Evaluation von aktuellen Planungssystemen anhand künstlicher Domänen und einer Daily Life Domäne

Stundentenprojekt im Fach Kognitive Systeme

Sommersemester 2005

Florian Brosch, Matthias Raps

florian.brosch@stud.uni-bamberg.de

matthias.raps@stud.uni-bamberg.de

Abstract: Diese Arbeit beschäftigt sich mit der Evaluierung der Einsetzbarkeit von Planungssystemen im Bereich von Daily Life Domänen. Dazu wird kurz auf den Hintergrund eingegangen, warum so genannte Cognitive Orthotics Systeme in der nahen Zukunft einen wachsenden Stellenwert einnehmen werden. Anschließend wird die Modellierung in PDDL und die Möglichkeiten, Daily Life Domänen mit aktuellen Planungssystemen zu modellieren genauer erläutert. Exemplarisch werden zwei Planer anhand strukturgleicher künstlicher Domänen empirisch verglichen und auf ihre Einsatzmöglichkeiten in diesem Bereich untersucht. Darüber hinaus wird eine komplette Daily Life Domäne vorgestellt und die Besonderheiten ihrer Modellierung in PDDL aufgezeigt.

1 Einleitung

Betrachtet man die demographische Entwicklung der deutschen Bevölkerung von 1950 und die Prognosen des Statistischen Bundesamtes Deutschland für die Bevölkerungsentwicklung bis 2050 (Abbildung 1.1), so fällt auf, dass die deutsche Bevölkerung im Durchschnitt immer älter wird

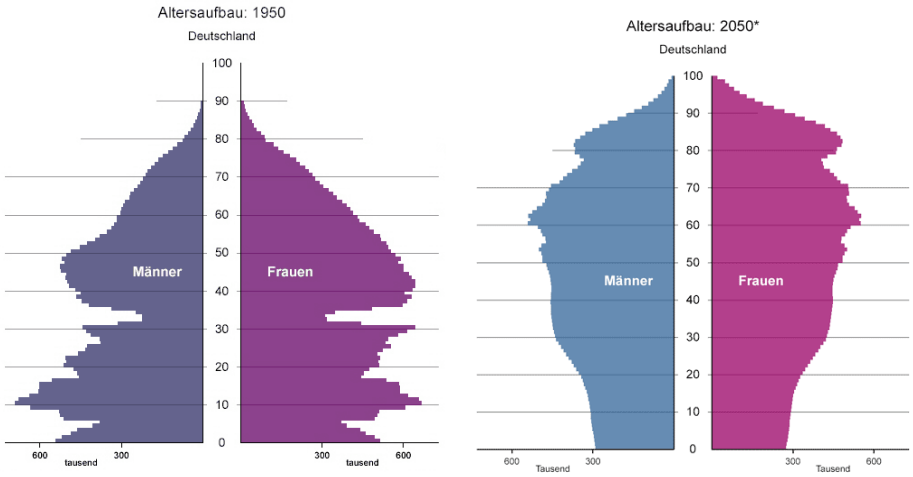

Abbildung 1.1: Bevölkerungspyramide 1950 (links) und 2050 (rechts)

In wissenschaftlichen Analysen wird häufig der so genannte *Altersquotient* als Maßzahl für das Phänomen der Überalterung verwendet. Er ist definiert als die Relation der 65-Jährigen und Älteren zur Bevölkerung im erwerbsfähigen Alter zwischen 15 und 64 Jahren. Der Vorsitzende des Sachverständigenrates zur Begutachtung der gesamtwirtschaftlichen Situation, Professor Bert Rürup, bezifferte den Altersquotient im Jahre 2000 auf 24,2 % und prognostizierte ihn auf 52,6 % im Jahre 2040.

Diese Entwicklung in Deutschland ist repräsentativ für die meisten Industrienationen der Welt. Neben volkswirtschaftlichen, politischen und sozialen Auswirkungen auf eine Gesellschaft, stellt das Phänomen insbesondere die Forschung und Wissenschaft vor neue Herausforderungen. So rückt der Forschungsansatz vermehrt in den Vordergrund, sowohl ältere Menschen, als auch deren Betreuer, mit neuer Technologie bei den Schwierigkeiten und Herausforderungen ihres Lebens zu unterstützen.

Je älter ein Mensch wird, desto vielfältiger können die Probleme sein, vor die er gestellt wird: Neben körperlichen und sozialen Auswirkungen haben Menschen bei zunehmendem Alter vor allem mit kognitiven Veränderungen zu kämpfen. Die Reduzierung der kognitiver Kapazität eines Menschen kann sowohl Folge der normalen körperlichen Alterung, als auch von Krankheiten sein.

Zu den häufigsten dieser Krankheiten zählt die Demenz. Man versteht darunter Störungen der geistigen Leistungen wie Gedächtnis, Denkvermögen, Sprache und Motorik. Die Alzheimer-Krankheit ist die bekannteste der Demenzerkrankungen. Rund 70% aller Demenzen werden durch sie hervorgerufen. Die Häufigkeit des Auftretens von Demenzerkrankungen nimmt mit dem Lebensalter zu: in der Altersgruppe 70 bis 74 sind 4% betroffen, in der Altersgruppe 85-89 knapp 21%. Zurzeit leiden in Deutschland etwa 1 Million Menschen an mittelschwer und schwer ausgeprägten Demenzerkrankungen. Bis zum Jahre 2005 werden etwa 1,2 Millionen Menschen betroffen sein. Bis zum Jahr 2050 wird sich die Zahl auf etwa 2 Millionen erhöhen.

Die Demenzerkrankungen sind hirnorganische Krankheiten, die sich durch das fortschreitende Absterben von Nervenzellen und Interneuronverbindungen auszeichnen. Das Krankheitsbild ist gekennzeichnet durch Gedächtnis- und Orientierungsstörungen, sowie Störungen des Denk- und Urteilsvermögens, die die Bewältigung eines normalen Alltagslebens erschweren. Patienten sind auf zunehmende Hilfe und Unterstützung angewiesen: Durch die Beeinträchtigung der kognitiven Kapazität, vorausschauend zu Handeln und zu Planen, einhergehend mit fortschreitender Vergesslichkeit, werden insbesondere die so genannten Aktivitäten des täglichen Lebens beeinträchtigt. Hierzu zählen zum Beispiel Aktivitäten wie Essen, Trinken, Einkaufen, Haushaltsführung oder die Nutzung öffentlicher Verkehrsmittel.

Je stärker ein Mensch beeinträchtigt ist, Aktivitäten des täglichen Lebens auszuführen, desto geringer wird seine Möglichkeit, auf sich allein gestellt zu Hause zu wohnen und sich in seiner Umwelt zurechtzufinden. Als Folge dessen werden häufig Maßnahmen von Betreuung bis hin zu institutioneller Einweisung nötig. Neben der finanziellen Belastung bedeutet dies meist eine erhebliche Einschränkung der Lebensqualität der betroffenen Menschen.

In den letzten Jahren wurden die verschiedensten Maßnahmen und Technologien entwickelt, um ältere Menschen bei der Ausübung von Aktivitäten des täglichen Lebens zu unterstützen. Diese Technologien werden allgemein als „Cognitive Orthotics“, also kognitive Hilfen oder kognitive Hilfsmaßnahmen bezeichnet.

Die Idee, Computer als Cognitive Orthotic zu verwenden, entstand 1963 von Douglas Englehart an der Universität von Stanfors, USA. Er schlug vor, kognitiv beeinträchtigte Menschern mit sprechenden Uhren und Kalendersystemen bei Aktivitäten des täglichen Lebens zu unterstützen. Mitte der 90er Jahre des letzten Jahrhunderts entwickelten Dowds und Robinson ein Cognitive Orthotic – System, dass ältere Menschen auf einem PDA unterstützt. Zur selben Zeit entwickelten Hersh und Treadgold ein ähnliches System, dass über Pager betrieben wurde.

Seit Anfang 2000 wird das Automider System als Teil der Initiative on Personal Robotic Assistance entwickelt. Ziel der Initiative ist es, einen autonomen mobilen Roboter zu entwickeln, der mit älteren Menschen zusammen „wohnt“ und sie aktiv bei der Planung und Umsetzung von Aktivitäten des täglichen Lebens unterstützt. Eines der Kernkomponenten des Roboters ist die Software Autominder. Sie soll auf adaptive, vorrausschauende Weise die täglichen Pläne des Patienten überwachen, gegebenenfalls neu planen und zu gegebenem Zeitpunkt daran erinnern, was zu tun ist. Wenn zum Beispiel der Patient zu einer bestimmten Zeit Medikamente einnehmen muss, er sich zu diesem Zeitpunkt allerdings nicht zu Hause befindet, sondern auf einer gesellschaftlichen Veranstaltung, erinnert der Autominder den Patienten daran, seine Medikamente mit- und zum gewünschten Zeitpunkt einzunehmen.

Autominder verwendet die Sprache der Disjunktiven Temporalprobleme (DTP), von Tsamardinos und Pollack [TP03]. Die Konstraints von Planungsproblemen, die in DTP beschrieben werden genügen der Form:

$$lb_1 \leq X_1 - Y_1 \leq ub_1 \vee \ldots \vee lb_n \leq X_n - Y_n \leq ub_n$$

Abbildung 1.2 Constraints in DTP

Hierbei beziehen sich Xi und Yi auf Start- und Endzeitpunkt eines Planschrittes und die zeitlichen Ober- und Untergrenzen lbi und ubi (lower bound und upper bound) repräsentieren reelle Zahlen. Abbildung 1.3 beschreibt, wie mit diesen Constraints Start- und Endzeitpunkt, sowie die Dauer eines Planschrittes und die Zeit zwischen den einzelnen Schritten beschrieben werden können. „TR“ bezeichnet den so genannten „Time Reference Point“, mit dem Uhrzeit-abhängigen Constraints beschrieben werden können, z.B. könnte TR „Mitternacht“ bedeuten.

