardConsAll}(\mathsf{N_{min}},\mathsf{N_{max}}),\mathsf{N_{max}} \neq \infty.$$
$$\mathsf{N_{min}}\{\mathsf{chosenQuestion}(\mathsf{Q}) : \mathsf{question}(\mathsf{Q})\} \leftarrow \mathsf{cardConsAll}(\mathsf{N_{min}},\infty)$$

Es soll nun davon ausgegangen werden, dass alle Fragen die *prop*-Rollen-Semantik „einer Rolle“ haben. Die Regeln zur Spezifikation und Behandlung dieser Semantik wären dann einfach wie folgt:

$$\mathsf{roleCons}(\mathsf{Q},\mathsf{one}) \leftarrow \mathsf{question}(\mathsf{Q}).$$
$$1\{\mathsf{chosenProp}(\mathsf{Q},\mathsf{V_1},\mathsf{V_2},\mathsf{V_3}) : \mathsf{prop}(\mathsf{Q},\mathsf{V_1},\mathsf{V_2},\mathsf{V_3})\}1 \leftarrow \mathsf{chosenQuestion}(\mathsf{Q}),\mathsf{question}(\mathsf{Q}),\mathsf{roleCons}(\mathsf{Q},\mathsf{one})$$

Nun zum Lösungsmodell: Werden ausgehend von diesem Programm die Modelle (Antwortmengen) berechnet, z. B. mit der Anweisung[12]

$$\#\ \mathsf{compute}\ 0\{\}.$$

so erhält man in diesem Fall 12 verschiedene Lösungen.[13] Eine dieser Antwortmengen ist die folgende:

$\mathsf{chosenQuestion}(\mathsf{q_1})$ $\mathsf{chosenQuestion}(\mathsf{q_2})$
$\mathsf{chosenProp}(q_1, 1, easy, topicA)$ $\mathsf{chosenProp}(q_2, 5, medium, topicB)$
$\mathsf{prop}(q_1, 1, easy, topicA)$ $\mathsf{prop}(q_2, 5, medium, topicB)$ $\mathsf{prop}(q_2, 5, difficult, topicC)$
$\mathsf{prop}(q_3, 1, medium, topicC)$
$\mathsf{contain}(3, topicTop, topicX)$ $\mathsf{contain}(3, topicTop, topicY)$ $\mathsf{contain}(3, topicX, topicA)$
$\mathsf{contain}(3, topicX, topicB)$ $\mathsf{contain}(3, topicX, topicC)$ $\mathsf{contain}(3, topicY, topicC)$
$\mathsf{contain}(3, topicY, topicD)$
$\mathsf{domain}(1, 1)$ $\mathsf{domain}(1, 5)$ $\mathsf{domain}(2, easy)$ $\mathsf{domain}(2, medium)$ $\mathsf{domain}(2, difficult)$
$\mathsf{domain}(3, topicA)$ $\mathsf{domain}(3, topicB)$ $\mathsf{domain}(3, topicC)$ $\mathsf{domain}(3, topicD)$
$\mathsf{domain}(3, topicX)$ $\mathsf{domain}(3, topicY)$ $\mathsf{domain}(3, topicTop)$
$\mathsf{roleCons}(\mathsf{q_1},\mathsf{one})$ $\mathsf{roleCons}(\mathsf{q_2},\mathsf{one})$ $\mathsf{roleCons}(\mathsf{q_3},\mathsf{one})$
$\mathsf{cardCons}(0,\infty)$

[12] Ist das Programm in einer Datei mit Namen `example.lp` gespeichert, so könnte der entsprechende Kommandozeilenaufruf etwa `lparse example.lp | smodels 0 >output_example.txt` lauten. Die Antwortmengen werden dann in einer Datei `output_example` ausgegeben. Die Angabe der „0“ entspricht gerade dem Parameter des `compute`-Statements.

[13] Ist $q_2$ im Modell nicht enthalten, so gibt es nämlich $\binom{2}{0} + \binom{2}{1} + \binom{2}{2} = 4$ Möglichkeiten der Auswahl. Ist $q_2$ enthalten, so hat man wegen der zu wählenden *prop*-Rollen-Semantik von $q_2$ genau $2 \cdot \left(\binom{2}{0} + \binom{2}{1} + \binom{2}{2}\right) = 8$ Möglichkeiten der Auswahl. Insgesamt gibt es daher 12 verschiedene Antwortmengen.

In dieser Lösung wurden die Fragen $q_1$ und $q_2$ gewählt. Für die Frage $q_2$ wurde als *mittelschwere (medium)* Frage zu Thema *topicB* gewertet.

Neben den gewählten Fragen und den gewählten Attributwerten – also den für einen Test interessanten Angaben – sind auch die Domänenprädikat-Atome vertreten. Letztere sind für alle Lösungen dieser Spezifikation identisch. Eine Ausgabe dieser Atome kann auf verschiedene Art und Weise unterbunden werden. Folgende Statements leisten das Gewünschte:[14]

$$\#\ \mathsf{hide}.$$
$$\#\ \mathsf{show}\ \mathsf{chosenQuestion}(\mathsf{Q}),\ \mathsf{chosenProp}(\mathsf{Q},\mathsf{V1},\mathsf{V2},\mathsf{V3}).$$

Obige Antwortmenge würde damit in der Form

$$\mathsf{chosenQuestion}(\mathsf{q}_1)\quad \mathsf{chosenQuestion}(\mathsf{q}_2)$$
$$\mathsf{chosenProp}(q_1, 1, \mathit{easy}, \mathit{topicA})\quad \mathsf{chosenProp}(q_2, 5, \mathit{medium}, \mathit{topicB})$$

ausgegeben werden. Sollen nun insgesamt zwischen 1 und 2 Fragen gewählt werden, so wird dieses Anzahlconstraint

$$\mathsf{cardConsAll}(1,2).$$

einfach noch hinzugefügt. Die Anzahl der Antwortmenge reduziert sich hiermit auf 9. Soll nun zusätzlich als „weiche Anforderung“ gelten, dass jeweils genau eine Frage zu Thema *topicB* (Constraint $c_1$) und zu Thema *topicC* (Constraint $c_2$) gewählt werden soll, so kann dies gemäß obigen Ausführungen wie folgt transformiert werden – hier in grundierter Form:[15]

$$\begin{aligned}
&\mathsf{wanted}(\mathsf{c}_1).\\
&\mathsf{wanted}(\mathsf{c}_2).\\
\mathsf{check}(\mathsf{c}_1) &\leftarrow 1\{\mathsf{chosenProp}(\mathsf{Q},\mathsf{V}_1,\mathsf{V}_2,\mathit{topicB}) : \mathsf{prop}(\mathsf{Q},\mathsf{V}_1,\mathsf{V}_2,\mathit{topicB})\}1.\\
\mathsf{check}(\mathsf{c}_2) &\leftarrow 1\{\mathsf{chosenProp}(\mathsf{Q},\mathsf{V}_1,\mathsf{V}_2,\mathit{topicC}) : \mathsf{prop}(\mathsf{Q},\mathsf{V}_1,\mathsf{V}_2,\mathit{topicC})\}1.\\
\bot &\leftarrow \mathsf{not}\ \mathsf{check}(\mathsf{Id}), \mathsf{hard}(\mathsf{Id}), \mathsf{not}\ \mathsf{wanted}(\mathsf{Id}).\\
\mathsf{wantedFalse}(\mathsf{Id}) &\leftarrow \mathsf{not}\ \mathsf{check}(\mathsf{Id}), \mathsf{wanted}(\mathsf{Id}).
\end{aligned}$$

*Ohne* das noch erforderliche Optimierungs-Statement

$$\#\,\mathsf{maximize}\{\mathsf{check}(\mathsf{Id}) : \mathsf{wanted}(\mathsf{Id})\}.$$

[14] Es gibt einige LPARSE-Optionen, mit deren Hilfe man die Ausgabe auf seine Bedürfnisse anpassen kann [Syr]. Beispielsweise unterdrückt auch der Aufruf `lparse -d none example.lp | smodels 0 >output_example.txt` die Ausgabe der Domänenliterale.

[15] Hier sei an die Beispiele 5.2.30 und 5.2.32 erinnert. Zu beachten ist auch, dass hier nicht die Regeln mit $\mathsf{chosenPropProj}_3$ nötig sind, da in diesem konkreten Anwendungsbereich sämtliche Fragen in der *prop*-Rollen-Semantik „einer Rolle“ auftreten. Es kann zu einer gewählten Frage also nur ein gewähltes prop- bzw. chosenProp-Tupel geben.

erhält man auch hier wieder 9 Antwortmengen. Eine mögliche wäre

$\mathsf{chosenQuestion}(\mathsf{q}_2)$ $\mathsf{chosenProp}(q_2, 5, medium, topicB)$
$\mathsf{check}(\mathsf{c}_1)$ $\mathsf{wantedFalse}(\mathsf{c}_2)$

Die Domänenliterale sind hier nicht notiert. In dieser Antwortmenge ist die Frage $q_2$ gewählt und zwar als *mittelschwere (medium)* Frage zu Thema *topicB*. Daher ist die Anforderung $c_1$ erfüllt ($\mathsf{check}(\mathsf{c}_1)$), $c_2$ jedoch nicht ($\mathsf{wantedFalse}(\mathsf{c}_2)$). Fügt man das oben angeführte Optimierungs-Statement noch hinzu, so werden nicht mehr sämtliche Antwortmengen berechnet. Das „optimale“ Modell ist schließlich wie folgt gegeben:

$\mathsf{chosenQuestion}(\mathsf{q}_2)$ $\mathsf{chosenQuestion}(\mathsf{q}_3)$
$\mathsf{chosenProp}(q_2, 5, medium, topicB)$ $\mathsf{chosenProp}(q_3, 5, medium, topicC)$
$\mathsf{check}(\mathsf{c}_1)$ $\mathsf{check}(\mathsf{c}_2)$

Modifiziert man obige Anforderung in der Art, dass von den Constraints $c_1$ und $c_2$ mindestens eines davon erfüllt sein muss, so ist dies als Auswahlconstraint $c_{choice}$ zu modellieren – auch hier in grundierter Form:

$$\mathsf{hard}(\mathsf{c}_{\mathsf{choice}}).$$
$$\mathsf{check}(\mathsf{c}_{\mathsf{choice}}) \leftarrow 1\{\mathsf{check}(\mathsf{c}_1), \mathsf{check}(\mathsf{c}_2)\}2.$$

wobei auch die Regel

$$\bot \leftarrow \mathsf{not\ check}(\mathsf{Id}), \mathsf{hard}(\mathsf{Id}), \mathsf{not\ wanted}(\mathsf{Id}).$$

im Programm enthalten sein muss. Hierbei gibt es 7 mögliche Lösungen. Die check-Atome geben wieder darüber Aufschluss, welche der Constraints $c_1$ und/oder $c_2$ erfüllt sind; $c_{choice}$ muss im Modell ja auf jeden Fall erfüllt sein, da es als hartes Constraint modelliert ist. Verändert man die Anforderungen hingegen in der Art, dass beide Constraints $c_1$ und $c_2$ zu erfüllen sind, so kann dies entweder durch deren Deklaration als harte Constraints

$$\mathsf{hard}(\mathsf{c}_1).\ \mathsf{hard}(\mathsf{c}_2).$$

oder durch die Modifikation von $c_{choice}$ geschehen:

$$\mathsf{check}(\mathsf{c}_{\mathsf{choice}}) \leftarrow 2\{\mathsf{check}(\mathsf{c}_1), \mathsf{check}(\mathsf{c}_2)\}2.$$

In beiden Fällen erhält man eine Antwortmenge – hier wieder ohne Domänenliterale:

$\mathsf{chosenQuestion}(\mathsf{q}_2)$ $\mathsf{chosenQuestion}(\mathsf{q}_3)$
$\mathsf{chosenProp}(q_2, 5, medium, topicB)$ $\mathsf{chosenProp}(q_3, 5, medium, topicC)$
$\mathsf{check}(\mathsf{c}_1)$ $\mathsf{check}(\mathsf{c}_2)$ $\mathsf{check}(\mathsf{c}_{\mathsf{choice}})$

Durch die zahlreichen Möglichkeiten der Modellierung von Constraints und durch die in den Antwortmengen codierbaren Hinweise auf Inkonsistenzen können bei der Generierung von Tests die relevanten Fragestellungen umfassend behandelt werden. Ein hohes Maß an Benutzerfreundlichkeit ist realisierbar.

#### 6.1.1.5 Verschiedene Modellebenen einer Testspezifikation, Fortsetzung

Im Folgenden wird das Beispiel aus Abschnitt 5.2.4 fortgesetzt. Es soll hier die noch fehlende interne Repräsentation in *Answer Set Programming (smodels)* gezeigt werden. Die Regeln für die Repräsentation des Strukturmodells und sonstige Hilfsregeln seien schon vorhanden. Die Bedingungen werden wie folgt modelliert:

- Wertung der Fragen im Test:

  $$\mathsf{role}(\mathsf{Q}, \mathrm{all}) \leftarrow \mathsf{question}(\mathsf{Q}).$$

  zusammen mit

  $$\mathsf{chosenProp}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n) \leftarrow \mathsf{chosenQuestion}(\mathsf{Q}), \mathsf{prop}(\mathsf{Q}, \mathsf{V}_1, \ldots, \mathsf{V}_n), \mathsf{roleCons}(\mathsf{Q}, \mathrm{all}).$$

- Modellierung der Anforderungen:

  Constraint $c_1$:

  In „variabler Version“:

  $$\mathsf{cardConsAll}(25, 30).$$
  $$\mathsf{N}_{\mathsf{min}}\{\mathsf{chosenQuestion}(\mathsf{Q}) : \mathsf{question}(\mathsf{Q})\}\mathsf{N}_{\mathsf{max}} \leftarrow \mathsf{cardConsAll}(\mathsf{N}_{\mathsf{min}}, \mathsf{N}_{\mathsf{max}}).$$

  oder einfach in „grundierter Version“:

  $$25\{\mathsf{chosenQuestion}(\mathsf{Q}) : \mathsf{question}(\mathsf{Q})\}30 \leftarrow$$

  Constraint $c_2$:

  In „variabler Version“: Spezifikation

  $$\mathsf{cardCons}(\mathsf{c}_2, \mathsf{x}, \mathsf{x}, \mathit{topicB}, 10, 12).$$

  und Behandlung

  $$\mathsf{check}(\mathsf{Id}) \leftarrow \mathsf{N}_{\mathsf{min}}\{\mathsf{chosenPropTransProj}_3(\mathsf{Q}, \mathsf{V}_3) : \mathsf{propTransProj}_3(\mathsf{Q}, \mathsf{V}_3)\}\mathsf{N}_{\mathsf{max}}, \mathsf{cardCons}(\mathsf{Id}, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{N}_{\mathsf{min}}, \mathsf{N}_{\mathsf{max}}), \mathsf{V}_1 = \mathsf{x}, \mathsf{V}_2 = \mathsf{x}, \mathsf{V}_3 \neq \mathsf{x}, \mathsf{N}_{\mathsf{max}} \neq \infty.$$

oder in „grundierter Version“:

$$\mathsf{check}(\mathsf{c}_2) \leftarrow 10\{\mathsf{chosenPropTransProj}_3(\mathsf{Q}, \mathit{topicB}) : \mathsf{propTransProj}_3(\mathsf{Q}, \mathit{topicB})\}12.$$

Constraint $c_3$:

Spezifikation:

$$\mathsf{weightsumCons}(\mathsf{c}_3, \mathsf{x}, \mathsf{x}, \mathsf{x}, 55, 60).$$
$$\#\mathsf{weight}\ \mathsf{chosenPropTransProj}_3(\mathsf{Q}, \mathsf{V}_3) = \mathsf{V}_3.$$

Entweder in „variabler Version“:

$$\mathsf{check}(\mathsf{Id}) \leftarrow \mathsf{N}_{\mathsf{min}}[\mathsf{chosenPropTransProj}_3(\mathsf{Q}, \mathsf{V}_3) : \mathsf{propTransProj}_3(\mathsf{Q}, \mathsf{V}_3)]\mathsf{N}_{\mathsf{max}},$$
$$\mathsf{weightsumCons}(\mathsf{Id}, \mathsf{x}, \mathsf{x}, \mathsf{x}, \mathsf{N}_{\mathsf{min}}, \mathsf{N}_{\mathsf{max}}), \mathsf{N}_{\mathsf{max}} \neq \infty.$$

oder in „grundierter Version“:

$$\mathsf{check}(\mathsf{c}_3) \leftarrow 55[\mathsf{chosenPropTransProj}_3(\mathsf{Q}, \mathsf{V}_3) : \mathsf{propTransProj}_3(\mathsf{Q}, \mathsf{V}_3)]60.$$

Constraint $c_4$:

Spezifikation:

$$\mathsf{cardCons}(\mathsf{c}_4, \mathit{5}, \mathit{medium}, \mathit{topicA}, \mathrm{all}).$$

In „variabler Version“

$$\mathsf{checkFalse}(\mathsf{Id}) \leftarrow \mathsf{not\ rolePropTrans}(\mathsf{Q}, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3), \mathsf{propTrans}(\mathsf{Q}, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3),$$
$$\mathsf{cardCons}(\mathsf{Id}, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3).$$
$$\mathsf{check}(\mathsf{Id}) \leftarrow \mathsf{not\ checkFalse}(\mathsf{Id}), \mathsf{cardCons}(\mathsf{Id}, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3).$$

oder in „grundierter Version“

$$\mathsf{checkFalse}(\mathsf{c}_4) \leftarrow \mathsf{not\ chosenQuestion}(\mathsf{Q}), \mathsf{propTrans}(\mathsf{Q}, \mathit{5}, \mathit{medium}, \mathit{topicA}).$$
$$\mathsf{check}(\mathsf{c}_4) \leftarrow \mathsf{not\ checkFalse}(\mathsf{c}_4).$$

Constraint $c_5$:

Deklaration als hartes Constraint:

$$\mathsf{hard}(\mathsf{c}_5).$$

Spezifikation:

$$\mathsf{excludeCons}(\mathsf{c}_5, \mathsf{q}_1).$$
$$\mathsf{excludeCons}(\mathsf{c}_5, \mathsf{q}_{27}).$$
$$\mathsf{excludeCons}(\mathsf{c}_5, \mathsf{q}_{40}).$$

zusammen mit der Umsetzung:

$$\mathsf{checkFalse(Id)} \leftarrow \mathsf{chosenQuestion(Q)}, \mathsf{question(Q)}, \mathsf{excludeCons(Id, Q)}.$$
$$\mathsf{check(Id)} \leftarrow \mathsf{not\ checkFalse(Id)}, \mathsf{excludeCons(Id, Q)}.$$

oder in einfacher Version:

$$\bot \leftarrow \mathsf{chosenQuestion(Q)}, \mathsf{excludeCons(Q)}.$$

Constraint $c_6$:

Deklaration als hartes Constraint:

$$\mathsf{hard(c_6)}.$$

Behandlung der Anforderung:

$$\mathsf{check(c_6)} \leftarrow 2\{\mathsf{check(c_2)}, \mathsf{check(c_3)}, \mathsf{check(c_4)}\}2.$$

Behandlung harter Constraints:

$$\bot \leftarrow \mathsf{not\ check(Id)}, \mathsf{hard(Id)}.$$

### 6.1.2 Studienberatung und Konsistenzprüfung von Studienordnungen

In einem ersten Schritt wird hier die Transformation der Kernstruktur gezeigt, in einem zweiten werden die Erweiterungen (vgl. Abschnitt 5.3.4) betrachtet.

#### 6.1.2.1 Kernkomponenten

Das logische ASP-Programm (*smodels*) umfasst folgende Bestandteile:

1. *Strukturelemente des Modulkatalogs* (realisierbar als extensionale Datenbank, EDB)
2. *Strukturelle Ableitungsregeln* (realisierbar als intensionale Datenbank, IDB)
3. *Allgemeine Regeln zur Modellierung der Constraints bzw. Constrainteigenschaften* (realisierbar als IDB)
4. *Regeln zur Repräsentation der Constraints der Studienordnung* (realisierbar als EDB und teilweise als IDB)

Im Folgenden ist ein Modulkatalog wie im Modell aus den Abschnitten 5.1 und 5.3 gegeben.

#### 6.1.2.2 Strukturmodell: Modulkatalog und Eigenschaften

Wie im Anwendungsbereich „Generierung von Tests“ wird auch hier nur ein Teil der Daten aus dem Strukturmodell direkt repräsentiert, wohingegen ein anderer Teil mit Hilfe von Regeln abgeleitet werden kann. Wichtiger Bestandteil der Datenstruktur ist der Modulkatalog-Objektgraph. Die Strukturtypen sind die darin auftretenden Typen der sogenannten Strukturelemente. Um hier die Darstellung der Regeln nicht unnötig kompliziert zu machen, wird davon ausgegangen, dass bezüglich der inneren Knoten neben dem Identifikator-Attribut `ID` nur noch ein weiteres Attribut, nämlich `Sort` (Art) bedeutsam ist. Für die Blattknoten, also bei den $Coursetemplate$-Strukturelementen wird der Einfachheit halber davon ausgegangen, dass als zusätzliche Attribute nur noch `ECTS` (ECTS-Leistungspunkte) und `HPW` (Semesterwochenstundenzahl) relevant sind. Diese Annahmen schränken die Allgemeingültigkeit des Ansatzes keineswegs ein, da weitere Attribute in analoger Weise hinzuzufügen sind.

**Strukturelemente und ihre Eigenschaften** Jedes Strukturelement $e$ des Modulkatalog-Objektgraphen wird durch ein Tupel

$$\mathsf{elementProp}(\mathsf{id}, \mathsf{type}, \mathsf{sort}).$$

repräsentiert. Hierbei stellen id und sort die Attributwerte zu den Attributen `Id` und `Sort` und ferner type den Strukturtyp des Objekts dar. Enthält ein Objekt kein Attribut `Sort`, so werden NULL-Werte (null) an entsprechender Stelle eingefügt. Für die $Coursetemplate$-Strukturelemente, also die Knoten der Blattebene werden die noch fehlenden Attributwerte („additional properties“) zu den Attributen `ECTS` und `HPW` durch Fakten der Gestalt

$$\mathsf{ctPropAdd}(\mathsf{id}, \mathsf{ects}, \mathsf{hpw}).$$

dargestellt.

**Beispiel 6.1.8 (Strukturelemente und ihre Eigenschaften)** Bezüglich des Modulkatalog-Objektgraphen aus den Abbildungen 5.27 und 5.28 sind das Modulecatalogue -Strukturelement *mc* und die Elemente des linken Teilbaums von *mc* in folgender Weise zu repräsentieren:

$$\mathsf{elementProp}(mc, \mathsf{Modulecatalogue}, null).$$
$$\mathsf{elementProp}(mg_A, \mathsf{Modulegroup}, null).$$
$$\mathsf{elementProp}(bm_1, \mathsf{Module}, basic).$$
$$\mathsf{elementProp}(bm_2, \mathsf{Module}, basic).$$
$$\mathsf{elementProp}(ctg_5, \mathsf{Coursetemplategroup}, choice).$$
$$\mathsf{elementProp}(ctg_7, \mathsf{Coursetemplategroup}, compulsory).$$
$$\mathsf{elementProp}(c_{10}, \mathsf{Coursetemplate}, basic\ course).$$
$$\mathsf{elementProp}(c_{15}, \mathsf{Coursetemplate}, proseminar).$$
$$\mathsf{elementProp}(c_{20}, \mathsf{Coursetemplate}, lecture).$$

zusammen mit den zusätzlichen Eigenschaften der Coursetemplate -Elemente:

$\mathsf{ctPropAdd}(c_{10}, 5, 2).$
$\mathsf{ctPropAdd}(c_{16}, 3, 2).$
$\mathsf{ctPropAdd}(c_{20}, 7, 3).$

**Hierarchische Beziehungen im Modulkatalog** Jedes Tupel $(e_1, e_2) \in contain$, das die Eltern-Kind-Beziehung widerspiegelt, wird durch ein Fakt

$\mathsf{contain}(\mathsf{e}_1, \mathsf{e}_2).$

repräsentiert. Zusätzlich wird für das Wurzelelement $e_w$

$\mathsf{contain}(\mathsf{root}, \mathsf{e}_\mathsf{w}).$

der Pseudo-Vaterknoten root, eine im Programm noch nicht vorkommende Konstante, hinzugefügt. Zur Darstellung der Eigenschaften der Strukturelemente zusammen mit dem zugehörigen Strukturtypen werden entsprechende Informationen gejoint:

$$\mathsf{containProp}(\mathsf{ParentId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}) \leftarrow \mathsf{contain}(\mathsf{ParentId}, \mathsf{Id}), \mathsf{elementProp}(\mathsf{Id}, \mathsf{Type}, \mathsf{Sort}).$$

**Beispiel 6.1.9 (Hierarchische Beziehungen im Modulkatalog)** In Ergänzung zu Beispiel 6.1.8 und bezüglich des Modulkatalog-Objektgraphen aus den Abbildungen 5.27 und 5.28 sind unter anderem folgende Fakten hinzuzufügen:

Pseudo-Vater-Knoten für *mc*:

$\mathsf{contain}(\mathsf{root}, mc).$

Umsetzung der $contain$-Instanzen:

$\mathsf{contain}(mc, mg_A).$
$\mathsf{contain}(mg_A, bm_1).\ \mathsf{contain}(mg_A, bm_2).$
$\mathsf{contain}(bm_2, ctg_5).\ \mathsf{contain}(bm_2, ctg_7).$
$\mathsf{contain}(ctg_5, c_{10}).\ \mathsf{contain}(ctg_5, c_{15}).\ \mathsf{contain}(ctg_7, c_{20}).$

Wird die Regel zum Joinen relevanter Informationen

$$\mathsf{containProp}(\mathsf{ParentId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}) \leftarrow \mathsf{contain}(\mathsf{ParentId}, \mathsf{Id}), \mathsf{elementProp}(\mathsf{Id}, \mathsf{Type}, \mathsf{Sort}).$$

zusammen mit den relevanten Fakten ausgewertet, so erhält man unter anderem folgende containProp-Atome:

containProp($root$, $mc$, Modulecatalogue, $null$).
containProp($mc$, $mg_A$, Modulegroup, $null$).
containProp($mg_A$, $bm_1$, Module, $basic$).
containProp($mg_A$, $bm_2$, Module, $basic$).
...
containProp($ctg_7$, $c_{20}$, Coursetemplate, $lecture$).

**Abgeleitete Eigenschaften** Die vorherigen Fakten enthalten auch noch weitere Informationen, die hier abgeleitet werden können. Die Attributwerte und Strukturtypen werden durch Projektionsregeln bestimmt:

element(Id) ← elementProp(Id, Type, Sort).
elementType(Type) ← elementProp(Id, Type, Sort).
elementSort(Sort) ← elementProp(Id, Type, Sort).
elementTypeSort(Type, Sort) ← elementProp(Id, Type, Sort).

Zur Unterscheidung von Wurzel-, Blatt- und inneren Knoten sind weitere Prädikate hilfreich:

elementRoot(Id) ← contain(root, Id).
elementNotLeaf(Id) ← contain(Id, ChildId).
elementLeaf(Id) ← element(Id),not elementNotLeaf(Id).

**Einfache transitive Beziehungen** Je nach Blickwinkel drückt sich die Element-Nachfahren- bzw. Element-Vorfahren-Beziehung durch das Prädikat containTrans aus:[16]

containTrans(Id, ChildId) ← contain(Id, ChildId).
containTrans(Id, SuccId) ← contain(Id, ChildId),containTrans(ChildId, SuccId),
element(SuccId).

Die Verknüpfung mit den Eigenschaften des Nachfahren-Knotens geschieht wie folgt:

containTransProp(AncId, Id, Type, Sort) ← containTrans(AncId, Id),
elementProp(Id, Type, Sort).

[16] Analog zu Bemerkung 6.1.1 kann auch in diesem Anwendungsbereich die Stratifizierung der Prädikate vorgenommen werden. Wie man leicht sehen kann, befindet sich sowohl das Prädikat contain als auch das Prädikat element in einem echt niedrigeren Stratum als RcontainTrans. In nachfolgenden Regeln taucht also jede Variable in einem Prädikat eines echt niedrigeren Stratums als der jeweilige Kopf der Regel auf. Dies garantiert die Domänenbeschränktheit der Regeln. Daher ist die Rekursion im Hinblick auf Terminierung des ASP-Programms unproblematisch.

**Transitive Beziehungen für Elemente mit zusätzlichen Eigenschaften** Wie einleitend erwähnt, sollen der Einfachheit halber weitere Eigenschaften nur bei den Veranstaltungstemplates auftreten. Hierzu wurde schon das Prädikat ctPropAdd eingeführt. Diesbezügliche gejointe Informationen werden mit Hilfe eines zusätzlichen Prädikats, welches kurz ctProp bezeichnet werden soll, dargestellt:

$$\mathsf{ctProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}, \mathsf{Ects}, \mathsf{Hpw}) \leftarrow \mathsf{containTransProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}), \\ \mathsf{ctPropAdd}(\mathsf{Id}, \mathsf{Ects}, \mathsf{Hpw}).$$

**Voraussetzungen** Ist eine Assoziation *require* gegeben, so wird diese direkt transformiert. Jedes Tupel $(e_1, e_2) \in \mathit{require}$ erscheint im Programm als einfaches Fakt:

$$\mathsf{require}(\mathsf{e}_1, \mathsf{e}_2).$$

**Beispiel 6.1.10 (Voraussetzungen)** Die im Modulkatalog-Objektgraphen aus den Abbildungen 5.27 und 5.28 auftretenden *require*-Instanzen können durch die Fakten

$$\mathsf{require}(bm_{10}, bm_2).$$
$$\mathsf{require}(c_{45}, c_{20}).$$
$$\mathsf{require}(c_{45}, c_{40}).$$

repräsentiert werden.

**Konkrete Lehrveranstaltungen** Wie schon im Abschnitt 5.3.4 des Modellierungskapitels erwähnt, können die konkreten Lehrveranstaltungen mit den bekannten bzw. den im nächsten Abschnitt gezeigten Mitteln und Methoden einfach realisiert werden.

#### 6.1.2.3 Constraintmodell: Bedingungen

Aufbauend auf den Gegebenheiten des Strukturmodells bzw. der entsprechenden Prädikate und Atome im vorherigen Abschnitt 6.1.2.2 werden nun die Bedingungen repräsentiert. Bevor auf die Realisierung der elementaren und komplexen Constraints eingegangen wird, sind einige Grundforderungen (vgl. auch Abschnitt 5.3.3) an ein gültiges Studium zu repräsentieren.

**Gewählte Elemente** In diesem Anwendungsbereich sind genau die Strukturobjekte die wählbaren Elemente. In einem Modell sollen diese durch ein Prädikat chosen vertreten sein. Befindet sich das Atom

$$\mathsf{chosen}(\mathsf{e})$$

in einem Modell, so bedeutet dies, dass das Element e gewählt ist, also Bestandteil eines gültigen Studiums ist. Da im allgemeinen Fall der Modulkatalog-Objektgraph kein Baum ist, ist es ferner wichtig zu wissen, welcher Elternknoten für einen Knoten der gerade relevante sein soll. In gewisser Weise sind daher auch Kanten im Objektgraphen des Modulkatalogs zu wählen. Hierzu ist das Prädikat chosenContain dienlich. Ein Fakt

$$\mathsf{chosenContain}(\mathsf{e}_1, \mathsf{e}_2)$$

in einem Modell soll ausdrücken, dass zum Knoten $\mathsf{e}_2$ der Elternknoten $\mathsf{e}_1$ relevant sein soll. Unter welchen Voraussetzungen derartige Fakten erzeugt werden, wird in nachfolgenden Ausführungen deutlich.

**Bemerkung 6.1.11 (Gewähltes und Folgerungen)** Wie in Abschnitt 5.3.3 erläutert, entspricht einem gültigen Studium ein Teilbaum des Modulkatalog-Objektgraphen, dessen Pfade von der Wurzel zu den Blättern des Modulkatalog-Objektgraphen geht. Um ein gültiges Studium im Modell eindeutig zu repräsentieren, sind neben den gewählten Strukturelementen des Baums auch die zugehörigen Kanten zu wählen, um die jeweilige Eindeutigkeit des Elternknotens zu gewährleisten. Man spricht in diesem Zusammenhang von „gewählten Kanten“. Im Modell wird dies mit Hilfe der Prädikate chosen und chosenContain ausgedrückt (vgl. Abbildung 6.2). Während der Modellberechnung ist es auch möglich, dass neben echten Kanten im Modulkatalog-Objektgraphen auch Verbindungen zwischen Elementen und Nachfahren-Elementen gewählt werden. In diesem Fall soll von „gewählten Hyperkanten“ (Prädikat chosenContainTrans) gesprochen werden. In Abbildung 6.2 ist dies für die Elemente $e_1$ und den Nachfahren $e_3$ der Fall. Zu beachten ist hierbei, dass hier noch keine Entscheidung darüber getroffen wurde, welcher Elternknoten von $e_3$ als gewählt gelten soll. Damit die

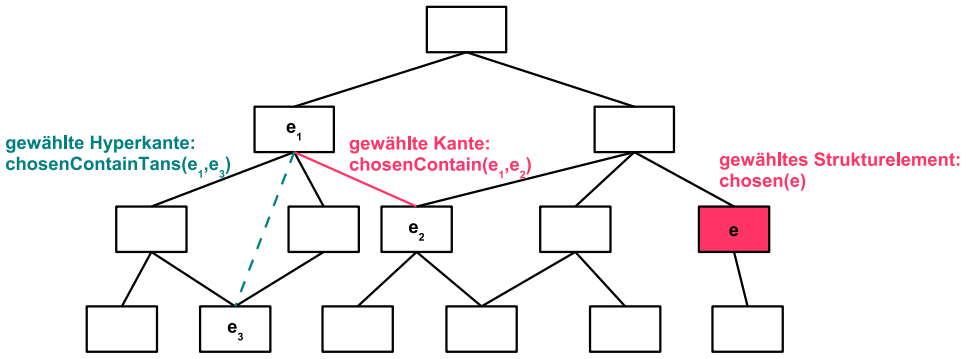



Abbildung 6.2: Gewählte Strukturelemente, Kanten und Hyperkanten

Auswahl von Strukturelementen, Kanten und Hyperkanten konsistent verläuft und den Grundanforderungen eines gültigen Studiums entspricht, sind folgende Aspekte zu beachten:

1. Die Endknoten gewählter Kanten und Hyperkanten sollen als gewählt gelten.

2. Eine gewählte Kante ist auch eine gewählte Hyperkante.
3. Das Aneinanderhängen von gewählten Hyperkanten liefert wieder eine gewählte Hyperkante.
4. Eine gewählte Hyperkante, die auch eine Kante ist, ist eine gewählte Kante.

