Technische Universität Hamburg-Harburg

Anwendung von ASP auf Planungsprobleme

Studienarbeit

Benjamin Jürgens
15.11.2013

Inhaltsverzeichnis

1 Einleitung

Das Ziel der Studienarbeit war es, die Anwendbarkeit von Answer set programming auf eine reale Problemstellung zu ergründen. Als System kam DLV zum Einsatz, ein System zur logischen Programmierung, welches die stable model-Semantik unter dem Paradigma von answer set programming implementiert. (1)

Mit diesem System sollte versucht werden ein System zu programmieren, mit dem Übernachtungen in einer Unterkunftsverwaltung geplant werden können. Dazu sollte das System in der Lage sein, aus Bedarfsmeldungen und einer gegebenen Unterkunftsstruktur mögliche Belegungen zu erzeugen, die anhand verschiedener Regeln als optimal zu bezeichnen ist.

Die Umsetzung sollte sich ähnlich wie die Objektorientierte Programmierung so eng wie möglich an der Realität orientieren, künstlich festgelegte Einschränkungen sollte so weit wie möglich vermieden werden.

Hierbei sollte der Frage nachgegangen werden, wie gut sich ASP zur Lösung eines Planungsproblems eignet und wie leistungsfähig das resultierende System ist. Diese Frage ist vor allem vor dem Hintergrund, dass ASP als eine Form der deklarativen Programmierung zur Schlussfolgerung auf einer Menge von Fakten und Annahmen gedacht ist, da es im gewählten Szenario nur wenige Möglichkeiten gibt, eine Lösung als objektiv als richtig oder falsch zu klassifizieren.

2 Datenstrukturen

Die verschiedenen nötigen Eingabedaten wurden entsprechend ihrer logischen Struktur in verschiedene Dateien aufgeteilt, um verschiedene Bereiche klar gegeneinander abzugrenzen und eine übersichtliche und leicht verständliche Struktur zu erreichen. Als Nebeneffekt wurde dadurch ein hohes Maß an Flexibilität erreicht; um verschiedene Szenarien schnell testen und vergleichen zu können, müssen bei Aufruf des Programmes lediglich die entsprechenden Dateien angegeben werden.

Bei der Auswahl der zur Verfügung stehenden Attribute habe ich mich an der realen Datenstruktur orientiert, die ihrerseits auf der Basis von Kundenanforderungen festgelegt wurde. Der Übersichtlichkeit halber wurden alle Attribute weggelassen, bei denen von Anfang an klar war, dass sie für die Aufgabenstellung nicht relevant sind.

Hier tat sich auch gleich die erste Frage zur Umsetzung auf:

Welches Schema ist am besten geeignet, um Zusammenhänge und Eigenschaften eines objektorientiert entworfenen Systems abzubilden?

Es gibt im Wesentlichen zwei mögliche Umsetzungen:

- Für jede Klasse wird ein Prädikat mit dem Namen der Klasse definiert, in dem alle Eigenschaften festgelegt werden
- Für jede Eigenschaft bzw. jede eng zusammenhängende Gruppe von Eigenschaften wird ein Prädikat mit dem Namen der Klasse, gefolgt von dem Namen der Eigenschaft definiert

Ich habe mich für die Definition eines Prädikates pro Klasse entschieden. Überlegungen zu den Vor- und Nachteilen sind in Kapitel 6.1.1 zu finden.

2.1 Unterkunft

Unterkunftsdaten sind alle Daten, die die generelle, nur äußerst selten veränderliche Struktur der Unterkunftsgebäude beschreiben. Die verschiedenen Objekte sind in einer Baumstruktur organisiert, siehe Abbildung 1 Um die Zusammenhänge zu definieren, verfügt jede Klasse über ein ID-Feld und in jedem Objekt ist eine Referenz zur übergeordneten Struktur definiert. Außerdem fällt in diesen Bereich die Definition der Größenkategorien für die Zimmer und die Definition eines abgeleiteten Prädikates, das die Anzahl der Betten in einem Zimmer zur Verfügung stellt.

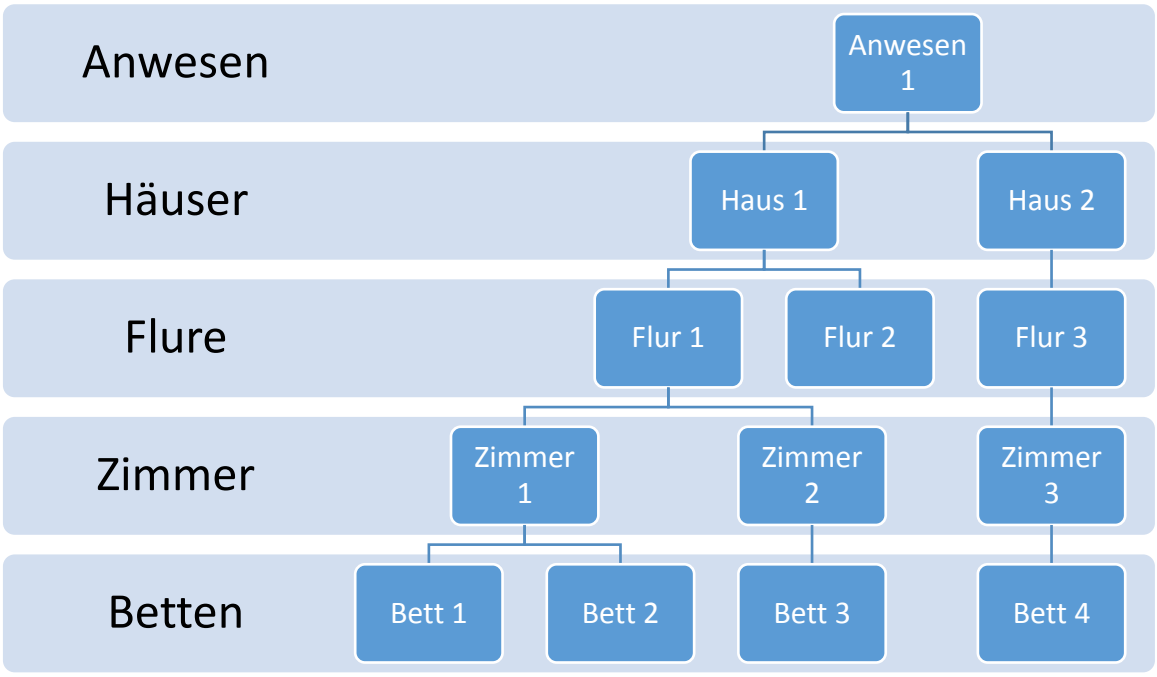

Abbildung 1: Struktur der Unterkünfte

2.1.1 Betten

Für Betten sind folgende Attribute definiert:

- ID
- Raum in dem das Bett steht
- Länge in cm

Die Länge könnte verwendet werden, um sicherzustellen, dass besonders große Personen auch ein entsprechend großes Bett zugewiesen bekommen. Diese Funktion wurde noch nicht umgesetzt.