"Toileting should begin between 11:00 and 11:15."
$660 \leq Toileting_S - TR \leq 675$
"Toileting takes between 1 and 3 minutes."
$1 \leq Toileting_E - Toileting_S \leq 3$
"Watching the TV news can begin at 18:00 or 23:00."
$1800 \leq WatchNews_S - TR \leq 1802 \vee$
$2300 \leq WatchNews_S - TR \leq 2302$
"The news takes exactly 30 minutes."
$30 \leq WatchNews_E - WatchNews_S \leq 30$
"Medicine should be taken within 1 hour of finishing breakfast."
$0 \leq TakeMeds_S - EatBreakfast_E \leq 60$
"Toileting and watching the news cannot overlap."
$0 \leq WatchNews_S - Toileting_E \leq \infty \vee$
$0 \leq Toileting_S - WatchNews_E \leq \infty$

Abbildung 1.3: Beispiel zur Verwendung von DTP Constraints

Das grundlegende Problem jeder Temporal-Logik ist deren Komplexität. Bei der in [TP03] beschriebenen Modellierungssprache handelt es sich darüber hinaus um einen individuellen Ansatz der speziell für Autominder im Rahmen der Forschungstätigkeit von Tsamardinos und Pollack entwickelt wurde. Im Bereich der Forschungen zum Automated Planning in der KI hat sich allerdings die Modellierungssprache PDDL durchgesetzt und kommt auch auf den zwei-jährlich stattfindenden Veranstaltungen der „International Planning Cometition" zu Einsatz. PDDL ist demnach die verbreitetere Modellierungssprache für Planungsprobleme.

Ziel dieser Arbeit ist es, automatisierte Planungssysteme auf ihre Einsatzmöglichkeit im Bereich der Cognitive Orthics zu untersuchen. Für die Evaluierung wurden künstliche Domänen entwickelt, die eine empirische Vergleichbarkeit ermöglichen. Darüber hinaus wurden die Möglichkeiten untersucht, mit diesen Planungssystemen, Aktivitäten des täglichen Lebens zu planen (Daily Life Domäne).

2. Planung mit PDDL

2.1 International Planning Competitions

Bei der International Planning Competition handelt es sich um eine seit 1998 alle zwei Jahre stattfindende Veranstaltung im Rahmen der Artificial Intelligence Planning and Scheduling Conference Series. Das Hauptaugenmerk dieser Veranstaltung liegt dabei auf dem wettbewerbsmäßigen empirischen Vergleich unterschiedlicher Planungssysteme. Weitere Ziele sind laut Homepage der Veranstaltung das Bereitstellen einer möglichst großen Anzahl von Daten für die Community, die sich mit diesem Thema beschäftigt, sowie das Einbringen neuer Forschungsansätze. Weiterhin sollen eine Reihe von Benchmarkproblemen und ein einheitlicher Formalismus zur Repräsentation zur Verfügung gestellt werden. Letzterer ist notwendig, um einen Vergleich und eine Evaluierung der einzelnen Systeme zu ermöglichen. Bei weiterführendem Interesse bezüglich der Veranstaltung und insbesondere deren Ergebnissen sei auf die Internetseiten der Competitions im Jahre 2002 und 2004 verwiesen [IPC02 und IPC04].

Als einheitliche Repräsentationssprache für die Competitions wird PDDL verwendet. Auf dessen Entwicklung und insbesondere auf die Erweiterungen PDDL 2.1 und PDDL 2.2, deren Konstrukte für unsere Modellierungen von Bedeutung sein werden, soll im folgenden Abschnitt eingegangen werden.

2.2 PDDL

Beginnend mit der Veranstaltung im Jahre 1998 wurde die einheitliche Spezifikationssprache PDDL (Planning Domain Definition Language), deren Mächtigkeit im Laufe der Zeit sukzessive erweitert wurde, von Drew McDermott zusammen mit dem damaligen Planungskomitee entwickelt. Die grundsätzliche Idee von PDDL ist dabei nach [Mc98], „die 'Physik' einer Domäne auszudrücken, d.h. welche Prädikate existieren, welche Aktionen sind möglich, was ist die Struktur zusammengesetzter Aktionen, und was sind die Auswirkungen von Aktionen". PDDL ist an den Formalismus von UCPOP angelehnt und es erfolgt eine STRIPS-artige Beschreibung der unterschiedlichen Aktionen. Für den genauen Aufbau von PDDL 1.2 sei auf das Manual für AIPS-98 [Mc99] verwiesen.

Im Rahmen von AIPS-2000 wurde eine Teilmenge der ursprünglichen PDDL Version eingesetzt [Ba00], während für die Competition im Jahre 2002 folgende Erweiterungen von Maria Fox and Derek Long vorgestellt wurden:

- Behandlung von numeric-valued fluents
- Darstellung von Zeit und Zeitdauer, sowie
- Spezifikation von Planmetriken als Teil von Planungsproblemen

Auf diese Erweiterungen wird im Teil 2.3 anhand der Beispieldomäne Depots gezielter eingegangen.

Für den zuletzt stattgefundenen Wettbewerb 2004 wurde PDDL von Hoffmann und Edelkamp noch um zwei weitere Konstrukte zu PDDL 2.2 erweitert. Zum einen handelt es sich dabei um so genannte timed initial literals. Mit diesen ist es möglich, bestimmte externe Ereignisse darzustellen, indem facts zu gewissen Zeitpunkten wahr oder falsch werden. Diese Zeitpunkte sind dabei unabhängig von den Aktionen, die ein Planer für die Ausführung auswählt. Timed initial literals werden im Initialzustand des Planungsproblems angegeben und werden oftmals zur Modellierung von Zeitfenstern genutzt. Ein einfaches Beispiel für timed initial literals ist in Abbildung 2-1 zu sehen.

```
(:init (at 8 (laden_geoeffnet)) (at 18 (not (laden_geoffnet))))
```

Abbildung 2-1: timed initial literals

Die zweite Erweiterung von PDDL2.2 sind derived predicates (abgeleitete Prädikate). Dabei handelt es sich um Prädikate, die nicht durch zur Verfügung stehende Aktionen beeinflusst werden. Stattdessen werden die Wahrheitswerte durch eine Menge von Regeln der Form 'if Formel(x) then Prädikat(x)' abgeleitet. Eine Instanz eines abgeleiteten Prädikats ist dabei genau dann wahr, wenn es durch den Einsatz der verfügbaren Regeln abgeleitet werden kann. Ein Beispiel aus der Blockwelt soll dies kurz verdeutlichen. Ein Block x liegt oberhalb eines anderen Blocks y `(above (x,y))`, wenn Block x direkt auf Block y liegt `(on (x,y))` oder wenn ein anderer Block z existiert, so dass Block x auf Block z `(on (x,z))` liegt und sich Block z oberhalb von Block y `(above (z,y))` befindet (vgl. Abbildung 2-2).

```
if on(x,y) OR (exists z: on(x,z) AND above(z,y)) then above(x,y)
```

Abbildung 2-2: Beispiel zu derived predicates nach [IPC04]

2.3 Domänen der IPC 2002

Um einen Eindruck von möglichen Planungsdomänen zu geben, werden nun im Folgenden die Domänen des Wettbewerbs 2002 kurz vorgestellt. Auf die Domäne Depots wird hierbei ausführlicher eingegangen, um einige Konstrukte von PDDL 2.1 genauer zu erläutern.

Im Rahmen des Wettbewerbs existieren sieben unterschiedliche Domänen, die in die drei Hauptbereiche Transportprobleme, Weltraumanwendungen und Sonstige aufgeteilt sind. In den beiden Bereichen Transport und Weltraum werden die einzelnen Domänen wiederum in unterschiedlichen Varianten präsentiert, um einen genaueren Vergleich der Planer zu ermöglichen. Es erfolgte eine Aufteilung in STRIPS-, numerische, einfache Zeit-, Zeit- und zum Teil komplexere Probleme. Zwischen den dadurch entstandenen Varianten der einzelnen Domänen bestehen durchaus substantielle Unterschiede, so dass man prinzipiell jede Domänenvariante als eigenständige Domäne betrachten kann.

Für den Bereich der Transportprobleme existieren die Domänen Depots, DriverLog und ZenoTravel. In der Depots Domäne werden Lastwagen zum Transport von Kisten verwendet. Diese Kisten müssen dann an ihren Zielorten mit Hilfe von Flaschenzügen auf Paletten gestapelt werden. Hier kann man eine große Ähnlichkeit zur Blockwelt erkennen, da das Stapeln der Kisten vergleichbar ist mit dem Aufeinandertürmen von Blöcken. Bei den DriverLog Problemen muss zur Auslieferung von Paketen mit Lieferwägen zu verschiedenen Empfängern gefahren werden. Dabei kann es vorkommen, dass die Fahrer zwischen den Transportern wechseln müssen und unterschiedliche Möglichkeiten existieren, um zu den Empfängern zu gelangen. Die letzte Transportdomäne ZenoTravel zielt in erster Linie auf numerische Probleme ab. Zum Transport von Menschen werden dabei Flugzeuge verwendet, mit denen eine langsame und eine schnelle Beförderung möglich sind. Davon abhängig ist dann der Spritverbrauch, der bei qualitativ guten Plänen möglichst klein sein soll.