Abbildung 6.3 deutet diese Sachverhalte an:

1. Die Elemente $e_1$, $e_2$, $e_3$ und $e_4$ sind als Endknoten gewählter Kanten bzw. Hyperkanten ebenso zu wählen.
2. Die gewählte Kante zwischen $e_1$ und $e_3$ soll auch als Hyperkante gewählt sein.
3. Die gewählten Hyperkanten zwischen $e_1$ und $e_2$ bzw. $e_2$ und $e_4$ liefern eine gewählte Hyperkante zwischen $e_1$ und $e_4$.
4. Die gewählte Hyperkante zwischen $e_1$ und $e_2$ ist auch als Kante zu wählen.

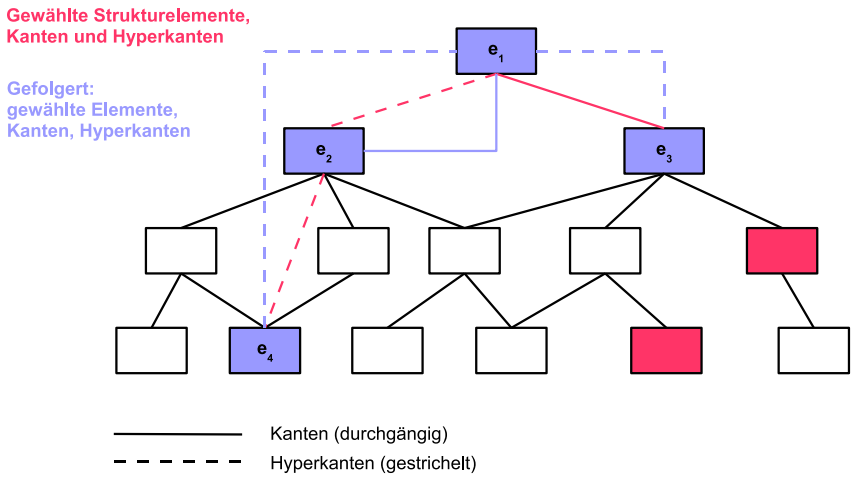



Abbildung 6.3: Gefolgerte Wahlen

**Gefolgerte Wahlen** In Anlehnung an die Ausführungen von Bemerkung 6.1.11 werden folgende Regeln aufgestellt:[17]

[17] Auch bei diesen Regeln sind keine Terminierungsprobleme aufgrund der Domänenbeschränktheit zu befürchten. Eine Untersuchung der Stratifizierung der Prädikate zeigt, dass nachfolgende Regeln vermöge der Domänenprädikate contain und containTrans „sicher“ sind.

Die Endknoten gewählter Kanten und Hyperkanten sollen als gewählt gelten:

$$\begin{aligned}
&\mathsf{chosen}(\mathsf{Id}) \leftarrow \mathsf{chosenContain}(\mathsf{Id}, \mathsf{ChildId}),\mathsf{contain}(\mathsf{Id}, \mathsf{ChildId}).\\
&\mathsf{chosen}(\mathsf{Id}) \leftarrow \mathsf{chosenContain}(\mathsf{ParentId}, \mathsf{Id}),\mathsf{contain}(\mathsf{ParentId}, \mathsf{Id}).\\
&\mathsf{chosen}(\mathsf{Id}) \leftarrow \mathsf{chosenContainTrans}(\mathsf{Id}, \mathsf{SuccId}),\mathsf{containTrans}(\mathsf{Id}, \mathsf{SuccId}).\\
&\mathsf{chosen}(\mathsf{Id}) \leftarrow \mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}),\mathsf{containTrans}(\mathsf{AncId}, \mathsf{Id}).
\end{aligned}$$

Eine gewählte Kante ist auch eine gewählte Hyperkante und das Aneinanderhängen von gewählten Hyperkanten liefert wieder eine gewählte Hyperkante:

$$\begin{aligned}
\mathsf{chosenContainTrans}(\mathsf{Id}, \mathsf{ChildId}) &\leftarrow \mathsf{chosenContain}(\mathsf{Id}, \mathsf{ChildId}),\mathsf{contain}(\mathsf{Id}, \mathsf{ChildId}).\\
\mathsf{chosenContainTrans}(\mathsf{Id}, \mathsf{SuccId}) &\leftarrow \mathsf{chosenContain}(\mathsf{Id}, \mathsf{ChildId}),\\
&\quad\ \mathsf{chosenContainTrans}(\mathsf{ChildId}, \mathsf{SuccId}),\\
&\quad\ \mathsf{contain}(\mathsf{Id}, \mathsf{ChildId}),\mathsf{containTrans}(\mathsf{ChildId}, \mathsf{SuccId}).
\end{aligned}$$

Auf der anderen Seite ist eine gewählte Hyperkante, die auch eine Kante ist, eine gewählte Kante:

$$\mathsf{chosenContain}(\mathsf{Id}, \mathsf{ChildId}) \leftarrow \mathsf{chosenContainTrans}(\mathsf{Id}, \mathsf{ChildId}),\mathsf{contain}(\mathsf{Id}, \mathsf{ChildId}).$$

**Grundforderungen an ein gültiges Studium** Hier sei besonders an die Forderungen im Lösungsmodell 5.3.3 erinnert.

In einem gültigen Studium müssen *ganze Pfade* des Strukturgraphen *von der Wurzel zur Blattebene* vertreten sein. Die Bedingung hierzu ist, dass es zu jedem gewählten Element *mindestens einen gewählten Elternknoten* und *einen gewählten Kindknoten* gibt, sofern dies möglich ist. Da ferner gefordert ist, dass ein gültiges Studium einen *Teilbaum* des Strukturgraphen darstellt, darf ein Knoten auch höchstens einen gewählten Elternknoten besitzen.

Die Umsetzung der Auswahl der Eltern- bzw. Kindknoten erfolgt implizit durch das Wählen entsprechender Kanten zu den Eltern- bzw. Kindknoten. Abbildung 6.4 verdeutlicht dies.

Hierzu folgende Regeln: Zu jedem Nicht-Wurzelelement ist *genau ein* Vater bzw. eine Kante zu einem Vater zu wählen:

$$\begin{aligned}
1\{\mathsf{chosenContain}(\mathsf{ParentId}, \mathsf{Id}) : \mathsf{contain}(\mathsf{ParentId}, \mathsf{Id})\}1 \leftarrow\ & \mathsf{chosen}(\mathsf{Id}), \mathsf{element}(\mathsf{Id}),\\
& \mathsf{not}\ \mathsf{elementRoot}(\mathsf{Id}).
\end{aligned}$$

Zu jedem Nicht-Blattelement ist mindestens ein Kindknoten bzw. eine Kante zu einem Kindknoten zu wählen:

$$\begin{aligned}
1\{\mathsf{chosenContain}(\mathsf{Id}, \mathsf{ChildId}) : \mathsf{contain}(\mathsf{Id}, \mathsf{ChildId})\} \leftarrow\ & \mathsf{chosen}(\mathsf{Id}), \mathsf{element}(\mathsf{Id}),\\
& \mathsf{not}\ \mathsf{elementLeaf}(\mathsf{Id}).
\end{aligned}$$

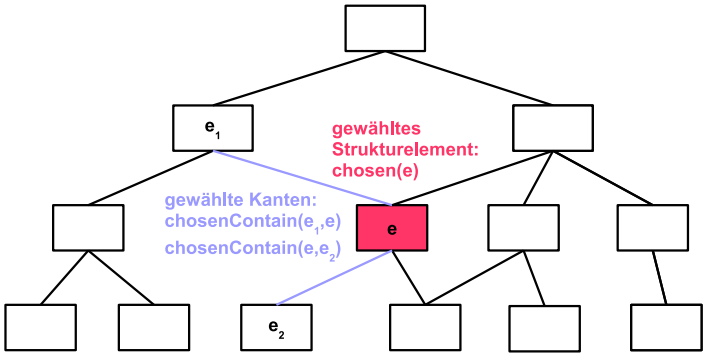



Abbildung 6.4: Gewählte „Eltern und Kinder“

Zur Umsetzung der Forderung „*Teilbaum*“ ist noch eine weitere Regel erforderlich. In Abbildung 6.5 ist die Problematik durch entsprechende Markierungen skizziert: Obwohl bis auf die Wurzel jeder gewählte Knoten genau einen gewählten Elternknoten besitzt und jedes gewählte innere Element mindestens einen gewählten Kindknoten besitzt, so ist doch eine nicht gewünschte Wahl erfolgt: Der Knoten e besitzt zwei verschiedene gewählte Vorfahren, nämlich $e_{anc,path1}$ und $e_{anc,path2}$, welche sich *nicht auf derselben Vorfahrenlinie* befinden.

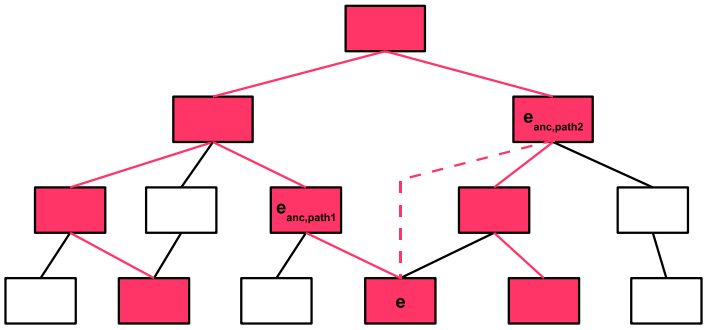



Abbildung 6.5: Problem falsch gewählter Vorfahren

Mit Hilfe einer Integritätsregel werden derartige Wahlen vermieden:[18]

$$\begin{aligned}\bot \leftarrow\ & \mathsf{chosenContainTrans}(\mathsf{AncId1}, \mathsf{Id}), \mathsf{chosenContainTrans}(\mathsf{AncId2}, \mathsf{Id}),\\ & \mathsf{containTrans}(\mathsf{AncId1}, \mathsf{Id}), \mathsf{containTrans}(\mathsf{AncId2}, \mathsf{Id}),\\ & \mathsf{not\ containTrans}(\mathsf{AncId1}, \mathsf{AncId2}), \mathsf{not\ containTrans}(\mathsf{AncId2}, \mathsf{AncId1}),\\ & \mathsf{not\ eq}(\mathsf{AncId1}, \mathsf{AncId2}).\end{aligned}$$

[18] Erinnerung: Built-in-Hilfsfunktion eq für die Prüfung der Gleichheit!

Eine informelle Formulierung dieser Integritätsbedingung wäre etwa: Sind die Hyperkanten von den verschiedenen Elementen $\mathsf{e}_{\mathsf{anc1}}$ bzw. $\mathsf{e}_{\mathsf{anc2}}$ zu einem gemeinsamen Nachfahren-Element e gewählt, so ist es nicht möglich, dass weder $\mathsf{e}_{\mathsf{anc1}}$ ein Vorfahre von $\mathsf{e}_{\mathsf{anc2}}$ noch $\mathsf{e}_{\mathsf{anc2}}$ ein Vorfahre von $\mathsf{e}_{\mathsf{anc1}}$ ist.

**Berücksichtigung der Voraussetzungen** Ist eine *require*-Assoziation gegeben, so muss gewährleistet sein, dass ein zu einem gewählten Element vorausgesetztes Element ebenso im Modell enthalten ist. Hierzu folgende Regel:

$$\mathsf{chosen(Id2)} \leftarrow \mathsf{chosen(Id1),element(Id1),element(Id2),require(Id1,Id2)}.$$

**Hilfsprädikat** Wie im Anwendungsbereich Generierung von Tests soll hier gegebenenfalls ein Hilfsprädikat number Verwendung finden, um einen endlichen Wertebereich für zahlenwertige Variablen vorzugeben. Durch

$$\mathsf{number(0..N)}.$$

sind die $N + 1$ Fakten „number(i).“ $(0 \leq i \leq N)$ definiert. Es wird davon ausgegangen, dass $N$ geeignet (groß) gewählt wird.

**Elementare Anzahlconstraints mit expliziter Spezifikation** Wie üblich soll auch hier $\mathcal{S}$ wieder die Menge aller wählbaren Elemente, also aller Strukturelemente bezeichnen. Bezüglich der Anzahlconstraints soll als Erstes der Fall expliziter Spezifikation mit numerischen Grenzen betrachtet werden. Es handelt sich also um die Anforderung

$$\mathit{cardinalityCons}(\{e_i \mid 1 \leq i \leq m, m \in \mathbb{N}\}, n_{min}, n_{max})$$

wobei $e_i \in \mathcal{S}, 1 \leq i \leq m, m \in \mathbb{N}$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$.

Die Idee ist dabei, die extensionale Constraintreferenzmenge aufzuzählen und die Anforderung an die Anzahl der zu wählenden Elemente in einem Atom eines zusätzlichen Prädikats zu spezifizieren.

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $e_1, \ldots, e_n \in \mathcal{S}$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$

TEMPLATE: Spezifikation:

$$\mathsf{cardConsExpl(id, n_{min}, n_{max})}.$$
$$\mathsf{cardConsExplBasicset(id, e_1)}.$$
$$\ldots$$
$$\mathsf{cardConsExplBasicset(id, e_m)}.$$

Hierbei stellt id eine zu vergebende Constraint-Id dar. Die Umsetzung geschieht wie folgt:

$$\mathsf{N_{min}}\{\mathsf{chosen(Id)} : \mathsf{cardConsExplBasicset(Cid, Id)}\}\mathsf{N_{max}} \leftarrow \mathsf{cardConsExpl(Cid, N_{min}, N_{max})}, \mathsf{not\ eq(N_{max}, \infty)}.$$

$$\mathsf{N_{min}}\{\mathsf{chosen(Id)} : \mathsf{cardConsExplBasicset(Cid, Id)}\} \leftarrow \mathsf{cardConsExpl(Cid, N_{min}, N_{max})}, \mathsf{eq(N_{max}, \infty)}.$$

Sind alle explizit spezifizierten Elemente zu wählen, also

$$\mathit{cardinalityCons}(\{e_1, \ldots, e_m\}, \text{all}),$$

wobei $e_i \in \mathcal{S}, 1 \leq i \leq m, m \in \mathbb{N}$, so wird dies wie folgt transformiert:

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $e_1, \ldots, e_n \in \mathcal{S}$

TEMPLATE: Spezifikation:

$\mathsf{cardConsExpl(id, all)}.$
$\mathsf{cardConsExplBasicset(id, e_1)}.$
...
$\mathsf{cardConsExplBasicset(id, e_m)}.$

Kernregeln der Umsetzung – jedes Element der Constraintreferenzmenge ist zu wählen:

$$\mathsf{chosen(Id)} \leftarrow \mathsf{cardConsExpl(Cid, all)}, \mathsf{cardConsExplBasicset(Cid, Id)}, \mathsf{element(Id)}.$$

**Bemerkung 6.1.12 (Implizit spezifizierte Constraintreferenzmengen)** Typischerweise treten bei impliziten Spezifikationen von Constraintreferenzmengen der Strukturtyp, ein möglicher Wert des Attributs `Sort` und der Vorfahre der zu wählenden Elemente auf (vgl. Abschnitt 5.3.2.1 und Definition 5.3.18):

$$\mathcal{S}_{\mathit{Type}=\mathsf{T},\mathtt{Sort}=\mathit{sort},\mathit{anc}=e_{\mathit{anc}}}, \quad \text{Abkürzung: } \mathcal{S}_{\mathsf{T},\mathit{sort},e_{\mathit{anc}}}$$

**Elementare Anzahlconstraints bei impliziter Spezifikation mit numerischen Grenzen** Wie eben schon in Bemerkung 6.1.12 erwähnt, sind typische implizit spezifizierte Constraintreferenzmengen von der Gestalt:

$$\mathcal{S}_{\mathit{Type}=\mathsf{T},\mathtt{Sort}=\mathit{sort},\mathit{anc}=e_{\mathit{anc}}}$$

Die Anforderung

$$cardinalityCons(\mathcal{S}_{Type=\mathsf{type},\texttt{Sort}=\mathsf{sort},anc=\mathsf{e}_{anc}}, n_{min}, n_{max})$$

wobei $e_{anc} \in \mathcal{S}$, $\mathsf{type} \in Structuretype^{cl}$ und $\mathsf{sort} \in domain(\texttt{Sort})$[19] und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ gilt, wird wie folgt transformiert.

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $e_{anc} \in \mathcal{S}$, $\mathsf{type} \in Structuretype^{cl}$, $sort \in domain(\texttt{Sort})$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$

TEMPLATE: Spezifikation:[20]

$$\mathsf{cardConsImpl}(\mathsf{e_{anc}}, \mathsf{type}, sort, \mathsf{n_{min}}, \mathsf{n_{max}}).$$

Umsetzungsregeln für den Fall, dass alle drei Parameterwerte der Constraintreferenzmenge mit „echten Werten“ – ungleich * bzw. *x* für einen beliebigen Wert – spezifiziert sind und für $n_{max} \neq \infty$ bzw. $n_{max} = \infty$:

$$\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort})\}\mathsf{N_{max}} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{AncId}, \mathsf{Type}, \mathsf{Sort}, \mathsf{N_{min}}, \mathsf{N_{max}}),$$
$$\mathsf{number}(\mathsf{N_{min}}), \mathsf{number}(\mathsf{N_{max}}), \mathsf{elementTypeSort}(\mathsf{Type}, \mathsf{Sort}), \mathsf{not\ eq}(\mathsf{Sort}, x), \mathsf{not\ eq}(\mathsf{N_{max}}, \infty).$$

$$\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort})\} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{AncId}, \mathsf{Type}, \mathsf{Sort}, \mathsf{N_{min}}, \mathsf{N_{max}}),$$
$$\mathsf{number}(\mathsf{N_{min}}), \mathsf{elementTypeSort}(\mathsf{Type}, \mathsf{Sort}), \mathsf{not\ eq}(\mathsf{Sort}, x), \mathsf{eq}(\mathsf{N_{max}}, \infty).$$

Für den Fall, dass der Attributwert des Attributs `Sort` beliebig, also *x* ist, werden weitere Regeln benötigt:

$$\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort})\}\mathsf{N_{max}} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{AncId}, \mathsf{Type}, x, \mathsf{N_{min}}, \mathsf{N_{max}}),$$
$$\mathsf{number}(\mathsf{N_{min}}), \mathsf{number}(\mathsf{N_{max}}), \mathsf{elementType}(\mathsf{Type}), \mathsf{not\ eq}(\mathsf{N_{max}}, \infty).$$

$$\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort})\} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{AncId}, \mathsf{Type}, x, \mathsf{N_{min}}, \mathsf{N_{max}}),$$
$$\mathsf{number}(\mathsf{N_{min}}), \mathsf{elementType}(\mathsf{Type}), \mathsf{eq}(\mathsf{N_{max}}, \infty).$$

In Abbildung 6.6 werden die gegebenen Bestandteile der Spezifikation veranschaulicht.

[19] $domain(\texttt{Sort})$ meint den Wertebereich des Attributs `Sort`, das zur Klasse von *type* gehört; man könnte also genauer `type-Sort` schreiben. Dies geht aus dem Zusammenhang stets hervor, ist hier also nicht notwendig.

[20] Hier ist kein Constraint-Identifikator angegeben, wäre aber ebenso möglich.

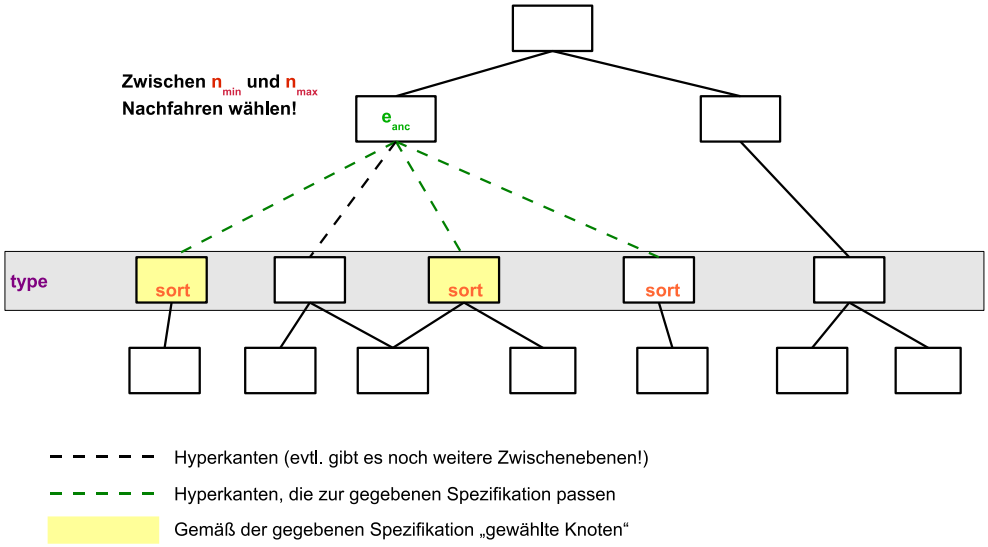



Abbildung 6.6: Anzahlconstraints mit impliziter Spezifikation

**Bemerkung 6.1.13 (Fallunterscheidungen und Ähnlichkeit von Regeln)** Wie man hier sieht, liefern die Fallunterscheidungen $n_{max} \neq \infty$, $n_{max} = \infty$, *sort* $\neq x$, *sort* $= x$ vier ähnliche Regeln. Im Fall endlicher Werte ist die Variable $N_{max}$ im Anzahlconstraint-Literal enthalten, ansonsten darf sie dort nicht erscheinen. Ist *sort* ein echter Wert, so ist er im Anzahlconstraint-Literal zu binden. Im Falle des beliebigen Wertes *x* muss die Variable Sort im Anzahlconstraint-Literal eine lokale, freie Variable sein, um sämtliche Grundierungmöglichkeiten offen zu halten. Daher sind diese Fallunterscheidungen notwendig.

Auch in weiteren zu betrachtenden Constraints sind oftmals diese Variationen der Regel in analoger Weise vorzunehmen. Dies soll für die weiteren Ausführungen jedoch aus Gründen der Einfachheit, Klarheit und Übersichtlichkeit meist unterlassen bleiben. Auf diesen Sachverhalt soll auch nicht stets hingewiesen werden.

**Bemerkung 6.1.14 (Auswertung der Regeln und Sideways Information Passing)** Obige Regeln sollen im Hinblick auf Aufbau und Auswertung im Folgenden näher analysiert werden. Vergleiche hierzu auch Abschnitt 4.2.2.4 und insbesondere Beispiel 4.2.57.

Es sei eine konkrete Spezifikation

$$\mathsf{cardConsImpl}(\mathsf{e_{anc}}, \mathsf{type}, \mathit{sort}, \mathsf{n_{min}}, \mathsf{n_{max}}).$$

gegeben. Hierbei soll beispielsweise $\mathit{sort} \neq \mathsf{x}$ und $\mathsf{n_{max}} \neq \infty$ gelten. Damit ist die erste der gezeigten

Regeln relevant:

$$\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId},\mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId},\mathsf{Id},\mathsf{Type},\mathsf{Sort})\}\mathsf{N_{max}} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{AncId},\mathsf{Type},\mathsf{Sort},\mathsf{N_{min}},\mathsf{N_{max}}),$$
$$\mathsf{number}(\mathsf{N_{min}}),\mathsf{number}(\mathsf{N_{max}}),\mathsf{elementTypeSort}(\mathsf{Type},\mathsf{Sort}),\mathsf{not\ eq}(\mathsf{Sort},\mathsf{x}),\mathsf{not\ eq}(\mathsf{N_{max}},\infty).$$

Die Bestandteile der Regel lassen sich in verschiedene Gruppen einteilen:

1. *Literale zur einfachen Fallunterscheidung:* Die Literale $\mathsf{not\ eq}(\mathsf{Sort},\mathsf{x})$ und $\mathsf{not\ eq}(\mathsf{N_{max}},\infty)$ dienen zur Angabe des betrachteten Falls $sort \neq \mathsf{x}$ und $\mathsf{n_{max}} \neq \infty$.

2. *Prädikate/Literale zur Sicherung der Domänenbeschränktheit:* Um die Regel sicher zu machen, werden die Literale $\mathsf{number}(\mathsf{N_{min}})$, $\mathsf{number}(\mathsf{N_{max}})$ und $\mathsf{elementTypeSort}(\mathsf{Type},\mathsf{Sort})$ verwendet.

3. *Literal als kontextbestimmendes Constraint:* Das Hauptliteral zur Weitergabe der konkreten Werte (Kontext) ist: $\mathsf{cardConsImpl}(\mathsf{AncId},\mathsf{Type},\mathsf{Sort},\mathsf{N_{min}},\mathsf{N_{max}})$.

4. *Literal als kontextkonsumierendes Constraint:* An die Variablen im Anzahlconstraint-Literal $\mathsf{N_{min}}\{\mathsf{chosenContainTrans}(\mathsf{AncId},\mathsf{Id}) : \mathsf{containTransProp}(\mathsf{AncId},\mathsf{Id},\mathsf{Type},\mathsf{Sort})\}\mathsf{N_{max}}$ werden vermöge der Variablenbindungen die Kontextinformationen des kontextbestimmenden Constraints weitergegeben.

Durch die Angabe des Spezifikations-Atoms werden die Variablen des kontextbestimmenden Constraints der Regel mit den entsprechenden Werten belegt. Durch die Variablenbindungen und das stattfindende Sideways Information Passing – die verwendeten Farben deuten dies an – erhält man daher im Prinzip folgende Form:[21]

$$\mathsf{n_{min}}\{\mathsf{chosenContainTrans}(\mathsf{e_{anc}},\mathsf{Id}) : \mathsf{containTransProp}(\mathsf{e_{anc}},\mathsf{Id},\mathsf{type},\mathsf{sort})\}\mathsf{n_{max}} \leftarrow$$
$$\mathsf{cardConsImpl}(\mathsf{e_{anc}},\mathsf{type},\mathsf{sort},\mathsf{n_{min}},\mathsf{n_{max}}),$$
$$\mathsf{number}(\mathsf{n_{min}}),\mathsf{number}(\mathsf{n_{max}}),\mathsf{elementTypeSort}(\mathsf{type},\mathsf{sort}),\mathsf{not\ eq}(\mathsf{sort},\mathsf{x}),\mathsf{not\ eq}(\mathsf{n_{max}},\infty).$$

Dies liefert im Wesentlichen die Regel der „grundierten Version“:

$$\mathsf{n_{min}}\{\mathsf{chosenContainTrans}(\mathsf{e_{anc}},\mathsf{Id}) : \mathsf{containTransProp}(\mathsf{e_{anc}},\mathsf{Id},\mathsf{type},\mathsf{sort})\}\mathsf{n_{max}} \leftarrow$$

Durch diese Auswahlregel – verbunden mit einem Bedingungsliteral – wird im Prinzip gewährleistet, dass aus der Menge der Hyperkanten, die von $\mathsf{e_{anc}}$ ausgehen und in ein Element des Typs $\mathsf{type}$ und der Art $\mathsf{sort}$ gehen, genau zwischen $n_{min}$ und $n_{max}$ Hyperkanten gewählt werden.

[21] Zu beachten ist, dass hierbei nicht nur „first-order Kontexte“, sondern „higher-order Kontexte“ (Anzahlen!) weitergegeben werden. Die entsprechenden Variablen sind im Anzahlconstraint-Literal vorgesehen.

**Elementare Anzahlconstraints bei impliziter Spezifikation und dem Schlüsselwort „all“** Schließlich sei noch der Fall impliziter Spezifikation unter Verwendung des Schlüsselworts „all“ betrachtet:

$$cardinalityCons(\mathcal{S}_{Type=\mathsf{type},\texttt{Sort}=sort,anc=e_{anc}}, \text{all}),$$

wobei $e_{anc} \in \mathcal{S}$, $\mathsf{type} \in Structuretype^{cl}$ und $sort \in domain(\texttt{Sort})$

CONSTRAINTART: Anzahlconstraint *cardinalityCons*

PARAMETER: $e_{anc} \in \mathcal{S}$, $\mathsf{type} \in Structuretype^{cl}$, $sort \in domain(\texttt{Sort})$

TEMPLATE: Spezifikation:

cardConsImpl($\mathsf{e_{anc}}$, type, *sort*, all).

Umsetzung:

chosenContainTrans(AncId, Id) ← containTransProp(AncId, Id, Type, Sort)
cardConsImpl(AncId, Type, Sort, all).

**Local-Successorconstraints mit implizitem Kontext und numerischen Anzahlforderungen** Local-Successorconstraints mit implizitem Kontext können nicht für sich alleine stehen, da sich die relevante Kontextmenge aus der „Auswertung“ eines vorangehenden Constraints ergibt. Gegeben ist also ein Constraint der Art

$$\mathfrak{C}(\mathcal{S}_{ancType,ancSort,e_{root}})^{local}\, cardinalityCons(\mathcal{S}_{type,\texttt{sort}}, n_{min}, n_{max}),$$

wobei $\mathfrak{C} \in \{cardinalityCons, weightsumCons, containCons\}$, $e_{root} \in \mathcal{S}$, $ancType, type \in Structuretype^{cl}$, $ancSort \in domain(\texttt{anctype-Sort})$, $sort \in domain(\texttt{type-Sort})$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$. Hierbei stellt $e_{root}$ den Vorfahrenknoten des vorherigen Constraints dar und ist damit der Wurzelknoten des Teilgraphen des Modulkatalogs, auf den sich das nachfolgende Constraint bezieht. Die Abbildung 6.7 veranschaulicht die nichtnumerischen Spezifikationsparameter.

CONSTRAINTART: Anzahl-local-Successorconstraint mit implizitem Kontext $^{local}\, cardinalityCons$

PARAMETER: $e_{root} \in \mathcal{S}$, $\mathsf{ancType} \in Structuretype^{cl}$, $ancSort \in domain(\texttt{anctype-Sort})$, $\mathsf{type} \in Structuretype^{cl}$, $sort \in domain(\texttt{type-Sort})$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$

TEMPLATE: Spezifikation:

cardConsLocal($\mathsf{e_{root}}$, ancType, *ancSort*, type, *sort*, $\mathsf{n_{min}}$, $\mathsf{n_{max}}$).

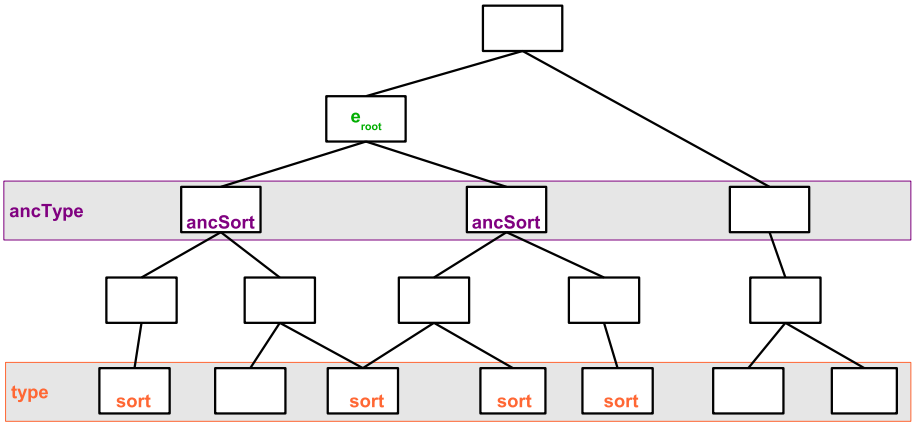



Abbildung 6.7: Successorconstraint mit implizitem Kontext: Veranschaulichung der nichtnumerischen Spezifikationsparameter

Umsetzung der Anforderung für $n_{max} \neq \infty$ und *ancsort*,*sort*$\neq$ *x*:

$N_{min}\{\text{chosenContainTrans}(\text{AncId}, \text{Id}) : \text{containTransProp}(\text{AncId}, \text{Id}, \text{Type}, \text{Sort})\}N_{max} \leftarrow$
$\text{cardConsLocal}(\text{RootId}, \text{AncType}, \text{AncSort}, \text{Type}, \text{Sort}, N_{min}, N_{max}),$
$\text{chosenContainTrans}(\text{RootId}, \text{AncId}), \text{containTransProp}(\text{RootId}, \text{AncId}, \text{AncType}, \text{AncSort}),$
$\text{number}(N_{min}), \text{number}(N_{max}), \text{element}(\text{RootId}), \text{not eq}(\text{AncSort}, x), \text{not eq}(\text{Sort}, x),$
$\text{not eq}(N_{max}, \infty).$

**Bemerkung 6.1.15 (Auswertung der local-Successorconstraint-Regel)** In Anlehnung an Bemerkung 6.1.14 sollen auch im Fall der local-Successorconstraints mit implizitem Kontext die entscheidenden Variablenbindungen näher untersucht werden.