Die resultierende Definition sieht folgendermaßen aus:

```
bed(Bed, Room, Length)
```

2.1.2 Zimmer

Für Zimmer sind folgende Attribute definiert:

- ID
- Flur auf dem sich das Zimmer befindet
- Größe in m²

- Wahrheitswert, ob in dem Zimmer ein Badezimmer vorhanden ist
- Wahrheitswert, ob der Zimmer behindertengerecht ist

Die Größe und die Information über das Badezimmer werden zur Definition von Kategorien verwendet.

Die Information, ob das Zimmer behindertengerecht ist wird verwendet, um sicherzustellen, dass behinderte Personen in einem behindertengerechtem Zimmer untergebracht werden.

Die resultierende Definition sieht folgendermaßen aus:

```
room(Room, Floor, Size, HasBathroom, IsAccessible)
```

2.1.3 Flure

Für Flure sind folgende Attribute definiert:

- ID
- Haus in dem sich der Flur befindet

Die Definition sieht folgendermaßen aus:

```
floor(Floor, House)
```

2.1.4 Häuser

Für Häuser sind folgende Attribute definiert:

- ID
- Anwesen auf dem das Haus steht

Die Definition sieht folgendermaßen aus:

```
house(House, Estate)
```

2.1.5 Anwesen

Für Anwesen sind außer der ID keine Attribute definiert, die Definition sieht folgendermaßen aus:

```
estate(Estate)
```

2.1.6 Zimmerkategorien

Da es für die Formulierung von Bedarfsmeldungen (siehe Kapitel 0) wenig sinnvoll erscheint, die gewünschte Zimmergröße in Quadratmetern anzugeben, werden dafür Kategorien definiert. Hierbei wird ein bestimmter Zahlenbereich einer Kategorie zugeordnet.

Die Definition ist wie folgt:

```
sizeCategory(Category, MinimumSize, MaximumSize)
```

Anmerkung: Diese Struktur erlaubt und unterstützt es auch, sich überlappende Bereiche zu definieren. So wäre es auch möglich, eine Kategorie zu definieren, die sämtliche Zimmer beinhaltet. Diese könnte verwendet werden, wenn die Zimmergröße keine Rolle spielt.

2.2 Personen

Für Personen sind folgende Attribute definiert:

- ID
- Name
- Geschlecht
- Wahrheitswert, on die Person eine Behinderung hat

Die Definition sieht folgendermaßen aus:

```
person(Person, Name, Gender, IsDisabled)
```

Außerdem sind die Prädikate

```
participant(Person)
```

und

```
teacher(Person)
```

definiert, um einer Person eine Rolle zuzuweisen. Im objektorientierten Datenmodell entspricht die Rolle einer Vererbung der Klasse Person. Dieser Punkt ist in Kapitel 6.1.1 näher ausgeführt.

2.2.1 Gruppen

Personen können für die Unterkunftsplanung zu Gruppen zusammengefasst werden. Diese Möglichkeit wurde eingeführt, um als gleichwertig angesehene Permutationen bei der Belegung zu reduzieren (siehe Kapitel 5.1). Eine Gruppe wird durch eine ID identifiziert und enthält eine Liste von Mitgliedern.

Die Definition sieht folgendermaßen aus:

```
group(Group, [Person_1, …, Person_n])
```

2.3 Kurse

Da es sich bei der modellierten Einrichtung um eine Bildungseinrichtung handelt, an der verschiedene Arten von Lehrgängen stattfinden, müssen auch diese Lehrgänge abgebildet werden, um sie bei der Planung berücksichtigen zu können. Es gibt drei verschiedene Arten von Lehrgängen:

- Studiengruppen (entspricht einer Schulklasse)
- Wahl- und Wahlpflichtfächer, die von Studenten zusätzlich zu ihrer Studiengruppe gewählt werden können
- Seminare, die für externe Teilnehmer angeboten werden

Die Unterschiede spielen für die Unterkunftsplanung allerdings keine Rolle und aufgrund der Tatsache, dass alle Lehrgänge ungeachtet ihres Typs in der gleichen Tabelle der Datenbank gespeichert sind, wird für alle Lehrgänge die gleiche Definition verwendet.

Folgende Eigenschaften können für einen Lehrgang definiert werden:

- ID
- Titel
- Anfang
- Ende
- Ort (Anwesen)

Letzteres stellt natürlich eine Vereinfachung der vorhandenen Daten dar, eigentlich wird der Raum, in dem der Unterricht stattfindet, gespeichert. Da diese Information hier irrelevant ist, wurde sie auf das Anwesen generalisiert. Datenstrukturen zur Abbildung der Unterrichtsräume sind dadurch nicht nötig.

Die Definition der Kurse sieht folgendermaßen aus:

```
course(Course, Title, Begin, End, Estate)
```

2.3.1 Kurs-Teilnehmer

Für die Zuordnung von Teilnehmern zu Kursen wird ein Prädikat definiert, das der Zuordnungstabelle einer N*M-Beziehung entspricht. Da im vorliegenden Kontext lediglich die reine Zuordnung benötigt wird, sieht die Definition folgendermaßen aus:

```
course_participant(Course, Participant)
```

2.4 Bedarfsmeldungen

Um einen Übernachtungswunsch auszudrücken, wird ein Prädikat benutzt. Durch diese Bedarfsmeldung soll die Anzahl der in Frage kommenden Zimmer möglichst klein gehalten werden, ohne dabei künstliche Einschränkungen aufzubauen. Folgende Eigenschaften sind für eine Bedarfsmeldung definiert:

- Person
- Anfang
- Ende
- Ort (an diesem Ort sollten auch die Kurse stattfinden, an denen die Person teilnimmt, falls es welche gibt)
- Gewünschte Zimmergröße
- Gewünschte Betten im Zimmer (meistens Einzel- oder Doppelzimmer, es gibt aber auch Mehrbettzimmer)
- Eigenes Badezimmer gewünscht (Wahrheitswert)

Die Definition sieht damit folgendermaßen aus:

```
demand(Person, Begin, End, Estate, Category, BedsInRoom,
BathroomWanted)
```

2.4.1 Gruppen-Bedarfsmeldungen

Um auch für Gruppen Bedarfsmeldungen definieren zu können, wird ein weiteres Prädikat definiert. Dieses unterscheidet sich von einer Bedarfsmeldung für eine Person nur dadurch, dass statt einer Person eine Gruppe referenziert wird:

```
groupDemand(Group, Begin, End, Estate, SizeCategory, BedsInRoom,
BathroomWanted)
```

2.5 Übernachtungen

Aus den Bedarfsmeldungen werden von der Programmlogik Übernachtungen erzeugt. Dazu wird anhand der in der Bedarfsmeldung festgelegten Werte für Anwesen und gewünschter Art des Zimmers (Kategorie, Anzahl Betten, Badezimmer inklusive) ein Zimmer festgelegt (siehe Kapitel 3). Folgende Definition gilt:

```
booking(Person, Begin, End, Room)
```

2.5.1 Gruppen-Übernachtungen

Die Definition unterscheidet sich wieder nur im ersten Term:

```
groupBooking(Group, Begin, End, Room)
```

3 Regelwerk

Das Regelwerk enthält sämtliche Logik und stellt damit das Kernstück des Programms dar. Es ist in drei Bereiche unterteilt.