Für den Bereich der Weltraumanwendungen existieren die Domänen Satellite und Rovers. Bei ersterer müssen Beobachtungsaufgaben zwischen mehreren Satelliten, die jeweils eine andere Ausstattung besitzen, verteilt werden. Eine Reihe von Erkundungsfahrzeugen müssen bei Rovers auf der Oberfläche eines Planeten navigiert werden. Diese nehmen Proben und kommunizieren ihre Daten zurück zum Landungsfahrzeug.

FreeCell, Settlers und UMTranslog-2 fallen in den letzten Bereich von sonstigen Problemen. Bei FreeCell handelt es sich um das von vielen Systemen her bekannte Solitärspiel. Das Ziel ist, ausgehend von einer gegebenen Anordnung von Spielkarten, eine bestimmte Sortierung von diesen auf Stapeln unter der Einhaltung fester Regeln zu erreichen. Bei Settlers werden vorhandene Ressourcen dazu genutzt, um Fahrzeuge unterschiedlicher Art zu konstruieren. Da diese Fahrzeuge zu Beginn noch nicht vorhanden sind, handelt es sich um ein Problem, bei dem neue Objekte generiert werden. Da allerdings PDDL dieses Konzept momentan noch nicht unterstützt, müssen potenziell mögliche Fahrzeuge in der Problembeschreibung mit aufgeführt werden. Die Qualität eines Planes wird durch eine lineare Kombination von Arbeitsaufwand, Ressourcenverbrauch und Schadstofferzeugung festgelegt. Bei UMTranslog-2 handelt es sich um eine numerische Domäne, die lediglich für hand-coding planners im Wettbewerb verwendet wurde und somit hier nicht weiter von Interesse ist.

Wie bereits oben erwähnt wurde, existieren von den einzelnen Domänen weitere Varianten. Diese werden anhand der Depots Domäne genauer betrachtet, da dort Konstrukte genutzt werden, die auch in unseren selbst modellierten Domänen Eingang gefunden haben. Wir beginnen mit der einfachen STRIPS–Domäne. (vgl Abbildung 2-2). Das Augenmerk wird dabei zuerst auf das typing gerichtet, welches auch als requirement benötigt wird. Mit Hilfe von typing ist es möglich, eine Typhierarchie aufzubauen, die dann innerhalb späterer Definitionen (Aktionen etc.) genutzt werden kann. So existieren hier beispielsweise die Typen *place* und *locatable* vom Typ *object*. Bei object handelt es sich um einen sog. built-in type, der keine genauere Definition benötigt.

```
(define (domain Depot)
(:requirements :typing)
(:types        place locatable - object
       depot distributor - place
       truck hoist surface - locatable
       pallet crate - surface)
…
(:action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (lifting ?x ?y))
:effect (and (not (lifting ?x ?y)) (in ?y ?z) (available ?x)))
```

Abbildung 2-3: Auszug aus der STRIPS Domäne

Bei der beispielhaften Aktion Load handelt es sich um eine “klassische” Aktionsdefinition mit Parametern (parameters), Vorbedingungen (preconditions) und Auswirkungen (effects). Um die Aktion Load ausführen zu können, müssen sich der LKW und der Flaschenzug am gleichen Ort befinden, sowie die Kiste vom Flaschenzug angehoben worden sein. Die Auswirkungen der Load-Aktion sind dann, dass die Kiste nicht mehr angehoben ist, die Kiste sich im LKW befindet und der Flaschenzug wieder einsetzbar ist.

Weitaus interessanter ist die Betrachtung von Depots numeric. Hier werden die requirements um fluents erweitert, wodurch es möglich wird, functions zu definieren, Zuweisungsoperatoren in den effects einzusetzen und arithmetische Vorbedingungen zu nutzen.

```
(define (domain Depot)
(:requirements :typing :fluents)
…
(:functions
        (load_limit ?t - truck)
        (current_load ?t - truck)
        (weight ?c - crate)
        (fuel-cost)
)
…
(:action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (lifting ?x ?y)
                (<= (+ (current_load ?z) (weight ?y)) (load_limit ?z)))
:effect (and (not (lifting ?x ?y)) (in ?y ?z) (available ?x)
                (increase (current_load ?z) (weight ?x))))
```

Abbildung 2-4: Auszug aus der numeric Domäne

Als function wird hier beispielsweise das aktuelle Zuladungsgewicht eines LKWs definiert (`current_load ?t - truck`). Dieses Gewicht kann dann im Planungsproblem festgelegt und durch Aktionen verändert werden. Eine Aktion, die eine solche Veränderung vornimmt, ist die Ladeaktion. Hier wird zuerst in der precondition abgeprüft, ob das Höchstladegewicht des LKWs kleiner oder gleich der Summe aus aktuellem Zuladungsgewicht und dem Gewicht der einzuladenden Kiste ist. Ist auch diese Voraussetzung erfüllt, so kann die Aktion ausgeführt werden, und als zusätzlicher Effekt der Aktion wird das aktuelle Zuladungsgewicht des LKWs um das Gewicht der Kiste erhöht.

In der Depots Simple Time Domäne wird vorerst auf fluents verzichtet und stattdessen werden die sog. durative-actions eingeführt, welche ebenfalls als requirement angegeben werden müssen. Die Definition einer durative-action enthält zusätzlich die Angabe einer Zeitdauer, die für die Ausführung der Aktion benötigt wird.

```
(:durative-action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:duration (= ?duration 3)
:condition (and (over all (at ?x ?p)) (over all (at ?z ?p)) (over all
        (lifting ?x ?y)))
:effect (and (at end (not (lifting ?x ?y))) (at end (in ?y ?z)) (at end
        (available ?x))))
```

Abbildung 2-5: Durative-action Load

In Abbildung 2-5 ist wiederum die Aktion zum Aufladen einer Kiste zu sehen. Dieser Vorgang hat eine Ausführungsdauer von 3 Zeiteinheiten (`:duration (= ?duration 3)`). Zusätzlich erkennt man, dass in der Aktionsdefinition nicht mehr von precondition gesprochen wird, sondern von condition. Hierbei wird der Tatsache Rechnung getragen, dass Bedingungen für eine Aktion auf drei unterschiedliche Zeitpunkte bzw. Zeitspannen bezogen sein können. Es gibt Bedingungen, die zu Beginn einer Aktion (at start), während der gesamten Dauer der Aktion (over all) oder auch erst am Ende der Aktion (at end) erfüllt sein müssen. Beispielsweise muss die Bedingung, dass sich der LKW am Ort des Ladevorgangs aufhält, über den gesamten Zeitraum der Aktion erfüllt sein. Möchte man dagegen eine Aktion "Fahren von A nach B" modellieren, so erscheint es logisch, dass die Bedingung "Fahrzeug hält sich am Ort A auf" nur zu Beginn der Aktion (at start) gelten muss und kann. Die Auswirkungen einer Aktion sind ebenfalls auf den Zeitpunkt ihres Auftretens bezogen, d.h. ein Effekt kann bereits zu Beginn einer Aktion auftreten (at start) oder bei Beendigung der Aktion (at end).

Anhand der Ladeaktion aus der Domäne Depots Time soll noch kurz auf eine Modellierungsalternative von durative-actions eingegangen werden. Es ist nicht zwangsläufig notwendig, dass die Zeitspanne für die Durchführung einer Aktion konstant ist, sondern es ist auch möglich, dass die Dauer der Aktion von anderen Faktoren abhängig gemacht wird.

```
(:durative-action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:duration (= ?duration (/ (weight ?y) (power ?x)))
…
```

Abbildung 2-6: Durative-action Load mit variabler duration

In Abbildung 2-6 erkennt man, dass die Ladezeit jetzt vom Gewicht der Kiste und von der Leistung des Flaschenzugs abhängt und durch Division berechnet wird.

Zum Abschluss dieses Abschnitts soll anhand eines möglichen Planungsproblems für die Domäne Depots Time u.a. der Einsatz von Metriken gezeigt werden.

```
(define (problem depotprob1818) (:domain Depot)
(:objects
        depot0 - Depot
        distributor0 distributor1 - Distributor
        truck0 truck1 - Truck
        …
(:init
        (at pallet0 depot0)
        (clear crate1)
        …
(at truck0 distributor1)
        (= (speed truck0) 4)
        …
(= (weight crate1) 86)
        (= (distance depot0 depot0) 0)
        (= (distance depot0 distributor0) 5)
        …
)

(:goal (and
                (on crate0 pallet2)
                (on crate1 pallet1)
        )
)

(:metric minimize (total-time)))
```

Abbildung 2-6: Ausschnitt aus einem Planungsproblem der Depots Time Domäne

Wie man erkennen kann, ist die hier eingesetzte Metrik (`minimize (total-time))`. Dies bedeutet, dass das Ergebnis der Planung derart sein soll, dass die Gesamtdauer eines Plans möglichst klein ist. Es steht folglich nicht mehr im Vordergrund, mit möglichst wenigen Aktionen das Ziel zu erreichen, sondern mit Bezug auf die durations möglichst schnell das Ziel zu ereichen. Total-time ist dabei nicht zwangsläufig die einzusetzende Metrik. Es ist durchaus auch vorstellbar, dass beispielsweise die zurückgelegten Wege minimiert werden sollen. In Abbildung 2-6 kann man weiterhin erkennen, dass innerhalb des Planungsproblems die Initialisierung der functions erfolgt. So wird hier z.B. festgelegt, dass die Geschwindigkeit des Trucks0 4 ist.