Geht man vom konkreten Spezifikations-Atom

$\text{cardConsLocal}(e_{root}, \text{ancType}, \textit{ancSort}, \text{type}, \textit{sort}, n_{min}, n_{max})$

mit $n_{max} \neq \infty$ und *ancsort*,*sort*$\neq$ *x* aus, so hat die Regel zur Behandlung des Constraints im Wesentlichen folgende Gestalt:[22]

$N_{min}\{\text{chosenContainTrans}(\text{AncId}, \text{Id}) : \text{containTransProp}(\text{AncId}, \text{Id}, \text{Type}, \text{Sort})\}N_{max} \leftarrow$
$\text{cardConsLocal}(\text{RootId}, \text{AncType}, \text{AncSort}, \text{Type}, \text{Sort}, N_{min}, N_{max}),$
$\text{chosenContainTrans}(\text{RootId}, \text{AncId}), \text{containTransProp}(\text{RootId}, \text{AncId}, \text{AncType}, \text{AncSort}).$

[22] Die Literale zur einfachen Fallunterscheidung und die Literale zur Sicherung der Domänenbeschränktheit werden hier der Einfachheit halber nicht notiert.

Das Spezifikationsatom stellt das kontextbestimmende Constraint dar. Werden die konkreten Werte dieses Atoms in der Regel gebunden, so erhält man folgende Form:

$$\begin{aligned}&n_{min}\{chosenContainTrans(AncId, Id) : containTransProp(AncId, Id, type, sort)\}n_{max} \leftarrow\\ &\quad cardConsLocal(e_{root}, ancType, ancSort, type, sort, n_{min}, n_{max}),\\ &\quad chosenContainTrans(e_{root}, AncId), containTransProp(e_{root}, AncId, ancType, ancSort).\end{aligned}$$

bzw. nach Wegnahme des Spezifikations-Atoms:

$$\begin{aligned}&n_{min}\{chosenContainTrans(AncId, Id) : containTransProp(AncId, Id, type, sort)\}n_{max} \leftarrow\\ &\quad chosenContainTrans(e_{root}, AncId), containTransProp(e_{root}, AncId, ancType, ancSort).\end{aligned}$$

Diese Auswahl-Regel enthält noch zwei Variablen: AncId und Id. Hierbei tritt AncId als globale und Id als lokale Variable im Anzahlconstraint auf. Nun bestimmt der Rumpf den Kontext, also die möglichen Belegungen für AncId. Für *jedes* Strukturelement $e_{anc}$ , das den Rumpf der Regel erfüllt, d. h. für jedes gewählte ancType-Strukturelement mit `anctype-Sort`-Attributwert ancSort, das im $e_{root}$-Teilgraphen liegt, ist folgende Regel zu bilden:

$$\begin{aligned}&n_{min}\{chosenContainTrans(e_{anc}, Id) : containTransProp(e_{anc}, Id, type, sort)\}n_{max} \leftarrow\\ &\quad chosenContainTrans(e_{root}, e_{anc}), containTransProp(e_{root}, e_{anc}, ancType, ancSort).\end{aligned}$$

bzw. nach Entfernen der „vollständig ausgewerteten“ Atome:

$$n_{min}\{chosenContainTrans(e_{anc}, Id) : containTransProp(e_{anc}, Id, type, sort)\}n_{max} \leftarrow$$

Schließlich werden vermöge des Prädikats chosenContainTrans im Bedingungsliteral die möglichen Belegungen e für Id bestimmt. Das Anzahlconstraint-Literal in der Auswahlregel bewirkt eine Auswahl von zwischen $n_{min}$ und $n_{max}$ chosenContainTrans($e_{anc}$, e)-Atomen gewünschten Strukturtyps type und gewünschter Art sort.

**Local-Successorconstraints mit implizitem Kontext und Anzahlforderung „all“** Der Fall „Wahl aller Elemente“ also

$$\mathfrak{C}(\mathcal{S}_{ancType,ancSort,e_{root}})^{local}\, cardinalityCons(\mathcal{S}_{type,\texttt{sort}}, \text{all})$$

mit ansonsten gleichen Parametern wie oben kann entsprechend behandelt werden. Zu bemerken ist, dass hier keine Auswahlregel erforderlich ist, sondern die Umsetzung mit einer einfachen normalen Regel erfolgen kann.

CONSTRAINTART: per-Anzahl-Successorconstraint mit implizitem Kontext $^{local}\, cardinalityCons$

PARAMETER: $e_{root} \in \mathcal{S}$, ancType $\in Structuretype^{cl}$, $ancSort \in domain(\texttt{anctype-Sort})$, type $\in Structuretype^{cl}$, $sort \in domain(\texttt{type-Sort})$

TEMPLATE: Spezifikation:

cardConsLocal($e_{root}$, ancType, *ancSort*, type, *sort*, all).

Umsetzung der Anforderung für *ancsort*, *sort*$\neq x$:

chosenContainTrans(AncId, Id) ← chosenContainTrans(RootId, AncId),
containTransProp(RootId, AncId, AncType, AncSort),
containTransProp(AncId, Id, Type, Sort),
cardConsLocal(RootId, AncType, AncSort, Type, Sort, all),
element(RootId),not eq(AncSort, x),not eq(Sort, x).

**Global-Successorconstraints mit implizitem Kontext und Anzahlforderungen** Im Gegensatz zum local-Successorconstraint bezieht sich hier die zu wählende Anzahl an Elementen auf *die Gesamtheit* der entsprechenden Nachfahrenknoten von spezifizierten Knoten und *nicht* auf *jeden einzelnen* dieser Knoten.

Die Spezifikation geschieht auf die gleiche Weise wie im local-Successorconstraint, die Umsetzung hingegen muss eine andere Semantik implementieren.

CONSTRAINTART: Anzahl-global-Successorconstraint mit implizitem Kontext $^{global}\,cardinalityCons$

PARAMETER: $e_{root} \in \mathcal{S}$, ancType $\in$ $Structuretype^{cl}$, *ancSort* $\in$ $domain(\texttt{anctype-Sort})$, type $\in$ $Structuretype^{cl}$, *sort* $\in$ $domain(\texttt{type-Sort})$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^{\infty}$

TEMPLATE: Spezifikation:

cardConsGlobal($e_{root}$, ancType, *ancSort*, type, *sort*, $n_{min}$, $n_{max}$).

Umsetzung der Anforderung für $n_{max} \neq \infty$ und *ancsort*,*sort*$\neq x$:

$N_{min}${chosenContainTrans(AncId, Id):
containTransProp(RootId, AncId, AncType, AncSort):
containTransProp(AncId, Id, Type, Sort)}$N_{max}$ ←
cardConsGlobal(RootId, AncType, AncSort, Type, Sort, $N_{min}$, $N_{max}$),
chosen(RootId),element(RootId),
number($N_{min}$), number($N_{max}$),not eq(AncSort, x),not eq(Sort, x),not eq($N_{max}$, $\infty$).

zusammen mit der Integritätsconstraintregel („relevante AncId“ müssen gewählt sein!):

$\bot$ ← chosenContainTrans(AncId, Id),containTransProp(RootId, AncId, AncType, AncSort)
not chosenContainTrans(RootId, AncId)
cardConsGlobal(RootId, AncType, AncSort, Type, Sort, $N_{min}$, $N_{max}$).

Der Fall „Wahl aller Elemente“ also

$$\mathfrak{C}(\mathcal{S}_{ancType,ancSort,e_{root}})^{global}\, cardinalityCons(\mathcal{S}_{type,\mathtt{sort}}, \text{all})$$

entspricht dem analogen local-Successorconstraint und hat die gleiche ASP-Transformation.

**Bedingte Constraints mit Anzahlforderungen** Sind gewisse Strukturelemente nur dann zu wählen, wenn ein anderes Element gewählt worden ist, so kann dies mit Hilfe eines Bedingungsconstraints realisiert werden. Es sollen die Constraints der Art

$$conditionCons\left(cardinalityCons(\mathcal{S}_{type,sort,e_{anc}}, n_{min}, n_{max}), containCons(\{e_{cond}\})\right)$$

bzw. in anderer Schreibweise

$$cardinalityCons(\mathcal{S}_{type,sort,e_{anc}}, n_{min}, n_{max}) \leftarrow containCons(\{e_{cond}\}),$$

wobei $e_{cond}, e_{anc} \in \mathcal{S}$, *type* $\in Structuretype^{cl}$, *sort* $\in domain(\mathtt{Sort})$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ gilt, betrachtet werden.

CONSTRAINTART: Bedingungsconstraint mit Anzahlforderung *conditionCons* verbunden mit *cardinalityCons*

PARAMETER: $e_{anc}, e_{cond} \in \mathcal{S}$, type $\in Structuretype^{cl}$, *sort* $\in domain(\mathtt{Sort})$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$

TEMPLATE: Spezifikation:

cardConsImplCond($e_{cond}$, $e_{anc}$, type, *sort*, $n_{min}$, $n_{max}$).

Umsetzung unter Verwendung von elementaren Anzahlconstraints mit impliziter Spezifikation für $n_{max} \neq \infty$, *sort* $\neq x$:

```
cardConsImpl(AncId, Type, Sort, Nmin, Nmax) ← chosen(CondId),element(CondId),
                cardConsImplCond(CondId, AncId, Type, Sort, Nmin, Nmax),
                element(AncId),elementTypeSort(Type, Sort),
                number(Nmin), number(Nmax).
```

Für den Fall der „Wahl aller Elemente“, also :

$$cardinalityCons(\mathcal{S}_{type,sort,e_{anc}}, \text{all}) \leftarrow containCons(\{e_{cond}\})$$

mit den ansonsten gleichen Parametern wie oben, geschieht die Umsetzung analog:

CONSTRAINTART: Bedingungsconstraint mit Anzahlforderung *conditionCons* verbunden mit *cardinalityCons*

PARAMETER: $e_{anc}, e_{cond} \in \mathcal{S}$, type $\in$ *Structuretype*$^{cl}$, *sort* $\in$ *domain*(`Sort`)

TEMPLATE: Spezifikation:

cardConsImpl(e$_{\text{cond}}$, e$_{\text{anc}}$, type, *sort*, all).

Umsetzung unter Verwendung von elementaren Anzahlconstraints mit impliziter Spezifikation für *sort* $\neq$ *x*:

cardConsImplCond(AncId, Type, Sort, all) $\leftarrow$ chosen(CondId),element(CondId),
cardConsImplCond(CondId, AncId, Type, Sort, all),
element(AncId),elementTypeSort(Type, Sort).

**Auswahlconstraints** Besonders für die Möglichkeit der individuellen Gewichtung sind Auswahlconstraints dienlich. Es sollen Constraints der Gestalt

$$choiceCons(\{c_i \mid 1 \leq i \leq m, m \in \mathbb{N}\}, n_{min}, n_{max}),$$

wobei $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ und Constraints $c_i$, betrachtet werden. Exemplarisch soll die Umsetzung für implizit spezifizierte Anzahlconstraints $c_i, 1 \leq i \leq m, m \in \mathbb{N}$ gezeigt werden. Der Einfachheit halber erhalten die Constraints $c_i$ eine Art Namen, über die sie identifiziert werden. Ist dann der entsprechende Name „gewählt“, so soll auch das Constraint erfüllt werden. Nun zur Transformation:

CONSTRAINTART: Auswahlconstraints *choiceCons* bezogen auf Anzahlconstraints *cardinalityCons*

PARAMETER: Constraints $c_i$ bzw. deren „Bezeichner“ $choice_i$, $e_{anc,i} \in \mathcal{S}$, type$_i$ $\in$ *Structuretype*$^{cl}$, *sort*$_i$ $\in$ *domain*(`Sort`), $n_{min,i} \in \mathbb{N}_0$, $n_{max,i} \in \mathbb{N}_0^\infty$ $1 \leq i \leq m, m \in \mathbb{N}$, $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$

TEMPLATE: Spezifikation des Auswahlconstraints:

choiceCons(cid, n$_{\text{min}}$, n$_{\text{max}}$).
choiceConsCondition(cid, choice$_1$).
...
choiceConsCondition(cid, choice$_{\text{m}}$).

Spezifikation der auftretenden Anzahlconstraints, $1 \leq i \leq m, m \in \mathbb{N}$:

cardConsImplChoice(choice$_{\text{i}}$, e$_{\text{anc}}$, type, *sort*, n$_{\text{min,i}}$, n$_{\text{max,i}}$).

Umsetzung der Auswahl an zu erfüllenden Constraints für $n_{max} \neq \infty$:[23]

$$\mathsf{N_{min}}\{\mathsf{chosenCondition(Choice)} : \mathsf{choiceConsCondition(Cid, Choice)}\}\mathsf{N_{max}} \leftarrow \mathsf{choiceCons(Cid, N_{min}, N_{max})}.$$

Zur Umsetzung der implizit spezifizierten elementaren Anzahlconstraints kann ähnlich zur Behandlung von bedingten Constraints mit Anzahlforderung vorgegangen werden (siehe oben), auch hier wieder ohne die Werte $\infty$ und $x$:

$$\begin{aligned}\mathsf{cardConsImpl(AncId, Type, Sort, N_{min}, N_{max})} \leftarrow\ & \mathsf{chosenCondition(Choice)},\\ & \mathsf{choiceConsCondition(Cid, Choice)},\\ & \mathsf{cardConsImplChoice(Choice, AncId, Type, Sort, N_{min}, N_{max})},\\ & \mathsf{element(AncId)},\mathsf{elementTypeSort(Type, Sort)},\mathsf{number(N_{min})},\mathsf{number(N_{max})}.\end{aligned}$$

Diese Art der Transformation kann auch für den Fall „all“ und andere Constraintarten durchgeführt werden. Dies soll hier aber nicht gezeigt werden.

**Ausschlussconstraints bei expliziter Spezifikation** Der Ausschluss von explizit spezifizierten Elementen

$$excludeCons(\{e_i \in \mathcal{S} \mid 1 \leq i \leq m, m \in \mathbb{N}\})$$

kann wie folgt vorgenommen werden:

CONSTRAINTART: Ausschlussconstraints $excludeCons$

PARAMETER: Strukturelemente $e_i \in \mathcal{S}, 1 \leq i \leq m, m \in \mathbb{N}$

TEMPLATE: Spezifikation des Ausschlussconstraints: Für alle $1 \leq i \leq m$:

$$\mathsf{excludeCons(e_i)}.$$

Umsetzung:

$$\bot \leftarrow \mathsf{chosen(Id)},\mathsf{element(Id)},\mathsf{excludeCons(Id)}.$$

**Bedingte Constraints mit Ausschluss von Elementen** Gewisse Strukturelemente dürfen nicht gewählt werden, wenn andere Strukturelemente gewählt worden sind. Gelegentlich ist dies eine gegenseitige Forderung. Es sollen die Constraints der Art

$$excludeCons(\{e_1\}) \leftarrow containCons(\{e_2\}),$$

[23] Man hat ferner zu sorgen, dass choiceConsCondition ein Domänenprädikat ist.

wobei $e_1, e_2 \in \mathcal{S}$, betrachtet werden.

CONSTRAINTART: Bedingte Constraints *conditionCons* mit *containCons* und *excludeCons*

PARAMETER: Strukturelemente $e_1, e_2 \in \mathcal{S}$

TEMPLATE: Spezifikation des Ausschlussconstraints:

conditionExcludeCons($e_1$, $e_2$).

Umsetzung:

excludeCons(Id1) $\leftarrow$ chosen(Id2),element(Id2),conditionExcludeCons(Id1, Id2).

zusammen mit der Umsetzung des Ausschlussconstraints bei expliziter Spezifikation. Der gegenseitige Ausschluss lässt sich daher einfach mit der zusätzlichen Spezifikation

conditionExcludeCons($e_2$, $e_1$).

realisieren. Auch weitere Fälle sind leicht umzusetzen.

**Überlegungen zur Transformation von Gewichtssummenconstraints** Für Anforderungen an die Summe der Gewichte von Elementen

$$weightsumCons(D, n_{min}, n_{max}),$$

wobei $D \subseteq \mathcal{S}$, $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ und bei vorgegebener Gewichtsfunktion $weight$ sei ohne Einschränkung der Einfachheit halber angenommen, dass diese nur in der Form von Anforderungen an die Summe der ECTS-Punkte (Gewichtung durch die Attributwerte des Attributs `ECTS`) oder der Semesterwochenstunden (Gewichtung durch die Attributwerte des Attributs `HPW`) auftreten. Ferner soll davon ausgegangen werden, dass nur die $Coursetemplate$-Strukturelemente diese Attribute besitzen.

Die Ausführungen im Anwendungsbereich Generierung von Tests bezüglich des Umgangs mit mehreren Gewichtssummenconstraints und bei verschiedenen Fällen der Spezifikation und die im Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen geführten Diskussionen bezüglich der Transformation der verschiedenen Anzahlconstraints lassen sich hier übertragen. Daher soll das Prinzip der Umsetzung von Gewichtssummenconstraints nur exemplarisch für elementare Gewichtssummenconstraints mit expliziter Spezifikation gezeigt werden. Zuvor sind allerdings noch einige neue Prädikate einzuführen.

**Hilfsregeln für die Umsetzung von Gewichtssummenconstraints** Um den gewählten Atomen Gewichte zuordnen zu können, werden neue chosen-Prädikate benötigt. So zeigt beispielsweise das Atom

$$\mathsf{chosenEcts}(\mathsf{e_{anc}}, \mathsf{e}, \mathsf{ects}).$$

an, dass das $Coursetemplate$-Strukturelement *e* mit *ects* ECTS-Punkten und dem Vorfahren $e_{anc}$ gewählt wird. Analoges gilt für

$$\mathsf{chosenHpw}(\mathsf{e_{anc}}, \mathsf{e}, \mathsf{hpw}).$$

Die Umsetzung der entprechenden Gewichtsfunktionen erfolgt über

$$\#\mathsf{weight}\ \mathsf{chosenEcts}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Ects}) = \mathsf{Ects}.$$
$$\#\mathsf{weight}\ \mathsf{chosenHpw}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Hpw}) = \mathsf{Hpw}.$$

Ist beispielsweise das Atom $\mathsf{chosenEcts}(\mathsf{e_{anc}}, \mathsf{e}, \mathsf{ects})$ gewählt, so müssen es auch die enthaltenen Strukturelemente sein. Daher folgende Regeln:

$$\begin{aligned}\mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) \leftarrow\ & \mathsf{chosenEcts}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Ects}),\\ & \mathsf{ctProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}, \mathsf{Ects}, \mathsf{Hpw}).\\ \mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}) \leftarrow\ & \mathsf{chosenHpw}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Hpw}),\\ & \mathsf{ctProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}, \mathsf{Ects}, \mathsf{Hpw}).\end{aligned}$$

Umgekehrt müssen entsprechend gewählte Hyperkanten zu $Coursetemplate$-Strukturelementen auch zur ECTS- bzw. HPW-Summe beitragen:

$$\begin{aligned}\mathsf{chosenEcts}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Ects}) \leftarrow\ & \mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}),\\ & \mathsf{ctProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}, \mathsf{Ects}, \mathsf{Hpw}).\\ \mathsf{chosenHpw}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Hpw}) \leftarrow\ & \mathsf{chosenContainTrans}(\mathsf{AncId}, \mathsf{Id}),\\ & \mathsf{ctProp}(\mathsf{AncId}, \mathsf{Id}, \mathsf{Type}, \mathsf{Sort}, \mathsf{Ects}, \mathsf{Hpw}).\end{aligned}$$

**Elementare Gewichtssummenconstraints mit impliziter Spezifikation** Es sollen Anforderungen der Art

$$weightsumCons(\mathcal{S}_{Type=\mathsf{type},\mathtt{Sort}=sort,anc=e_{anc}}, n_{min}, n_{max}),$$

wobei $e_{anc} \in \mathcal{S}$, $\mathsf{type} \in Structuretype^{cl}$ (bzw. hier der Einfachheit halber: $\mathsf{type} = Coursetemplate$) und *sort* $\in domain(\mathtt{Sort})$ und $n_{min} \in \mathbb{N}_0, n_{max} \in \mathbb{N}_0^\infty$ gilt, betrachtet werden. Außerdem ist eine Gewichtsfunktion gegeben.

CONSTRAINTART: Gewichtssummenconstraint $weightsumCons$

PARAMETER: $e_{anc} \in \mathcal{S}$, (type $=$ *Coursetemplate* $\in$ *Structuretype*$^{cl}$,) *sort* $\in$ *domain*(Sort), $n_{min} \in \mathbb{N}_0$, $n_{max} \in \mathbb{N}_0^\infty$ und die Gewichtsfunktion $weight_{Ects}$

Liefert die Gewichtsfunktion auf den *Coursetemplate*-Strukturelementen gerade deren Ects-Punkte, $weight_{Ects} : e \mapsto e.Ects$, so ist folgende Transformation relevant:

TEMPLATE: Spezifikation:

weightConsEctsImpl($e_{anc}$, type, *sort*, $n_{min}$, $n_{max}$).

Umsetzungsregeln für $n_{max} \neq \infty$ und ohne den Platzhalter für beliebige Attributwerte *x*:[24]

$N_{min}$[chosenEcts(AncId, Id, Ects) : ctProp(AncId, Id, Type, Sort, Ects, Hpw)]$N_{max}$ $\leftarrow$
weightConsEctsImpl(AncId, Type, Sort, $N_{min}$, $N_{max}$),
number($N_{min}$), number($N_{max}$),elementTypeSort(Type, Sort),
not eq(Sort, x),not eq($N_{max}$, $\infty$).

Soll hingegen eine Gewichtssumme über die HPW-Werte gebildet werden, so ist die Gewichtsfunktion $weight_{Hpw} : e \mapsto e.Hpw$ entscheidend. Eine analoge Spezifikation wie eben kann einfach vorgenommen werden.

#### 6.1.2.4 Erweiterungen

Im Folgenden wird die Transformation der Erweiterungen aus Abschnitt 5.3.4 vorgestellt. Der Einfachheit halber wird die Modellierung nur für einen Studenten vorgenommen, so dass die Assoziationen zwischen Student und Strukturelementen ohne die Matrikelnummer modelliert werden. Die Prinzipien der Umsetzung werden trotzdem in vollem Umfang gezeigt, so dass dies keine Einschränkung der Allgemeingültigkeit des Ansatzes darstellt. Falls notwendig, kann der Identifkator für den Studierenden ohne Probleme hinzugenommen werden. Ferner soll auf die Modellierung der *requireEcts*-Assoziation verzichtet werden.

**Erweiterungen im Strukturmodell** Die Elemente der zusätzlichen Assoziationen *recommendation* zur Modellierung der Empfehlungen und zur Angabe des Turnus *ctCycle* können eins zu eins in ASP-Fakten transformiert werden:

Jedes Tupel $(e, min, max) \in recommendation$ wird durch ein Fakt

recommendation(e, min, max).

und jedes Tupel $ctCycle(e_c, cycle) \in ctCycle$ durch ein Fakt

ctCycle($e_c$, cycle).

repräsentiert.

[24] Zu beachten ist hier die Notation des Gewichtsconstraint-Literals. Dieses ist erkennbar an den eckigen Klammern im Gegensatz zu einem Anzahlconstraint-Literal, welches mit geschweiften Klammern notiert wird.

**Bestandene Strukturelemente** Die bestandenen Veranstaltungstemplates (Assoziation $passedCt$) des Studierenden werden in Form von Fakten dem Programm hinzugefügt (EDB). Ein Tupel $e \in passedCt$[25] wird durch

$$\mathsf{passed(e)}.$$

repräsentiert. Die bestandenen Elemente müssen einem gültigen Studium angehören, also „gewählt" worden sein:

$$\mathsf{chosen(Id)} \leftarrow \mathsf{passed(Id), element(Id)}.$$

Darüber hinaus müssen vorausgesetzte $Coursetemplate$-Strukturelemente[26] ebenso bestanden sein:

$$\begin{aligned}\mathsf{passedFalse(Id)} \leftarrow\ & \mathsf{passed(Id1),require(Id1, Id),not\ passed(Id),}\\ & \mathsf{elementLeaf(Id),element(Id1)}.\\ \bot \leftarrow\ & \mathsf{passedFalse(Id), elementLeaf(Id)}.\end{aligned}$$

Ein übergeordnetes Strukturelement ist bezüglich eines gültigen Studiums (Modells) bestanden, wenn zum einen die „erforderlichen", d. h. im betrachteten Modell befindlichen, Nachfahren und zum anderen die vorausgesetzten Elemente bestanden sind. Dies lässt sich in folgender Form realisieren:

$$\begin{aligned}\mathsf{passedFalse(AncId)} \leftarrow\ & \mathsf{chosenContainTrans(AncId, Id),containTrans(AncId, Id)}.\\ & \mathsf{chosen(Id),not\ passed(Id)}.\\ \mathsf{passedFalse(Id)} \leftarrow\ & \mathsf{chosen(Id),require(Id, IdReq),chosen(IdReq),not\ passed(IdReq),}\\ & \mathsf{elementNotLeaf(Id)}.\\ \mathsf{passed(Id)} \leftarrow\ & \mathsf{not\ passedFalse(Id),elementNotLeaf(Id),element(Id)}.\end{aligned}$$

**Bemerkung 6.1.16 (Bestandenes Studium)** Ist $e_{mc}$ der Wurzelknoten des Modulkatalog-Objektgraphen, so hat der Studierende das Studium bestanden, wenn im Modell des Programms das Atom $\mathsf{passed(e_{mc})}$ vertreten ist.

**Elemente mit Voraussetzungen** Die Assoziation $require$ modelliert die direkten Voraussetzungen von Strukturelementen. Hat ein innerer Knoten des Strukturgraphen eine Voraussetzung, so „erben" diese in gewisser Weise die Nachfahren. Dies wird durch die Relation $inheritedRequire$ bzw. das Prädikat inheritedRequire ausgedrückt.

$$\begin{aligned}&\mathsf{inheritedRequire(Id, R)} \leftarrow \mathsf{require(Id, R),element(Id), element(R)}.\\ &\mathsf{inheritedRequire(Id, R)} \leftarrow \mathsf{require(AncId, R),containTrans(AncId, Id)}.\end{aligned}$$

[25] Zu beachten ist, dass die Studenten-ID der Einfachheit halber hier nicht notiert wird.

[26] Erinnerung: Die $Coursetemplate$-Strukturelemente sind genau die Blätter im Modulkatalog-Objektgraphen.

Elemente, die dann auf diese Weise eine Voraussetzung besitzen, werden schließlich durch Projektion bestimmt (Relation $hasRequirement$ bzw. Prädikat hasRequirement).

$$\mathsf{hasRequirement(Id)} \leftarrow \mathsf{inheritedRequire(Id, R)}, \mathsf{element(Id)}, \mathsf{element(R)}.$$

**Erlaubte Strukturelemente** Die Assoziation $allowed$ wird wie folgt in Answer Set Programming transformiert: Strukturelemente, die keinerlei Voraussetzung besitzen und noch nicht bestanden sind, sind erlaubt:

$$\mathsf{allowed(Id)} \leftarrow \mathsf{element(Id)}, \mathsf{not\ hasRequirement(Id)}, \mathsf{not\ passed(Id)}.$$

Sie sind auch erlaubt, wenn sie keine nicht-bestandenen Voraussetzungen besitzen:

$$\begin{aligned} \mathsf{allowed(Id)} \leftarrow\ & \mathsf{element(Id)}, \mathsf{not\ hasRequirementNotPassed(Id)}, \\ & \mathsf{not\ passed(Id)}. \\ \mathsf{hasRequirementNotPassed(Id)} \leftarrow\ & \mathsf{inheritedRequire(Id, R)}, \mathsf{not\ passed(Id)}, \\ & \mathsf{element(Id)}, \mathsf{element(R)}. \end{aligned}$$

**Zeitliche Empfehlungen** Mit Hilfe entsprechender Algorithmen und Regeln können die zeitlichen Empfehlungen gemäß der Hinweise im Abschnitt 5.3.4 in Answer Set Programming realisiert werden. Dies soll hier nicht gezeigt werden.

**Konkrete Lehrveranstaltungen** Wie schon im Abschnitt 5.3.4 des Modellierungskapitels erwähnt können die konkreten Lehrveranstaltungen mit den bekannten Mitteln und Methoden einfach realisiert werden.

#### 6.1.2.5 Lösungsmodell: Gültiges Studium

Die Umsetzung der Grundanforderungen an ein gültiges Studium aus Abschnitt 5.3.3 wurde schon im Rahmen der Umsetzung des Strukturmodells (vgl. Abschnitt 6.1.2.2) gezeigt.

Die Antwortmengen stellen gerade eine Codierung der gültigen Studien bezüglich der gegebenen Spezifikation einer Studienordnung dar.

**Berechnung der Antwortmengen** Wie auch im Anwendungsbereich „Generierung von Tests“ geschieht die Berechnung aller Antwortmengen durch die Anweisung

$$\#\ \mathsf{compute}\ 0\{\}.$$

Die Berechnung einer maximalen Anzahl $n$ von Antwortmengen kann durch die Anweisung

$$\#\ \mathsf{compute}\ \mathsf{n}\{\}.$$

realisiert werden. Auch hier kann die Ausgabe auf bestimmte Atome oder Prädikate mit Hilfe der hide- und show-Deklarationen eingeschränkt werden.

**Konsistenzprüfung von Studienordnungen** Die Frage eines Studienordnungskomitees, ob der Spezifikationsvorschlag einer neuen Studienordnung konsistent ist, kann durch das Anstoßen der Berechnung mindestens einer Antwortmenge des gegebenen Programms beantwortet werden: Ergibt sich keine Antwortmenge, so sind Inkonsistenzen vorhanden. Im anderen Fall ist die Studienordnung erfüllbar.

Im Fall der Inkonsistenz kann ein Modus realisiert werden, so dass ein Teil der kritischen Constraints $c_1, \ldots, c_m, m \in \mathbb{N}$ nicht als harte Constraints, sondern in Form eines Auswahlconstraints

$$choiceCons(\{c_1, \ldots, c_m\}, n_{min}, n_{max})$$

realisiert sind. Es sind dann nur zwischen $n_{min}$ und $n_{max}$ Constraints aus der gegebenen Constraintmenge obligatorisch. Im ASP-Programm werden die Spezfikationsatome der Constraints $c_i, 1 \leq i \leq m$ also nicht durch $m$ Fakten

$$\mathsf{SPECIFICATIONATOM}(\mathsf{c_1}), \ldots, \mathsf{SPECIFICATIONATOM}(\mathsf{c_m})$$

realisiert, sondern mit Hilfe einer Auswahlregel:

$$\mathsf{n_{min}}\{\mathsf{SPECIFICATIONATOM}(\mathsf{c_1}), \ldots, \mathsf{SPECIFICATIONATOM}(\mathsf{c_m})\}\mathsf{n_{max}}.$$

Wird $n_{min} = 0$ und $n_{max} = \infty$ eingestellt, so sind in einer Antwortmenge beliebig viele dieser Spezifikationsatome enthalten und daher die zugehörigen Constraints relevant. Durch eine entsprechende weiche # maximize-Anweisung kann erreicht werden, dass möglichst viele dieser Spezifikationsatome enthalten sind. Durch mehrere # maximize-Anweisungen kann eine gewisse „Hierarchie in der Dringlichkeit“ der Constraints ausgedrückt werden. Durch die Analyse der Ausgaben können Rückschlüsse auf die Ursachen der Inkonsistenz getroffen werden. Auch eine Variation von $n_{min}$ und $n_{max}$ kann hilfreiche Aussagen liefern.

Die Konsistenzprüfung ist also auf sehr einfache Weise durchzuführen. Außerdem gibt es vielfältige Ansätze, Hinweise auf die Ursachen und Möglichkeiten zur Behebung vorhandener Inkonsistenz zu erhalten. Eine flexible Handhabung ist hierdurch gewährleistet.

**Beratung für den Studiendekan der Fakultät** Mögliche Lehrangebote ergeben sich durch die in den Antwortmengen enthaltenen $Coursetemplate$-Atomen. Das „Mindesmaß“ an konkretem Lehrangebot ergibt sich aus einer der berechneten Antwortmengen: Zu jedem $Coursetemplate$-Atom ist mindestens eine konkrete Lehrveranstaltung anzubieten. In der Regel möchte allerdings eine Universität den Studierenden eine gewisse Wahlfreiheit bieten, so dass das konkrete Lehrangebot etwas reichhaltiger ausfallen sollte.

Wie schon erwähnt, werden die konkreten Lehrveranstaltungen einfach in Form einer zusätzlichen Ebene im Modulkatalog-Objektgraphen unter der $Coursetemplate$-Ebene realisiert. Darüber hinaus ist das Constraint

(*) „Zu jedem gewählten $Coursetemplate$ ist genau eine konkrete Veranstaltung zu wählen.“

zu realisieren.

Möchte der Studiendekan wissen, ob das konkrete Lehrangebot ausreicht, so hat er einfach nur zu prüfen, ob es unter den gerade besprochenen Bedingungen eine Lösung zur Spezifikation gibt. Gibt es eine Antwortmenge, so ist das konkrete Lehrangebot ausreichend, im anderen Fall nicht.