3.1 Zusätzliche Definitionen

Hier sind zusätzliche Prädikate definiert, die direkt auf den Fakten basieren.

3.1.1 Anzahl der Betten in einem Zimmer

Für die Planung ist es von Bedeutung, wie viele Betten sich in einem Zimmer befinden. Da die Größe der Betten noch nicht berücksichtigt wird, ist dies die einzige Information, die aus der Definition der Betten gewonnen wird. Zur Berechnung wird die Aggregatfunktion `#count{}` benutzt.

3.1.2 Anzahl der benötigten Zimmer für eine Gruppen-Bedarfsmeldung

Um für eine Gruppen-Bedarfsmeldung die korrekte Anzahl von Zimmern zu buchen, muss diese zunächst berechnet werden. Sie ist abhängig von der gewünschten Bettenanzahl, die in der Gruppen-Bedarfsmeldung angegeben ist und von der Anzahl männlicher und weiblicher Mitglieder der Gruppe. Die Formel zur Berechnung lautet wie folgt:

$$RoomsNeeded = round\_up\left(\frac{Males}{BedsPerRoom}\right) + round\_up\left(\frac{Females}{BedsPerRoom}\right)$$

Dabei gelten folgende Definitionen:

RoomsNeeded	Anzahl der benötigten Zimmer
BedsPerRoom	gewünschten Bettenanzahl pro Zimmer
Males	Anzahl der männlichen Gruppenmitglieder
Females	Anzahl der weiblichen Gruppenmitglieder
round_up(q)	Funktion, die die rationale Zahl q zur nächsthöheren ganzen Zahl aufrundet

Auf Grund der Tatsache, dass nur mit positiven ganzen Zahlen gerechnet wird und nicht ganzzahlige Ergebnisse immer abgeschnitten / abgerundet werden, gibt es keine Rundungsfunktionen. Deshalb muss die Formel umgeschrieben werden, um den Rundungsfehler auszugleichen:

$$RoomsNeeded = \frac{Males - 1}{BedsPerRoom} + 1 + \frac{Females - 1}{BedsPerRoom} + 1$$

Bei allen Brüchen, deren Ergebnis keine ganze Zahl ist, muss das Ergebnis um 1 erhöht werden, um auf die korrekte Zahl zu kommen. Um das Ergebnis für Brüche mit ganzzahligem Ergebnis zu korrigieren, wird der Zähler um 1 verringert. Diese Formel versagt allerdings, wenn eine Gruppe nur aus Personen eines Geschlechts besteht, weil einer der beiden Zähler dann negativ würde. Um auch diesen Fall abzudecken, müssen negative Zahlen vermieden werden. Dies wird erreicht, indem die 1 erweitert und in den Bruch integriert wird:

$$RoomsNeeded = \frac{Males + BedsPerRoom - 1}{BedsPerRoom} + \frac{Females + BedsPerRoom - 1}{BedsPerRoom}$$

Da als weitere Einschränkung Verkettung von Operatoren nicht unterstützt wird, müssen für jeden Rechenschritt Zwischenvariablen definiert werden. Damit ergibt sich folgende Definition:

```
roomsNeededForGroupDemand(Group, GroupDemandBegin,
GroupDemandEnd, RoomsNeeded) :-
      groupDemand(Group, GroupDemandBegin, GroupDemandEnd, _, _,
      BedsPerRoom, _),
      group(Group, Members),
            #count{ MalePerson : #member(MalePerson, Members),
      person(MalePerson, _, m, _) } = Males,
      M1 = Males + BedsPerRoom,
      #prec(M1, M2),
      M3 = M2 / BedsPerRoom,
            #count{ FemalePerson : #member(FemalePerson, Members),
      person(FemalePerson, _, f, _) } = Females,
      F1 = Females + BedsPerRoom,
      #prec(F1, F2),
      F3 = F2 / BedsPerRoom,
      RoomsNeeded = M3 + F3.
```

3.2 Buchungen

Bei den folgenden Regeln stellt sich immer wieder die Frage, wann sich zwei Zeitspannen, die jeweils durch Anfang und Ende definiert sind, überschneiden. Meine Erfahrung zeigt, dass zur Prüfung immer wieder unnötig komplizierte Bedingungen formuliert werden, weshalb der Sachverhalt an dieser Stelle unabhängig vom Kontext betrachtet werden soll.

Obwohl die Lösung eigentlich sehr simpel ist, habe ich schon des Öfteren und auch bei mir selbst beobachtet, dass zunächst begonnen wird, in dem die beiden Anfangszeiten miteinander verglichen werden. Danach wird dann meist geprüft, ob die spätere Anfangszeit vor dem Ende der anderen Zeitspanne liegt und für den allgemeinen Fall, dass die beiden Zeitspannen nicht beliebig vertauschbar sind, müssen die Beziehungen mit vertauschten Rollen auch noch formuliert werden.

Auf die wohl einfachste Lösung kommt man, wenn man sich die Grenzfälle anschaut, bzw. überlegt unter welchen Bedingungen sich die Zeitspannen nicht überschneiden. Diese Bedingung lässt sich für zwei gleichwertige Ereignisse sehr kompakt formulieren:

Zwei Ereignisse überscheiden sich nicht, wenn das eine endet, bevor das andere anfängt.

Der Grenzfall definiert sich dadurch, dass der Anfang des einen Ereignisses mit dem Ende des anderen zusammenfällt.

Damit müssen folgende Ungleichungen erfüllt sein, damit sich die Ereignisse E1 und E2 überscheiden.

$$Anfang_{E1} < Ende_{E2}$$

$$Anfang_{E2} < Ende_{E1}$$

Wenn mehrere Zeitspannen daraufhin geprüft werden sollen, ob es einen gemeinsamen Zeitraum gibt, in dem sich alle Zeitspannen überscheiden, kann ein willkürlich gewählter Zeitpunkt definiert werden und geprüft werden, ob es für diesen Zeitpunkt Werte gibt, so dass er nach allen Anfangszeiten und vor allen Endzeiten liegt. Alternativ kann man die späteste Anfangszeit bestimmen und prüfen, ob diese vor der frühesten Endzeit liegt.

3.2.1 Generierung der Buchungen

Zunächst werden für alle möglichen Kombinationen von Bedarfsmeldung und passendem Zimmer Buchungen generiert. Dies ist die einzig mögliche Vorgehensweise, denn um an dieser Stelle direkt eine Regel definieren zu können, die pro Bedarfsmeldung eine Buchung erzeugt, müsste der Oder-Operator wie eine Aggregatsfunktion auf ein symbolic set angewendet werden können, um auszudrücken, dass genau eine der möglichen Buchungen in einem Modell vorkommt.

Dies entspricht auch dem Vorgehen im DLV User manual (1)

Als nächster Schritt wird die Menge der Buchungen zu jedem Bedarf auf festgelegt, wodurch alle anderen Modelle wegfallen.