2.4 HSP und LPG-td

Im Hinblick auf die von uns angestrebten Vergleiche und Modellierungen stellte sich die Frage, welche Planer wir hierfür auswählen. Bei den Auswahlkriterien standen vor allem die Punkte einfache Installierbarkeit und Bedienung der Planer im Vordergrund. Deshalb fiel die Entscheidung auf das System HSP* von Patrick Haslum und Hector Geffner [HSP04], da es sich bei diesem auch um ein uns bereits bekanntes Planungssystem handelt. Der zweite ausgewählte Planer ist LPG-td von Alfonso Gerevini, Alessandro Saetti, Ivan Serina, und Paolo Tonelli [LPG04]. Für diesen Planer sprachen u.a. die sehr guten Ergebnisse bei der Competitions 2002 und 2004 und der Umstand, dass dieses System auch unter Windows lauffähig ist. Im Folgenden sollen beide Planer mit ihren möglichen Optionen kurz charakterisiert werden.

2.4.1 HSP* (Heuristic Search Planning)

Bei HSP* handelt es sich um eine Familie von domänenunabhängigen Planern, die alle auf heuristischer Suche basieren. Die genutzte Heuristik wird dabei aus der jeweiligen Problemrepräsentation automatisiert gewonnen, und die Ergebnisse der Planung sind optimal bezüglich eines angegebenen Maßes. Hie HSP* Planer basieren dabei alle auf ihren nicht optimalen Vorgängern HSP und HSPr. Für unsere weiteren Betrachtungen werden wir das Augenmerk auf den HSP_0 Planer richten, der auch unter dem Namen TP4 bekannt ist und an IPC 2002 und 2004 teilnahm.

HSP_0 führt mit Hilfe der zulässigen (admissible) Heuristik H^2 eine Rückwärtssuche (regression) durch. Bei dieser Heuristik werden die Kosten für die Erreichung einer Menge von Atomen dadurch angenähert, dass man die Kosten für die Erreichung aller Untermengen dieser Atome betrachtet. Die Kosten für die „teuerste“ Untermenge werden dann als Approximation für die Kosten der eigentlichen Menge von Atomen eingesetzt. Die notwendigen Berechnungen – die auf einer Methode der dynamischen Programmierung basieren - erfolgen dabei bereits vor der eigentlichen Suche und die Ergebnisse der Berechnung werden in einer Tabelle zwischengespeichert.

HSP_0 ist, wie auch alle anderen HSP* Planer, in der Lage, unterschiedliche Typen der Planung durchzuführen. Die grundsätzlichste Art ist dabei das parallele Planen, wobei die Anzahl der Schritte minimiert wird. Weiterhin ist es möglich, temporale Pläne zu erstellen, bei denen das Ziel ist, die Gesamtdauer zu minimieren. Bei dieser Planungsart können auch wieder verwendbare oder verbrauchbare Ressourcen eingesetzt werden. Die letzte mögliche Planungsart ist das sequentielle Planen. Dabei wird die Anzahl der Aktionen minimiert, und es ist möglich, mit Kosten oder Ressourcen oder auch mit beidem zu arbeiten. Beim Einsatz der Option –cost werden die Aktionskosten im Bezug auf die spezifizierte Metrik minimiert, während beim Einsatz von -res Beschränkungen bezüglich verbrauchbarer Ressourcen beachtet werden.

Als Input erhält HSP_0 Domänen- und Problemspezifikationen in einer PDDL 2.1 ähnlichen Sprache. Der Unterschied ergibt sich u.a. aus einer anderen Semantik bezüglich der durative actions. Man geht dabei von einer Art “Intervallsemantik” aus, d.h. für eine Aktion A, die über ein bestimmtes Intervall *[S,T]* (mit $T = S+dur(A)$), ausgeführt wird, wird Folgendes angenommen:

- alle Vorbedingungen müssen zum Zeitpunkt S gelten
- Vorbedingungen, die nicht durch die Aktion gelöscht werden, müssen innerhalb des gesamten Intervall gültig sein
- alle Auswirkungen einer Aktion finden zu einem nicht bestimmbaren Zeitpunkt innerhalb des Intervalls statt.

Dies hat zur Folge, dass sich zwei Aktionen nicht überlappen können, wenn diese zueinander mutex (mutually exclusive) sind, d.h. eine Aktion löscht die Vorbedingung bzw. den Effekt der anderen. Dadurch ist natürlich auch eine Aktion inkonsistent, die dieselbe proposition hinzufügt und entfernt. Eine Ausnahme hiervon stellen die später erläuterten „Ressourcenpatterns“ dar. Durative actions müssen außerdem eine duration, die größer als 0 ist, besitzen. Aktionen mit einer Zeitdauer von 0 werden zwar vom Planer akzeptiert, es kann allerdings vorkommen, dass diese dann nicht korrekt bzw. sinnvoll eingesetzt werden.

Numerische Variablen, d.h. functions, die in der Problemdefinition nicht konstant sind, und Bedingungen und Effekte die von diesen abhängen, werden als Ressourcen oder Kosten behandelt. Wird innerhalb einer Problemspezifikation eine andere Metrik als total-time angegeben, so werden die Gesamtkosten, die durch die Aktionsausführungen entstehen, minimiert. Bei Ressourcen unterscheidet man zwischen wieder verwendbaren und verbrauchbaren Ressourcen, wobei erstere lediglich beschränkt sind, d.h. von Aktionen weder verbraucht noch produziert werden, während verbrauchbare Ressourcen von Aktionen aufgezehrt werden. Die Ressourcen können dabei über eine Art von Patterns innerhalb der Aktionen erkannt werden. Falls eine Aktion die in Abbildung 2-7 gezeigten Effekte besitzt, so wird der Wert <amount> von der durch <fun> spezifizierten Ressource für die Dauer der Aktion „entliehen“.

```
(at start (decrease (<fun> ...)
<amount>))
(at end (increase (<fun> ...)
<amount>))
```

Abbildung 2-7: „Ressourcenpattern“ 1 nach [HSP04]

Außerdem besteht die Möglichkeit, eine Ressource zu sperren (vgl. Abbildung 2-8). Dabei wird durch eine Aktion, die die gezeigten Bedingungen erfüllt, die durch <pred> spezifizierte Ressource für andere Aktionen, die ebenfalls diese Ressource benötigen, gesperrt.

```
precondition:
(at start (<pred> ...))
and the effects:
(at start (not (<pred>
...)))
(at end (<pred> ...))
```

Abbildung 2-8: “Ressourcenpattern” 2 nach [HSP04]

2.4.2 LPG-td

LPG-td ist eine Erweiterung von LPG 1.2, um den neuen Anforderungen von PDDL 2.2 gerecht zu werden. Der Planer ist also in der Lage, timed initial literals und derived predicates zu handhaben. Da es sich bei LPG-td um einen suboptimalen Planer handelt, sind die Ergebnisse der Planung nicht zwangläufig die „bestmöglichen“, d.h. die optimalen Pläne. Laut den Ergebnissen der Competition 2004 sind Lösungen allerdings i.d.R. sehr ähnlich zu den optimalen Plänen. LPG-td basiert auf einer lokalen stochastischen Suche im Raum von speziellen Plangraphen, den sog. Aktionsgraphen, die aus der Problemrepräsentation abgeleitet werden. Dabei kommen verschiedene Heuristiken, die auf einer parametrisierten Funktion beruhen, zum Einsatz. Diese Parameter gewichten unterschiedliche Arten von Inkonsistenzen innerhalb eines partiellen Plans und werden während der Suche dynamisch mit Hilfe von Lagrange-Multiplikatoren ausgewertet. Für die detaillierte Funktionsweise von LPG-td sei auf [GS02] und [GSS04] verwiesen.

LPG-td ist in der Lage, in drei unterschiedlichen Modi zu arbeiten:

- speed (-speed): In diesem Modus wird ein Plan von beliebiger Qualität so schnell wie möglich gefunden und die Suche daraufhin eingestellt.

- incremental (-n [Anzahl der gewünschten Lösungen]): Hier ist das Ergebnis der Planung eine Abfolge von möglichen Plänen. Die erste Lösung wird mit Hilfe des speed Modus generiert und die darauf folgenden sind dann jeweils Verbesserungen bezüglich der spezifizierten Metrik im Vergleich zu dem vorhergehenden Plan. Dabei ist es nicht zwangsläufig notwendig, dass n Lösungen gefunden werden, da eine bereits vorhandene Lösung optimal sein kann und keine Verbesserung mehr möglich ist. In diesem Fall stoppt der Planer, wenn die angegeben Höchstplanzeit erreicht wurde.

- quality (-quality): Hier arbeitet LPG-td langsamer als im speed Modus, findet dabei aber eine qualitativ höherwertige Lösung. Nachdem eine erste Lösung gefunden wurde, wird automatisch entschieden, wie viel CPU-Zeit dafür verwendet werden soll, um die ursprüngliche Lösung zu verbessern.

3. Empirische Evaluation anhand künstlicher Domänen

3.1 Vorstellung der Domänen

Wie im letzten Kapitel erwähnt, handelt es sich bei LPG-TD und HSP* um sehr unterschiedliche Planer. Um eine empirische Evaluation der beiden zu ermöglichen, wurde davon Abstand genommen, die Evaluation mit einer Daily Life Domäne (siehe Kapitel 1 und 4) durchzuführen. Die Möglichkeiten eines empirischen Vergleiches auf Basis einer Problemdomäne mit Aktionen des täglichen Lebens (Daily Life Activities), sind zu stark eingeschränkt, als dass sich grundlegende Aussagen treffen lassen würden. Um dennoch eine Evaluierung der Einsatzmöglichkeiten der beiden Planer in Hinsicht auf einen Einsatz in diesem Bereich zu ermöglichen, wurden künstliche Domänen für die Evaluierung erstellt, die unterschiedliche Klassen und Komplexitäten von Daily Life Domänen jeweils einzeln repräsentieren.