Reicht das aktuelle bzw. aktuell geplante Lehrangebot nicht aus, so ist es natürlich interessant, welche Lehrveranstaltungen fehlen und somit zusätzlich angeboten werden sollten. In diesem Fall ist das oben in (*) besprochene Constraint etwas abzuwandeln, was sehr leicht umgesetzt werden kann: Statt „genau eine“ ist „keine oder eine“ konkrete Veranstaltung zu jedem gewählten $Coursetemplate$-Strukturelement auszusuchen. Zu den $Coursetemplate$-Elementen, zu denen es keine konkrete Veranstaltung im Modell gibt, sind noch weitere konkrete Veranstaltungen anzubieten.

Auch für die Erstellung eines „Musterstudienplans“ erhält der Studiendekan geeignete Hinweise.

Es zeigt sich also, dass die Realisierung der Beratungsmöglichkeiten für den Studiendekan sich einfach gestaltet. Die relevanten Fragenstellungen können gut behandelt werden.

**Studienberatung** Die verschiedenen Möglichkeiten, ein Studium zu organisieren, ergeben sich aus den Antwortmengen der Spezifikation der Studienordnung.

Hat ein Studierender schon Veranstaltungstemplates[27] besucht, so kann dies im Programm durch Fakten

$$\mathsf{chosen}(\mathsf{id}).$$

umgesetzt werden. Das Wählen von inneren Strukturelementen des Modulkatalog-Graphen geschieht analog. Die Antwortmengen, die diese Atome enthalten, geben Auskunft über weitere Möglichkeiten, das Studium zu absolvieren. Hierzu gehört auch die Wahl ensprechender Vorfahren- bzw. Nachfahren-Strukturelemente.

[27] bzw. konkrete Veranstaltungen, die zu den Veranstaltungstemplates assoziiert sind

Fragt sich ein Student, ob die gewünschten Entscheidungen mit der Studienordnung konform sind, so kann er seine Wahlen angeben. Ergibt sich eine Antwortmenge, so sind sie konform, im anderen Fall nicht.

Zu beachten ist, dass bestandene Veranstaltungstemplates in der erweiterten Programmversion auch durch Fakten

$$\mathsf{passed}(\mathsf{id}).$$

umgesetzt werden können. Wie an entsprechender Stelle erwähnt ist, können Informationen über Voraussetzungen gegeben und Empfehlungen über den zeitlichen Ablauf des Studiums getroffen werden.

Das Programm gewährleistet also eine umfassende und flexible Studienberatung, die den Bedürfnissen der Studierenden gerecht wird.

Zusammenfassend lässt sich hiermit also feststellen, dass sämtliche Fragestellungen der in Abschnitt 2.4.1 aufgelisteten relevanten Anwendungsfälle mit Hilfe des Programms gut behandelt werden können.

## 6.2 Pfadbasierter Ansatz

Die folgenden Ausführungen dienen der Abgrenzung der gewählten logikbasierten Methode zu gängigen Ansätzen (XML, XPath und XQuery) mit anderer, hier pfadbasierter, Philosophie. Es soll keine vollständige Transformation vorgenommen werden. Die Überlegungen können auch als Art Gedankenspiel aufgefasst werden. Zur einfacheren Darstellung und ohne Verlust an Aussage wird hier nur ein Anwendungsbereich, nämlich „Studienberatung und Konsistenzprüfung von Studienordnungen“ betrachtet; dessen Domäne ist nämlich durch den Modulkatalog „von Natur aus“ hierarchisch strukturiert, was naheliegt ein hierarchisches Datenmodell in Betracht zu ziehen.

In einem ersten Schritt soll zunächst nur XML und XPath Anwendung finden. Für die nachfolgenden Überlegungen sei hier auch an die Modellierung in Kapitel 5 und den Abschnitt 6.1.2 erinnert.

### 6.2.1 XML und XPath

#### 6.2.1.1 Ansätze

Bei der Transformation des Modells in einen internen Formalismus sind die drei Bestandteile, Struktur-, Constraint- und Lösungsmodell, von Bedeutung. Es bietet sich folgendes Vorgehen bei der Repräsentation an:

($S$) *Strukturmodell:* Modulkatalog-Objektgraph als XML-Dokument.

$(C)$ *Constraintmodell:* Ein oder mehrere XPath-Ausdrücke zur Beschreibung der Anforderungen.

$(L)$ *Lösungsmodell:* Eine Lösung als Teilgraph des Modulkatalog-Objektgraphen bzw. als Teildokument des entsprechenden XML-Dokuments.

Es gibt nun prinzipiell zwei verschiedene Ansätze, nämlich den des Modeltestings und den des Modelgeneratings.

1. *Modeltesting:*

   - Eine hypothetische Lösung, also ein Teilgraph des Modulkatalog-Objektgraphen bzw. das entsprechende XML-Dokument $M$, ist gegeben.
   - Mittels eines Modeltesting-Verfahrens wird *geprüft*, ob $M$ die die Constraints beschreibenden XPath-Ausdrücke erfüllt.
   - Ist dies der Fall, so ist $M$ eine gültige Lösung.

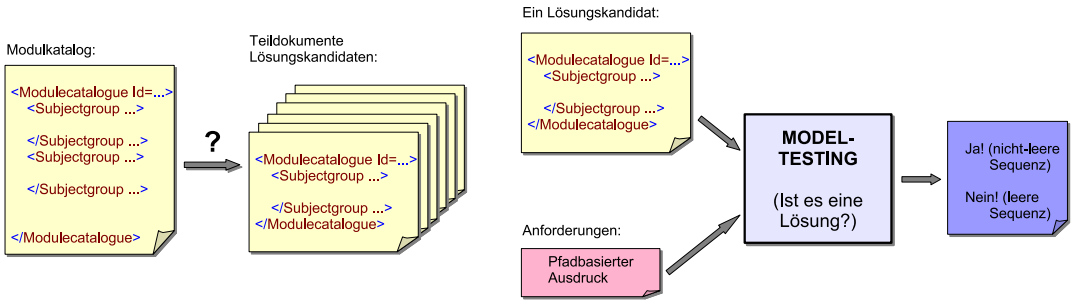



Abbildung 6.8: Modeltesting

   Dieses Prinzip veranschaulicht Abbildung 6.8.

2. *Modelgenerating:*

   - Mittels eines Modelgenerating-Verfahrens und der Datenstrukturen von $(S)$ und $(C)$ werden die möglichen Lösungen *generiert*. Diese liefern die Datenstrukur für $(L)$.

   Dieses Prinzip veranschaulicht Abbildung 6.9.

Die Erläuterung der Prinzipien und der Problematik soll weitgehend beispielsbasiert erfolgen. Eine Übertragung auf allgemeinere Fälle kann in der Regel leicht erfolgen.

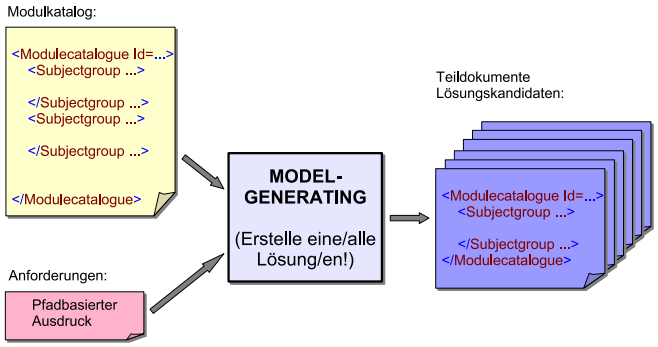



Abbildung 6.9: Modelgenerating

### 6.2.1.2 Modeltesting

**Beispiel 6.2.1** Gegeben sei der Ausschnitt eines Modulkatalog-Objektgraphen einer Studienordnung – der Einfachheit und Kompaktheit halber sind die *Coursetemplate*-Strukturelemente hier nicht eingezeichnet – aus Abbildung 6.10 und die Anforderung an ein gültiges Studium aus Abbildung 6.11.

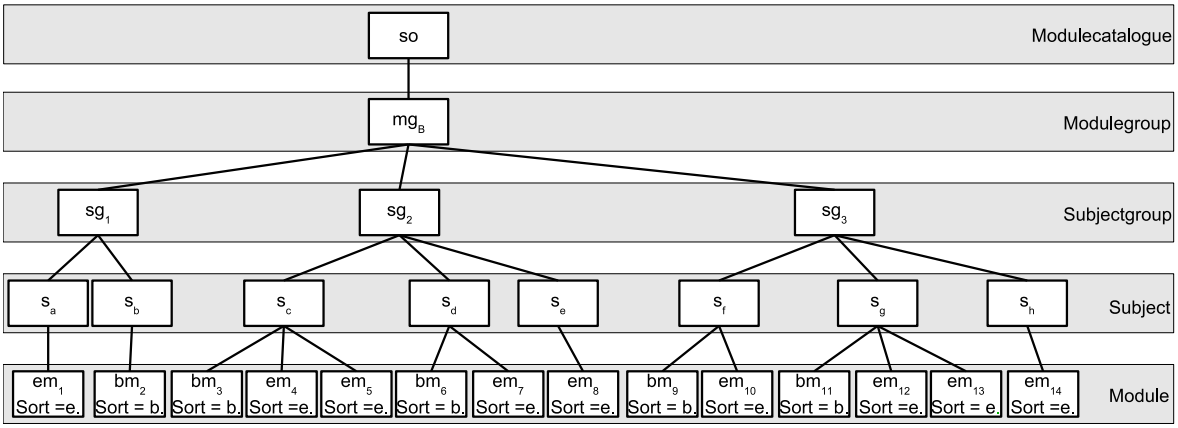



Abbildung 6.10: Modulkatalog-Objekt-Graph, Beispiel

**Strukturmodell** Der Modulkatalog-Objektgraph aus Beispiel 6.2.1 könnte durch folgene XML-Struktur straight-forward repräsentiert werden:

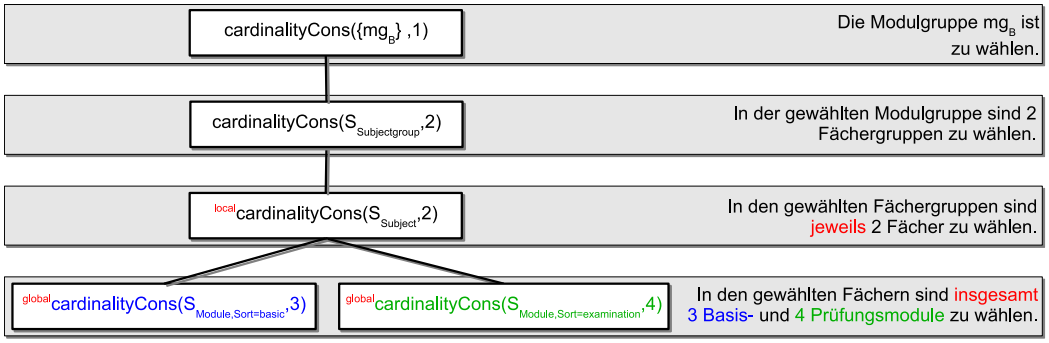



Abbildung 6.11: Baumconstraint, Beispiel

```
<Modulecatalogue Id="so">
    <Modulegroup Id="mgB">
        <Subjectgroup Id="sg1">
            <Subject Id="sa">
                <Module Id="em1" Sort="examination"/>
            </Subject>
            <Subject Id="sb">
                <Module Id="bm2" Sort="basic"/>
            </Subject>
        </Subjectgroup>
        <Subjectgroup Id="sg2">
            <Subject Id="sc">
                <Module Id="bm3" Sort="basic"/>
                <Module Id="em4" Sort="examination"/>
                <Module Id="em5" Sort="examination"/>
            </Subject>
            <Subject Id="sd">
                <Module Id="bm6" Sort="basic"/>
                <Module Id="em7" Sort="examination"/>
            </Subject>
            <Subject Id="se">
                <Module Id="em8" Sort="examination"/>
            </Subject>
        </Subjectgroup>
        <Subjectgroup Id="sg3">
            <Subject Id="sf">
                <Module Id="bm9" Sort="basic"/>
                <Module Id="em10" Sort="examination"/>
            </Subject>
            <Subject Id="sg">
                <Module Id="bm11" Sort="basic"/>
                <Module Id="em12" Sort="examination"/>
                <Module Id="em13" Sort="examination"/>
```

```
            </Subject>
            <Subject Id="sh">
                  <Module Id="em14" Sort="examination"/>
37          </Subject>
        </Subjectgroup>
    </Modulegroup>
</Modulecatalogue>
```

Listing 6.1: XML-Repräsentation des Modulkataloges aus Beispiel 6.2.1

**Lösungen** Ein bezüglich der gegebenen Anforderung gültiges Studium wäre in Beispiel 6.2.1 ein Teilgraph des Graphen aus Abbildung 6.10 bzw. ein entsprechendes Teildokument des obigen XML-Dokuments. Eine mögliche Lösung ist in Abbildung 6.12 farblich markiert bzw. in nachfolgendem XML-Dokument repräsentiert:

```
<Modulecatalogue Id="so">
    <Modulegroup Id="mgB">
        <Subjectgroup Id="sg2">
            <Subject Id="sc">
5                <Module Id="bm3" Sort="basic"/>
                 <Module Id="em5" Sort="examination"/>
            </Subject>
            <Subject Id="sd">
                  <Module Id="bm6" Sort="basic"/>
10                <Module Id="em7" Sort="examination"/>
            </Subject>
        </Subjectgroup>
        <Subjectgroup Id="sg3">
            <Subject Id="sf">
15                <Module Id="bm9" Sort="basic"/>
                  <Module Id="em10" Sort="examination"/>
            </Subject>
            <Subject Id="sh">
                  <Module Id="em14" Sort="examination"/>
20          </Subject>
        </Subjectgroup>
    </Modulegroup>
</Modulecatalogue>
```

Listing 6.2: Ein gültiges Studium / XML-Repräsentation von Abbildung 6.12

Hingegen stellen der Teilgraph aus Abbildung 6.13 und das entsprechende XML-Dokument bezüglich der gegebenen Anforderung kein gültiges Studium dar: In einer der Fächergruppen ist nur ein Fach gewählt und außerdem sind zu wenige Basismodule vertreten.

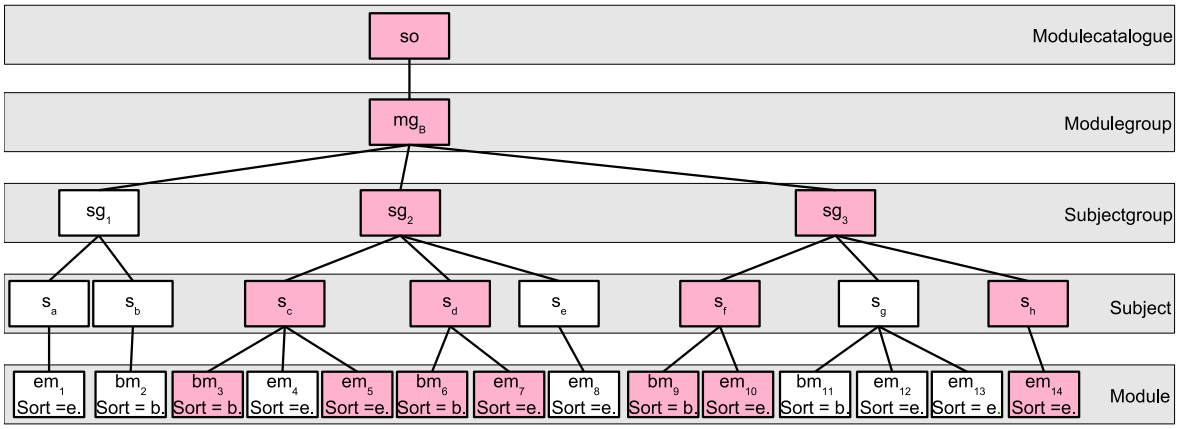



Abbildung 6.12: Eine Möglichkeit für ein gültiges Studium: Teilgraph

```
<Modulecatalogue Id="so">
    <Modulegroup Id="mgB">
        <Subjectgroup Id="sg2">
            <Subject Id="sc">
                <Module Id="bm3" Sort="basic"/>
                <Module Id="em4" Sort="examination"/>
                <Module Id="em5" Sort="examination"/>
            </Subject>
            <Subject Id="se">
                <Module Id="em8" Sort="examination"/>
            </Subject>
        </Subjectgroup>
        <Subjectgroup Id="sg3">
            <Subject Id="sh">
                <Module Id="em14" Sort="examination"/>
            </Subject>
        </Subjectgroup>
    </Modulegroup>
</Modulecatalogue>
```

Listing 6.3: Kein gültiges Studium / XML-Repräsentation von Abbildung 6.13

**Anforderungen und Constraints** Die Anforderungen an ein gültiges Studium sind als XPath-Ausdrücke $p_1, \ldots, p_r, r \in \mathbb{N}$ so zu formulieren, dass bei entsprechenden Anfragen an einen „ungültigen Lösungskandidaten“ mindestens einmal eine leere Sequenz als Ergebnis geliefert wird. Wird hingegen die Anfrage an eine „richtige Lösung“ gestellt, so sind die Ergebnissequenzen nicht leer.

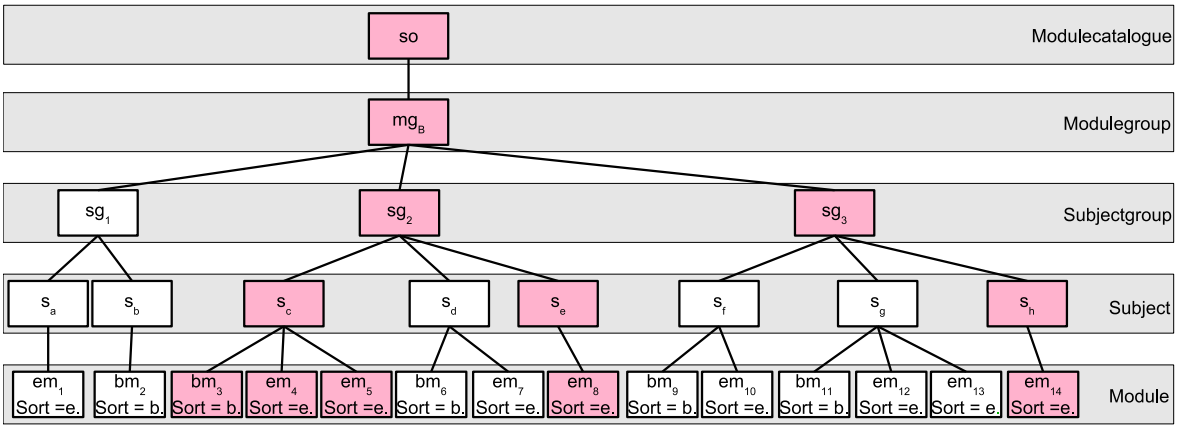



Abbildung 6.13: Kein gültiges Studium

Im Beispiel 6.2.1 könnte die Anforderung als XPath-Ausdruck schrittweise wie folgt aufgebaut werden:

- Die Modulgruppe $mg_B$ muss enthalten sein:

  ```
  //Modulegroup[@Id="mgB"]
  ```

- In dieser sind 2 Fächergruppen zu wählen:

  ```
  //Modulegroup[@Id="mgB"] [count( Subjectgroup )=2]
  ```

- Je gewählter Fächergruppe müssen 2 Fächer belegt werden (insgesamt: 4 Fächer):

  ```
  //Modulegroup[@Id="mgB"][count( Subjectgroup )=2]
  /Subjectgroup[count( Subject )=2] [count( //Modulegroup[@Id="mgB"]//Subject )=4]
  ```

  Es ist zu gewährleisten, dass vorherige „Wahlen“ durch zusätzliche Bedingungen nicht „verloren“ gehen.

- In den gewählten Fächern sind insgesamt 3 Basis- und 4 Prüfungsmodule zu belegen:

  ```
  //Modulegroup[@Id="mgB"][count( Subjectgroup )=2]
  /Subjectgroup[count( Subject )=2]  [count(//Modulegroup[@Id="mgB"]//Subject)=4]
  /Subject[count( //Modulegroup[@Id="mgB"]//Module[@Sort="basic"] )=3]
  [count( //Modulegroup[@Id="mgB"]//Module[@Sort="examination"] )=4]
  ```

Wird nun also der Pfadausdruck

```
//Modulegroup[@Id=”mgB”][count( Subjectgroup )=2]
/Subjectgroup[count( Subject )=2]  [count(//Modulegroup[@Id=”mgB”]//Subject)=4]
/Subject[count( //Modulegroup[@Id=”mgB”]//Module[@Sort=”basic”] )=3]
[count( //Modulegroup[@Id=”mgB”]//Module[@Sort=”examination”] )=4]
```

an ein XML-Dokument gestellt, welches ein gültiges Studium repräsentiert, so ergibt sich keine leere Ergebnissequenz. In diesem Fall würden sämtliche belegten Fächer ausgegeben werden. Wird hingegen die Anfrage an ein XML-Dokument gestellt, welches kein gültiges Studium darstellt, so ist das Ergebnis die leere Sequenz. Möchte man Hinweise auf die Ursache der Inkonsistenz erhalten, so könnte man entsprechende Teilpfadausdrücke evaluieren.

Zur Implementierung von *Gewichtssummenconstraints*, z. B. der Anforderung, dass insgesamt mehr als 24 ECTS-Punkte in den gewählten Veranstaltungstemplates erreicht werden müssen, wäre im Sturkturmodell folgende Repräsentation möglich:

```
<Modulecatalogue Id="so">
    <Modulegroup Id="mgB">
          ...
          <Coursetemplate Id="c7">
              <Ects>5</Ects>
          </Coursetemplate
          ...
          <Coursetemplate Id="c19">
              <Ects>10</Ects>
          </Coursetemplate>
    </Modulegroup>
</Modulecatalogue>
```

Listing 6.4: Repräsentation numerischer Attribute von Veranstaltungstemplates

Der entprechende Part im XPath-Ausdruck könnte

```
//Coursetemplate[sum(Ects)>24]
```

lauten.

Wie man sich leicht überlegen kann, sind eine Reihe von Constraints in XPath darstellbar, nämlich:

- *Anzahl:* mit Hilfe von count , mit Einschränkungen bei $cardinalityCons(C, \text{all})$: hier wäre ein Vergleich mit dem Ursprungsdokument notwendig; als Ausweg ist die Anforderung numerisch auszudrücken
- *Gewichtssumme:* mit Hilfe von sum und entsprechenden Tags
- *Enthaltensein:* durch Angabe der entsprechenden *id* als Bedingung, etwa

  ```
  Structuretype[@Id=”id”]
  ```

- *Ausschluss:* Ähnlich zum Enthaltenseinsconstraint, nur mit „ungleich“:

  ```
  Structuretype[@Id!=”id”]
  ```

- *Successorconstraint und Konjunktion:* durch entsprechende Formulierung der Bedingungen und Verknüpfung der Pfadausdrücke (vgl. obiges Beispiel)

Nicht betrachtet sind hier allerdings mögliche Voraussetzungen innerhalb der Strukturelemente des Modulkatalog-Objektgraphen.

#### 6.2.1.3 Modelgenerating

Betrachten wir auch hier das Beispiel 6.2.1. Das Strukturmodell und die Lösungen sind die gleichen wie beim Modeltesting mit dem Unterschied, dass beim Modelgenerating die Lösungen oder zumindest eine beliebige davon generiert werden soll. Ausgangsdokument ist das XML-Dokument, welche den Modulkatalog repräsentiert, im Beispiel Listing 6.1.

Das Grundproblem ist die Transformation eines Graphen in geeignete Teilgraphen, welche genau die Lösungen der Anforderungen darstellen.

Im Beispiel 6.2.1 könnte die Anforderung als XPath-Ausdruck schrittweise wie folgt aufgebaut werden:

- Die Modulgruppe $mg_B$ muss enthalten sein:

  ```
  //Modulegroup[@Id=”mgB”]
  ```

- In dieser sind 2 Fächergruppen zu wählen:

  ```
  //Modulegroup[@Id=”mgB”] [count( Subjectgroup )>=2]/Subjectgroup[position()<=2]
  ```

  Es gilt hier aus der Menge (Sequenz) der Fächergruppen genau 2 beliebige auszuwählen. Eine Bedingung ist, dass Modulgruppe $mg_B$ mindestens 2 Fächergruppen enthält. Die einfachste Möglichkeit ist, die ersten beiden zu wählen. Diese Anfrage würde die Sequenz dieser beiden Fächergruppen als Ergebnis liefern.

- Je gewählter Fächergruppe müssen 2 Fächer belegt werden:

  ```
  //Modulegroup[@Id=”mgB”][count( Subjectgroup [count(Subject)>=2] )>=2]
  /Subjectgroup[count(Subject)>=2][position()<=2]
  /Subject[position()<=2]
  ```

Enthält eine der Fächergruppen weniger als 2 Fächer, so entfällt diese bzw. deren Kindknoten in der Resultatsequenz. Es ist daher zu gewährleisten, dass die Fächergruppenwahl schon geschickt genug war, so dass die gewählten Fächergruppen auch mindestens 2 Fächer enthalten. Das Ergebnis dieser Anfrage liefert die Sequenz der gewählten Fächer.

- In den gewählten Fächern sind insgesamt 3 Basis- und 4 Prüfungsmodule zu belegen:

  In diesem Fall ist ein $^{global}$-Constraint nicht straight-forward zu lösen, position() bezieht sich auf die Position innerhalb der Geschwisterknoten und nicht innerhalb der Knoten „der gleichen Generation“; der Umweg über die Elternknoten wie beim Modeltesting ist auch nicht möglich, da möglicherweise nicht gewählte Geschwisterknoten wieder in Betracht gezogen würden. Man müsste mehr Kenntnis über den Aufbau des Baumes mit einbeziehen, z. B. konkrete Knoten oder Knoten mit bestimmter Position und gewünschten Eigenschaften wählen. Die entsprechende Anfrage würde die Module als Ergebnissequenz liefern.

Trotz der guten Navigationsmöglichkeiten innerhalb des Modulkatalog-Objektgraphen stößt man mit diesem Ansatz schnell an die Grenzen, da beispielsweise die Möglichkeiten des Zählens sehr begrenzt sind. Hierzu mehr in Abschnitt 7.4.1. Aus diesen Gründen wird ein Ansatz untersucht, der XPath beeinhaltet, jedoch mächtiger ist.

### 6.2.2 Erweiterung mit XQuery

#### 6.2.2.1 Prinzipielle Überlegungen zum Algorithmus

Zur Implementierung des Modelgenerating-Ansatzes mit den Konzepten von XPath und XQuery ist Folgendes zu überlegen:

- *Eingabe:* XML-Dokument, das den Modulkatalog-Objektgraphen der Studienordnung repräsentiert (Quellbaum oder -graph)[28]
- *Ausgabe:* XML-Dokument, welches ein gültiges Studium (Lösung) repräsentiert (Zielbaum oder -graph)
  - Zieldokumente sind Teildokumente der Eingabe
  - Evtl. auch Ausgabe aller Lösungen
- *Algorithmus, Transformation (Modelgenerating):* Überführung von Quellbaum bzw. -graph in Zielbaum bzw. -graph
  - *Ziele:* Finden einer allgemeinen Lösung bzw. aller Lösungen

[28] Der Einfachheit halber wird im Folgenden meist davon ausgegangen, dass eine Baumstruktur vorliegt.

  - *Dualität:* Pfadanfragen und Generierung von Lösungen (vgl. Abschnitt 6.2.1)

- *Ansatz:* Schrittweises „Abrollen“, von „einem“ Knoten ausgehend verschiedene Möglichkeiten „generieren“. Zu beachten ist ferner, dass möglicherweise getroffene Entscheidungen von Studierenden auch berücksichtigt werden sollen. Diese können in Erweiterung zu obiger Aussage auch eine Art Eingabe darstellen.

#### 6.2.2.2 Grundoperationen

Bezüglich der Implementierung kann hier der Vorteil von XQuery, rekursive Funktionen definieren zu können, zum Tragen kommen. Nachfolgend werden einige gewünschte Grundoperationen aufgezeigt. Zu beachten ist hierbei, dass hinter XQuery ein Sequenz- und nicht ein Mengenkonzept steht.

**Bestimmung von Teilsequenzen** In einer Sequenz von Elementen sind sämtliche Teilsequenzen einer vorgegebenen Länge zu bestimmen (vgl. Abbildung 6.14).

Operation SUBSEQUENCES:

1. *Eingabeparameter:*
   - Sequenz von Elementen: $seq$
   - Länge der gewünschten Teilsequenzen: $n \in \mathbb{N}_0$
2. *Ausgabe:* Sequenz der Teilsequenzen der Länge $n$ von $seq$

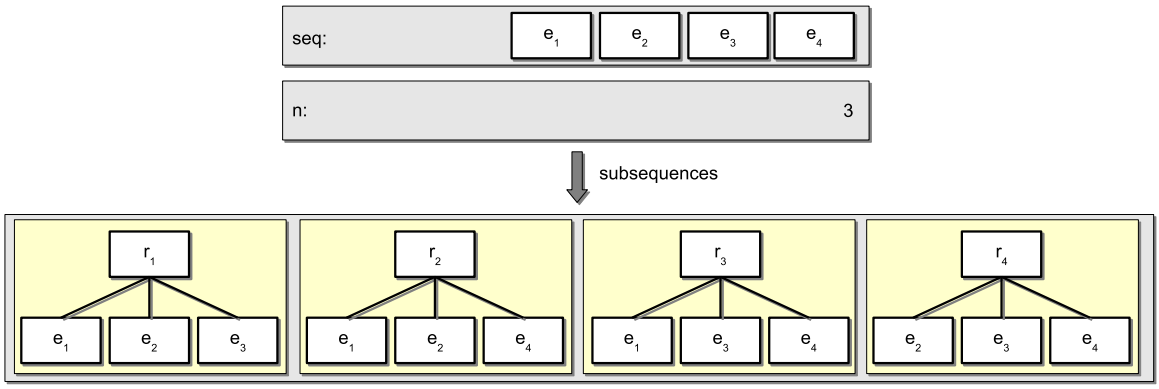



Abbildung 6.14: Operation subsequences, Beispiel

Beispielhaft sei eine einfache Implementierung in XQuery gezeigt:

```
declare function local:subsequences($seq as element()*, $n as xs:integer)
  as element()*
  {
   let $head := $seq[position()=1]
   let $body := $seq[position()>1]
   let $number := fn:count($seq)

   return
   if ($n>0)
   then
      if ($number = $n)
      then <tm>{$seq}</tm>
      else
         if ($number > $n)
         then
            local:concatTo($head, local:subsets($body, $n -1))
                  | local:subsets($body,$n)
         else empty
   else <tm/>   (:wenn n=0 (n<=0):)
};

(:Hilfsfunktion:)
declare function local:concatTo($h as element()*, $s as element()*)
  as element()*
  {
   for $t in $s
   return <tm>{($h,$t/child::*)}</tm>
  };
```

Listing 6.5: Beispiel einer XQuery-Implementierung von SUBSEQUENCES

Darüber hinaus möchte man die Möglichkeit haben, obligatorische Elemente anzugeben. Dies führt zu einer Sonderform der Operation SUBSEQUENCES: In einer Sequenz von Elementen sind sämtliche Teilsequenzen vorgegebener Länge zu bestimmen, mit der zusätzlichen Bedingung, dass bestimmte Elemente auf jeden Fall enthalten sein müssen (vgl. Abbildung 6.15).

Operation SUBSEQUENCESINCLUDE:

1. *Eingabeparameter:*
   - Sequenz von Elementen: $seq$
   - Sequenz von obligatorischen Elementen (aus $seq$): $obl$
   - Länge der gewünschten Teilsequenzen: $n \in \mathbb{N}_0$
2. *Ausgabe:* Sequenz der Teilsequenzen der Länge $n$ von $seq$, welche die Elemente aus $obl$ enthalten.

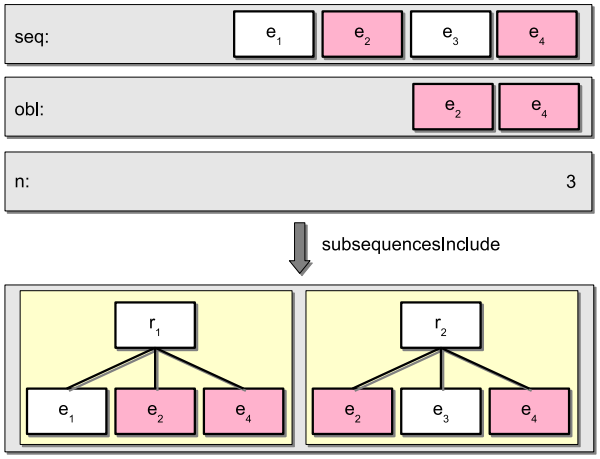



Abbildung 6.15: Operation subsequencesInclude, Beispiel

**Wählen** Unter mehreren Möglichkeiten ist eine Wahl zu treffen, beispielsweise bei der Wahl der Fächer o. Ä. Hierbei ist aus einer Sequenz von Elementen eines auszuwählen, anzugeben über die Position. Operation CHOOSE:[29]

1. *Eingabeparameter:*
   - Sequenz von Elementen: $seq$
   - Position des zu wählenden Elements: $i \in \mathbb{N}$
2. *Ausgabe:* $i$-tes Element von $seq$, falls möglich

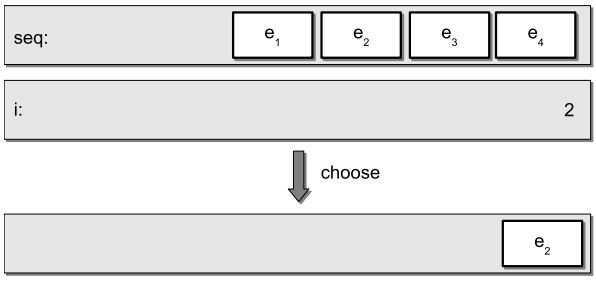



Abbildung 6.16: Operation choose, Beispiel

[29] Im Prinzip ist dies in XQuery durch position() schon realisiert, jedoch ohne Fehlerbehandlung.