3.2.2 Grundregeln

Folgende Grundregeln sind definiert:

- Ein Zimmer darf zu keiner Zeit mehr Buchungen haben, als Betten vorhanden sind
- Zimmer dürfen nicht von Personen mit unterschiedlichem Geschlecht belegt werden
- Behinderte Personen müssen in einem behindertengerechten Zimmer untergebracht werden
- Wenn ein Badezimmer gewünscht ist, muss auch ein Zimmer mit Badezimmer gebucht werden
- Lehrer und Schüler können nicht im gleichen Zimmer untergebracht werden

3.2.3 Erweiterte Regeln

Eine weitere Gruppe von neuen Regeln dient dazu, bestimmte nicht erwünschte Belegungen auszuschließen sowie eine möglichst dichte Belegung zu erreichen:

- Lehrer und Schüler sollen möglichst nicht auf dem gleichen Flur untergebracht werden (weak constraint)
- Nicht behinderte Personen dürfen nur dann in einem behindertengerechten Zimmer untergebracht werden, wenn kein anderes passendes Zimmer frei ist
- Personen die kein eigenes Badezimmer brauchen dürfen nur dann in einem Raum mit Badezimmer untergebracht werden, wenn kein anderes passendes Zimmer frei ist
- Teilnehmer des gleichen Lehrgangs dürfen nur dann in unterschiedlichen Fluren untergebracht werden, wenn in keinem Flur Platz für beide ist
- Schließe Lösungen aus, in denen Personen des gleichen Geschlechts in unterschiedlichen Räumen mit gleichen Eigenschaften untergebracht sind, obwohl in beiden Räumen noch freie Betten verfügbar sind

3.2.4 Ordnungsregeln

Schließlich wurden Regeln definiert, welche die möglichen Permutationen anhand der Zimmer-ID bzw. der Personen-ID einschränken, indem sie auf Basis der ID eine Ordnung definieren:

- Schließe Lösungen aus, in denen ein Zimmer frei bleibt, während ein gleichwertiges Zimmer mit höherer ID belegt wird
- Schließe Lösungen aus, in denen gleichwertige Zimmer von Personen des gleichen Geschlechts belegt werden und die Reihenfolge der Personen-ID nicht der Reihenfolge der Zimmer-ID entspricht.

Bei allen Regeln, die bestimmte Lösungen ausschließen muss sichergestellt sein, dass es auch mindestens eine Lösung gibt, die durch diese Regel nicht gefiltert wird. Außerdem muss auf Grund des Anspruchs, dass diese Regeln nur die Menge der als gleichwertig angesehenen Permutationen einschränken sollen, sichergestellt sein, dass bei der Kombination der Regeln noch gültige Lösungen übrig bleiben, dass die Schnittmenge der gültigen Lösungen aus allen Regeln also nicht leer ist.

3.3 Gruppen-Buchungen

Das Regelwerk für Gruppen-Buchungen enthält im Wesentlichen die gleichen Regeln wie das Regelwerk für die Einzelbuchungen, kommt aber mit weniger Regeln aus, da die Eigenschaften der Personen nicht berücksichtigt werden müssen und Permutationen in der Belegung durch die Abstrahierung auf die Gruppe ausgeblendet werden.

Die Generierung der Gruppenbuchungen erfolgt nach dem gleichen Prinzip wie für die Einzelbuchungen.

Da für einen Bedarf einer Gruppe im Allgemeinen mehrere Zimmer gebucht werden müssen, sieht die Regel zur Auswahl der richtigen Anzahl von Buchungen etwas anders aus und bedient sich des in Kapitel 3.1.2 vorgestellten Prädikates.

Analog zu den Einzelbuchungen müssen Räume mit Badezimmer gebucht werden, wenn dies gewünscht ist.

Ebenfalls analog zu den Einzelbuchungen gibt es eine Regel, die verhindert, dass eine Gruppe unnötigerweise auf verschiedenen Fluren untergebracht wird.

Schließlich wird eine Regel definiert, um zu verhindern, dass Zimmer mit kleiner ID frei bleiben, während Zimmer mit höherer ID belegt werden.

4 Erste Implementierung

4.1 Regelwerk

Im ersten Schritt wurde ein minimaler Ansatz implementiert, der alle in Frage kommenden Lösungen berechnet. Sämtliche Regeln, die die Ergebnismenge auf eine möglichst dichte Belegung eingrenzen, wurden weggelassen.

Das Regelwerk beschränkt sich damit auf folgende Regeln

- Generiere für jede Bedarfsmeldung genau eine Buchung
- Stelle sicher, dass ein Zimmer zu keiner Zeit für mehr Person gebucht ist, als es Betten hat
- Verhindere, dass männliche und weibliche Personen gleichzeitig im gleichen Zimmer untergebracht sind.

4.2 Testszenario

Für den Test wurde eine Unterkunft modelliert, die aus sechs Doppelzimmern der gleichen Kategorie besteht. In allen Bedarfsmeldungen wurde folglich diese Kategorie angegeben. Als weitere Vereinfachung verwenden alle Bedarfsmeldungen den gleichen Zeitraum. Der Integer-Wertebereich ist wie auch bei allen weiteren Tests auf 731 begrenzt. Diese Zahl wird für die Definition von gültigen Zeitpunkten benutzt und ergibt sich aus einer Begrenzung des Planungshorizontes auf zwei Jahre, was einen realitätsnahen Wert darstellt. Beginnend bei einer Bedarfsmeldung wurde das Programm nacheinander für eine steigende Anzahl an Bedarfsmeldungen aufgerufen. Dabei wurde abwechselnd eine männliche und eine weibliche Person hinzugefügt.

Durch dieses Setup wurde ein weitgehend lineares Verhalten der Eingangsdaten erreicht, was die Beurteilung und mathematische Analyse der Ergebnisse sehr vereinfacht, ohne dass das Modell mathematisch trivial wird.

Trivial würde in diesem Fall bedeuten:

Es gibt nur Einzelzimmer, nur eine Kategorie und es wird immer der gleiche Zeitraum verwendet. Das Geschlecht der Personen wäre dann irrelevant und Anzahl der möglichen Lösungen berechnet sich als

Formel 1: Anzahl der Answer sets

$$AnswerSets = \frac{Rooms!}{(Rooms - Demands)!}$$

Dies entspricht der Formel für Variationen ohne Wiederholung.

4.2.1 Ergebnisse

Der Testlauf ergab die statistischen Werte in **Fehler! Verweisquelle konnte nicht gefunden werden.**, von denen vor allem die Anzahl der Answer Sets und die Laufzeit interessant sind:

Tabelle 1: Ergebnisse des ersten Tests

demands	Rules	Constraints	Structural Size	Choice Points	Answer sets	Sekunden
1	762	37	1687	1	6	1
2	1524	86	3309	7	30	1
3	2286	873	7883	37	150	2
4	3048	934	10267	193	630	4
5	3810	995	12651	763	2520	7
6	4572	1068	15059	3241	8280	21
7	5334	1141	17467	10597	24840	59
8	6096	1226	19899	28603	59400	162
9	6858	1311	22331	64849	118800	381
10	7620	1408	24787	95587	162000	703
11	8382	1505	27243	113533	162000	1298
12	9144	1614	29723	133873	162000	1964

Die Anzahl der Regeln ist exakt linear zur Anzahl der Bedarfsmeldungen und wie weitere Tests ergeben haben außerdem linear zur Anzahl der Räume. Die strukturelle Größe und besonders die Anzahl der Constraints machen in Zeile drei einen deutlichen Sprung, was wahrscheinlich daran liegt, dass es ab Zeile drei mehrere Bedarfsmeldungen für Personen des gleichen Geschlechts gibt. Abgesehen von diesem Sprung können sie aber auch durch eine lineare Funktion approximiert werden.