Diese künstlichen Domänen unterscheiden sich stark von klassischen Domänen, wie sie zum Beispiel bei der IPSC verwendet werden. Die Komplexität bei klassischen Domänen wird vor allem durch die Komplexität des Initialzustandes der Domäne und der Anzahl der Prädikate bestimmt. Betrachtet man zum Beispiel ein klassisches Blocksworld – Problem, so steigt dessen Komplexität, desto mehr Blöcke in der Domäne vorgesehen sind und je aufwendiger der Initialzustand in den gewünschten Goal-Zustand zu überführen ist. Diese Art der Steigerung der Komplexität ist bei Daily Life Domänen nur sehr begrenzt umsetzbar. Zwar könnte man zum Beispiel die Anzahl an Pillen, die ein Patient zu nehmen hat, erhöhen, allerdings stellt sich die Frage, inwieweit der Bezug zu konkreten Daily Life Domänen gewahrt bleibt, wenn für signifikante Unterschiede in den Verarbeitungszeiten der Planer, ein Patient über 1000 Tabletten zu nehmen hätte. Aus dem Bereich der Aktionen des täglichen Lebens gibt es nur sehr wenige, die eine sinnvolle Erhöhung der Komplexität der Daily Life Domäne durch eine Erhöhung der Anzahl der Prädikate zur Folge hätten. Vielmehr zeichnen sich Daily Life Domänen dadurch aus, dass die Anzahl der Aktionen, also Operatoren, bei wachsender Datailiertheit der Modellierung zunimmt. Bei dem Versuch, Aktionen eines Menschen während seines Tagesablaufs zu modellieren, steigt die Anzahl der möglichen Aktionen viel stärker, als die der möglichen Prädikate.

Um diesem Phänomen gerecht zu werden, wurden die künstlichen Domänen derart gestaltet, dass sie steigende Komplexität in der Form einer wachsenden Anzahl von Aktionen abbilden. Es gibt insgesamt 5 Typen von Domänen, die bei der Evaluierung berücksichtigt wurden:

Erhöhung der Komplexität durch Erhöhung der

- sequentiell ausgeführten Aktionen
- parallel ausgeführten Aktionen
- sequentiell und parallel ausgeführten Aktionen

- sequentiell und parallel ausgeführten Aktionen mit der Abwandlung, dass die Vorbedingung jeder Aktion nach deren Ausführung gelöscht wird

- Dauer der durative actions

Letzter Typ von Domänen soll vor allem dazu dienen, zu untersuchen, ob die Dauer der Aktion eine Auswirkung auf das Ergebnis des Planers hat.

3.1.1 Die Sequentielle Domäne

Wie bereits kurz erwähnt, stellt dieser Typ von künstlicher Domäne eine Erhöhung der Komplexität des Planungsproblemes durch die Erhöhung der Anzahl von nacheinander ausgeführten Aktionen dar. Die Grundidee dieser Domäne ist, dass Aktion A1 vor Aktion A2 ausgeführt wird, diese vor Aktion A3, bis zum Schluss Aktion An ausgeführt wird.

```
(define (domain Kunst2)

(:predicates 

(nothing)
(done1)
(done2)
…
)

(:action A1
:parameters ()
:precondition (and (nothing))
:effect (and (done1))
)

(:action A2
:parameters ()
:precondition (and (done1))
:effect (and (done2))
)
…
)
```

Abbildung 3.1: Domänendefinition der sequentiellen Domäne

Wie in der Domänendefinition in Abbildung 3.1 dargestellt, werden die Prädikate „nothing “, „done1“, „done2“ bis hin zu „done n“ definiert. Das Prädikat „nothing“ dient der Beschreibung des Startzustandes, bei dem keine Aktion bisher ausgeführt wurde. Die Aktion, die als Erstes ausgeführt werden kann, ist Aktion A1, die als Vorbedingung „nothing“ hat und als Effekt „done1“ liefert. Erst nachdem Aktion A1 ausgeführt wurde, kann die nächste Aktion ausgeführt werden.

```
(define (problem Seq)(:domain Kunst2)

(:init (nothing))

(:goal (and
(done200)

)))
```

Abbildung 3.2: Init und Goal der sequentiellen Domäne

Der Initialzustand der sequentiellen Domäne ist in Abbildung XXXX abgebildet. Der Init – Zustand ist ausschließlich „nothing“. Ziel, also goal, ist es, die letzte Aktion in der Sequenz der Aktionen ausgeführt zu haben. In diesem Fall ist das Aktion A200, die als Ergebnis „done200“ liefert.

Getestet wurden Sequentielle Domänen mit 100, 200, 300 bis 1000 sequentiellen Aktionen.

3.1.2 Die parallele Domäne

Der Aufbau dieser Domäne ist analog zu der sequentiellen Domäne, allerdings die Aktionen nicht sequentiell abgearbeitet, sondern parallel zu einen Zeitpunkt.

```
(define (domain Kunst1)
(:predicates   (nothing)
               (done1)
               (done2)
               (done3)
               …

…

(:action A1
:parameters ()
:precondition (and (nothing))
:effect (and (done1))
)

(:action A2
:parameters ()
:precondition (and (nothing))
:effect (and (done2))
)

(:action A3
:parameters ()
:precondition (and (nothing))
:effect (and (done3))
)

…
```

Abbildung 3.3 Domänendefinition der parallelen Domäne

In Abbildung 3.3 erkennt man den grundlegenden Unterschied zu der sequentiellen Domäne: Die einzelnen Aktionen besitzen alle die gleiche Vorbedingung, dass „nothing“ gelten muss. Dadurch wird erreicht, dass alle definierten Aktionen zum erstmöglichen Zeitpunkt ausgeführt werden können. Der Initialzustand aus Abbildung XXXXX beschreibt, dass im Init - Zustand „nothing“ gilt und alle „done n“ Prädikate (Ergebnis der Aktion „action n“) für den Goal – Zustand gelten müssen.

Getestet wurden 100, 200, 300 bis 1000 parallele Aktionen.

```
(define (problem Par)(:domain Kunst1)

 (:init (nothing))
(:goal  (and
                (done1)
                (done2)
                (done3)
                (done4)
                (done5)
)
```

Abbildung 3.4 Initialzustand der parallelen Domäne

3.1.3 Die sequenitelle und parallele Domäne

Der nächste Schritt nach dem Test von sequentiell oder parallel ausgeführten Aktionen, ist der Test wie sich sequentiell und parallel gleichzeitig ausgeführte Aktionen auf die Planung auswirken. Bisher wurde die Anzahl der Aktionen in Hunderter - Schritten erhöht. Da sich in der sequentiellen und parallelen Domäne die Gesamtanzahl der auszuführenden Aktionen aus dem Produkt der sequetiellen und parallelen Aktionen errechnet, ist es nicht mehr möglich, die Anzahl der Aktionen in Hunderter – Schritten zu erhöhen. Stattdessen wurde die Anzahl der Aktionen von 10*10=100 in einem Zweierinkrement über 12*12=144 bis 32*32=1024 Aktionen erhöht. Dadurch ist zumindest eine ungefähre Vergleichbarkeit zu den beiden vorherigen Domänen gewährleistet.

```
(define (domain Kunst4)
 (:predicates  (nothing)
               (done1-1)
               (done1-2)
               …
               (done1-10)
               (done2-1)
               (done2-2)
               …
               (done2-10)

               …
```

Abbildung 3.5 Prädikate der sequentiellen und parallelen Domäne

```
(:action A1-1
:parameters ()
:precondition (and (nothing))
:effect (and (done1-1))
)

(:action A1-2
:parameters ()
:precondition (and (done1-1))
:effect (and (done1-2) (not (done1-1)))
)

(:action A1-3
:parameters ()
:precondition (and (done1-2))
:effect (and (done1-3) (not (done1-2)))
)

 ...

(:action A2-1
:parameters ()
:precondition (and (nothing))
:effect (and (done2-1))
)

(:action A2-2
:parameters ()
:precondition (and (done2-1))
:effect (and (done2-2) (not (done2-1)))
)

…
```

Abbildung 3.6 Aktionen der parallelen und sequentiellen Domäne

Die Domänendefinition in Abbildung 3.5 und 3.6 unterscheidet sich dahingehend von den vorigen Definitionen, dass die Namen der Prädikate eine zweidimensionale Komponente besitzen. Allgemein sind die Prädikate nach dem Muster „done X – Y“ aufgebaut. X bezeichnet dabei den Zeitpunkt X zu dem sie ausgeführt wurde und Y die Anzahl der zu Zeitpunkt X ausgeführten parallelen Aktionen. Die Aktionen „Action X - Y“ haben als Ergebnis eine zweidimensionale Struktur von Prädikaten, wie sie in Abbildung 3.7 beschrieben sind:

Zeitpunkt 1	*Zeitpunkt 2*	*Zeitpunkt 3*	*...*	*Zeitpunkt X*
Done 1 - 1	Done 2 - 1	Done 3 - 1	...	Done X - 1
Done 1 - 2	Done 2 - 2	Done 3 - 2	...	Done X - 2
Done 1 - 3	Done 2 - 3	Done 3 - 3	...	Done X - 3
...	...	...	...	...
Done 1 - Y	Done 2 - Y	Done 3 - Y		Done X - Y