**Gleichheitsprüfung** Da bei diesem Verfahren neue XML-Dokumente aufgrund der Transformation von Baum (Graph) zu Baum (Graph) entstehen, ist eine neue Art von Gleichheit zu definieren. Bestimmte XML-Elemente in verschiedenen XML-Dokumenten sind miteinander zu identifizieren.

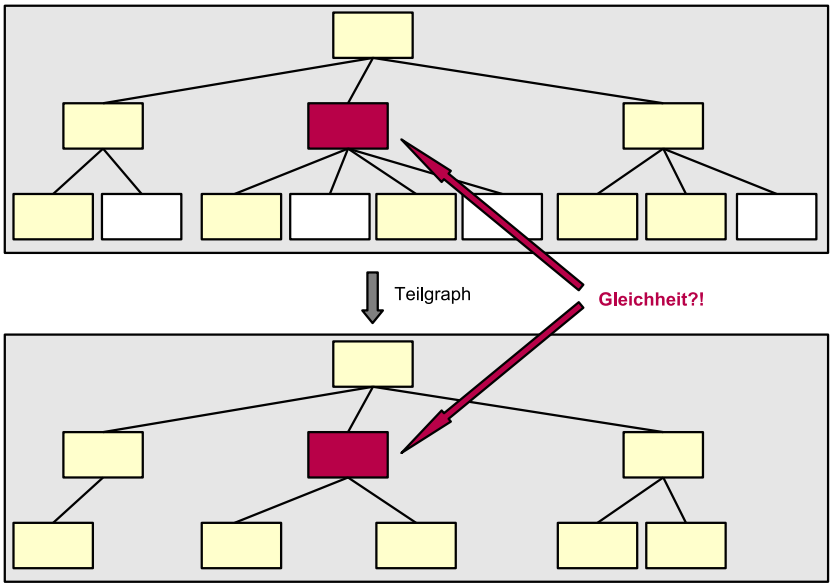



Abbildung 6.17: Definition von Gleichheit

In Abbildung 6.17 sind die markierten Elemente als XML-Elemente nicht gleich, sie befinden sich in verschiedenen XML-Dokumenten. Als mögliche Elemente eines Modulkatalogs sollten sie jedoch miteinander identifizierbar sein, was in folgener Operation zum Ausdruck kommt:

Operation ISEQUAL:

1. *Eingabeparameter:*
   - Element: $e_1$
   - Element: $e_2$
2. *Ausgabe:* Wahrheitswert über die Gleichheit der Elemente. Hierbei sollen sie genau dann als gleich gelten, wenn sie die gleichen Elemente im Anwendungsbereich, also die gleichen Elemente des Modulkatalog-Objektgraphen repräsentieren.[30]

Mit Hilfe der Operation ISEQUAL können noch weitere Gleichheitsprüfungs-Operationen definiert werden, z. B. eine Operation AREEQUAL, welche zwei Sequenzen von Elementen miteinander vergleicht. Diese Operationen werden unter anderem bei Entfernen-Operationen benötigt.

[30] Eine implementierungtechnische Realisierung könnte beispielsweise über die Identifikation von Attribut-ID's erfolgen.

**Entfernen von Elementen** Hierbei gibt es verschiedene Ausprägungen einer möglichen Operation REMOVE:

1. Entfernen *eines Elements*
   (a) aus einer *Sequenz von Elementen*, wobei
      - nur das „erste Vorkommen“ gelöscht wird.
      - alle Vorkommen des entsprechenden Elements aus der Sequenz gelöscht werden.
   (b) aus einem *Wald von Elementen*, wobei
      - alle Vorkommen des entsprechenden Elements innerhalb des Waldes gelöscht werden; im Gegensatz zum Löschen aus einer Sequenz werden hier die Nachfahren auch betrachtet.
2. Entsprechendes Entfernen von *mehreren Elementen*.

Als Beispiel sei eine Operation REMOVEFOREST betrachtet:

1. *Eingabeparameter:*
   - Wald von Elementen bzw. Sequenz von Wurzelelementen: $forest$
   - Element: $e$
2. *Ausgabe:* Wald von Elementen, der aus dem Eingabe-Wald dadurch hervorgeht, dass alle Vorkommen des Elements $e$ innerhalb des Waldes entfernt werden.[31]

#### 6.2.2.3 Operationen zum Bestimmen von Teillösungen

Die Anforderungen in der Studienordnung beschränken die Möglichkeiten der Bildung von Teilgraphen aus dem Modulkatalog. Examplarisch sollen im Folgenden nur einige Constraints bezogen auf Anzahlen betrachtet werden.

Durch das schrittweise Vorgehen wünscht man sich Operationen, welche auf der Basis einer $k$-ten Zwischenlösung und eines Constraints eine oder sämtliche $(k + 1)$-Teillösungen (Lösungen nach dem $(k + 1)$-ten Schritt errechnen. Werden in einem Schritt sämtliche Teillösungen bestimmt, so könnte man sich vorstellen, dass ein Studierender auf deren Grundlage die Möglichkeit erhält, sich für eine zu entscheiden und diese zu wählen. Eine einfache Form wäre, dies über die Nummer der Teillösung zu tun.

Als Beispiel sei eine Operation CARDINALITYSOLUTIONS betrachtet:

[31] In einer XQuery-Implementierung kann der Element-Konstruktor dienlich sein.

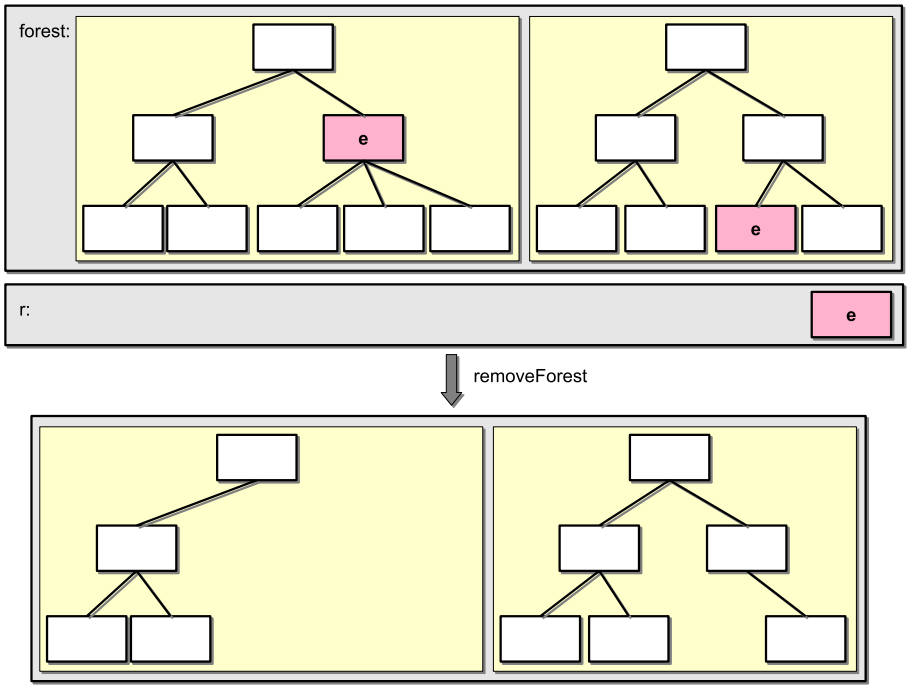



Abbildung 6.18: Operation removeForest, Beispiel

1. *Eingabeparameter:*
   - Baum: $soTemp$
   - Constraintreferenz-Elemente: $ref$
   - Anzahl: $n \in \mathbb{N}_0$
2. *Ausgabe:* Sequenz aller Teilbäume von $soTemp$, welche aus $soTemp$ dadurch hervorgehen, dass genau $n$ Elemente aus $ref$ noch enthalten sind (und die anderen aus $ref$ gestrichen wurden).
3. *Hinweise zu einer möglichen Realisierung:* In XQuery kann $soTemp$ als Element und $ref$ als Sequenz von Elementen realisiert werden. Beim Aufruf kann in der Regel $ref$ als Pfadausdruck angegeben werden. Algorithmisch kann diese Operation durch Verwendung der Grundoperationen SUBSEQUENCES ($n$-Teilsequenzen von $ref$), CHOOSE (Zwischenwahl einer $n$-Teilsequenz von $ref$), REMOVE-Operationen (Bestimmung der zu löschenden Elemente, Löschen dieser Elemente) realisiert werden.

Soll nur eine Teillösung ausgegeben werden, so kann dies mit einer Operation CARDINALITYCHOICESOLUTION mit ähnlicher Funktionalität wie CARDINALITYSOLUTIONS umgesetzt werden, mit

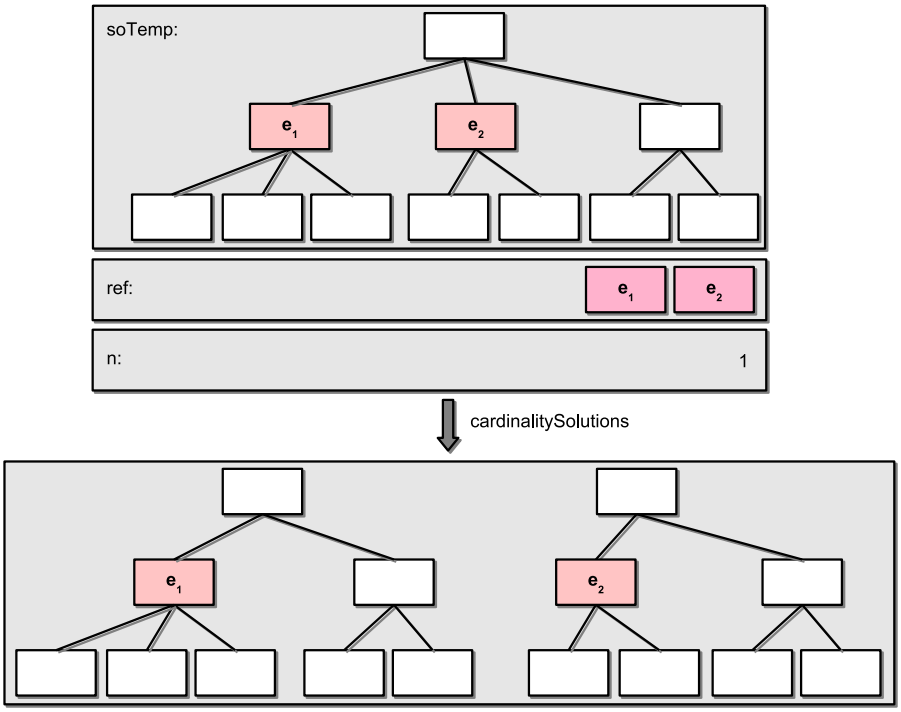



Abbildung 6.19: Operation cardinalitySolutions, Beispiel

dem Unterschied, dass ein zusätzlicher Parameter zur Angabe der Nummer der gewünschten Lösung notwendig ist.

Analoge Operationen CARDINALITYLOCALSOLUTIONS und CARDINALITYGLOBALSOLUTIONS bzw. CARDINALITYLOCALCHOICESOLUTION und CARDINALITYGLOBALCHOICESOLUTION zur Umsetzung der Successorconstraints können festgelegt werden. In diesen kann die Operation CARDINALITYSOLUTIONS Anwendung finden.

**Beispiel 6.2.2 (Studienberatung)** Die Anforderung „Wähle in der Modulgruppe $mg_B$ genau 2 Fächer.“ aus Beispiel 6.2.1 kann durch einen Aufruf der Art

$$\text{CARDINALITYSOLUTIONS}(so, \text{so/Modulegroup/Subject}, 2)$$

gelöst werden. Hierbei repräsentiert $so$ den Modulkatalog (bzw. eine entsprechende Zwischenlösung). Die Wahl einer der Resultate kann durch die Angabe der entsprechenden Nummer erfolgen, z. B. über CHOOSE oder gleich über CARDINALITYCHOICESOLUTION.

Weiteres kann noch realisiert werden, soll hier aber nicht weiter ausgeführt werden. Erwähnt sei nur, dass beispielsweise bei der Implementierung von Voraussetzungen das Konzept von KEY und KEYREF von Vorteil ist.

# Kapitel 7

# Evaluation

Eine Bewertung des eigenen Ansatzes wird in diesem Kapitel vorgenommen. Bezüglich der internen Repräsentation soll zunächst vor allem die favorisierte Methode Answer Set Programming mit Gewichten im Blickfeld sein. Ergänzend werden weitere Methoden in Augenschein genommen.

## 7.1 Funktionale, systembezogene Anforderungen

Zunächst soll auf die funktionalen, systembezogenen Anforderungen aus den Abschnitten 2.3.2 und 2.4.2 eingegangen werden.

### 7.1.1 Repräsentation der Daten in geeigneter Form

#### 7.1.1.1 Grunddaten

Die Grunddaten, also im Anwendungsbereich „Generierung von Tests“ (kurz: AB GT) die Fragen und ihre Eigenschaften und im Anwendungsbereich „Studienberatung und Konsistenzprüfung von Studienordnungen“ (kurz: AB SO) der Modulkatalog zusammen mit den konkreten Veranstaltungen, müssen geeignet repräsentiert werden.

In Abschnitt 6.1 ist gezeigt, dass diese Daten sehr einfach in der Sprache Answer Set Programming (ASP) in Form von Fakten oder mit Hilfe einfacher struktureller Ableitungsregeln und Domänenprädikate darzustellen sind. Je nachdem welche Infrastruktur bevorzugt wird, können diese Daten auch in einer relationalen Datenbank gehalten werden und gegebenenfalls auf einfache Weise in die Syntax der Logiksprache in gewünschter Form transformiert werden.

#### 7.1.1.2 Anforderungen

Die Anforderungen können meist in Form von zwei Stufen auf modulare Art und Weise umgesetzt werden: Zum einen durch die Spezifikation eines bestimmten Constraints mit bestimmten Parameterwerten, was in Form eines Faktes geschehen kann, und zum anderen durch die invarianten Regeln zur „Behandlung der Constraints“. Mit Hilfe von nur wenigen Regeln können ganze „Klassen“ von Constraints behandelt werden.

In den Abschnitten 6.1.1.3 und 6.1.2.3 ist gezeigt, dass dies templatebasiert erfolgen kann. Der jeweilige Administrator des Testmanagement-Systems bzw. StARC-Systems hat die für den konkreten Anwendungsbereich erforderlichen Regeln nach vorgegebenem Muster umzusetzen. Im Falle des StARC-Systems sind vom Administrator auch noch die Spezifikationsfakten zur Repräsentation der in der Studienordnung vertretenen Bedingungen anzugeben.

Die Anforderungen und Bedingungen, welche im Wesentlichen im AB GT der Testersteller nach seinen Wünschen und im AB SO der Studienordnungs-Admin nach den Vorgaben der Studienordnung spezifiziert, sind nach entsprechendem Muster umzusetzen (vgl. Abschnitte 6.1.1.3 und 6.1.2.3).

#### 7.1.1.3 Eingaben der Benutzer

Im Anwendungsbereich Generierung von Tests erfolgen die Eingaben in Form von Constraints durch den Testersteller. Wie gezeigt wurde (vgl. Abschnitt 6.1.1.3), kann dies durch eine entsprechende Spezifikation – in ASP meist durch ein Fakt – sehr einfach realisiert werden.

Auch für den Administrator des Testmanagement-Systems ist eine einfache Konfiguration des Systems denkbar. Er hat den Fragenkatalog mitsamt den Attributen zu spezifizieren. Im Abschnitt 7.3 wird hierzu in Abbildung 7.4 ein möglicher einfacher Dialog gezeigt (vgl. auch Abschnitt 5.2.4). Mit Hilfe eines weiteren Fensters könnten dann die tatsächlichen „Fragen-Daten“ eingetragen werden.

Für den Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen kann Folgendes festgestellt werden: Die studienordnungsimmanenten Anforderungen, die der Admin realisieren muss, sind nach Abschnitt 6.1.2.3 gut zu realisieren. Ebenso sind sämtliche weitere relevanten Eingaben und Anfragen von Benutzern wie den Studierenden, dem SO-Komitee und dem Studiendekan anwendungsnah umzusetzen (vgl. Abschnitt 6.1.2.5).

### 7.1.2 Bearbeitung von Anfragen

#### 7.1.2.1 Repräsentation und interne Bearbeitung von Anfragen

Das in Kapitel 5 definierte Modell dient als Schnittstelle sowohl zur internen Repräsentation als auch zur Bedienoberfläche. Dadurch sind geeignete Bedienschnittstellen für sämtliche Benutzer des Systems denkbar. [1]

[1] Eine systematische Entwicklung entsprechender Bedienoberflächen wird im Rahmen dieser Arbeit jedoch nicht untersucht.

Wie eben schon erwähnt, lassen sich die Anfragen durch die Benutzer einfach repräsentieren. Als Kern der intelligenten Engine kann beispielsweise der ASP-Solver *smodels* fungieren. Als interne Antwort treten hierbei die stabilen Modelle bzw. Antwortmengen auf.

#### 7.1.2.2 Bewertung der Modelle

Da es zu einer Spezifikation oder Anfrage sehr viele Antwortmengen geben kann, stellt sich die Frage, ob eine gewisse Bewertung der Modelle sinnvoll ist und vorgenommen werden kann.

Im Anwendungsbereich Generierung von Tests beispielsweise benötigt der Prüfer in der Regel nicht alle Möglichkeiten für eine gültige Testzusammenstellung, sondern nur eine oder wenige hiervon. Welcher dieser gültigen Tests sollte bevorzugt werden? Hat der Testersteller keine weiteren „Wünsche“, so kann ein beliebiger Test aus der Menge aller Modelle ausgewählt werden. Sämtliche Lösungen sind daher gleichwertig. Möglich ist jedoch auch, dass der Testersteller manche Lösungen bevorzugen würde. Diese könnten möglicherweise in Form von weichen Constraints (Optimierung einer Zielfunktion) repräsentiert werden. Optimierungsconstraints (vgl. Abschnitt 5.2.2.5) und Anforderungen der Art „Nicht der gleiche Test wie das letzte Mal“ (vgl. Abschnitt 5.2.2.9) fallen in diese Kategorie und können mit den Mitteln von ASP+Gewichte umgesetzt werden (vgl. Abschnitt 6.1.1.3).

Ähnliche Überlegungen können für den Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen angestellt werden. Eine mögliche Zielfunktion könnte die Gesamtstudiendauer („maximale Semesteranzahl“) sein. Diese wäre – ähnlich wie bei den Optimierungsconstraints im AB GT – zu optimieren, also zu minimieren. Auf eine ausführliche Diskussion soll hier allerdings verzichtet werden.

#### 7.1.2.3 Beantwortung von Anfragen

Die gewählten Antwortmengen können geeignet nachbearbeitet werden, um die Antwort der Anfrage in eine gewünschte Form zu bringen. So sind beispielsweise oftmals nicht die Gesamtheit aller Modelle gewünscht, sondern nur eine oder mehrere Teilmengen hiervon. Darüber hinaus sind bezogen auf ein Modell in der Regel nicht alle Atome interessant. Eine geeignete Schnittstelle zur Ausgabe der Antwort kann realisiert werden.

### 7.1.3 Berücksichtigung eines generierenden Aspekts (Modelgenerating)

Typischerweise treten in beiden Anwendungsbereichen je Problembeschreibung mehrere Lösungen auf. Bei der Erstellung eines Tests sollte dieser auf Grundlage der Spezifikation erstellt, also *generiert* werden. Ebenso möchte möglicherweise ein Studierender wissen, welche Veranstaltungen er belegen kann. Auch hier spielt der Aspekt des *Generierens* von Lösungen eine wichtige Rolle.

Die Verwirklichung des generierenden Aspekts ist eine der Antwortmengenprogrammierung bzw. deren Systemen immanente Eigenschaft. Durch SMODELS können sämtliche Antwortmengen berechnet, also *generiert* werden.

### 7.1.4 Berücksichtigung eines prüfenden Aspekts (Modeltesting)

Werden zu einer gegebenen Spezifikation zusätzliche Anforderungen hinzugenommen, so schränkt dies den Lösungsraum, also die möglichen Antwortmengen ein. Durch diese in der Antwortmengenprogrammierung verwirklichte Grundidee können auch Prinzipien des Modeltesting auf sehr einfache Art und Weise realisiert werden: Eine mögliche Lösung wird zu einer gegebenen Spezifikation hinzugenommen und die zugehörigen Antwortmengen berechnet. Ergibt sich als einzige Antwortmenge der betrachtete Lösungskandidat, so ist dieser tatsächlich eine Lösung und ist in sich und zur Spezifikation konsistent. Ist dies nicht der Fall, so deutet dies auf Inkonsistenzen hin.

### 7.1.5 Umgang mit loser Spezifikation und vorgegebenen Teillösungen

Sind nicht alle Anforderungen eines Problems spezifiziert, so können trotzdem die Lösungen der Teilspezifikation gefunden werden. Werden weitere Constraints hinzugenommen, so schränkt dies den aktuellen Lösungsraum nur noch weiter ein. Bei vollständiger Spezifikation erhält man schließlich die gewünschten Lösungen.

Analog zum Modeltesting können Teillösungs-Kandidaten daraufhin überprüft werden, ob sie es auch tatsächlich sind: Ein Teillösungs-Kandidat wird der Spezifikation hinzugegeben. Liefert die Berechnung der Antwortmengen mindestens eine Lösung, so handelt es sich tatsächlich um eine Teillösung, ist also zur Spezifikation konsistent. Fehlende Anteile zu einer Gesamtlösung können den verschiedenen Antwortmengen entnommen werden.

### 7.1.6 Beachtung der strukturierten Domäne (Strukturproblem)

Im Abschnitt 6.1 des Transformations-Kapitels, vor allem in den Ausführungen zur Umsetzung der jeweiligen Strukturmodelle (vgl. Abschnitte 6.1.1.2, 6.1.2.2) ist gezeigt, dass sich die strukturellen Eigenschaften problemnah transformieren lassen. Die Möglichkeit der Definition rekursiver Regeln lässt eine einfache Berechnung transitiver Zusammenhänge zu.

Zu bemerken ist allerdings, dass die *smodels*-Darstellung auf Strings basiert. Die „getypte Struktur“ kann nur bedingt repräsentiert werden.

## 7.2 Nichtfunktionale Anforderungen und Zielkriterien

Im Folgenden wird eine Bewertung bezüglich der nichtfunktionalen Anforderungen und Zielkriterien aus den Abschnitten 2.5, 2.3.3 und 2.4.3 vorgenommen. Hierbei soll der verwendete logikbasierte Ansatz mit Constraints im Zentrum der Betrachtungen stehen.

### 7.2.1 Ausdruckskraft und Mächtigkeit

In Kapitel 5 wird ein Modell der betrachteten Anwendungsbereiche definiert und in Abschnitt 6.1 des Kapitels 6 die Transformation in einen logikbasierten Ansatz mit Constraints (Answer Set Programming mit Gewichten) vorgenommen.

#### 7.2.1.1 Informelle Überlegungen

Bei der Transformation hat es sich gezeigt, dass sämtliche relevanten Gegebenheiten der Anwendungsbereiche in „natürlicher Weise“ abgebildet werden können. Sowohl die strukturellen Eigenschaften der Domäne als auch die komplexen Anforderungen und Constraints können auf geeignete Art und Weise repräsentiert werden. Die Anforderungen an eine Lösung, welche durch eine Verknüpfung von Struktur und Constraints gegeben ist, können in der internen Repräsentation gut umgesetzt werden. Sämtliche relevante Anfragen und Konsistenzbedingungen sind abbildbar. Auch komplexe Abhängigkeiten zwischen Datenobjekten, welche vor allem für den Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen eine Rolle spielen, sind geeignet realisierbar. Es können genau jeweils sämtliche Lösungen des Problems gefunden werden (Vollständigkeit und Korrektheit).

Vor allem auch im Vergleich zu anderen Ansätzen wie beispielsweise dem pfadbasierten (vgl. Abschnitt 6.2) treten die Vorteile des logikbasierten Ansatzes mit Constraints mit großer Deutlichkeit hervor. Beispielsweise durch die in den Anwendungsbereichen auftretenden lokalen und globalen Gewichtssummen- und Anzahlconstraints stoßen andere Ansätze bezüglich der Ausdruckskraft schnell an ihre Grenzen. In den Ausführungen von Abschnitt 7.4.1 wird hierauf noch näher eingegangen.

Bei Betrachtung der zugehörigen theoretischen Grundlagen aus dem Abschnitt 4.2.2 wird deutlich, dass eine Ursache hierfür neben dem hohen Grad an Deklarativität hauptsächlich darin liegt, dass vermöge der Anzahl- und Gewichtsconstraint-Konstrukte „higher order-Kontexte“ repräsentierbar sind (Anlehnungen an Logik 2. Ordnung).

#### 7.2.1.2 Einige „formale“ Überlegungen

Es sollen hier nicht detaillierte Komplexitätsbetrachtungen angestellt werden, sondern nur Ansätze hierzu.

Um einen Anhaltspunkt über die *Komplexität des zu lösenden Problems* zu bekommen, werden folgende Betrachtungen angestellt, hier angelehnt an den Anwendungsbereich Generierung von Tests:

1. *Beispiel eines Teilproblems:* Gegeben ist eine endliche Menge $q_1, \ldots, q_n$ $(n \in \mathbb{N})$ von Fragen, denen jeweils eine Bearbeitungsdauer $a_1, \ldots, a_n$ zugeordnet ist.
2. *Frage:* Ein Test soll aus einer Teilmenge dieser Fragen bestehen und soll eine Gesamtbearbeitungsdauer von $b \in \mathbb{N}$ Minuten haben. Gibt es eine solche?

Dieses Teilproblem entspricht dem NP-vollständigen Rucksackproblem. Es können noch weitere ähnlich gelagerte Teilprobleme – auch im Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen – definiert werden. Dies deutet auf die NP-Härte des zu lösenden Problems hin. Trivialmethoden für die Lösung des Problems scheiden daher aus.

Ausführliche Komplexitätsbetrachtungen zur Logikprogrammierung [EFF+03, DEGV97, MT91] bzw. zur Antwortmengenprogrammierung [SNS02, Syr03] sind in der Literatur zu finden. Hieraus geht hervor, dass die Berechnung stabiler Modelle auch für einfache Problemklassen schon NP-vollständig ist. Ebenso ist die Berechnung stabiler Modelle grundierter Gewichtsconstraint-Regeln ohne Optimierungs-Statements (F)NP-vollständig. Sind allerdings Funktionen oder Optimierungs-Statements enthalten, so liegt die Berechnung der stabilen Modelle in einer „höheren“ Komplexitätsklasse.

Die Ausführungen deuten darauf hin, dass der gewählte Ansatz somit für die betrachteten Anwendungsbereiche eine geeignete Ausdruckskraft und Mächtigkeit besitzt.

### 7.2.2 Adäquatheit

Neben der eben diskutierten geeigneten Ausdruckskraft und Mächtigkeit des gewählten Ansatzes erscheint dieser aus folgenden Überlegungen auch adäquat zu sein:

Sowohl im Modell (vgl. Kapitel 5) als auch in der internen logikbasierten Repräsentation mit Constraints (vgl. Abschnitt 6.1) sind *kompakte Spezifikationen nahe an der informellen Beschreibung der betrachteten Anwendungsbereiche* möglich. Die beschriebenen Transformationen in Kapitel 6.1 zeigen den einfachen, „natürlichen“ Zusammenhang zwischen Modell und ASP-Darstellung.

Darüber hinaus sind *lose, unvollständige Spezifikationen* möglich (vgl. Abschnitt 7.1.5). Dies bedeutet, dass nicht alle Entscheidungen und Bedingungen vollständig bekannt sein und angegeben bzw.

spezifiziert werden müssen, um sinnvolle Berechnungen anstellen zu können. So kann – um nur ein Beispiel zu nennen – sich im AB SO ein Studierender im Rahmen einer Studienberatung trotzdem mögliche zu belegende Veranstaltungen anzeigen lassen, auch wenn er noch nicht alle Module gewählt hat. Ebenso kann die Festlegung eines sogenannten Schwerpunktes auch nach der teilweisen Belegung von entsprechenden Modulen und Veranstaltungen erfolgen. Hierzu gibt es – auch im AB GT – eine Vielzahl analoger Anwendungsfälle.

Gerade durch die Möglichkeit der unvollständigen Spezifikation sind Interaktionsmöglichkeiten realisierbar, was die Benutzbarkeit des Systems unterstreicht. So können beispielsweise zu einer unvollständigen Spezifikation die Lösungen (Antwortmengen) berechnet und dem Anwender ausgegeben werden. Durch zusätzliche Angaben (Entscheidungen, Bedingungen) des Benutzers kann der Lösungsraum weiter eingeschränkt werden. Entsprechende Ergebnisse können wiederum an den Anfragesteller ausgegeben werden. Wiederholtes Interagieren liefert geeignete Anfrageergebnisse.

Als weiterer Aspekt ist die *Benutzerfreundlichkeit* zu nennen. Gerade im AB GT wurde auf die Möglichkeiten zum Erkennen und Lokalisieren von Inkonsistenzen hingewiesen. Entsprechendes Vorgehen ist auch im AB SO möglich.

### 7.2.3 Erweiterbarkeit und Modifizierbarkeit

Die Systeme sind auf Erweiterbarkeit hin angelegt. Die Repräsentationen besitzen ein hohes Maß an *Modularität*. Unter anderem zeigt sich dies in der Kernstruktur der Modelle (vgl. Abschnitt 5.1), wobei eine Einteilung in Struktur-, Constraint- und Lösungsmodell vorgenommen werden kann. Eine *weitgehende Lokalität* bei Änderungen und Erweiterungen kann festgestellt werden. Allerdings sind diese Eigenschaften aufgrund gewisser Abhängigkeiten zwischen den Daten bzw. Repräsentationseinheiten in natürlicher Weise eingeschränkt. Dies kann sich vor allem bei konzeptuellen Änderungen bemerkbar machen. Detaillierter wird dies in nachfolgenden Ausführungen diskutiert.

#### 7.2.3.1 Einfüge-Operationen (Insert)

**Modell** Das jeweilige *Strukturmodell* kann bei Einfüge-Operationen durch einfache Anpassungen im entsprechenden Objektmodell aktualisiert werden.

Werden beispielsweise im AB GT neue Fragen erstellt, so sind lediglich im Objektdiagramm die neuen Fragenobjekte hinzuzufügen und durch Instanzen der Assoziation $prop$ mit weiteren Attributobjekten zu verknüpfen. Wird die Struktur eines hierarchischen Attributs erweitert, so sind zusätzliche Attributobjekte im Hierarchiegraphen zu ergänzen. Die Instanz der Assoziation $contain$ ist einfach durch entsprechende Tupel zu ergänzen.

Analoges kann für den AB SO festgestellt werden. Entsprechende Elemente sind dem Strukturgraphen hinzuzufügen und die Instanzen der Assoziationen (z. B. $contain$, $require$) einfach zu ergänzen.

Da die Constraints der Spezifikation eines Tests beziehungsweise eines gültigen Studiums implizit konjunktiv verknüpft sind, können Insert-Änderungen im jeweiligen *Constraintmodell* in der Regel einfach durch Hinzunahme von konkreten Constraint-Spezifikationen vorgenommen werden.

Zusätzliche Elemente bei den Instanzen des Struktur- bzw. Constraintmodells führen natürlicherweise zu einer impliziten Veränderung des Lösungsraums.

**Interne Repräsentation (ASP+Gewichte)** In Abschnitt 6.1 ist ausführlich gezeigt, wie entsprechende Modelle in den internen Formalismus ASP+Gewichte transformiert werden können. Es ist leicht ersichtlich, dass eben angesprochene Einfüge-Operationen in der Regel durch Hinzunahme einfacher Fakten realisiert werden können.

#### 7.2.3.2 Lösch-Vorgänge (Delete)

Sowohl im Modell als auch in der ASP-Repräsentation sind Löschvorgänge in analog einfacher Weise durchzuführen wie die eben erläuterten Insert-Operationen: Die entsprechenden Objekte sind samt den zugehörigen Elementen der Assoziations-Instanzen im Strukturmodell bzw. den zugehörigen ASP-Fakten in der internen Darstellung einfach zu entfernen.

#### 7.2.3.3 Aktualisierungen im Datenbestand (Update)

Typische Aktualisierungen im AB GT sind Veränderungen in der Relation $prop$, also in der Zuordnung von Fragen zu ihren direkten Attributwerten oder Veränderungen in der Struktur der Domänen der Attribute, also in einer Assoziation $contain$. Im Modell sind hierbei lediglich die entsprechenden Tupel anzupassen. Dies überträgt sich in eine einfache Modifikation der Fakten in der ASP-Repräsentation. Im AB SO sind einfache strukturelle Änderungen im Datenbestand in analoger Weise durchzuführen.

Aktualisierungen in den Constraints sind jeweils durch einfache, lokale Anpassungen der Constraint-Spezifikationen in beiden Modellen vorzunehmen.

#### 7.2.3.4 Konzeptuelle Änderungen

Die bisher besprochenen Änderungen weisen ein sehr hohes Maß an Lokalität auf. Dies gilt für konzeptuelle Änderungen in eingeschränkter Weise, wie folgende Ausführungen zeigen:

**Modell** Die abstrakten Modelle sind Templates für eine Vielzahl konkreter Modelle. Relevante konzeptuelle Änderungen wirken sich in der Regel nur auf die konkreten Modelle und nicht auf die abstrakten Modelle aus.