Damit stellen die choice points und die answer sets den limitierenden Faktor dar und müssen näher betrachtet werden

Die Laufzeit hat nur informellen Charakter, da sie naturgemäß nicht reproduzierbar ist. Daher soll nur die Anzahl der Answer sets näher betrachtet werden.

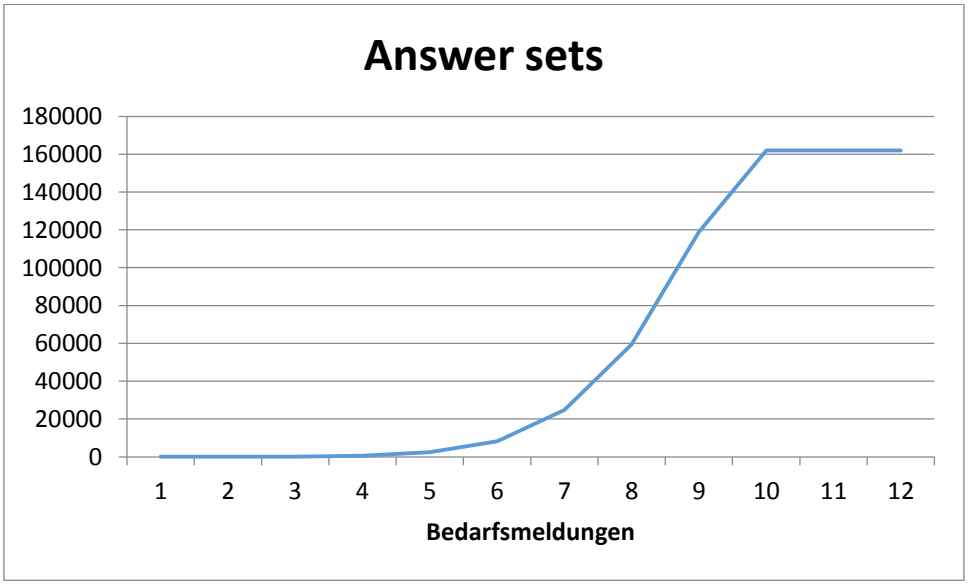

Diagramm 1: Anzahl der Answer sets in Test 1

In Diagramm 1 wird deutlich, dass die Anzahl der Permutationsmöglichkeiten auch für ein überschaubares Testszenario sehr schnell Dimensionen annehmen kann, die auch auf leistungsfähiger

Hardware nicht in annehmbarer Zeit verarbeitet werden können. Auf Grund des gegenüber dem trivialen Fall nur leicht veränderten Szenarios stellt sich die Frage, ob das Wachstum der Answer sets sich ähnlich verhält, wie in Formel 1 angegeben. Bei Betrachten der Ergebnisse der ersten Zeilen erkennt man, dass sich die Anzahl der Answer sets im getesteten Szenario nach einer ähnlichen Formel berechnet. Diagramm 2 veranschaulicht eine weitere Überlegung zum Verhalten der Answer sets: Die Menge der Answer sets kann gut durch eine Fakultätsfunktion beschrieben werden. Diese kann wiederum für eingeschränkte Bereiche durch eine Exponentialfunktion approximiert werden. Dies gilt besonders für eine große Anzahl verfügbarer Zimmer und besonders für wenige Bedarfsmeldungen. Je voller das System belegt ist, desto geringer wird das Wachstum der answer sets. Bereits bei zehn Gästen (fünf männlichen und fünf weiblichen) sind alle Zimmer belegt, so dass sich die Menge der Answer sets nicht mehr ändert. Auch vorher ist bereits ein Sättigungseffekt zu erkennen.

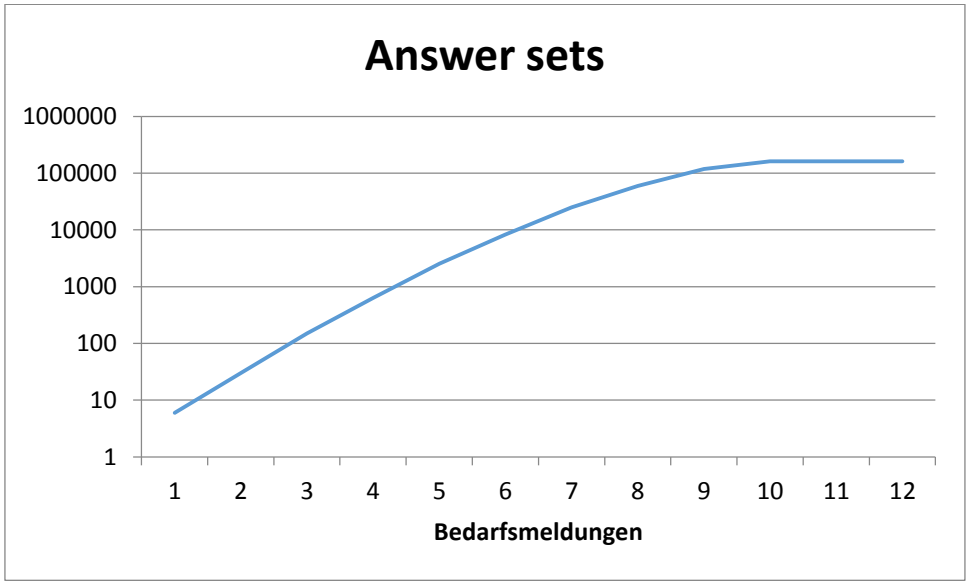

Diagramm 2: Anzahl der Answer sets in Test 1 (logarithmisch)

4.3 Optimierungsmöglichkeiten

Nach diesem kombinatorisch nachvollziehbaren aber dennoch nicht sehr zufriedenstellenden Ergebnis soll im nächsten Schritt nach Möglichkeiten gesucht werden, die Leistungsfähigkeit zu optimieren und die Menge der Answer sets nach möglichst nachvollziehbaren Regeln einzuschränken. Dabei sollten möglichst nur Regeln definiert werden, die die vorhandenen Informationen wie die Zugehörigkeit der Teilnehmer zu Kursen nutzen.

5 Zweite Implementierung

5.1 Änderungen gegenüber erster Implementierung

5.1.1 Optimierungen bestehender Regeln

Die Regeln zur Generierung der Buchungen aus den Bedarfsmeldungen wurden unter Verwendung des Oder-Operators (v) zusammengefasst und um eine Einschränkung auf das gewünschte Anwesen erweitert, die in der ersten Version noch nicht vorhanden war. Dies hatte jedoch keinen Einfluss auf die Ergebnisse, da im Testszenario nur ein Anwesen vorhanden war.