Tabelle 3.7 Struktur der Prädikate in der parallelen und sequentiellen Domäne

Der Initialzustand, der in dieser Domäne verwendet wurde, sieht wie folgt aus. Der Goal – Zustand ist erreicht, wenn zu jedem Zeitpunkt X alle Aktionen Y für diesen Zeitpunkt ausgeführt wurden.

```
(define (problem SPD)(:domain Kunst4)
(:init (nothing))
(:goal (and
               (done1-10)
               (done2-10)
               (done3-10)
               (done4-10)
               (done5-10)
               (done6-10)
               (done7-10)
               (done8-10)
               (done9-10)
               (done10-10)
)))
```

Abbildung 3.8 Initialzustand der sequentiellen und parallelen Domäne

3.1.4 Sequentielle und parallele Domäne mit Delete

Als Abwandlung der Domäne aus 3.1.3 wurde Untersucht, ob es einen signifikanten Unterschied macht, ob bei deren Modellierung der Domäne darauf geachtet wurde, die Vorbedingungen der jeweiligen Aktion gelöscht werden. Dadurch wird der Baum der zu durchsuchenden Möglichkeiten den der Planer aufspannt reduziert. Hintergrund dieser Domain ist die Frage inwieweit die Vollständigkeit der Modellierung das Planungsergebnis beeinflusst. An dieser Stelle sei angemerkt, dass die Domäne ohne das Löschen der Vorbedingung von den Planern vollständig bearbeitet werden konnte, so dass das Löschen der Vorbedingung keinen Einfluss auf die Korrektheit der Modellierung hat.

```
(define (domain Kunst4)

(:action A1-1
:parameters ()
:precondition (and (nothing))
:effect (and (done1-1))
)

(:action A1-2
:parameters ()
:precondition (and (done1-1))
:effect (and (done1-2) (not (done1-1)))
)

(:action A1-3
:parameters ()
:precondition (and (done1-2))
:effect (and (done1-3) (not (done1-2)))
)
```

Abbildung 3.9 Domänendefinition mit Delete

Die Definition der Prädikate und des Initialzustandes erfolgt analog zu der ursprünglichen Domäne aus 3.1.3.

3.1.5 Durative Actions Domäne

Der elementare Unterschied der Durative Actions Domäne zu den bisherigen Domänen liegt darin, dass die Aktionen eine zeitliche Dauer besitzen. Formal wird dies dadurch erreicht, dass die „actions“ in der Domänendefinition durch „durative actions“ mit einer spezifizierten Ausführungsdauer ersetzt wurden (siehe 2.3). Diese Domäne ist ein Derivat der Domäne aus 3.1.3, in der die beschrieben Änderung der Aktionen vorgenommen wurde. Untersuchungsgegenstand hierbei war, ob die Dauer der Aktion einen Einfluss auf die Planung hat.

```
define (domain Kunst5)
(:requirements :durative-actions)
(:predicates   (nothing)
               (done1-1)
               (done1-2)
               (done1-3)
               (done1-4)
               (done1-5)
               (done1-6)
               (done1-7)
               (done1-8)
               (done1-9)
               (done1-10)
               (done2-1)
               (done2-2)
               (done2-3)
               (done2-4)
               (done2-5)
               (done2-6)
               (done2-7)
               (done2-8)
               (done2-9)
               (done2-10)

(:durative-action A1-1
:parameters ()
:duration (= ?duration 500)
:condition (and (at start (nothing)))
:effect (and (at end (done1-1)))
)

(:durative-action A1-2
:parameters ()
:duration (= ?duration 500)
:condition (and (at start (done1-1)))
:effect (and (at end (done1-2)) (at end (not (done1-1))))
)
```

Abbildung 3.10 Domänendefinition Durative Actions

3.2 Empirischer Vergleich

Dieser Abschnitt beschäftigt sich mit den Ergebnissen der durchgeführten Tests der im vorherigen Unterkapitel vorgestellten Domains. Bei dem Test handelt es sich nicht um ein Experiment im klassischen Sinne, sondern vielmehr um ein Quasi-Experiment. Es wurde aufgrund des Aufwandes bezüglich der manuellen Interaktion mit dem Testsystem und des zeitlichen Aufwandes darauf verzichtet, eine klassische empirische Untersuchung durchzuführen. So benötigte LPG-TD zum Beispiel knapp 1,5 Minuten, um die sequentielle Domäne aus 3.1.1 mit 1000 sequentiellen Aktionen zu planen. Hätte man eine hinreichende Anzahl an Stichproben in allen Domänen und Komplexitätsvariationen gesammelt, so hätte dies den gesetzten Umfang dieser Arbeit bei Weitem überschritten. Deshalb beschränkt sich der empirische Vergleich auf eine rein deskriptive, quasi-experimentelle Untersuchung. Aus jeder Domäne wurde bei einer Komplexitätsvariante jeweils ein Planungsdurchlauf pro Planer getätigt. Getestet wurde auf einem Intel Celeron 2,6 Ghz System mit 512 Mb RAM mit einer Debian 3.1 Distribution.

In den folgenden Abschnitten werden die Anzahl der Aktionen und die benötigte Zeit der beiden Planer, das jeweilige Problem zu planen miteinander verglichen. Die Zeiteinheit sei grundsätzlich Sekunden.

3.2.1 Auswertung Sequentielle Domäne

Sequentielle Aktionen	HSP total time	LPG total time
100	0,0309950	0,06
200	0,1299800	0,19
300	0,3099530	0,79
400	0,5319190	1,69
500	0,8638690	3,11
600	1,2488100	13,25
700	1,7297400	24,58
800	2,2506600	36,38
900	2,8815600	57,94
1000	3,5674600	99,91

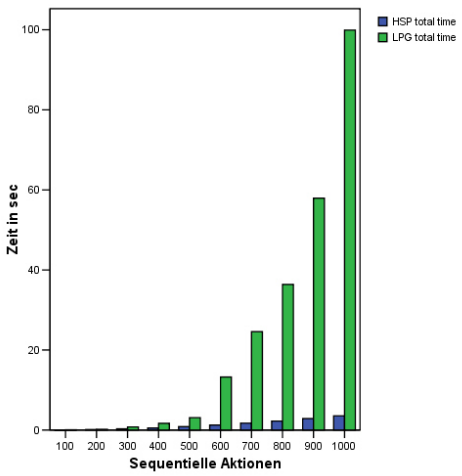

Abbildung 3.11 Sequentielle Domäne

In Abbildung 3.11 sind die Zeiten, die HSP und LPG-TD im Test für die sequentielle Domäne benötigten abgebildet. Auffällig hierbei ist, dass die Zeit, die LPG-TD im Vergleich zu HSP benötigt, die sequentielle Domäne zu planen übermassig stark ansteigt. Während HSP bei einem Anstieg der Anzahl der auszuführenden Aktionen um den Faktor 10 in der Planungszeit (HSP total time) ebenfalls nur um knapp den Faktor 100 ansteigt, benötigt LPT-TD im Vergleich dazu über den Faktor 1500.

3.2.2 Auswertung parallele Domäne

Parallel	HSP total time	LPG total time
100	0,0259960	0,05
200	0,1039850	0,12
300	0,2359640	n.m. > 250
400	0,4339340	
500	0,6918940	
600	1,0178500	
700	1,4047900	
800	1,8447200	
900	2,3616400	
1000	2,9195600	

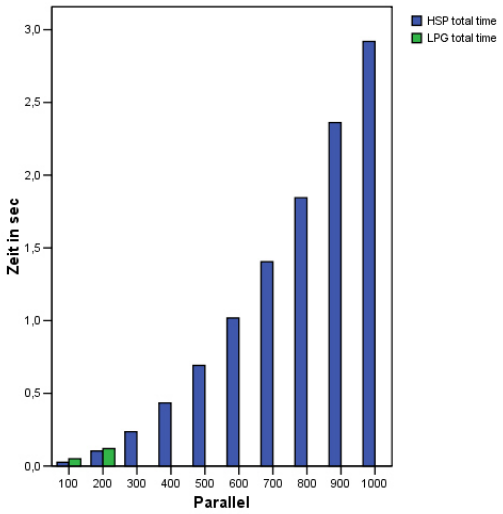

Abbildung 3.12 Parallele Domäne

Zu den in Abbildung 3.12 dargestellten Planungszeiten sei angemerkt, dass LPG-TD der Programmierung nach keine Planungsprobleme unterstützt, die mehr als 250 parallele Aktionen beinhalten. Von daher ist die parallele Domäne auf beiden Systemen nur bedingt vergleichbar. HSP hingegen ist bezüglich der Anzahl an parallel ausführbaren Aktionen bis 1000 nicht beschränkt und war in diesem Test sogar schneller, als bei der Planung von der gleichen Anzahl an sequentiellen Aktionen.

3.2.3 Auswertung sequentiell und parallelen Domäne

Wie in Kapitel 3.1.2 beschrieben, wurde die Anzahl der Aktionen so erhöht, dass eine ungefähre Vergleichbarkeit mit den anderen Domänen besteht. In der Tabelle aus Abbildung 3.1.5 wurde die Anzahl der Aktionen jeweils um die Quadrate der angegebenen Anzahlen von sequentiellen Mal parallelen Aktionen erhöht.