Im AB GT sind konzeptuelle Änderungen in den *konkreten Strukturmodellen* typischerweise die Hinzu- oder Wegnahme von Attributen und die veränderte Strukurierung der Domänen der Attribute. Beispielsweise könnte ein zusätzliches Attribut `Security relevance` (Sicherheitsrelevanz) zur Beschreibung der Fragen dienen oder es könnte das flache Attribut `Topic` (Thema) durch eine Hierarchie strukturiert werden.

Wie in Bemerkung 5.2.18 schon angedeutet, müssen bei Veränderungen in der Anzahl der Attribute Anpassungen in sämtlichen Tupeln der Assoziation $prop$ vorgenommen werden. Bezüglich der Constraints sind bei der impliziten Spezifikation von Constraintreferenzmengen schließlich auch entsprechende Anpassungen bei den Spezifikationstupeln vorzunehmen (vgl. auch Abschnitt 5.2.2.8).

Wird jedoch nur die Struktur der Domäne eines Attributs verändert, so sind weitgehend lokale Änderungen möglich. Beispielsweise kann das flache Attribut `Topic` einfach durch die zusätzliche Definition einer entprechenden Assoziation $contain_{Topic}$ mit den zugehörigen Instanzen strukturiert werden und damit in ein hierarchisches Attribut verwandelt werden.

Im AB SO sind typische strukturelle Änderungen in der Hinzu- oder Wegnahme von Strukturtypen ($Structuretype^{cl}$) zu sehen. Im konkreten Klassendiagramm kann dies meist durch eine einfache, lokale Änderung geschehen. Ebenso sind die Anpassungen im Strukturobjektmodell durch mehr oder weniger starke lokale Änderungen durchzuführen. Bei der Spezifikation der Constraints können die aktualisierten Strukturtypen auftreten. Sollen bei der impliziten Spezifikation der Constraintreferenzmengen nach anderen als den angegebenen Eigenschaften selektiert werden, so müssen diese bei der Spezifikation Berücksichtigung finden.

**Interne Repräsentation (ASP+Gewichte)** Aus der Beschreibung der Transformation der Modelle in die interne Repräsentation und die eben dargestellten erforderlichen Anpassungen im Modell ergeben sich die entsprechenden notwendigen Änderungen bei den Regeln.

Beispielsweise wird bei der Veränderung der Anzahl der Attribute im AB GT die Stelligkeit von prop verändert. So müssen einerseits sowohl sämtliche Tupel von prop als auch die Regeln angepasst werden, in denen das Prädikat prop auftritt. Eine Veränderung der Stelligkeit von prop zieht eine „simultane“ Veränderung der Stelligkeit von propTrans und weiteren „direkt abhängigen“ Prädikaten nach sich. Dies trifft auch auf die Regeln zur Spezifikation und Behandlung der Constraints zu.

Veränderungen in der Struktur einer Domäne sind durch Hinzu- oder Wegnahme von contain-Fakten und zugehörigen transitiven Regeln zu bewerkstelligen.

Im AB SO sind die Objekte neuer Strukturtypen durch zusätzliche elementProp-Fakten darzustellen, so dass diese Änderung im Wesentlichen lokal durchgeführt werden kann.

Soll bei den Constraints nach anderen als den schon besprochenen Eigenschaften selektiert werden, so sind Anpassungen in der Stelligkeit von elementProp und daher sämtlichen abhängigen Regeln vorzunehmen.

### 7.2.4 Integrierbarkeit

Der verfolgte Ansatz beinhaltet *geeignete Schnittstellen zur Integrierbarkeit in Informationssysteme*. Dies wird schon in der Beschreibung des Lösungsansatzes (vgl. Kapitel 3) deutlich: Das Modell stellt eine zentrale Schnittstelle sowohl zur internen Repräsentation als auch zur Bedienoberfläche dar. Hierdurch ist eine Offenheit zu verschiedenen Systemen (interne Formalismen, Benutzer-GUIs) gegeben. Der verfolgte Ansatz ist auf *Plattformunabhängigkeit* ausgelegt.

### 7.2.5 Performanz und Skalierbarkeit

Für beide Anwendungsbereiche sind umfangreiche Laufzeitmessungen durchgeführt worden. Im Folgenden sollen die wesentlichen Ergebnisse gezeigt werden:

#### 7.2.5.1 Generierung von Tests

Um einen Vergleich mit einem anderen logikbasierten Programmierparadigma anstellen zu können, wurde eine Vergleichsimplementierung mit dem CLP-Solver ECLiPSe (vgl. [ACD+], [Pir06]) durchgeführt. Einige Ergebnisse sind schon in [SF06] gezeigt.

**Einige Parameter** Unter anderem wurden folgende Parameter bei den Laufzeittests variiert:

1. Größe des Fragenpools
2. Anzahl der auszuwählenden Fragen
3. Anzahl der Attribute
4. Anzahl der Attributwerte eines Attributs
5. Anzahl der mehrwertigen Attributwerte
6. Größe der Hierarchie eines Attributs mit strukturierter Domäne
7. Anzahl der Anzahlconstraints
8. Anzahl der prozentualen Constraints

**Wichtigste Ergebnisse** Vorwegnehmend sei erwähnt, dass natürlich die gemessenen Laufzeiten von der gewählten Implementierung abhängen können. Das asymptotische Laufzeitverhalten von SMODELS ist bei den meisten Parametern unkritisch; in vielen Fällen ist keine Abhängigkeit bzw. eine lineare Abhängigkeit zu beobachten. Hingegen ist eine exponentielle Abhängigkeit in der Größe des Fragenpools festzustellen. Trotzdem scheint die SMODELS-Implementierung für die Praxis einsetzbar zu sein. Realistische industrielle Fragenpools enthalten bis zu 2000 oder 3000 Fragen. Im universitären Umfeld sind es meist weniger, dafür aber speziellere Fragen im Fragenpool. Große Fragenpools können oftmals in kleinere, disjunkte Teilmengen eingeteilt werden, so dass sich bestimmte Anfragen auf diese Teilmengen beziehen können.

Bei einem Fragenpool mit 3000 Fragen wurde eine Laufzeit von wenigen Sekunden im Fall von flachen, nicht-hierarchischen Attributen bis zu wenigen Minuten im Fall von mehrwertigen, hierarchischen Attributen gemessen.

Vergleicht man das Laufzeitverhalten der vorhandenen SMODELS- mit der ECLIPSE-Implementierung, so ist festzustellen, dass ECLIPSE bei großen Fragenpools und bei einer großen Hierarchietiefe vorteilhafter erscheint. Hingegen hat bei mehrwertigen Attributen, bei einer großen Anzahl von Attributen und bei einer großen Anzahl von Attributwerten die verwendete SMODELS-Implementierung bessere Eigenschaften.

Der Einsatz als Just-in-Time-Applikation ist in der Regel daher nicht anzuraten, jedoch ist dies auch oftmals gar nicht notwendig und erwünscht.

#### 7.2.5.2 Studienberatung und Konsistenzprüfung von Studienordnungen

Zahlreiche Testreihen wurden mit variierenden Parametern durchgeführt. Sie wurden durchgeführt mit Rechnern, welche mit Intel Core 2 Duo-2,13-GHz-Prozessoren, 2-Gigabyte-Arbeitsspeicher und Windows XP Professional SP2 ausgestattet sind. Zur Errechnung der Antwortmengen wurden LPARSE 1.0.4 und SMODELS 2.26 verwendet. Ausgangspunkt der Laufzeittests bildeten zwei Standardtestfälle. Es wurden jeweils (maximal) 200 Modelle errechnet.

**Standardkonfigurationen** Die realen, modularen Studienordnungen unterscheiden sich stark in den Entscheidungsmöglichkeiten. Manche Studienordnungen geben kaum Freiheit für eigene Wahlen des Studierenden, manche hingegen bieten diesbezüglich eine Vielzahl von Möglichkeiten. Aus diesem Grunde wurden zwei Standardtestfälle gewählt:

- STANDARDTESTFALL A (ST-A): Die Konfiguration der Parameter orientiert sich an Studienordnungen mit einem hohen Maß an Wahlmöglichkeiten. Als Beispiel dient der derzeitige Bachelorstudiengang European Studies der Universität Passau [SoP].
- STANDARDTESTFALL B (ST-B): Die Konfiguration der Parameter orientiert sich an Studienordnungen mit einem geringen Maß an Wahlmöglichkeiten, wie es beispielsweise beim derzeitigen Bachelorstudiengang Informatik der Universität Passau [SoP] der Fall ist.

**Ergebnisse aus den Laufzeittests** Anzumerken ist, dass es zwischen einigen Paramatern natürlicherweise auch Abhängigkeiten gibt, beispielsweise zwischen der Anzahl der $Coursetemplate$-Strukturobjekte und der Anzahl der konkreten Lehrveranstaltungen oder auch zwischen der Tiefe des Strukturgraphen und der Anzahl aller Strukturelemente $|Structuretype^{ob}|$. Hier wurden sinnvolle Anpassungen vorgenommen. Gemessen wurde sowohl die Laufzeit des Parsers LPARSE als auch der des eigentlichen Reasoners zur Berechnung der Antwortmengen SMODELS (vgl. Abschnitt 4.2.2.6).

1. Parameter mit Bezug zum Modulkatalog-Objektgraphen:
   (a) ERGEBNIS: Exponentielle Abhängigkeit der Laufzeit:
      - Anzahl der konkreten Lehrveranstaltungen bis 600 (ST-A) bzw. 1200 (ST-B) (vgl. Abbildung 7.1)
      - Anzahl der $Coursetemplate$-Objekte bis 300 (ST-A) bzw. 450 (ST-B)
      - Tiefe des Strukturgraphen bis 11 (ST-A) bzw. 15 (ST-B)

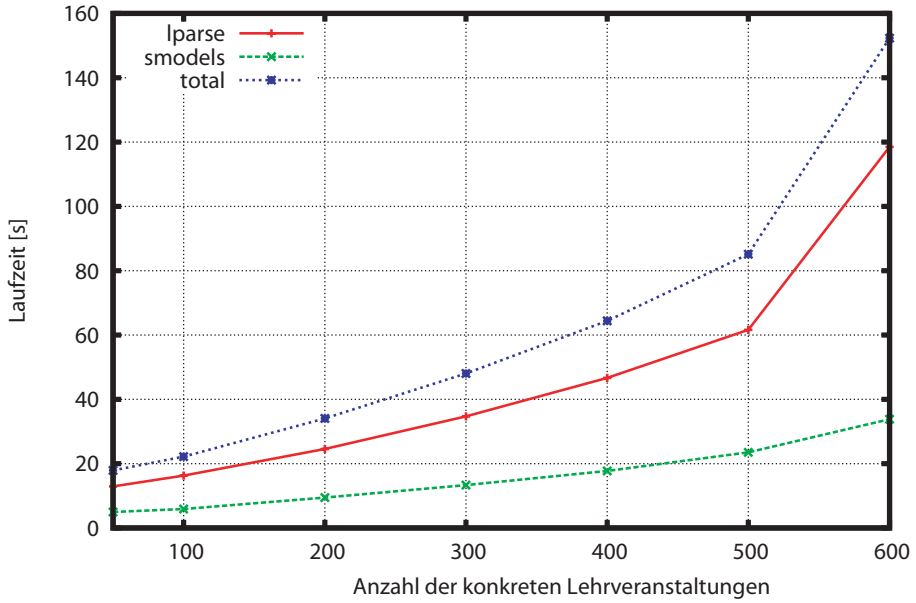
   (b) ERGEBNIS: Lineare Abhängigkeit:
      - Stärke der Mehrwertigkeit

   (c) ERGEBNIS: Keine Abhängigkeit:
      - Prozentsatz aller Strukturelemente, die eine Voraussetzens-Bedingung enthalten

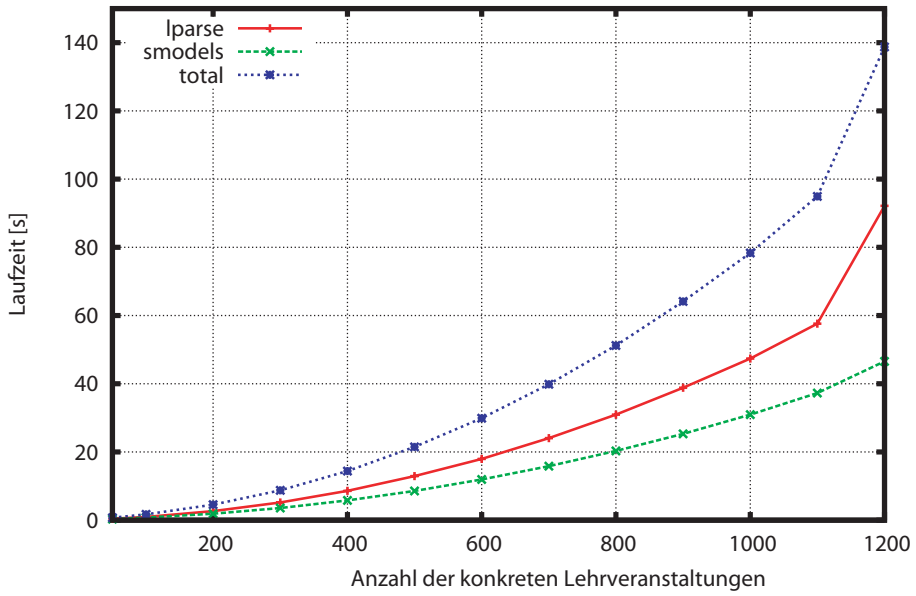
2. Parameter mit Bezug zum Constraintmodell
   (a) ERGEBNIS: Keine oder lineare Abhängigkeit
   (Gemessene Größenordnung: Faktor 100 - 1000 verglichen mit realistischen Größen):
      - Prozentsatz von obligatorischen $Coursetemplate$-Objekten
      - Anzahl der (harten) Anzahlconstraints
      - Anzahl der (harten) Gewichtssummenconstraints (vgl. Abbildung 7.2)
      - Anzahl der (harten) Successorconstraints mit Anzahlauswahl
      - Anzahl der (harten) Successorconstraints mit Gewichtssummenauswahl

   (b) ERGEBNIS: Lineare bzw. exponentielle Abhängigkeit
   (Gemessene Größenordnung: Faktor 100 verglichen mit realistischen Größen)
      - Anzahl der Auswahlconstraints (vgl. Abbildung 7.3)

3. Parameter mit Bezug zum Studierenden
   (a) ERGEBNIS: Keine Abhängigkeit
      - Fortschritt (Semesteranzahl) des Studenten

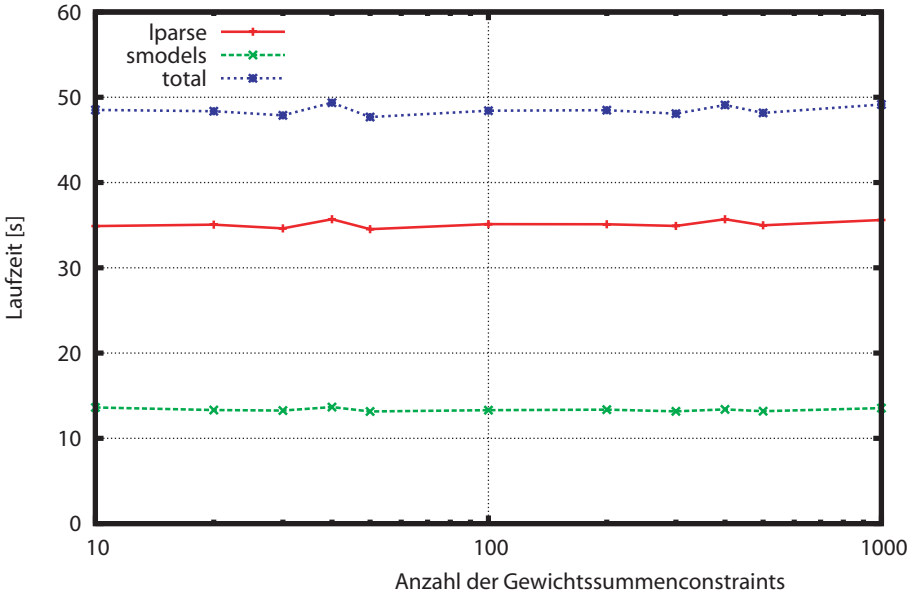



(a) Ausgangspunkt: Konfiguration ST-A

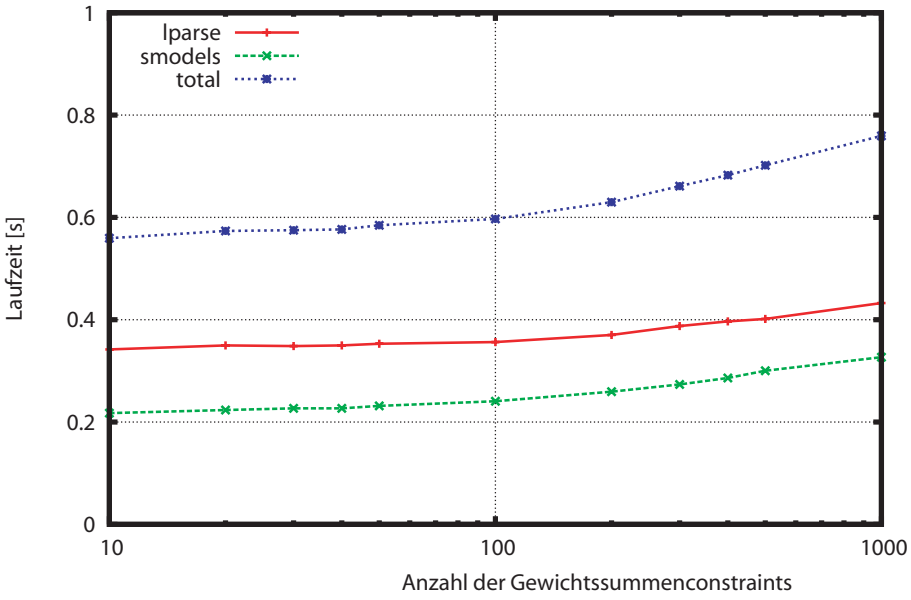



(b) Ausgangspunkt: Konfiguration ST-B

Abbildung 7.1: Laufzeit bei Variation der Anzahl konkreter Lehrveranstaltungen

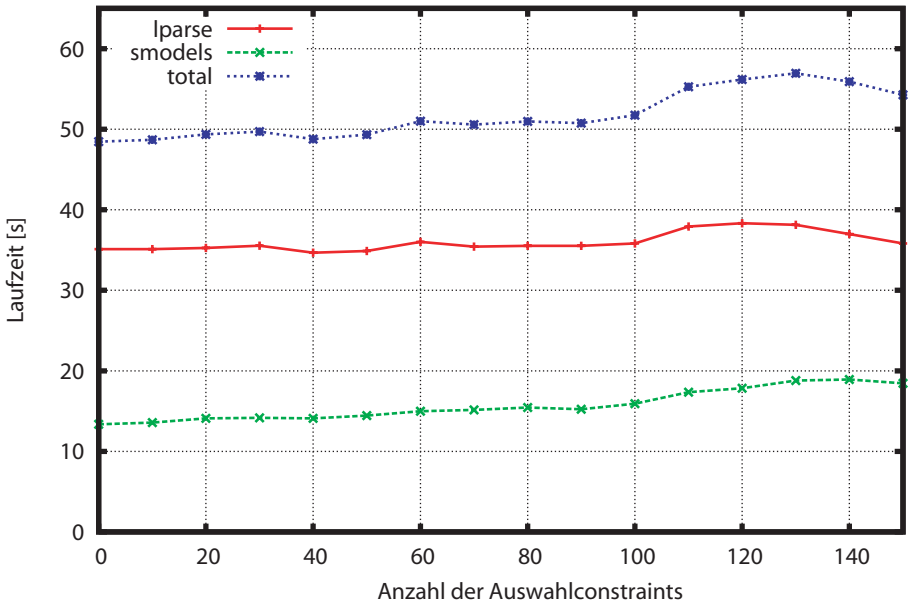



(a) Ausgangspunkt: Konfiguration ST-A

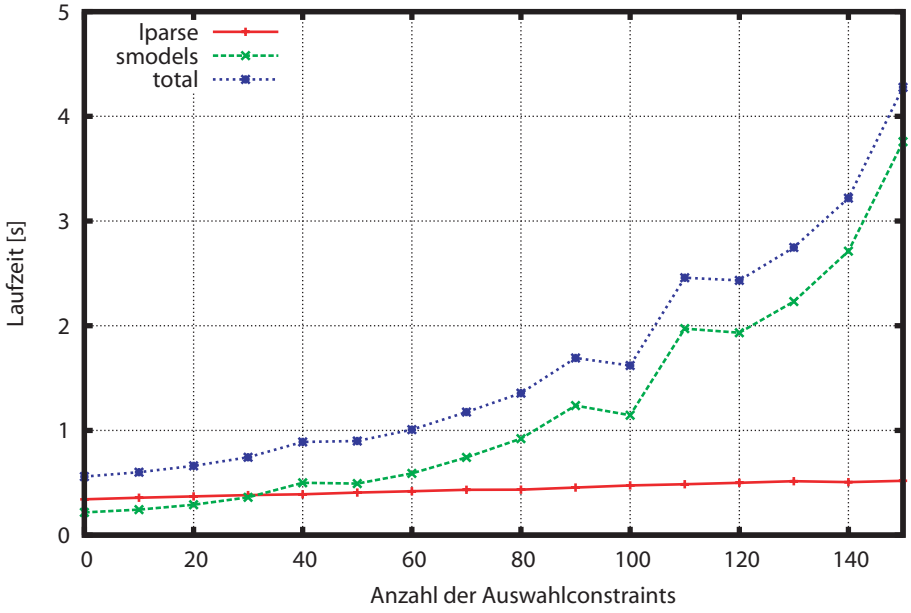



(b) Ausgangspunkt: Konfiguration ST-B

Abbildung 7.2: Laufzeit bei Variation der Anzahl der (harten) Gewichtsconstraints




(a) Ausgangspunkt: Konfiguration ST-A




(b) Ausgangspunkt: Konfiguration ST-B

Abbildung 7.3: Laufzeit bei Variation der Anzahl der Auswahlconstraints

**Zusammenfassung** Die Laufzeittests zeigen unter anderem:

1. Bei den realistischen Standardkonfigurationen variieren die Laufzeiten von einer halben Sekunde im einfacheren Fall bis zu wenigen Sekunden im komplexeren Fall.

2. Sämtliche gemessene Laufzeiten liegen im Bereich von Millisekunden bis maximal 200 s.

3. In den meisten Testfällen hat der Parser LPARSE im Gegensatz zu SMODELS an der Gesamtlaufzeit den größeren Anteil. Eine geeignete Vorgrundierung der Programme würde daher die Gesamtlaufzeit entscheidend optimieren.

4. Vor allem bei der Variation der Variablen des Strukturmodells, zum Beispiel der Anzahl der konkreten Lehrveranstaltungen, wurde die obere Messgrenze bei den Benchmarks durch Speicherplatzprobleme eingeschränkt, wobei mindestens die Hälfte des Speicherplatzes durch das Java-Benchmark-Programm ohne den eigentlichen Reasoner LPARSE+SMODELS okkupiert war. Trotzdem waren auch hier Messungen mit Problemgrößen, die den Faktor 2 - 10 im Vergleich zu realistischen Größen hatten, noch gut möglich.

5. Die Variation der Parameter mit Bezug zum Constraintmodell oder zum Studierenden im Bereich realistischer Größen wirkt sich kaum nachteilig – in vielen Fällen nur im Bereich von Millisekunden – auf die Gesamtlaufzeit aus.[2]

6. Erkennbar ist, dass Parameter mit Bezug zum Constraintmodell oder zum Studierenden einen wesentlich geringeren Einfluss auf die Laufzeitfunktion haben, als Parameter mit Bezug zum Strukturmodell. Hier ist teilweise exponentielles Verhalten zu beobachten. Dies zeigt, dass hier die „Graphenproblematik“ überwiegt. Zu bedenken ist auch, dass sich die Anzahl der Lösungen bezüglich eines Modulkatalog-Objektgraphen im Wesentlichen durch das *Produkt* der Anzahl der Teillösungen von entsprechenden Teilbäumen(-graphen) ergeben. Hierdurch ist ein exponentielles Anwachsen der Anzahl der Lösungen mit der „Größe“ des Strukturgraphen gegeben.

7. Zu berücksichtigen ist ferner, dass im Rahmen der Benchmarks die Studienordnungen automatisiert und „zufällig“ erzeugt worden sind. Das Optimierungspotential konnte hier noch nicht vollständig ausgeschöpft werden.

Als Folgerungen für mögliche Optimierungen können folgende Maßnahmen ergriffen werden:

1. Vorgrundieren zur Minimierung der Laufzeit des Parsers

2. Auswertung ausschließlich der relevanten Regeln des Constraintmodells

[2] Zu beachten ist, dass die Messungen zum Teil in Größenordnungen stattgefunden haben, die den Faktor 100 - 1000 im Vergleich zu realistischen Größen hatten.

3. Einschränkung der Berechnungen auf den aktuell relevanten Teilgraphen des Modulkatalog-Objektdiagramms.[3]

4. Vorberechnen und Tabellieren, Berechnung von „Teilmodellen“ statt des gesamten Modells und Berücksichtigung von aktuell relevanten Teillösungen.

Die Realisierung eines Demonstrators zur automatisierten Studienberatung mit ASP (vgl. [Mal07]) hat gezeigt, dass unter Berücksichtigung wesentlicher Optimierungspotentiale bezüglich der zu erwartenden Laufzeit eine interaktive Handhabung und Nutzung gut möglich ist.

Zusammenfassend lässt sich also feststellen, dass sich im Anwendungsbereich Studienberatung und Konsistenzprüfung von Studienordnungen adäquate Antwortzeiten bei der Anfragebearbeitung bei realistischen Datenmengen und akzeptable Laufzeitfunktionen bei der Variation im Datenvolumen ergeben.

### 7.2.6 Programmverständnis und Wartbarkeit

Die Diskussionen zur Repräsentation der Daten, zur Adäquatheit und zur Erweiterbarkeit und Modifizierbarkeit zeigen, dass eine *gute Handhabbarkeit* des Programms bzw. Systems realisierbar ist. Durch die vorgenommene genaue und kompakte Beschreibung der Transformation der Modelle in eine entsprechende interne ASP-Repräsentation wird dies unterstrichen. Der System- und Programm-Experte bzw. der Administrator kann sich mit Hilfe der Ausführungen im Rahmen dieser Arbeit *leicht* in das Programm *einarbeiten*. Dadurch ist auch eine *gute Wartbarkeit* des Systems zu gewährleisten.

### 7.2.7 Relevanz

Die Internationalisierung der akademischen Bildung, die Modularisierung der Studienordnungen und die damit einhergehenden zunehmenden Erfordernisse der Studienberatung machen die Notwendigkeit und Relevanz des Anwendungsbereichs Studienberatung und Konsistenzprüfung von Studienordnungen und damit eines StARC-Systems deutlich (vgl. Abschnitt 2.1).

Durch die zunehmende Bedeutung der Bildung in der Gesellschaft sind Lern- und Bildungsangebote ständig weiterzuentwickeln. Trainingsprogramme in Universitäten und Industrie sind durch automatisierte Systeme wie Testmanagement-Systeme zu unterstützen. Die erfolgreiche Integration der *smodels*-Lösung in ein reales System, nämlich in das Prüfungsmanagementsystem der Deutschen Bahn zur Schulung der Lokführer [IFI], unterstreicht die Relevanz und Praxistauglichkeit des eigenen Ansatzes.

[3] Die Anfragen beziehen sich höchstens auf den Modulkatalog-Objektgraphen einer einzigen Studienordnung. Anfragen über mehrere Modulkatalog-Objektgraphen sind in der Regel nicht notwendig. Darüber hinaus kann in vielen Fällen eine Einschränkung der Berechnungen auf einen Teilgraphen erfolgen, was einen wesentlichen Beitrag zur Laufzeitoptimierung liefert.

## 7.3 Bedienung der Systeme und Qualifikation der Benutzer

Im Folgenden soll ein grober Überblick über einige mögliche Bedienabläufe und die damit verbundene notwendige Qualifikation der Benutzer gegeben werden (vgl. auch Abbildungen 2.5 und 2.10).[4]

### 7.3.1 Testmanagement-System

Die Grundlage hierzu ist ein „intelligentes System“, in welchem ein großer Pool von Fragen geeignet verwaltet werden kann. Typische Benutzer sind neben dem Administrator der Ersteller von Fragen, der Trainer und der Lerner. Natürlich ist eine Personalunion bei den verschiedenen Rollen möglich.

#### 7.3.1.1 Administrator

Der Administrator hat ein konkretes konzeptuelles Modell umzusetzen oder dieses zu definieren. Hierzu sind die Attribute und die auftretenden Constraintarten zu spezifizieren. Abbildung 7.4 zeigt hierzu einen vereinfachten, möglichen Dialog.

Auch für Änderungen am konzeptuellen Modell sind Eingabemöglichkeiten zu schaffen.

#### 7.3.1.2 Ersteller von Fragen

Für die Behandlung des bei der Testerstellung zugrundeliegenden informatischen Problems ist die eigentliche Ausprägung der Frage nicht relevant (vgl. Bemerkungen 2.3.3 und 5.2.1). Nichtsdestotrotz hat ein Ersteller von Fragen diese festzulegen (vgl. Abbildung 7.5).

Darüber hinaus sind die Fragen den Attributwerten zuzuordnen und die Hierarchien der strukturierten Domänen hiearchischer Attribute zu definieren (vgl. Abbildung 7.6). Die Angabe der möglichen Attributwerte könnte in einem eigenen Dialog oder implizit bei der Festlegung der Fragen-Eigenschaften erfolgen.

#### 7.3.1.3 Trainer und Lerner

Der Trainer, Prüfer und gegebenenfalls der Lerner möchten einen Test konfiguieren. Hierzu sind die an ihn gestellten Bedingungen zu spezifizieren. Ein entsprechendes Interface könnte wie in Abbildung 7.7 aussehen (vgl. auch Abschnitt 5.2.4). Durch explizite Spezifikation von Fragen, die im Test enthalten sein sollen, können (Teil-)Lösungen vorgegeben werden. Implizit kann hiermit eine Konsistenzprüfung durchgeführt werden.

[4] Die gezeigten Darstellungen der Bedienoberflächen zeigen lediglich einen ersten einfach Ansatz zu deren Definition und dienen lediglich der Veranschaulichung. Eine systematische Entwicklung der graphischen Bedienoberflächen (GUIs) wird im Rahmen dieser Arbeit nicht behandelt. In professionellen Systemen können die GUIs natürlich eine andere Gestalt haben.

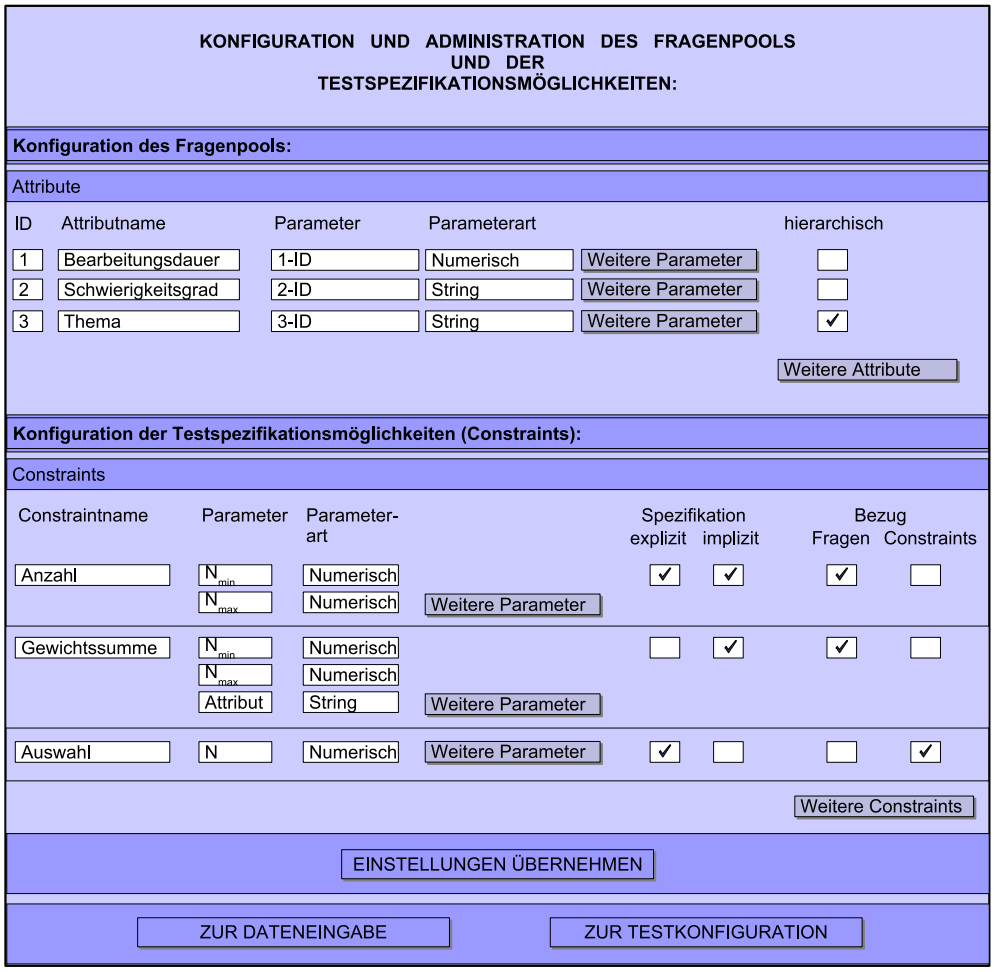



Abbildung 7.4: Einfacher Dialog für den Admin zur Umsetzung des konkreten Modells

Der Trainer bzw. der Lerner möchte (mindestens) einen gültigen Test erhalten. Auch hierzu muss es eine entsprechende Ausgabemöglichkeit geben. Der Lerner erhält schließlich einen ausgewählten Test zur Bearbeitung.