Die Regeln, um die erzeugten Buchungen auf die gültigen Fälle einzuschränken, in denen zu jeder Bedarfsmeldung genau eine Buchung existiert, wurden zusammengefasst. Dies wurde durch Verwendung der Aggregatfunktion `#count{}` erreicht und kam nebenbei auch der Lesbarkeit des Codes zu Gute.

5.1.2 Neue Regeln

Die Grundregeln wurden um einige Regeln erweitert, die in der ersten Version noch nicht umgesetzt waren. Des Weiteren wurden Regeln definiert, um eine zu beliebige Verteilung der Buchungen zu verhindern. Außerdem wurden Regeln aufgestellt, um die Anzahl der Permutationen einzuschränken.

Für eine vollständige Liste der Regeln siehe Kapitel 3.2 und 3.3.

5.1.3 Neue Datenstrukturen

Um Permutationen der zugeordneten Zimmer innerhalb einer Gruppe zu reduzieren, wurde durch Definition neuer Prädikate die Möglichkeit geschaffen, Gruppen von Personen zu bilden und Bedarfsmeldungen für die ganze Gruppe zu formulieren. Aus einer Gruppen-Bedarfsmeldung werden dann mehrere Buchungen generiert, um die ganze Gruppe in der gewünschten Kategorie unterzubringen. Diese Strukturen sind in Kapitel 2.2.1, 2.4.1 und 2.5.1 genauer beschrieben. Bei der Definition wurde versucht, Anforderungen, die in der Planungsrealität oft vorkommen möglichst direkt abbildbar zu machen. Je nachdem wofür eine Unterkunft benutzt wird reist ein mehr oder weniger großer Teil der Übernachtungsgäste in Gruppen an und jede Gruppe benötigt mehrere Zimmer. Wie genau die Zimmer dann innerhalb der Gruppe belegt werden ist Sache der Gruppe und braucht deshalb nicht in der Unterkunftsverwaltung festgelegt werden, außerdem kann man davon ausgehen, dass dies von den Gruppen auch nicht gewünscht ist.

5.2 Testszenario 1

Als Testszenario wurde wieder die gleiche Struktur der Unterkunft mit sechs Doppelzimmern der gleichen Kategorie gewählt, um eine gute Vergleichbarkeit mit der ersten Implementierung zu haben. Für diese wurden wie im ersten Test für eine zunehmende Menge an Bedarfsmeldungen Lösungen berechnet.

Zusätzlich wurde der gleiche Test für eine Gruppe durchgeführt, zu der nacheinander die Personen hinzugefügt wurden.

5.2.1 Ergebnisse

5.2.1.1 Einzelbuchungen

Tabelle 2: Ergebnisse des zweiten Tests

demands	Rules	Constraints	Structural Size	Choice Points	Answer sets
1	786	46	1803	1	1
2	1572	89	3854	1	2
3	2358	945	9487	3	2
4	3144	1075	13290	7	2
5	3930	1280	18323	14	6
6	4716	1497	23980	29	24
7	5502	1789	31347	36	12
8	6288	2093	39578	57	6
9	7074	2472	49999	90	30
10	7860	2863	61524	127	180
11	8646	3329	75719	43	60
12	9432	3807	91258	17	20

Hier zeigt sich ein Ergebnis, das nicht nur eine wesentliche Verbesserung gegenüber dem ersten Test zeigt, sondern auch eine auf den ersten Blick kaum nachvollziehbare Schwankung der Ergebnismenge aufweist. Die im Diagramm besonders deutlich wird.

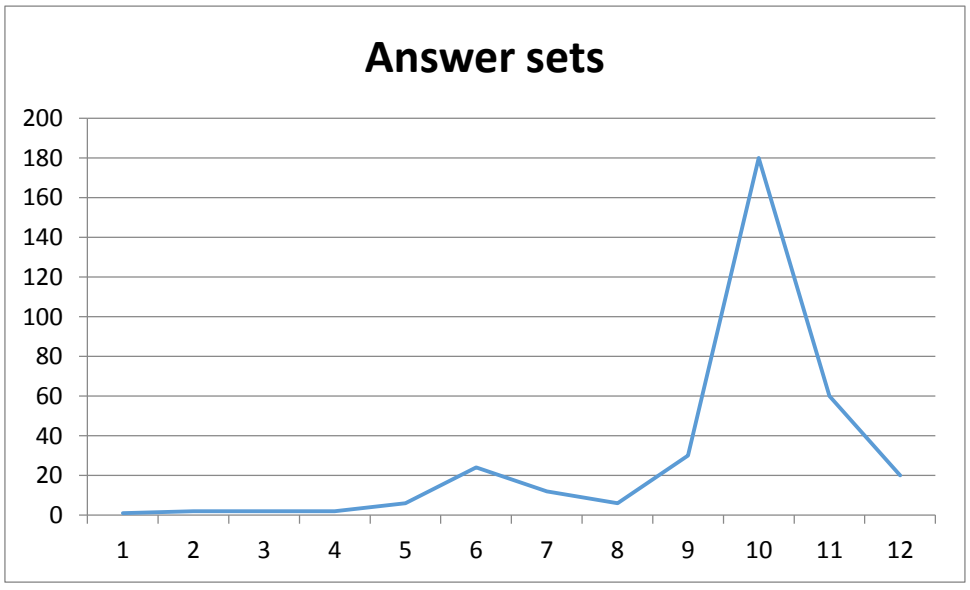

Diagramm 3: Anzahl der Answer sets in Test 2

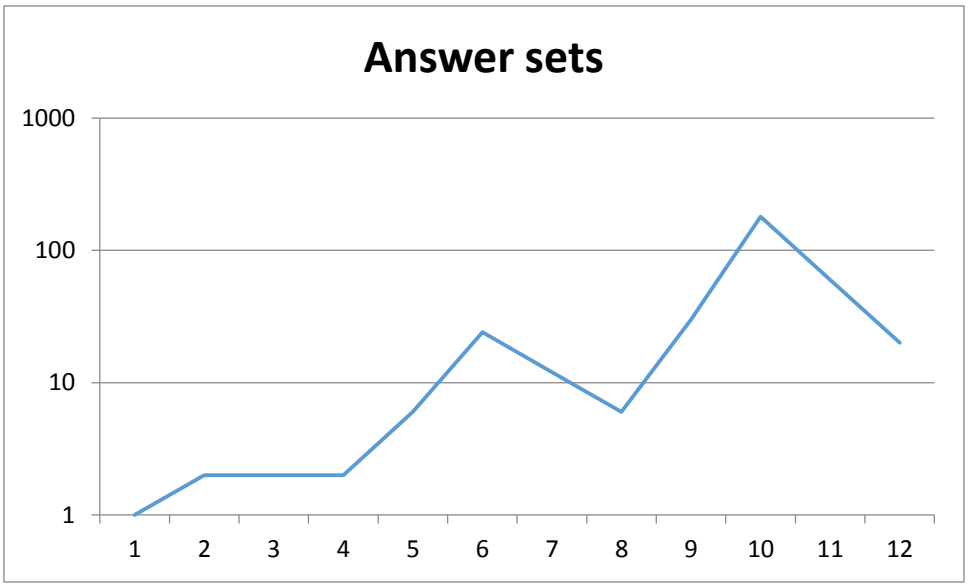

Diagramm 4: Anzahl der Answer sets in Test 2 (logarithmisch)

Um die Ergebnisse nachzuvollziehen, muss man sich die beiden letzten neuen Regeln anschauen, die eine bestimmte Ordnung in den Buchungen forcieren.