Seqentiell * parallel	HSP total time	LPG total time
10	0,0539920	0,07
12	0,1129830	0,15
14	0,2099670	0,35
16	0,3689430	0,96
17	0,4749270	1,39
18	0,6009080	3,78
20	0,9368570	10,6
22	1,3857900	29,09
24	1,9957000	51,24
26	2,7605800	95,94
28	3,7464300	113,39
30	4,9652500	Max time exeeded
32	6,4380200	

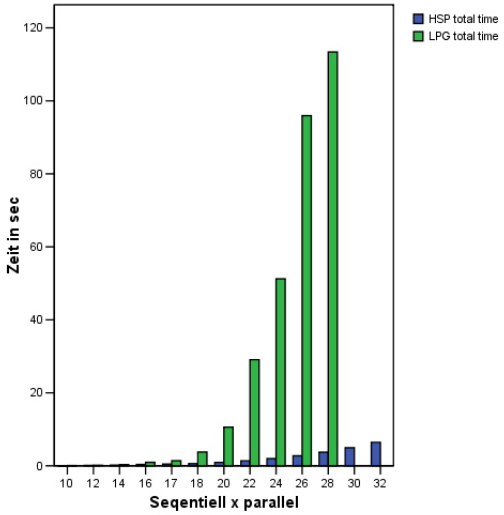

Abbildung 3.13 Sequentiell und parallelen Domäne

In diesem Fall sei angemerkt, dass LPG-TD bei einer Anzahl von jeweils 30, also 30 * 30 = 900 Aktionen das Maximum der maximal erlaubten Zeit für eine Planung überschritten hat. Während HSP selbst bei 32 parallelen Aktionen pro insgesamt 32 sequentiellen Planungsschritten nur knapp das doppelte der Planungszeit wie bei 1000 sequentiellen Aktionen benötigte, stieß LPT-TD bei dieser Planungsdomäne an seine Grenzen.

3.2.4 Auswertung sequentielle und parallele Domäne mit Delete

Wie in Kapitel 3.1.4 beschrieben, soll bei dieser Domäne untersucht werden, ob das Löschen der Vorbedingung einer Aktion nach deren Ausführung eine Auswirkung auf die Performanz der beiden Planer hat.

Die Anzahl der Aktionen wurde analog zu der Domäne in 3.2.3 erhöht. Die Ergebnisse der Auswertung sind in Abbildung 3.14 dargestellt:

Seqentiell * parallel	HSP total time	LPG total time
10	0,0539920	0,07
12	0,1129830	0,13
14	0,2099670	0,3
16	0,3689430	0,89
17	0,4749270	1,22
18	0,6009080	4,41
20	0,9368570	10,2
22	1,3857900	26,13
24	1,9957000	49,41
26	2,7605800	92,58
28	3,7464300	124,08
30	4,9652500	Max time exeeded
32	6,4380200	

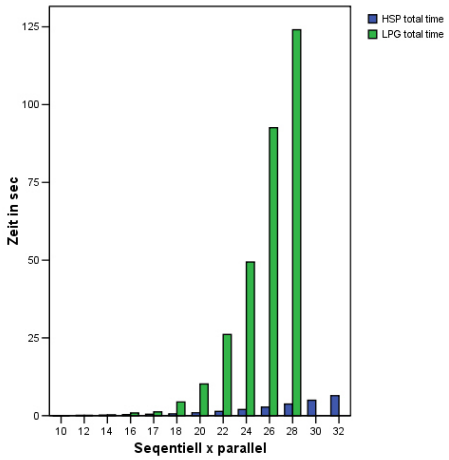

Abbildung 3.14 Sequentielle und Parallele Domäne mit Delete

Vergleicht man dieses Ergebnis mit der Domäne ohne Löschen der Vorbedingung, so lässt sich kein signifikanter Unterschied feststellen. Das Löschen der Vorbedingungen hat keine Auswirkung auf die Performanz der Planer.

3.2.5 Auswertung Durative Actions Domäne

Gegenstand der Untersuchung bezüglich dieser Domäne war die Frage, ob die Länge der Duration eine Auswirkung auf die Dauer der Planung hat. In Abbildung 3.15 sind die Planungszeiten für HSP* bei einer Länge jeder Durative-Action von 1 bis 1000 Zeiteinheiten. Wie in den beiden vorigen Domänen errechnet sich die Gesamtzahl an Domänen aus den Quadraten der jeweils angegebenen Anzahl an sequentieller Mal paralleler Domänen. Abbildung 3.16 stellt die Ergebnisse für LPG-TD in dieser Domäne dar.

Seqentiell * parallel	Duration 1	Duration 10	Duration 100	Duration 500	Duration 1000
10	0,06	0,05	0,05	0,05	0,06
14	0,22	0,22	0,22	0,22	0,22
18	0,61	0,62	0,62	0,62	0,63
22	1,44	1,44	1,43	1,43	1,45
26	2,87	2,86	2,86	2,86	2,92
30	5,12	5,12	5,13	5,12	5,19

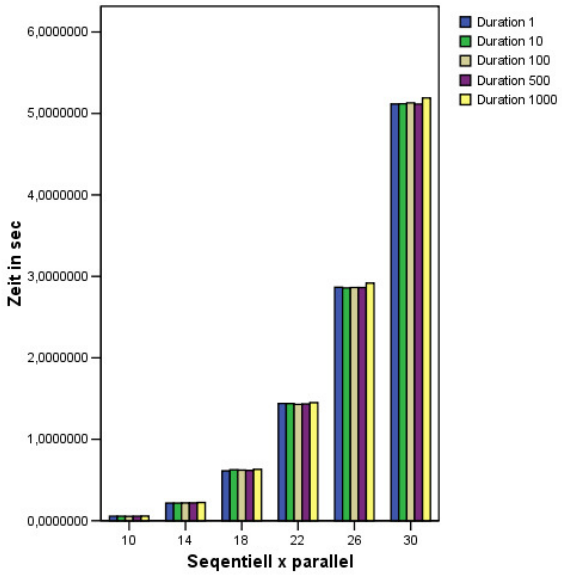

Abbildung 3.15 Durative Actions bei HSP*

Seqentiell * parallel	Duration 1	Duration 10	Duration 100	Duration 500	Duration 1000
10	0,06	0,06	0,07	0,07	0,07
14	0,38	0,35	0,29	0,4	0,31
18	1,89	1,72	3,29	2,98	1,44
22	28,76	18,56	27,86	21	29,14
26	94,28	97,96	102,13	94,67	105,32
30	n.m	n.m	n.m	n.m	n.m

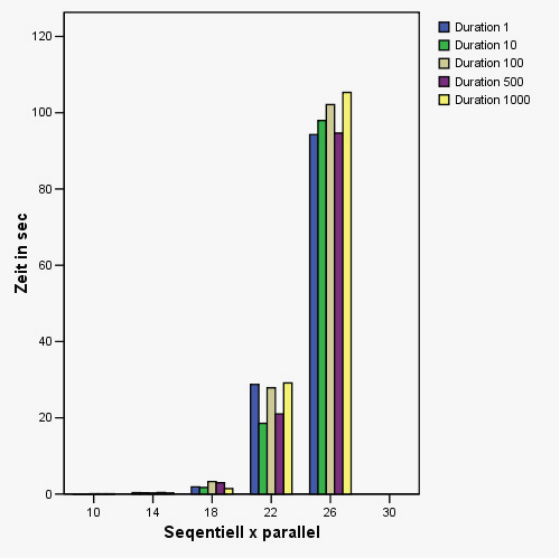

Abbildung 3.16 Durative Actions bei LPG-TD

Es wird ersichtlich, dass die Duration der Aktionen keine signifikante Auswirkung auf die Planungsgeschwindigkeit der beiden Planer hat. LPG-TD überschritt allerdings bei 30 * 30=900 Aktionen die zulässige Gesamtzeit, die in der Programmierung vorgesehen wurde.

Obwohl LPG-TD bis zu einer Anzahl von 18*18=324 Aktionen vergleichbare Planungszeiten, wie HSP* aufwies, wuchs Zeit ab 324 Aktionen wesentlich stärker an. Bei 26*26=676 Aktionen benötigte LPG-TD beinahe 30 Mal so lange eine, Lösung des Planungsproblems zu finden, als HSP.

4 Eigene Daily Life Domäne

Der Ausgangspunkt für die Entwicklung der Daily Life Domäne ist die grundsätzliche Überlegung, den Tagesablauf einer älteren Person abzubilden. D.h. es sollen die wichtigsten Tätigkeiten dieser Person in Form von Aktionen dargestellt, sowie die wichtigsten Ziele eines Tages festgehalten werden. Damit besitzt man den Input für die eingesetzten Planer, die einen möglichst optimalen Tagesablauf in Bezug auf eine Reihe von Bedingungen vorschlagen sollen.

Eine erste wichtige Vorüberlegung hierfür war, dass man es mit Tätigkeiten zu tun hat, die alle eine unterschiedliche Zeitdauer aufweisen. Aus diesem Grund fiel die Entscheidung für den Einsatz von durative-actions, mit deren Hilfe es möglich ist, Aktionen eine feste oder von anderen Parametern abhängige Dauer zuzuweisen.

Eine weitere Feststellung war, dass eine sinnvolle und realitätsnahe Modellierung nur möglich ist, wenn functions eingesetzt werden, um bestimmte Ressourcen abzubilden. Ein einfaches Beispiel hierfür ist das Essen. Eine Person kann nur etwas essen, wenn ausreichend Lebensmittel im Haushalt vorhanden sind. Deshalb werden diese in Form einer function abgebildet, die mit jeder Mahlzeit abnimmt. Eine genauere Erläuterung erfolgt im Abschnitt 4.1.