### 7.3.2 StARC-System

Vom Grundprinzip her ist die Bedienung des StARC-Systems ähnlich der des Testmanagement-Systems. Für sämtliche Benutzer sind geeignete Dialoge erforderlich.

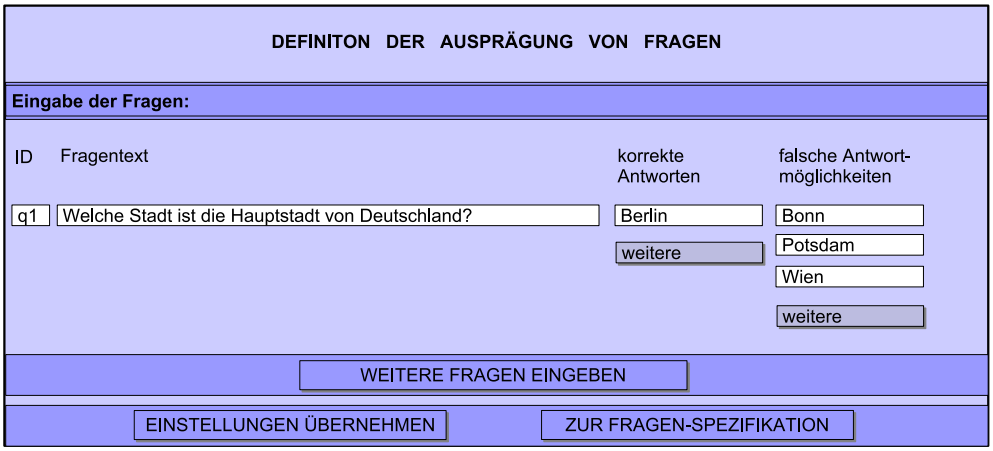



Abbildung 7.5: Einfaches Fenster zur Definition der Ausprägung von Fragen

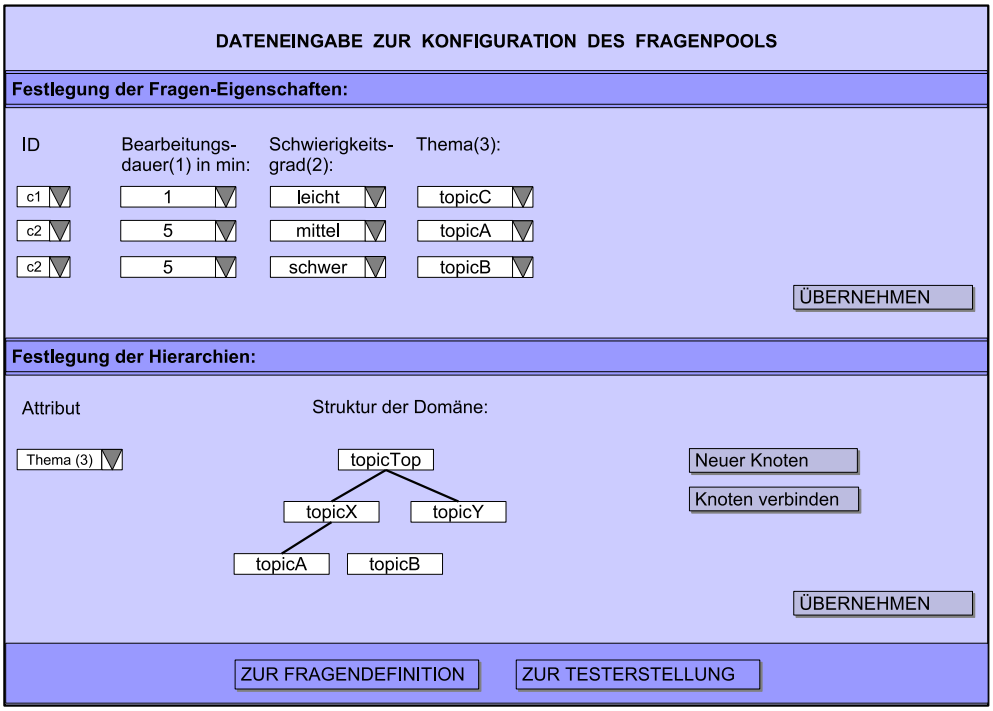



Abbildung 7.6: Dateneingabe zur Definition der Eigenschaften von Fragen

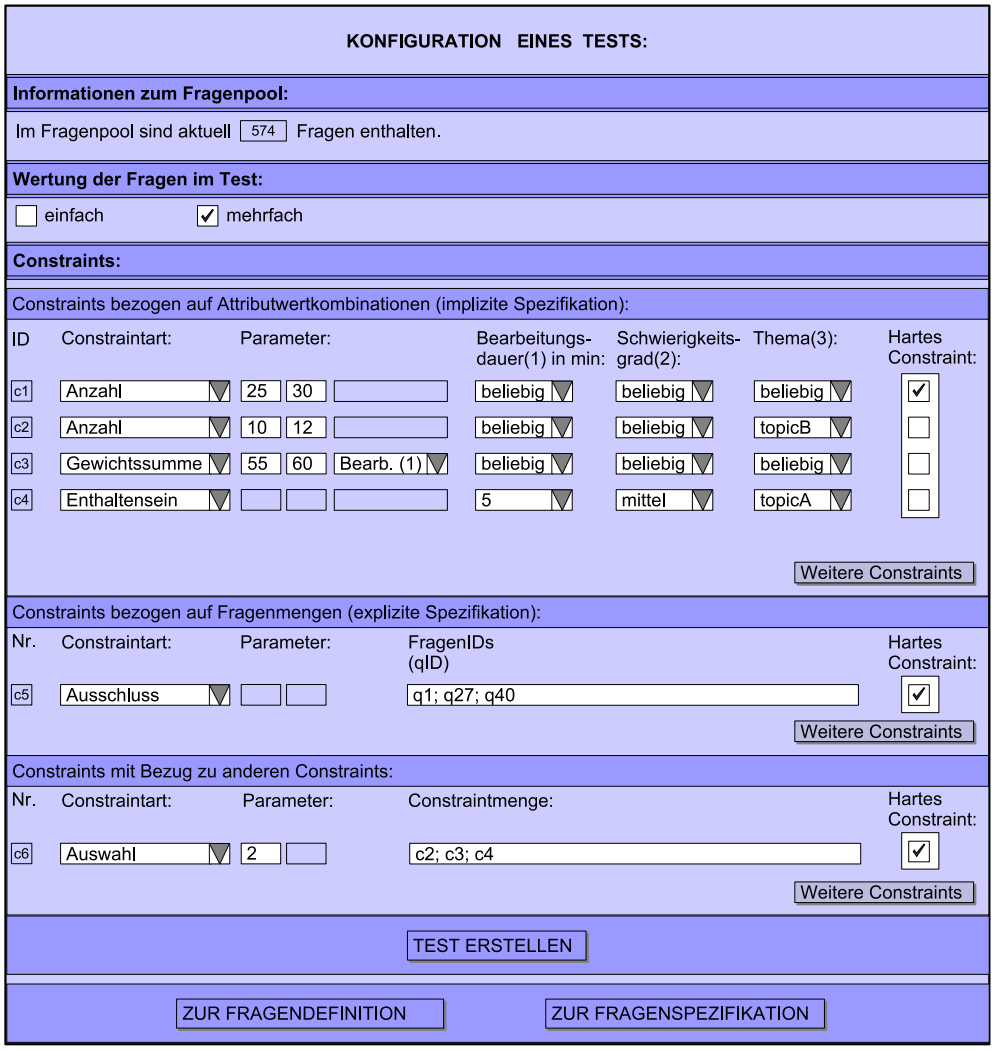



Abbildung 7.7: Bedienoberfläche zur Konfiguration eines Tests

#### 7.3.2.1 Administrator

Der Administrator legt mit Hilfe eines Bedieninterfaces für eine Studienordnung die Strukturtypen und den Modulkatalog-Objektgraphen samt sämtlicher Strukturelemente fest. Darüber hinaus müssen die zu der Studienordnung gehörigen komplexen Bedingungen und Constraints geeignet spezifiziert werden.

Gibt es Änderungen in der Studienordnung, so werden mit Hilfe eines entsprechenden Dialogs An-

passungen an der Spezifikation vorgenommen bzw., falls die alte Studienordnung noch für eine Übergangszeit gültig sein soll, eine neue Version hierzu erstellt.

Darüber hinaus kann der Administrator die konkreten Lehrveranstaltungen pflegen. Diese werden durch bestimmte Eigenschaften (z. B. SWS, ECTS, Art) ausgezeichnet und den Veranstaltungstemplates der Studienordnung zugeordnet.

Denkbar ist jedoch auch, dass die Dozenten für die eigenen Veranstaltungen die Verwaltung, Zuordnung und Pflege übernehmen.

#### 7.3.2.2 Studienordnungs-Komitee

Bezüglich der Fragestellung, ob der Entwurf einer Studienordnung konsistent ist, könnte der Dialog ähnlich dem des Administrators sein. Zusätzlich könnte ein „Entwurfsmodus“ sinnvoll sein (vgl. Abbildung 7.8). Möglicherweise gibt auch das Studienordnungs-Komitee den Auftrag an den Administrator, die Eingaben bezüglich des SO-Entwurfes zu machen.

#### 7.3.2.3 Studiendekan

Die Fragestellungen des Studiendekans betreffen typischerweise das zu einem Studiengang angebotene konkrete Lehrangebot, etwa „Reicht das konkrete Lehrangebot aus, damit der Studiengang studierbar ist?“ oder „Welche Möglichkeiten sinnvoller Lehrangebote für die nächsten Semester gibt es?“.

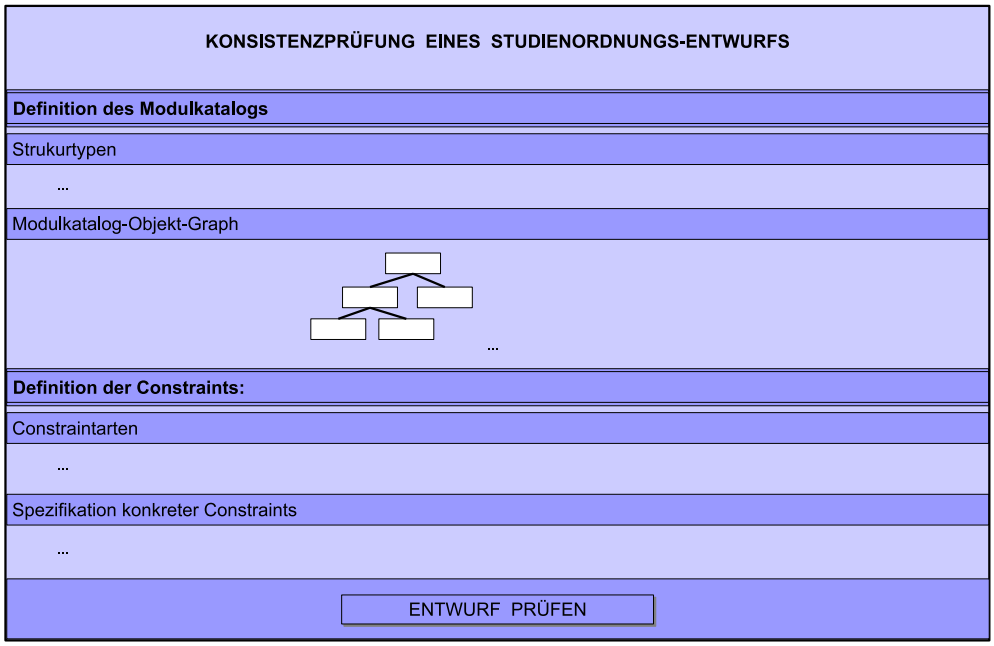
In Abschnitt 5.3.3 wird diskutiert, wie diese Fragestellungen technisch umzusetzen sind. Das Bedieninterface sollte diesen Möglichkeiten Rechnung tragen. Hierzu sind Eingabemöglichkeiten für aktuelle oder geplante konkrete Lehrveranstaltungen und verschiedene Modi der Berechnung je nach zu beantwortender Fragestellung vorzusehen. Ist bei der ersten Fragestellung das konkrete Lehrangebot nicht ausreichend, so sind oftmals Hinweise auf fehlende Veranstaltungen erwünscht. Eine entsprechende Ausgabe kann erstellt werden. Bei der zweiten Fragestellung ist als Antwort mindestens eine Möglichkeit anzuzeigen.

Interessiert sich der Studiendekan für mögliche Studienpläne, so sind die Veranstaltungen unter Berücksichtigung der partiellen Ordnung, die sich aus den Voraussetzungen und den zeitlichen Empfehlungen ergibt, anzuzeigen.

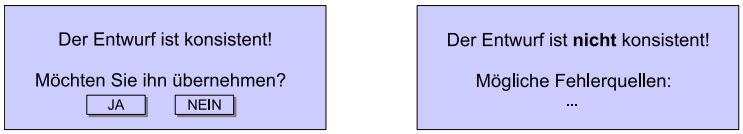
#### 7.3.2.4 Studierende

Wichtige Anwendungsfälle für ein StARC-System treten im Rahmen der Studienberatung auf. Hier sind verschiedene Interaktionsabläufe denkbar.

Möchte ein Student zu Beginn seines Studiums wissen, welche Möglichkeiten er hat, es zu absolvieren, wäre eine entsprechende Ausgabe einiger (aller) Möglichkeiten denkbar. Da dies aber sehr




(a) Hauptfenster zur Prüfung der Konsistenz




(b) Mögliche Antworten

Abbildung 7.8: Ausschnitt aus einer möglichen Bedienoberfläche zur Konsistenzprüfung von Studienordnungs-Entwürfen für das SO-Komitee

viele sein können und ein Teil untereinander ähnlich sein kann, ist oftmals eine iterative Erstellung der Möglichkeiten – unter Einbeziehung von Angaben oder Entscheidungen des Studenten – sinnvoller. Dem Studenten können schrittweise Entscheidungsmöglichkeiten angeboten werden. Die Berechnung erfolgt dann unter Berücksichtigung der getroffenen Einstellungen („Entscheidungen“) des Studenten. Abbildung 7.9 zeigt einen Ausschnitt eines möglichen Bedieninterfaces. Zu bemerken ist ferner, dass die Voraussetzens-Beziehung Berücksichtigung finden muss: Es können nur solche Elemente gewählt werden, deren Voraussetzung erfüllt ist. Außerdem sollte es auch möglich sein, nur einen Teil der geforderten Elemente zu wählen (z. B. Wahl von nur einem Modul im Beispiel der Abbildung 7.9) und auf dieser Grundlage trotzdem sinnvolle Berechnungen anzustellen.

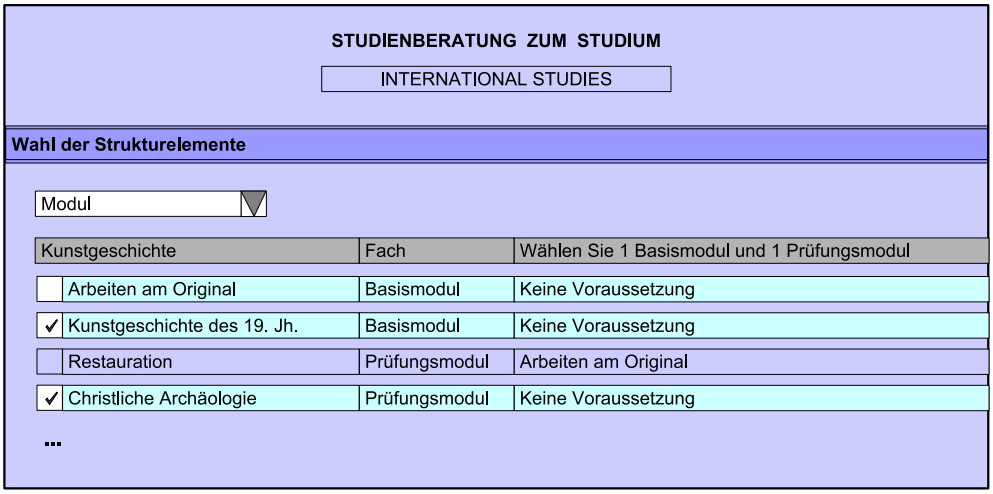



Abbildung 7.9: Ausschnitt einer GUI zur Studienberatung: Wahl der Module im (gewählten) Fach Kunstgeschichte

Ähnliche Abläufe sind denkbar, wenn ein Student schon einen gewissen Fortschritt im Studium hat, also einen Teil der Veranstaltungen schon bestanden hat. Auf der Grundlage entsprechender Angaben kann eine Beratung für das weitere Studium erfolgen. Wird eine externe Veranstaltung für ein bestimmtes Veranstaltungstemplate anerkannt, so kann dies im Prinzip dadurch ausgedrückt werden, dass das entsprechende Veranstaltungstemplate bestanden ist.

Interessiert sich ein Student für bestimmte Veranstaltungen oder möchte er bestimmte auf die Studienordnung bezogene Entscheidungen treffen, so kann er deren Konsistenz zur Studienordnung hin überprüfen. Eine entsprechende Eingabemöglichkeit sollte das Bedieninterface besitzen.

## 7.4 Bezug zu alternativen Ansätzen

### 7.4.1 Pfadbasierter Ansatz

In Abschnitt 6.2 werden Überlegungen zur Transformation in einen pfadbasierten Ansatz vorgenommen. Folgendes lässt sich feststellen:

#### 7.4.1.1 Modeltesting

Wie im Abschnitt 2.4.2 erläutert, ist für diesen Anwendungsbereich ein generierender Ansatz wünschenswert. Ein reiner Modeltesting-Ansatz trägt nicht allen Anforderungen Rechnung. Im Rahmen der „Modellprüfung mit XML+XPath“ müssen die möglichen Lösungskandidaten als XML-Dokumente schon vorliegen. Ein zusätzliches Verfahren ist notwendig, diese zu erzeugen.

#### 7.4.1.2 XML+XPath

**Vorteile und Stärken** Die pfadbasierten Aspekte dieser Methode bringen im Rahmen der Navigation innerhalb des strukturierten, „getypten“ Modulkatalog-Objektgraphen bzw. des entsprechenden Dokuments Vorteile. Gute Fähigkeiten der Navigation durch den Graphen sind gegeben. Vor allem bei Modellprüfungsaufgaben sind eine Reihe von Constraints in XPath einfach darstellbar. Hierbei sind die Möglichkeiten des Zählens mit count und die des Rechnens mit sum von Vorteil. Jedoch lässt hierzu auch der vorgestellte Modelgenerating-Ansatz Möglichkeiten zu. Darüber hinaus ermöglicht dieser Ansatz das einfache Aneinanderreihen von Bedingungen.

**Nachteile und Grenzen** Gerade auch im Hinblick auf den vorgestellten „generierenden Ansatz“ sind Grenzen und Nachteile von XPath gegeben:

- *Zählen:* Die Möglichkeiten des Zählens sind begrenzt.
- *Teilmengen(sequenz)auswahl:* XPath besitzt kein grundsätzliches Konzept, *beliebige* Teilsequenzen aus einer Sequenz von Knoten zu erzeugen. Möglich ist zwar,
  - alle Kindknoten mit bestimmten Eigenschaften oder
  - Knoten an bestimmter Position – falls es diese gibt –

  zu wählen. Methoden der Iteration oder des Auswählens sind begrenzt.
- *Transformationsart:* Durch die Navigation in tiefere Ebenen verliert man die „vorher schon gewählten“ Elemente. Die jeweiligen Vorfahren-Knoten gehören der Ergebnissequenz nicht an. In XPath ist es nicht vorgesehen, nur bestimmte Knoten des Baumes „wegzuschneiden“. Es besteht eine sehr eingeschränkte Möglichkeit der Transformation von Bäumen zu Bäumen.

- *Kompaktheit der Ausdrücke:* Darüber hinaus ist festzustellen, dass auch für einfache Fälle die zugehörigen XPath-Ausdrücke unübersichtlich sein können.

**Vergleich mit Auswahl-Baumconstraints** Im Folgenden sollen einige Aspekte von XPath mit den im Abschnitt 5.3.2.7 definierten Auswahl-Baumconstraints verglichen werden. Vergleiche hierzu auch die Bemerkungen 5.3.32 und 5.3.49.

1. *„Eingabe“*
   - *XPath:* Sequenz (von Elementknoten)
   - *Auswahl-Baumconstraints:* Menge (von Knoten im Graphen)
2. *Kontextbegriff*
   - *XPath:* Informationen, die das Resultat eines Ausdrucks beeinflussen können (dynamischer Kontext: statischer Kontext und weitere Informationen, unter anderem Eingabesequenz, Kontextdatenelement, Kontextposition, Kontextgröße)
   - *Auswahl-Baumconstraints:* Kontextmenge: Menge der Strukturelemente, die für die Auswertung des entsprechenden Constraints (meist Successorconstraints) gerade relevant ist

   Da bei XPath der Kontext in gewissem Sinne auch die Sequenz der „gerade zu berücksichtigenden Knoten“ darstellt, haben die Kontextbegriffe in beiden Ansätzen eine gewisse Ähnlichkeit, wenn man von dem Unterschied der Sequenz- und Mengensemantik absieht.
3. *Expressivität*
   - *XPath:* Anfragen in der Regel an *alle* Elemente eines Teilbaums mit bestimmtem Typ und bestimmten Eigenschaften oder explizite Selektion über die Position
   - *Auswahl-Baumconstraint:* Auswahl einer *beliebigen Anzahl (oder Gewichtssumme)* von Elementen mit bestimmtem Typ und bestimmten Eigenschaften und Auswahl einer beliebigen Anzahl von zu erfüllenden Constraints möglich

   In Auswahl-Baumconstraints sind also viel allgemeinere Bedingungen darstellbar als in XPath.
4. *Kompaktheit, Aufbau des Ausdrucks:*
   - *XPath:* Ausdruck besteht im Wesentlichen aus einem oder mehreren sogenannten Stepausdrücken, die durch das Zeichen „/“ oder „//“ getrennt sind und denen optional das Zeichen „/“ oder „//“ vorangestellt ist. In den Stepausdrücken wird üblicherweise auf die Typen der betrachteten semistrukturierten Daten Bezug genommen.
   - *Auswahl-Baumconstraints:* Constraint hat Baumstruktur, welche eine sehr kompakte und übersichtliche Darstellung ermöglicht. Die entsprechenden Constraintreferenzmengen stellen üblicherweise den Bezug zum betrachteten Graphen (z. B. Modulkatalog-Objektgraphen) her.

#### 7.4.1.3 Modelgenerating mit XML+XPath/XQuery

Die Überlegungen zur Transformation des Modells in XML und XPath/XQuery zeigen:

**Vorteile und Stärken** Für einen XML-basierten Ansatz sprechen:

1. Vereinigung der Vorteile einer funktionalen und einer pfadbasierten Sprache
   - Leichte Navigationmöglichkeit im strukturierten Dokument
   - Gute Spezifikationsmöglichkeit von Constraintreferenzmengen
   - Implementierung rekursiver Funktionen
2. Generierung von Teilbäumen, in denen die Struktur erhalten bleibt
3. Verwendung der weithin bekannten Methoden XML, XPath/XQuery
4. Visualisierungsmöglichkeiten

**Nachteile und Grenzen** Dennoch gibt es einige wesentliche Grenzen und Nachteile dieses Ansatzes:

1. *Architektonischer Vergleich mit ASP-Lösung:*
   - *Reasoner:* Sowohl für XML+XPath/XQuery als auch für ASP gibt es Systeme zur Anfragebearbeitung, jedoch arbeitet der Ansatz XML+XPath/XQuery in gewissem Sinne auf einer „niedrigeren Ebene“ hinsichtlich der Bedeutung der repräsentierten Strukturen.
   - *Modelgenerating:* Im Fall von „XML+XPath/XQuery“ ist die Funktionalität der Modellgenerierung vollständig zu implementieren, im Fall von ASP ist sie im Reasoner SMODELS vorhanden.
   - *Semantik:* XML+XPath/XQuery beinhaltet funktionale und pfadbasierte Aspekte, ASP die Antwortmengen-Semantik,[5] welche adäquat für Probleme mit mehreren möglichen Lösungen ist und lose Spezifikationen unterstützt.
2. *Repräsentation der Bedingungen:* Etliche Bedingungen können zwar dargestellt werden, jedoch sind durch die erforderlichen Gewichtsconstraints schnell Grenzen der Realisierbarkeit gesetzt. Sowohl die Möglichkeiten des „Zählens“ und einer gleichzeitigen Bewertung der Gesamtheit der gewählten Pfade als auch die der Optimierung sind begrenzt. Eine Art Pfadsprache 2. Ordnung wäre erforderlich.

[5] Manchmal wird die Antwortmengen-Semantik auch als Semantik stabiler Modelle bezeichnet.

3. *Transformationsart:* Wie oben erwähnt, verliert man bei einer XPath-Anfrage bei der Navigation in tiefere Ebenen die „vorher schon gewählten“ Elemente. Die Informationen über die Vorfahren-Knoten gehen verloren. Da ein gültiges Studium einen Teilbaum des Modulkatalog-Objektgraphen darstellt, ist strukturell gesehen eine Baum-zu-Baum-Transformation erforderlich. XML+XPath/XQuery trägt dieses Konzept jedoch *nicht* in „natürlicher“ Weise in sich. Mit Hilfe geeigneter Funktionen kann die gewünschte Funktionalität zwar weitgehend implementiert werden, der damit verbundene Programmieraufwand und eine gewisse „Unhandlichkeit“ schließen diesen Ansatz jedoch für komplexe anwendungsrelevante Probleme aus.

4. *„Technische Anmerkungen“:*

   - XML+XPath/XQuery sind weniger dazu gedacht, auf *mehr* als auf einem Dokument zu arbeiten. So müssen z. B. sich entsprechende Knoten verglichen werden oder auch eine Spezifikation in einem Dokument auf ein anderes übertragen werden. Dies ist deswegen bedeutsam, da der gezeigte Ansatz iterative Berechnungen vorsieht: Eine Anfrage auf ein Input-XML-Dokument liefert eine oder mehrere Output-XML-Dokumente, auf denen wiederum weitere Anfragen in der Rolle von Input-Dokumenten gestellt werden können.
   - Beim Streichen von Elementen aus einem Baum muss der resultierende Baum „ganz neu“ erzeugt werden. Eine Identifikation der Knoten über eine eindeutige ID ist notwendig.

5. *Benutzerfreundlichkeit:* Eine gewisse Umständlichkeit bei der Handhabung ist festzustellen. Für die Berechnung *sämtlicher* Lösungen ist dieser Ansatz nicht gut geeignet.

**Zusammenfassung** Obige Ausführungen zeigen: Der XML-basierte Ansatz scheint für die betrachteten komplexen Probleme, welche eine Kombination aus strukturellen Gegebenheiten und Constraints darstellen, nicht ädäquat zu sein.

### 7.4.2 Weitere Ansätze

#### 7.4.2.1 Andere Methoden

Schließlich soll noch kurz auf die noch nicht betrachteten Methoden aus Abschnitt 4.1 eingegangen werden.

Wegen der Komplexität des Problems (vgl. Abschnitt 7.2.1) scheiden Trivialmethodenaus. So sind auch die herkömmlichen Methoden wie

- *SQL*

  \+ effiziente Standardmethode

  \- (kombinierte) Anzahl- und Gewichtsbeschränkungen

- *Datalog*
  - \+ logikbasierte Methode
  - \- (kombinierte) Anzahl- und Gewichtsbeschränkungen
- *Operations Research (OR)*
  - \+ (Un)gleichungssysteme
  - \- Abhängigkeiten zwischen den Objekten

zu wenig ausdrucksstark und nicht adäquat.

#### 7.4.2.2 Andere Systeme und Arbeiten

**Testmanagement-System** Der Leser sei auf den Abschnitt 2.3.4 verwiesen, in dem existierende industrielle Lösungen für Testmanagement- oder ähnliche Systeme und deren Einschränkungen diskutiert werden.

**StARC-System** Erinnert sei hier auf die Ausführungen von Abschnitt 2.4.4, in welchem auf einige existierende verschiedene Assistenzsysteme wie Campus-, Kurs- und Lernmanagementsysteme und deren Einschränkungen und Grenzen eingegangen wird.

Einige weitere wissenschaftliche Arbeiten seien exemplarisch zitiert. Hierbei sind die Schwerpunkte in den Arbeiten unterschiedlich gelegt. So werden teilweise spezielle Logikansätze oder besondere Gegebenheiten des Anwendungsbereichs oder auch das Anliegen, das System als Webanwendung zu betreiben, in den Vordergrund gestellt. Es scheint jedoch keinen allgemeinen und umfassenden Ansatz zu geben, der mit den Ergebnissen dieser Arbeit vergleichbar ist.

So wird beispielsweise in [Abr00] ein logikbasiertes Informationssystem, in welchem Prolog und eine Beschreibungslogik zum Einsatz kommen, vorgestellt. Die Autoren von [IWL03] präsentieren ein intelligentes „online academic management system“, welches eine mächtige auf PT-Resolution[6] basierende Engine enthält. Hierbei werden Aspekte der Immatrikulation und der Studienberatung berücksichtigt. Ein Entscheidungsunterstützungssystem für akademische Beratung wird in [MB95] beschrieben. Es werden sogar gewisse Voraussetzungen einfacher Form zwischen Kursen bei der Modellierung berücksichtigt. Die Autoren von [MMGKB99] verwenden in ihrem Beratungssystem ebenso einen Ansatz mit Logikprogrammierung (Prolog) und berücksichtigen verschiedene Arten von Voraussetzungen, jedoch bei Weitem nicht in der Allgemeinheit wie in der vorliegenden Arbeit. Objektorientierte und wissensbasierte Paradigmen werden in dem Studienberatungssystem von [GLY95]

[6] „resolution with partial intersection and trunctation“

kombiniert. Webbasierte Aspekte werden in dem akademischen Beratungssystem von [MDH01] betont. In dem Artikel [NRD05] tauchen moderne Sprachen wie XML zur Unterstützung des Bologna-Prozesses in universitären Informationssystemen auf. Mit temporaler Logik wird in [BBM07] zum Modellieren und Checken von Curricula gearbeitet. Ein logikbasierter Ansatz wird auch in [BBP04] verfolgt. Schließlich sei noch [IC01] erwähnt.

## 7.5 Typische Bestandteile betrachteter Planungsprobleme

Die betrachteten Planungsprobleme „Studienberatung und Konsistenzprüfung von Studienordnungen“ und „Generierung von Tests“ weisen eine mehr oder weniger starke Strukturierung der Domäne auf. Im Kapitel 5 sind sowohl die Gemeinsamkeiten als auch die Besonderheiten der betrachteten Anwendungsbereiche herausgearbeitet.

Hier sollen nochmals grob die typischen Bestandteile des Problems aufgelistet werden:

Struktur und Bedingungen sind weitgehend unabhängig zu modellieren, wobei gewisse Abhängigkeiten natürlicherweise bestehen.

**Struktur** Im Strukturmodell sind folgende Aspekte bedeutsam:

1. (Struktur-)Typen mit Attributen
2. Objekte mit Attributwerten und Eigenschaften
3. Mehrdimensionalität
4. n:m-Beziehungen
5. Wählbare Elemente: Elemente, die in einer Lösung enthalten sein können
6. Strukturierung der Domäne: Strukturtypen, Strukturgraph
7. Abhängigkeiten zwischen den Daten

**Bedingungen** Im Rahmen des Constraintmodells bzw. der auftretenden Bedingungen sind folgende Aspekte relevant:

1. Verschiedene Klassen von Bedingungen
2. Unterscheidung der Bedingungen:
   - Elementare und komplexe Constraints

- Bezugsmenge: Constraintreferenzmenge (Teilmenge wählbarer Elemente) bzw. Constraintmenge (Teilmenge aller Constraints)

3. Elementare Bedingungen:
   - Anzahl
   - Gewichtssumme
   - Vorgabe von obligatorischen Elementen
   - Ausschluss von Elementen (Negation)
4. Komplexe Constraints:
   - Bedingtes Constraint
   - Auswahl von Constraints
   - Prozentuale Anforderung
   - Successorconstraint
   - Sequenz-, Baum- und Auswahl-Baumconstraint
5. Kontextbezogenheit bei Auswahlen
6. Harte und weiche Anforderungen
7. Lokale und globale Bedingungen
8. Bezug zum Strukturmodell:
   - Bezugsmenge der Constraints (Spezifikation: (Struktur-)Typen, Attribute)
   - Mögliche Struktur der Constraints

**Lösungen** Die Lösungen des Problems stellen Teilmengen der Menge wählbarer Elemente dar. Unvollständige Spezifikationen und Teillösungen sind möglich. Sowohl Modellgenerierung als auch Modellprüfung ist relevant. Sinnvolle Einschränkungen der Anfragen auf Teilmengen der Domäne („Zerlegung der Domäne“) ist in der Regel möglich.

Eine Übertragung des gewählten Ansatzes auf Planungsprobleme in strukturierten Domänen mit ähnlichen Rahmenbedingungen erscheint sinnvoll und möglich.