Die erste Regel legt im vorliegenden Fall mit gleichwertigen Zimmern und Bedarfsmeldungen die Zimmer, die verwendet werden dürfen eindeutig fest, so dass alle berechneten Lösungen Permutationen in der Belegung dieser Zimmer darstellen.

Die zweite Regel legt innerhalb der Gruppe der männlichen bzw. weiblichen Personen eine Reihenfolge für die Belegung fest. Wenn alle benutzten Zimmer voll belegt sind kann sie besonders gut greifen und die einzige Variabilität besteht in der Festlegung, in welchen Zimmern weibliche und in welchen männliche untergebracht werden. Die Anzahl der Answer sets berechnet sich dann als

$$AnswerSets = \frac{\frac{Demands}{2}!}{\left(\frac{Demands}{4}!\right)^2}$$

Das fakultative Verhalten wird durch die Wirkung dieser Regeln zwar stark verfälscht, lässt sich aber in der logarithmischen Darstellung immer noch gut erahnen, so dass auch hier in einem größeren Szenario ab einem gewissen Punkt die Lösungsmenge zu groß wird. Dieses Verhalten liegt aber in der Natur der Aufgabenstellung begründet die Grenze kann durch Optimierungen nur nach hinten verlagert werden.

5.2.1.2 Gruppenbuchungen

Für die Gruppenbuchung wurde in allen zwölf Fällen nur eine einzige Lösung berechnet. Daher gibt es zu diesem Test keine Ergebnisse, die erläutert werden müssten und es bleibt festzustellen, dass die Zusammenfassung von Personen zu Gruppen ein sehr wirkungsvolles Mittel darstellt, um die Komplexität der Aufgabe zu reduzieren.

5.3 Testszenario 2

Als zusätzliches Testszenario um die Leistungsfähigkeit in realitätsnahen Szenarien beurteilen zu können wurde eine Unterkunft mit 99 Zimmern unterschiedlicher Kategorien auf verschiedenen Fluren entworfen.

Getestet wurde das Programm beginnend bei einer einzigen Bedarfsmeldung für eine Zimmerkategorie, die auf mehreren Fluren verfügbar ist, die Zahl sollte im Folgenden schrittweise erhöht werden.

Die Zahl der Answer sets für eine Bedarfsmeldung entspricht wenig überraschend der Anzahl der Flure, da in jedem Flur nur das Zimmer mit der niedrigsten ID ausgewählt wird. Die Berechnung dauert allerdings schon 45 Sekunden, wobei der größte Teil auf die Instanziierung entfällt. Ein Aufruf für zwei Bedarfsmeldungen wurde nach 10 Minuten abgebrochen, bevor die Instanziierung abgeschlossen war.

Tabelle 3: Ergebnisse für Testszenario 2

demands	Rules	Constraints	Structural Size	Choice Points	Answer sets
1	3930	42286	260813	1	4

Ein Test des Systems mit einer Gruppen-Bedarfsmeldung brachte auch keine besseren Ergebnisse, die Instanziierung dauerte mit 79 Sekunden sogar noch wesentlich länger, obwohl die Anzahl der Regeln mit 150 deutlich geringer ist.

Das System skaliert also sehr schlecht mit der Anzahl der Zimmer, bzw. mit der Größe der Unterkunft und kann nicht als praxistauglich angesehen werden.

6 Technische Realisierung

6.1 Umsetzung objektorientierter Paradigmen

6.1.1 Repräsentation von Objekt-Eigenschaften

Da ich mich entschieden hatte, alle Eigenschaften eines Objektes in einem Prädikat zu definieren, musste ich mich zu Anfang möglichst auf die implementierten Attribute festlegen, da eine Erweiterung um weitere Attribute eine Anpassung an sämtlichen verwendeten Stellen nötig machen würde. Eine Alternative zu dieser Vorgehensweise wäre es, für jede Eigenschaft eines Objektes ein eigenes Prädikat zu definieren.

Für die Klasse Bett sehen die beiden Implementierungen folgendermaßen aus:

Definition in einem Prädikat:

```
bed(Bed, Room, Length).
```

Definition in mehreren Prädikaten:

```
bed(Bed).
bed_inRoom(Bed, Room).
bed_length(Bed, Length).
```

Eine Variante der Definition in mehreren Prädikaten, die Variable Bed über ein universelles Attribut `is(Bed,bed).` als Bett zu deklarieren, wurde zu Anfang in Erwägung gezogen, aber schnell verworfen, da sie keine weiteren Vorteile bietet.

Neben der besseren Lesbarkeit bei der Definition von Fakten, hat die Aufteilung in mehrere Attribute den Vorteil, dass sie einfacher um neue Eigenschaften zu erweitern ist und man sich bei der Definition von Fakten auf die Abbildung der tatsächlich verfügbaren Informationen beschränken kann. Außerdem wird die Lesbarkeit auch bei der Definition von Regeln verbessert, da man nur die relevanten Eigenschaften spezifizieren muss und die Verwendung anonymer Variablen vermeiden kann.

Negativ bei der Definition in mehreren Attributen ist jedoch, dass die Zusammengehörigkeit von Attributen eines Objektes nur über die ID gegeben ist, die bei jeder Wert-Definition angegeben werden muss. Diese Zusammengehörigkeit ist schwerer ersichtlich und bei Objekten wie Bedarfsmeldungen oder Buchungen, deren Schlüssel sich aus mehreren Eigenschaften (Person, Beginn, Ende) zusammensetzt, wird es noch unübersichtlicher.

6.1.2 Vererbung

Die Klassen Participant und Teacher sind ein Beispiel für eine Vererbung der Klasse Person. Diese Klassen haben im Klassenmodell das als Vorlage diente natürlich weitere Eigenschaften wie eine Matrikelnummer oder die Gehaltsgruppe und natürliche etliche weitere Beziehungen zu anderen Klassen wie z.B. Kursen. Diese waren jedoch für die Aufgabenstellung alle nicht relevant, so dass die Prädikate `participant()` und `teacher()` außer der ID keine weiteren Parameter haben. Sie hätten daher auch durch eine Eigenschaft „Rolle“ direkt im Prädikat `person()` definiert werden können.

6.2 Hilfsmittel

6.2.1 Microsoft Excel

Excel wurde neben der Aufbereitung der Testergebnisse auch zur Generierung der Fakten aus einer Tabelle mittels der vorhandenen Textfunktionen benutzt. Dadurch konnten die gewünschten Eigenschaften der Objekte, z.B. der Zimmer übersichtlich und komfortabel in einer Tabelle festgelegt werden und die daraus erzeugten Fakten mussten nur noch an die gewünschte Stelle kopiert werden.

6.2.2 Notepad++

Sehr hilfreich erwies sich der Einsatz von Notepad++ durch die Fähigkeiten zur Syntaxhervorhebung, Faltung von Textblöcken und Auto-Vervollständigung. Um diese Möglichkeiten voll ausschöpfen zu können, wurde basierend auf der Sprachdefinition für Prolog eine Sprachdefinition für Disjunctive Datalog wie DLV es verwendet erzeugt und im Laufe der Arbeit weiterentwickelt. Sprachdateien für Notepad++ sind im Internet unter der Adresse http://sourceforge.net/apps/mediawiki/notepad-plus/?title=User_Defined_Language_Files zu finden.

7 Einschränkungen von DLV

Bei der Programmierung wurden Einschränkungen mit unterschiedlicher Tragweite deutlich. Diese reichte von „muss etwas umständlich programmiert werden“ über „kann für die vorliegende Aufgabe noch mit einem Trick realisiert werden“ bis „ist nicht möglich“. Es sei angemerkt, dass diese Einschränkungen bei den Problemstellungen, auf die DLV bzw. ASP im Allgemeinen ausgerichtet ist, meistens keine große Rolle spielen.

7.1 Verkettung von Funktionen

Eine lediglich ärgerliche Einschränkung ist die Tatsache, dass arithmetische Funktionen nicht verkettet werden können. Wenn eine Verkettung benötigt wird, müssen deswegen für jeden Zwischenschritt temporäre Variablen definiert werden. Dies schränkt die Menge der Probleme, die mit DLV gelöst werden können nicht ein, lässt es für mathematische Aufgabenstellungen aber etwas weniger geeignet erscheinen. Der Grund für diese Einschränkung ist, dass alle Funktionen als Prädikate realisiert sind und Prädikate keinen gültigen Term für die Verwendung in anderen Prädikaten darstellen. Ergebnisse werden der Natur von ASP gemäß nicht berechnet sondern geschlussfolgert.

7.1.1 Verkettung von Aggregatfunktionen

Bei der Programmierung einer Regel wollte ich prüfen, ob es während einer bestimmten Zeitspanne Zeitpunkte gibt, zu denen die Anzahl der Buchungen eines bestimmten Raumes, die diesen Zeitpunkt betreffen, gleich der Anzahl der Betten in diesem Raum ist.

Bei anderen Regeln hatte ich festgestellt, dass eine Prüfung, ob es für eine Variable Werte gibt, die bestimmte Bedingungen erfüllen, am besten durch eine Zählung der Werte, die die Bedingungen erfüllen und einen Vergleich mit 0 programmiert werden kann. Außerdem ist diese Technik die einzige mir bekannte Möglichkeit, das gewünschte zu erreichen. Ein Beispiel ist folgendes Code-Fragment:

```
#count { Person1 :
      booking(Person1, Booking1Begin, Booking1End, Room1),
      Booking1Begin < ParticipantBooking2End,
      ParticipantBooking2Begin < Booking1End
} = 0,
```

Nun enthielten die Bedingungen bei der vorliegenden Problemstellung aber auch eine Aggregatfunktion über die in dem Zimmer untergebrachten Personen. Ich habe deshalb probiert, ob sich diese Regel durch eine Verkettung der Aggregatsfunktionen programmieren lässt. Das entsprechende Code-Fragment:

```
#count { Time1 :
      #int(Time1),
      Booking1Begin <= Time1, Time1 < Booking1End,
      Booking2Begin <= Time1, Time1 < Booking2End,
      #count { PersonInRoom1 :
            booking(PersonInRoom1, BookingInRoom1Begin,
BookingInRoom1End, Room1),
            BookingInRoom1Begin <= Time1, Time1 <
BookingInRoom1End
      } = BedsInRoom
} = 0,
```

Tests hatten dabei reproduzierbar den Absturz von DLV zur Folge. Je nachdem, welche zusätzlichen Bedingungen in der Regel standen, wurde noch eine Meldung ausgegeben:

```
Variables to be aggregated over can not be used in a different
aggregate atom.
```

Diese Meldung bezieht sich mutmaßlich auf die Variable Time1, doch eine Verwendung in der inneren Funktion ist hier ja explizit gewünscht und müsste aus logischen Überlegungen heraus möglich sein.

7.2 Fehlen von Gleitkommazahlen und negativen Zahlen

Das Fehlen von Datentypen wie Gleitkommazahlen und negativen Zahlen oder genauer gesagt die Beschränkung auf natürliche Zahlen ist in erster Linie eine Design-Entscheidung, führt jedoch dazu, dass viele Probleme mit DLV praktisch nicht lösbar sind. Diesem Umstand ist wahrscheinlich auch das Fehlen von Funktionen für den Durchschnitt etc. geschuldet.

8 Fazit

Die Ergebnisse haben gezeigt, dass ASP prinzipiell auch auf Planungsprobleme wie das vorliegende angewendet werden kann. Allerdings steigt die Komplexität der Berechnung schon bei kleinen Test-Szenarien schnell auf Werte, die nicht mehr zu handhaben sind, so dass hier bereits künstliche Einschränkungen nötig sind.

Wenn die Anzahl der vorhandenen Betten, Zimmer etc. dann auf realistische Dimensionen angehoben werden zeigt sich, dass selbst für die Verarbeitung einer einzigen Bedarfsmeldung ein enormer Rechenaufwand nötig ist, da die Instanziierung erhebliche Zeit in Anspruch nimmt.

Für kleine Unterkünfte wäre es eventuell noch vorstellbar, immer nur eine Bedarfsmeldung pro Aufruf zu verarbeiten und die schon festgelegten Buchungen dem System als Fakten mitzugeben, was prinzipiell möglich wäre. Die Grenzen dafür sind aber ziemlich eng gesetzt und so muss abschließend festgestellt werden, dass für das vorliegende Planungs- bzw. Optimierungsproblem andere Ansätze gewählt werden müssen.

Das Problem ist, dass ASP als Vertreter logischer Programmierung dazu da ist, aus einer Faktenbasis und einem Regelwerk alle möglichen Lösungen im Sinne von Schlussfolgerungen herzuleiten. Dieser Ansatz ist im Fall dieses Planungsproblems nicht sinnvoll, da die kombinatorische Vielfalt des Ergebnisraumes viel zu schnell wächst.

9 Literaturverzeichnis

1. **Wikipedia.** *DLV.* [Online] 7. 10 2013. [Zitat vom: 10. 11 2013.] http://en.wikipedia.org/wiki/DLV.

2. **Robert Bihlmeyer, Wolfgang Faber, Giuseppe Ielpa, Vincenzino Lio, Gerald Pfeifer.** *DLV - User Manual.* [Online] 13. 11 2013. [Zitat vom: 13. 11 2013.] http://www.dlvsystem.com/html/DLV_User_Manual.html#SEATING.