## 7.6 Methodische Einordnung und Bewertung des (Constraint-)Modells

Im Folgenden soll das in Kapitel 5 definierte Modell im Hinblick auf verwendete und verwandte Methoden eingeordnet werden. Im Zentrum der Betrachtungen soll vor allem das zugehörige Constraintmodell (Abschnitte 5.1.2, 5.2.2 und vor allem 5.3.2) stehen. Die definierten Constraints, vor allem die komplexen Sequenz-, Baum- und Auswahl-Baumconstraints sollen in Augenschein genommen werden.

Zur einfacheren Argumentation wird vor allem auf den zweiten Anwendungsbereich „Studienberatung und Konsistenzprüfung von Studienordnungen“ Bezug genommen. Entsprechende Aussagen gelten aber zum Teil auch für den ersten Anwendungsbereich „Generierung von Tests“.

**Strukturmodell** In den Strukturmodellen werden folgende Konzepte verwendet:

- *Klassen- und Objektmodell:* Strukturelemente und Strukturtypen des Modulkatalogs
- *Graphkonzept:* Modulkatalog-Objektgraph mit Strukturelementen als Knoten und $contain$-Instanzen als Kanten.

**Constraintmodell** Die Constraintmodelle beinhalten folgende Konzepte:

- *Relationen- und Mengensicht:* Definition der Constraints mit Hilfe von Relationen
- *Graphkonzept:* Auffassung der Baum- und Auswahl-Baumconstraints als Bäume von Constraints
- *Constraintsemantik:* Definition einer formalen Semantik

**Beziehungen zwischen Constraint- und Strukturmodell** Die Verzahnung von Constraint- und Strukturmodell geschieht über

- *Constraintreferenzmengen:* Sie bestehen aus einer Teilmenge $C \subseteq \mathcal{S}$ der Menge der wählbaren Elemente $\mathcal{S}$ und sind daher Elemente des Objektgraphen. Dadurch ist eine natürliche Beziehung zwischen den Constraints und den strukturellen Elementen gegeben.
- *Struktur des komplexen Constraints:* Vor allem die Sequenz-, Baum- und Auswahl-Baumconstraints spiegeln durch ihre Struktur die Struktur des Graphen, auf den sie sich beziehen, wider.

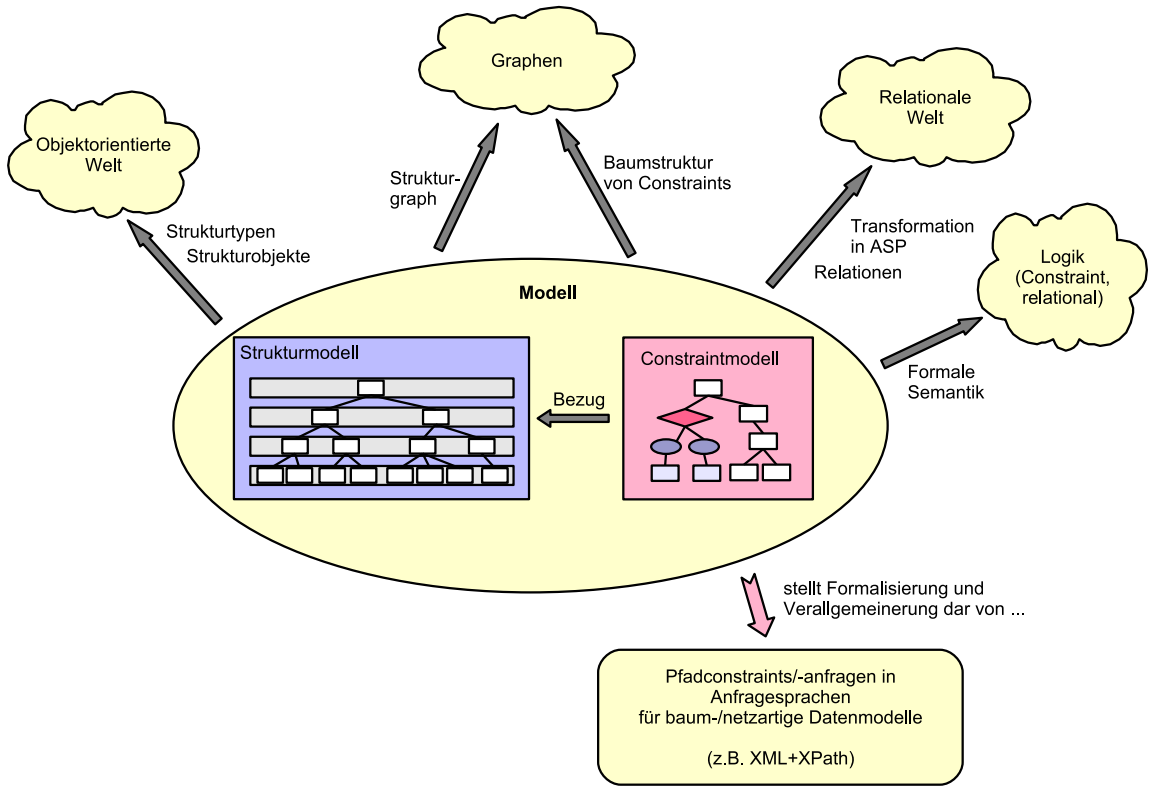



Abbildung 7.10: Modell: Methodische Einordnung

**Bewertung** Zusammen mit früheren Ausführungen (vgl. auch Bemerkungen 5.3.32 und 5.3.49 und Abschnitt 7.4.1) wird deutlich, dass die im Modell vorgenommene Definition von Constraints auf Objektbäumen (vgl. Baum-, Auswahl-Baumconstraints) Folgendes leistet:

- Sie stellt eine logische Formulierung und Formalisierung von Pfadanfragen/-constraints in baum-/netzartigen Datenmodellen wie XML+XPath dar. Es kann unter anderem als ein Konzept zur Definition und Weitergabe von Kontext auf Pfaden verstanden werden. Die pragmatisch gehandhabte Semantik der XPath-Ausdrücke wird hiermit gewissermaßen auf die Ebene einer Logik (Constraintlogik, relationale Logik) gestellt.
- Darüber hinaus stellt sie sogar eine Verallgemeinerung von derartigen Anfragesprachen, also beispielsweise XML+XPath dar. So können beispielsweise komplexe Anzahl-, Gewichtssummen- und Auswahlbedingungen formuliert werden.
- Die definierte „Modellierungssprache“ wird hiermit für allgemeine Constraintaufgaben nutzbar.

- Vermöge der Relationen bzw. der vorgenommenen Transformation in Answer Set Programming mit Gewichten wird eine Verbindung zur relationalen Welt hergestellt.

Nach bestem Wissen und ausgiebiger Recherche scheint es keinen vergleichbaren Ansatz zur Formulierung von Constraints auf Objektbäumen und der „Definition und Weitergabe von Kontexten in Pfaden“ zu geben (vgl. auch Bemerkung 5.3.50).

In Abbildung 7.10 werden die besprochenen Sachverhalte veranschaulicht.

# Kapitel 8

# Resumee und Ausblick

In der vorliegenden Arbeit wird ein methodisch-technisches Framework zur Behandlung von Planungsproblemen in strukturierten Domänen präsentiert. Im Besonderen wird auf zwei Anwendungsbereiche eingegangen:

1. Generierung von Tests
2. Studienberatung und Konsistenzprüfung von Studienordnungen

Unter anderem können folgende Resultate festgehalten werden:

- Der gezeigte Ansatz liefert eine Methodik zur Behandlung von Planungsproblemen in strukturierten Domänen wie bei der Generierung von Tests oder der Erstellung bzw. Nutzung von Studienordnungen.
- Das Modell (vgl. Kapitel 5) erweist sich als zentraler Bestandteil des Ansatzes. Es kann als Schnittstelle zur Bedienoberfläche und zur internen Repräsentation fungieren.
- Durch die verschiedenen Ebenen der Modellierung (abstrakte und konkrete Modelle) wird ein hohes Maß an Abstraktion und Allgemeingültigkeit verwirklicht. Sämtliche relevanten Probleminstanzen lassen sich im Modell geeignet repräsentieren. Darüber hinaus erscheint die Übertragbarkeit auf ähnliche Problemklassen gegeben zu sein.
- Im Rahmen des Frameworks wird die Integration von strukturellen und constraintbasierten Aspekten realisiert. Die weitgehend unabhängige Modellierung von Struktur und Constraints verwirklicht ein hohes Maß an Modularität.

- Im Zusammenhang mit der Constraintmodellierung (vgl. Abschnitte 5.1.2, 5.2.2, 5.3.2) erfolgt die Definition relevanter einfacher und komplexer Constraints gemäß den auftretenen Anforderungen. Es wird hiermit eine sehr kompakte „Sprache“ definiert, welche einen geeigneten Bezug der Constraints zu den vorhandenen strukturellen Gegebenheiten verwirklicht (vgl. z. B. Auswahl-Baumconstraints).

- Durch die vorgenommene formale Definition der Syntax und Semantik der Elemente des Constraintsmodells wird eine logische Formulierung, Formalisierung und Verallgemeinerung von modernen pfadbasierten Anfragesprachen wie XPath bzw. deren kontextbasierten Pfadsemantik erreicht (vgl. Abschnitt 7.6).

- Durch die Nähe des Modells zur informellen Beschreibung der Anwendungsbereiche wird die Realisierung geeigneter Bedienschnittstellen unterstützt (vgl. auch Abschnitte 5.2.4 und 7.3).

- Die exakte Transformation des (konkreten) Modells in den favorisierten internen Formalismus (ASP+Gewichte) kann in natürlicher Weise definiert werden (vgl. Kapitel 6). Zahlreiche „Pattern“ sind spezifiziert und definiert, die eine „musterbasierte Übersetzung“ in den internen Formalismus möglich machen.[1]

- Als interner Formalismus erweist sich der logikbasierte Ansatz mit Constraints (Answer Set Programming mit Gewichten) für derartige Planungsprobleme in strukturierten Domänen als am besten geeignet, auch wenn andere gängige Ansätze (vgl. Abschnitt 7.4) gewisse eigene Stärken aufweisen.

- Sowohl bezüglich der funktionalen, systembezogenen Anforderungen (Repräsentation der Daten in geeigneter Form, Berücksichtigung eines generierenden und auch eines prüfenden Aspekts, Umgang mit losen Spezifikationen und Teillösungen und Beachtung der strukturierten Domäne) als auch der nichtfunktionalen Anforderungen und Zielkriterien (Ausdruckskraft und Mächtigkeit, Adäquatheit, Erweiterbarkeit und Modifizierbarkeit, Integrierbarkeit, Performanz und Skalierbarkeit) erweist sich der verwendete Gesamtansatz als äußerst tragfähig (vgl. Kapitel 7).

- Durch die im Modell verwendeten Methoden wird eine Verbindung sowohl zur Logik als auch zur relationalen Welt geschaffen (vgl. Abschnitt 7.6).

- Es wird im Rahmen des vorgestellten Frameworks gezeigt, wie die verwendeten formalen Methoden wie etwa ASP+Gewichte in aktuellen Anwendungsbereichen hoher Relevanz nutzbar gemacht und adäquat verwendet werden können.

Aufgrund dieser und früherer Ausführungen lassen sich die wesentlichen Bestandteile des wissenschaftlichen Beitrags dieser Arbeit wie folgt kurz zusammenfassen:

[1] Diese Muster können auch als Art Problemmuster oder Module aufgefasst werden, wie sie für einfache Probleme in [Bar03] definiert sind.

*Im Rahmen dieser Arbeit wird ein Framework zur Behandlung von Planungsproblemen in strukturierten Domänen anhand der Anwendungsbereiche „Generierung von Tests“ und „Studienberatung und Konsistenzprüfung von Studienordnungen“ definiert und bewertet.*

Die zentralen innovativen Aspekte des Frameworks sind:

1. Als ein wichtiger Kernpunkt zur Lösung der Aufgabe erfolgt im Rahmen eines modellgetriebenen Ansatzes die *Definition eines formalen Modells*, welches sowohl die Schnittstelle zur Bedienoberfläche als auch zur internen Repräsentation darstellt.

   (a) Im darin enthaltenen Constraintmodell werden Constraints formal definiert, welche eine logische Formulierung, Formalisierung und Verallgemeinerung von gängigen Pfadanfragen in Anfragesprachen für baum-/netzartige Datenmodelle wie XML+XPath darstellen. Jene werden hierdurch für allgemeine Constraintaufgaben nutzbar.

   (b) Durch die im Modell verwendeten Methoden wird eine Verbindung sowohl zur Logik als auch zur relationalen Welt geschaffen. Es erfolgt eine adäquate Verzahnung von strukturellen und constraintbasierten Aspekten.

   (c) Durch die verschiedenen Ebenen der Modellierung (abstrakte und konkrete Modelle) und die weitgehend gemeinsame Grundmodellierung für die beiden betrachteten Anwendungsbereiche wird ein hohes Maß an Allgemeingültigkeit erreicht. Eine Übertragbarkeit auf weitere Problemklassen ist möglich.

2. Im Rahmen des Gesamtansatzes erfolgt die *Integration und Lösung verschiedener Grundproblematiken*. Modellgenerierung und -prüfung, späte Spezifikationen, Verfeinerung vorgegebener Teillösungen und die Beachtung der strukturierten Domäne werden berücksichtigt.

3. Darüber hinaus wird die *formale Methode „ASP+Gewichte“* als ein adäquater Formalismus für die interne Repräsentation erkannt und eine *Transformationsvorschrift vom Modell zur internen Repräsentation definiert*. Hierdurch erfolgt der *Nachweis der Relevanz und die Nutzbarmachung eines Spezifikations- und Ausführungsformalismus*, der nicht gerade im Zentrum der Aufmerksamkeit steht. Gerade auch im Vergleich mit modernen Sprachen wie XML+XPath wird die Praktikabilität und Stärke von „ASP+Gewichte“ nachgewiesen.

4. Im Rahmen der Definition der Transformation des Modells in den internen Formalismus „ASP+Gewichte“[2] werden *zahlreiche Problem-Muster definiert* und *templatebasiert* umgesetzt. Sie können im Prinzip als eine Art ASP-Module aufgefasst werden.

5. Im Zusammenhang mit den Diskussionen zur *Anwendbarkeit und Integrierbarkeit in Informationssysteme* (Prototypische Implementierungen, zahlreiche Benchmarks, Architektur, Bedienung der Systeme) wird die Praktikabilität und Relevanz des eigenen Ansatzes nachgewiesen.

[2] Answer Set Programming mit Gewichten, vgl. Abschnitt 4.2

Der verfolgte Ansatz für die betrachteten Planungsprobleme in strukturierten Domänen erweist sich als sehr mächtig und adäquat. Darüber hinaus ergeben sich im Rahmen der vorliegenden Arbeit einige Ansatzpunkte für weitere Forschungen:

- Die Entwicklung einer allgemeinen Spezifikationssprache für den Benutzer und einer entsprechenden graphischen Bedienoberfläche ist auf der Grundlage des im Kapitel 5 definierten Modells vorzunehmen.
- Der SMODELS-Algorithmus könnte speziell für die betrachteten Anwendungsbereiche angepasst werden. Ein Aspekt hierbei wäre die Realisierung einer Art „Gedächtnis“: Schon berechnete Modelle/Lösungen einer Probleminstanz werden bei der nächsten Berechnung nicht mehr ausgegeben, sondern die entsprechend „nächsten“ Modelle/Lösungen.
- Aufgrund des hohen Grads an Allgemeingültigkeit der (abstrakten) Modellierung erscheint eine Übertragung auf weitere Anwendungsbereiche, welche strukturelle und constraintbasierte Aspekte vereinigen, gut möglich. Um auch andere Probleme besser lösen zu können, erscheint eine Untersuchung weiterer Problemklassen (vgl. Kapitel 5 und Abschnitt 7.5) in dieser Hinsicht lohnend.

# Literaturverzeichnis


[Abr00] Salvador Abreu. A logic-based information system. In Enrico Pontelli and Vítor Santos Costa, editors, *PADL*, volume 1753 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2000.

[ACD+] Abderrahamane Aggoun, David Chan, Pierre Dufresne, Eamon Falvey, Hugh Grant, Warwick Harvey, Alexander Herold, Geoffrey Macartney, Micha Meier, David Miller, Shyam Mudambi, Bruno Perez, Emmanuel van Rossum, Joachim Schimpf, Kish Sehn, Periklis Andreas Tsahageas, and Diminique Henry de Villeneuve. ECLiPSe *User Manual*. ECRC, IC-Parc.

[AV97] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 122–133, New York, NY, USA, 1997. ACM.

[Bar03] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press (January 9, 2003), 2003.

[BBM07] Matteo Baldoni, Cristina Baroglio, and Elisa Marengo. Curricula modeling and checking. In Roberto Basili and Maria Teresa Pazienza, editors, *AI*IA*, volume 4733 of *Lecture Notes in Computer Science*, pages 471–482. Springer, 2007.

[BBP04] Matteo Baldoni, Cristina Baroglio, and Viviana Patti. Web-based adaptive tutoring: An approach based on logic agents and reasoning about actions. *Artif. Intell. Rev.*, 22(1):3–39, 2004.

[BDK97] Gerhard Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview*. Center for the Study of Language and Inf (June 1, 1997), 1997.

[BE98] Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 86–97. Morgan Kaufmann, San Francisco, California, 1998.

[BFW00] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000.

[BG04] A. Bauer and H. Günzel, editors. *Data-Warehouse-Systeme. Architektur - Entwicklung - Anwendung.* dpunkt.verlag, Heidelberg, 2. Auflage, 2004.

[Bla] Homepage BLACKBOARD LEARNING SYSTEM. http://www.blackboard.com/ products/as/, August 2007.

[Bre96] G. Brewka. Well-founded semantics for extended logic programs with dynamic preference information. In *Journal of Artificial Intelligence Research*, volume 4, 1996.

[Bre04] G. Brewka. Answer sets: From constraint programming towards qualitative optimization. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR-7*, pages 34–46. Springer Verlag, 2004.

[CDF+04] D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy, and P. Wadler. *XQuery from the Experts, A Guide to the W3C XML Query Langugage*. Addison-Wesley, 2004.

[CFL+01] Francesco Calimeri, Wolfgang Faber, Nicola Leone, Simona Perri, and Gerald Pfeifer. DLV - declarative problem solving using answer set programming. In *Proceedings of the Seventh Congress of the Italian Association for Artificial Intelligence AI*IA 2001*, Bari, Italy, 2001.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. Logic programming and databases. *Surveys in Computer Science*, 1990.

[Dat] Homepage DATENLOSEN. http://www.datenlotsen.de, August 2007.

[DD07] W. Domschke and A. Drexl. *Einführung in Operations Research*. Springer-Verlag, 2007.

[Dea] Homepage DEAKIN KM. KNOWLEDGE PRESENTER. http://www.deakinkm.com, August 2007.

[DEGV97] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. In *IEEE Conference on Computational Complexity*, pages 82–101, 1997.

[DFI+03a] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Aggregate functions in DLV. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation*, Messina, Italy, September 2003.

[DFI+03b] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Semantics and computation of aggregate functions in disjunctive logic programming. Technical Report Technical Report INFSYS RR-1843-03-07, Institut für Informationssysteme, Technische Universität Wien, April 2003.

[Die06] R. Diestel. *Graphentheorie*, volume 3. Springer, 2006.

[DSTW02] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. Towards a classification of preference handling approaches in nonmonotonic reasoning. In U. Junker, editor, *Proceedings of the Workshop on Preferences in Artificial Intelligence and Constraint Programming: Symbolic Approaches*, pages 16–24. AAAI Press, 2002.

[EFF+03] Thomas Eiter, Wolfgang Faber, Michael Fink, Gerald Pfeifer, and Stefan Woltran. Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation*, September 2003.

[EFL+03] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, ii: the small DLV$^K$ system. *Artificial Intelligence*, 144(1-2):157–211, March 2003.

[ELST04] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *KR*, pages 141–151. AAAI Press, 2004.

[EN02] Ramez Elmasri and Shamkant B. Navathe. *Grundlagen von Datenbanksystemen*, volume 3. Pearson Studium, 2002.

[Eura] Europäische Bildungsminister, Bologna Erklärung. http://www.bologna-berlin2003.de/pdf/bologna_declaration.pdf, August 2007.

[Eurb] Europäische Kommission, Der Bologna Prozess. http://ec.europa.eu/education/policies/educ/bologna/bologna_en.html, August 2007.

[Eurc] Europäische Kommission, Europäisches System zur Übertragung und Akkumulierung von Studienleistungen (ECTS) - Kernpunkte. http://ec.europa.eu/education/programmes/socrates/ects/doc/ectskey_de.pdf, August 2007.

[Eurd] Europäische Kommission, Bildung und Kultur, Allgemeine und berufliche Bildung 2010. http://ec.europa.eu/education/policies/2010/doc/compendium05_de.pdf, August 2007.

[Eure] European Commission. Education and Training 2010, Diverse System, Shared Goals. http://ec.europa.eu/education/policies/2010/et_2010_en.html, August 2007.

[Eurf] Europäische Union, Internet. Bologna-Prozess: Harmonisierung der Hochschulsysteme. http://europa.eu/scadplus/leg/de/cha/c11088.htm, August 2007.

[Exa] EXAM - Webbasierte Befragungen, Umfragen und Tests, Homepage. http://www.myexam.net/, August 2007.

[ExB] EXAM BUILDER e-learning solutions for professionals, Homepage. http://www.exambuilder.com, August 2007.

[FA97] Thom Frühwirth and Slim Abdennadher. *Constraint-Programmierung*. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[FP] Wolfgang Faber and Gerald Pfeifer. DLV Homepage, since 1996. http://www.dlvsystem.com/, August 2007.

[Fre05] Prof. Dr. Burkhard Freitag. Skript zur Vorlesung „Deduktive Datenbanken“. http://www.im.uni-passau.de/lehre/ss05/dedDBS/, Sommersemester 2005.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 579–597, Jerusalem, Israel, June 1990. MIT Press.

[GL94] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1994.

[GLY95] Himawan Gunadhi, Kwang-Hui Lim, and Wee-Yong Yeong. Pace: a planning advisor on curriculum and enrolment. In *HICSS (3)*, pages 23–31, 1995.

[GW03] M. Gumhold and M. Weber. SASy - A Study Assistance System. In *Proceedings of the 5th International Conference on New Educational Environments*, Luzern, 2003.

[Hoo] J. N. Hooker. Integrated methods for optimization. In *Series: International Series in Operations Research & Management Science*, volume 100.

[HW98] A. Hunger and S. Werner. A Course Curriculum and a Multimedia, Concept for an internationally orientated Degree Course. In *Proceedings of the CATE '98*, pages 25–28. IASTED/ACTA Press, September 1998.

[IC01] José Antonio Macías Iglesias and Pablo Castells. Interactive design of adaptive courses. In *Computers and Education. Towards an Interconnected Society*, pages 235–242. 2001.

[IFI] IFIS. Homepage ASSESSMENT.SUITE. http://www.assessment-suite.de/, August 2007.

[IIP+04] Giovambattista Ianni, Giuseppe Ielpa, Adriana Pietramala, Maria Carmela Santoro, and Francesco Calimeri. Enhancing answer set programming with templates. In James P. Delgrande and Torsten Schaub, editors, *NMR*, pages 233–239, 2004.

[Ili] Homepage ILIAS. http://www.ilias.de/docu/start.php, August 2007.

[IMS] IMS GLOBAL LEARNING CONSORTIUM, INC., question and test interoperability specification, Homepage. http://www.imsglobal.org/question/index.html, August 2007.

[Int] InteLeC. Homepage INTELEC. http://im.uni-passau.de/intelec/, August 2007.

[IPR05] Giovambattista Ianni, Claudio Panetta, and Francesco Ricca. Specification of assessment-test criteria through asp specifications. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[IWL03] I. Ivanto, J. Wang, and F. Liu. Intelligent online academic management system. In Wanlei Zhou, Paul Nicholson, Brian Corbitt, and Joseph Fong, editors, *ICWL*, volume 2783 of *Lecture Notes in Computer Science*, pages 320–326. Springer, 2003.

[JAV] Java Language Specification, Third Edition. http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html, August 2007.

[JBB+00] D. Jordan, D. K. Berry, M. Berler, R. G. Catell, J. Eastman, C. Russell, O. Schadow, T. Stanienda, and F. Velez, editors. *The Object Database Standard ODMG 3.0. Konzepte und Methoden.* Morgan Kaufmann, 2000.

[KE06] A. Kemper and A. Eickler. *Datenbanksysteme – Eine Einführung*. R. Oldenbourg Verlag, München, Wien, 6., aktualisierte und erweiterte edition, 2006.

[KLF06] Stefan Kurz, Markus Lehmann, and Burkhard Freitag. Towards a personalized and situated course guidance system. In *Proceedings of the E-Learn World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education*, Honolulu, Hawaii, 2006.

[Kul] Kultusministerkonferenz. Rahmenvorgaben für die Einführung von Leistungspunktsystemen und die Modularisierung von Studiengängen. http://www.acquin.org/acquincms/index/cms-filesystem-action?file=/ ModularisierungKMK221004.pdf, September 2007.

[KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise.* Addison-Wesley Professional, 2003.

[Leh03] W. Lehner. *Datenbanktechnologie für Data-Warehouse-Systeme. Konzepte und Methoden.* dpunkt.verlag, Heidelberg, 2. Auflage, 2003.

[Lif96] Vladimir Lifschitz. Foundations of logic programming. In *Principles of knowledge representation*, pages 69–127. Center for the Study of Language and Information, 1996.

[Lif99] Vladimir Lifschitz. Answer set planning. In *International Conference on Logic Programming*, pages 23–37, 1999.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[Llo87] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[LPF+02] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Francesco Calimeri, Tina Dell'Armi, Thomas Eiter, Georg Gottlob, Giovambattista Ianni, Giuseppe Ielpa, Christoph Koch, Simona Perri, and Axel Polleres. The DLV system. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, number 2424 in Lecture Notes in Computer Science, pages 537–540, Cosenza, Italy, September 2002.

[LS04] W. Lehner and H. Schöning. *XQuery, Grundlagen und fortgeschrittene Methoden.* dpunkt.verlag, 2004.

[Mal07] Sarah Mallach. Antwortmengenprogrammierung zur Studienberatung für modulare Studiengänge, Zulassungsarbeit zum Staatsexamen, 2007.

[MB95] W. Scott Murray and Louis A. Le Blanc. A decision support system for academic advising. In *SAC*, pages 22–26, 1995.

[MDH01] Oge Marques, Xundong Ding, and Sam Hsu. Design and development of a web-based academic advising system. citeseer.ist.psu.edu/marques01design.html, 2001.

[Min94] Jack Minker. Overview of disjunctive logic programming. *Ann. Math. Artif. Intell.*, 12(1-2):1–24, 1994.

[MMGKB99] J. E. Mendez-Mateos, G. Gupta, A. I. Karshmer, and J. P. Brown. Nada: Nmsu advising degree audit system. In *First International Conference on Practical Applications of Constraint Technologies and Logic Programming*, pages 181–196, 1999.

[Moo] Homepage MOODLE. http://moodle.org/, August 2007.

[MSU04] Stephen J. Mellor, Kendall Scott, and Axel Uhl. *MDA Distilled: principles of model-driven architecture*. Addison-Wesley Professional, 2004.

[MT91] Wiktor Marek and Miroslaw Truszczynski. Autoepistemic logic. *J. ACM*, 38(3):587–618, 1991.

[NRD05] Sérgio Nunes, Ligia Ribeiro, and Gabriel David. Supporting the Bologna Process in HE Information Systems. In *EUNIS*, 2005.

[NS97] Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume volume 1265 of Lecture Notes in Artificial Intelligence, pages 420–429, Dagstuhl, Germany, July 1997.

[NS00] Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, chapter 21, pages 491–521. Kluwer Academic Publishers, 2000.

[NSS99] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics of weight constraint rules. In *Logic Programming and Non-monotonic Reasoning*, pages 317–331, 1999.

[NSS00] Ilkka Niemelä, Patrik Simons, and Tommi Syrjänen. Smodels: a system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (cs.AI/0003073)*, Breckenridge, Colorado, USA, April 2000.

[Okt] OKTIS, Online Kursverwaltung- und Test-Informationssystem , Homepage. http://www.oktis.de/, August 2007.

[Ora06] Oracle Corporation. Information Connects - Peoplesoft Enterprise Campus Solutions 8.9. http://www.oracle.com/media/peoplesoft/en/pdf/datasheets/peoplesoft-enterprise-campus-solution.pdf, August 2007, 2006.

[PDB03] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Translation of aggregate programs to normal logic programs. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming*, volume 78 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[Pir06] Christoph Pirkl. Vergleich der Praxistauglichkeit von ASP-Systemen und Constraint-Solvern am Fallbeispiel Automatische Prüfungsgenerierung, Diplomarbeit, 2006.

[RGF06] Sven Radde, Liubov Gordienko, and Burkhard Freitag. An advanced interactive language teaching platform. In A. Mendez-Vilas et al, editor, *Current Developments in Technology-Assisted Education*, pages 270–274. FORMATEX, nov 2006.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[SAP04] SAP AG. SAP Campus Management - solution in detail, SAP for higher education & research. http://www.sap.com/industries/highered/ pdf/B-WP_SID_Campus_Management.pdf, August 2007, 2004.

[Sch05] Petra Schwaiger. Antwortmengenprogrammierung in der Praxis - eine Fallstudie. In Stefan Brass and Christian Goldberg, editors, *Tagungsband zum 17. GI-Workshop über Grundlagen von Datenbanken*. Institut für Informatik, Martin-Luther-Universität, 2005.

[SF06] Petra Schwaiger and Burkhard Freitag. Using answer set programming for the automatic compilation of assessment tests. In *Proceedings of the Twenty Second International Conference on Logic Programming*, volume 4079 of *LNCS*, pages 300–314, Seattle, USA, 2006. Springer-Verlag.

[Sim99] Patrik Simons. Extending the stable model semantics with more expressive rules. In *Logic Programming and Non-monotonic Reasoning*, pages 305–316, 1999.

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[SNTS01] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming*, pages 195–201, Stanford, USA, March 2001. AAAI Press.

[SoP] Studien- und Prüfungsordnungen an der Universität Passau. http://www.uni-passau.de/560.html, September 2007.

[SQL] Homepage SQL. http://www.sql.org/, August 2007.

[SS86] Leon Sterling and Ehud Shapiro. *The art of Prolog: advanced programming techniques*. MIT Press, Cambridge, MA, USA, 1986.

[Syr] Tommi Syrjänen. Lparse 1.0 – user's manual. http://www.tcs.hut.fi/Software/smodels/, Januar 2008.

[Syr01] Tommi Syrjänen. Omega-restricted logic programs. In *LPNMR '01: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 267–279, London, UK, 2001. Springer-Verlag.

[Syr03] Tommi Syrjänen. Logic programming with cardinality constraints. Research Report A86, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 2003.

[Syr04] Tommi Syrjänen. Cardinality constraint programs. In Jose Julio Alferes and Joao Leite, editors, *The 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, pages 187–200. Springer-Verlag, September 2004.

[TeB] TEST BUILDER - build scorm-based tests and quizzes, Homepage. http://www.e-learningconsulting.com/products/testbuilder.html, August 2007.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[W3Ca] W3C. Homepage W3C World Wide Web Consortium. http://www.w3.org/, August 2007.

[W3Cb] W3C. W3C World Wide Web Consortium , Technical Reports and Publications. http://www.w3.org/TR, August 2007.

[Web] Homepage WEBCT, A Blackboard Company. http://www.webct.com, August 2007.

[WHSB99] Stefan Werner, Axel Hunger, Frank Schwarz, and Sladjan Buzuk. CONGA: A Course Online/Offline Information and Guidance System to support an International Degree Course. In *Proceedings of ICCE 99*, pages 577–583. IOS Press, 1999.

# Index

# Anhang A

# Einige Abkürzungen

| Symbol | Bedeutung | Hinweis |
|---|---|---|
| $D^{prT}_{(v_1,\ldots,v_n)}$ | $(\pi_{A_0} \circ \sigma_{A_1=v_1 \wedge \cdots \wedge A_n=v_n}(prT)) \cap D = \{q \in D \mid (q, v_1, \ldots, v_n) \in prT\}$ | Definition 5.2.20 |
| $D^{prT}_{(V_1,\ldots,V_n)}$ | $\bigcup_{(v_1,\ldots,v_n) \in \prod_{i=1}^{n} V_i} D^{prT}_{(v_1,\ldots,v_n)}$ | Definition 5.2.20 |
| $D^{roleCons}_{\text{one}}$ | $(\pi_1 \circ roleCons)(D \times \{\text{one}\}) = \{q \in D \mid roleCons(q, \text{one})\}$ | Definition 5.2.34 |
| $D^{roleCons}_{\text{all}}$ | $(\pi_1 \circ roleCons)(D \times \{\text{all}\}) = \{q \in D \mid roleCons(q, \text{all})\}$ | Definition 5.2.34 |
| $\mathfrak{C}^u_{pr_iTrans}(t)$ | $\mathfrak{C}\left(\left(Q^{roleCons}_{\text{all}}\right)^{propTrans}_{t} \cup \left(Q^{roleCons}_{\text{one}}\right)^{pr_iTrans}_{t}\right)$ | Definition 5.2.52 |
| $\mathfrak{C}^u(t)$ | $\bigvee_{i=1}^{r} \mathfrak{C}^u_{pr_iTrans}(t)$ | Definition 5.2.52 |
| $D_{cond}$ | $\sigma_{cond}(D)$ | Definition 5.3.16 |
| $D_{T,v,B}$ | $D_{Type=T,\texttt{Sort}=v,anc=B}$ | Definition 5.3.18 |