Als weitaus größeres Problem zu Beginn der Modellierung war die Fragestellung, wie man zeitliche constraints in die Domäne einbaut. Man möchte beispielsweise verhindern, dass die Person bereits morgens zu Abend isst. Eine mögliche Lösung für dieses Problem stellen die timed initial literals dar, die seit PDDL2.2 zur Verfügung stehen. Diese ermöglichen es, ein Literal innerhalb eines bestimmten Zeitfensters true oder false werden zu lassen. Auf diese Weise kann man dann beispielsweise abbilden, dass das Abendessen nur zwischen 18.30 Uhr und 20.00 Uhr möglich sein soll.

Der Einsatz vom timed initial literals stellt uns allerdings vor das Problem, dass der vorher ausgewählte Planer HSP* diese nicht als Input akzeptiert. Da es uns aber nicht machbar erschien, eine realitätsnahe Modellierung ohne den Einsatz von Zeitfenstern durchzuführen, haben wir beschlossen, für die Daily Life Domäne auf den Einsatz von HSP* zu verzichten und lediglich auf LPG-td zurückzugreifen. Dies birgt natürlich das Risiko, dass die Domäne zu speziell auf diesen Planer zugeschnitten ist. Aus unserer Sicht ist dies allerdings von geringerer Bedeutung als die Entwicklung einer realitätsnahen Domäne. Weiterhin gehen wir davon aus, dass eine Anpassung an andere Planer grundsätzlich mit relativ geringem Aufwand möglich ist, sofern diese die benötigten Anforderungen zur Verfügung stellen.

Im Folgenden wird zuerst die modellierte Domäne mit ihren einzelnen Aktionen, den dahinter stehenden Überlegungen und möglichen Alternativen dargestellt und darauf aufbauend eine mögliches Planungsproblem für einen Tag erläutert. Zum Abschluss dieses Abschnitts sollen kurz die erhaltenen Planungsergebnisse des LPG-td Planers aufgezeigt werden.

Als requirements benötigen wir strips, fluents, durative-actions, timed initial literals und negative preconditions. Die jeweilige Notwendigkeit ergibt sich aus den folgenden Ausführungen zu den einzelnen Aktionen. Dort wird auch genauer auf die eingesetzten predicates und functions eingegangen.

Allgemein wurde versucht, die Benennung der Aktionen, Prädikate und Funktionen so zu wählen, dass sich deren Bedeutungen weitestgehend aus dem Kontext erschließen lassen. Auf diese Weise soll ein möglichst einfaches Verständnis der Domäne gewährleistet werden. Wir gehen dabei implizit davon aus, dass die Leistungsfähigkeit der Planer nicht durch Benennungen (z.B. Länge der Aktionsnamen) beeinflusst wird.

4.1 Aktionen der Daily Life Domäne

```
(:durative-action Fruehstuecken
:parameters (?y - Person)

:duration (=?duration 30)

:condition (and (at start (hat_Zeit ?y))(over all (wach ?y)) (over all
       (moeglich_Fruehstueck))  (at  start  (  >=  (vorhandenes_Essen)
       8))(over all (befindet_sich ?y Zu_Hause)))

:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))(at
       end (gefruehstueckt ?y)) (at end (decrease (vorhandenes_Essen)
       8)))
)
```

Abbildung 4-1: Aktion Fruehstuecken

Wie es sich für einen ordentlichen Tagesablauf gehört, soll auch hier mit dem Frühstück (vgl. Abbildung 4-1) begonnen werden. Wie bereits oben erwähnt, werden durative-actions eingesetzt, die Aktionen mit einer bestimmten (Zeit-)Dauer versehen. In unserem Falle hat diese Aktion eine Dauer von 30. Dabei wird vom Prinzip her nichts darüber ausgesagt, worum es sich bei den 30 handelt. D.h. es könnte sich um Sekunden, Minuten, Stunden oder auch andere Einheiten handeln. Für diese Domäne haben wir eine Aufteilung des Tages in Minuten zugrunde gelegt. Selbstverständlich kann diese Zeitdauer jederzeit - beispielsweise für den Einsatz der Domäne für eine andere Person - nach oben oder unten angepasst werden oder von anderen Faktoren, wie z.B. einem bestimmten Wochentag, abhängig gemacht werden.

Eine Vorbedingung für die Aktion ist, dass die Person, die frühstücken möchte, wach ist `(over all (wach ?y))`. Diese Bedingung wird in vielen der nachfolgenden Aktionen eingesetzt, um sicherzustellen, dass die Tätigkeiten der Person nur tagsüber ausgeführt werden können. Die Zeiten, in denen eine Person wach ist, werden dann innerhalb des Planungsproblems mit Hilfe von timed initial literals gesteuert. Es wird also angegeben, wann die Person aufsteht und wann diese zu Bett geht (z.B. `(at 420 (wach Oma))` und `(at 1290 (not (wach Oma)))`). Auf diese Weise könnte man als Erweiterung auch einen Mittagsschlaf modellieren, indem man ein weiteres Zeitintervall für das Prädikat `(wach Oma)` angibt.

Mit der Bedingung `(over all (wach ?y))` wird folglich bereits eine Einschränkung der möglichen Ausführungszeitpunkte der Aktion festgelegt. Allerdings wäre es jetzt noch immer realitätsfremd möglich, dass erst gegen Abend gefrühstückt wird. Um dies zu verhindern, wird zusätzlich die Bedingung `(over all (moeglich_Fruehstueck))` eingesetzt. Hierfür kommen wiederum timed initial literals zum Einsatz. An dieser Stelle ist natürlich die Frage berechtigt, warum man sich nicht auf das Prädikat `moeglich_Fruehstueck` beschränkt und auf `wach ?y` verzichtet. Dies hätte momentan auf das Planungsergebnis keinen Einfluss, da das Zeitintervall zum Frühstücken komplett innerhalb des Zeitintervalls, in der die Person wach ist, liegt. Zur Erläuterung soll folgendes Beispielszenario betrachtet werden. Mit Hilfe der Domäne sollen auch andere Tage modelliert werden, und an einem Sonntag möchte die Person eine Stunde länger schlafen, d.h. bis 8 Uhr. Würde jetzt nur auf das Prädikat `moeglich_Fruehstück` zurückgegriffen, so würde der Planer evtl. einen Plan ausgeben, der um 7 Uhr Frühstück vorschlägt, obwohl die Person zu diesem Zeitpunkt noch schläft. Um dies zu verhindern und um eine möglichst allgemeine und erweiterbare Modellierung zu erhalten, werden deshalb beide Prädikate innerhalb der conditions abgeprüft.

Ein weiteres Problem, das bei der Modellierung zum Tragen kommt, war, dass viele Aktionen sequentiell ausgeführt werden müssen. Es scheint beispielsweise nicht sinnvoll, dass es gleichzeitig möglich ist, zu frühstücken und einkaufen zu gehen. Dies kann selbstverständlich verhindert werden, indem ein Planer eingesetzt wird, der reines sequentielles Planen als Option unterstützt. Auf der anderen Seite würde damit die Modellierung eines Tagesablaufs natürlich sehr eingeschränkt werden, da es trotzdem Tätigkeiten gibt, die unter Umständen parallel ausgeführt werden können. So wäre es denkbar, dass die Person gleichzeitig zu Abend isst und Musik hört (diese Aktion wird später genauer erläutert). Aus diesem Grund haben wir beschlossen, auf eine Art Pattern, das bereits im Rahmen der Erläuterungen zu HSP* kurz vorgestellt wurde, zurückzugreifen. Mit diesem ist es möglich, wieder verwendbare Ressourcen darzustellen, die zu bestimmten Zeitpunkten belegt sein können. In unserem Fall ist diese Ressource die Person, die eine Aktion ausführt und die währenddessen keine andere Aktion ausführen kann, die ebenfalls diese Ressource benötigt. Auf diese Weise wird sichergestellt, dass einige Aktionen nur nacheinander ausgeführt werden können, aber dennoch eine parallele Ausführung von Aktionen nicht komplett ausgeschlossen wird. Das Pattern besteht aus drei Teilen. Zum einen wird in der condition abgeprüft, ob die Person zu Beginn der Aktion Zeit für diese hat `(at start (hat_Zeit ?y))`. Ist dies der Fall, so wird als Effekt zu Beginn der Aktion das Prädikat `hat_Zeit ?y` aus dem Zustandsraum entfernt. `(at start (not (hat_Zeit ?y)))`. Dies hat zur Folge, dass keine andere Aktion, die als Vorbedingung `(at start (hat_Zeit ?y))` besitzt, parallel ausgeführt werden kann. Ist die Aktion abgeschlossen, so wird das Prädikat wieder dem Zustandsraum hinzugefügt `(at end (hat_Zeit ?y))` und die Ressource steht wieder anderen Aktionen zur Verfügung.

Eine weitere Bedingung, damit eine Person frühstücken kann, ist, dass sie sich während der gesamten Dauer des Essens zu Hause befindet `(over all (befindet_sich ?y Zu_Hause))`. Damit wird verhindert, dass die Person nicht frühstückt, wenn sie sich z.B. beim Arzt oder im Laden aufhält.

Das Ergebnis der Aktion Frühstücken ist dann, dass gefrühstückt wurde (`(at end (gefruehstueckt ?y))`. Dies wird u.a. für das Erreichen des Zielzustandes gefordert, d.h. diese Aktion muss zwangsläufig in dem sich ergebenden Tagesplan vorhanden sein.

Ein weiterer Effekt des Frühstückens ist, dass das verfügbare Essen im Haushalt abgenommen hat. Dies wird mit Hilfe einer function, die um 8 abnimmt, dargestellt `(at end (decrease (vorhandenes_Essen) 8))`. Die 8 ist dabei beliebig gewählt und kann jederzeit auf einen anderen Wert abgeändert werden. In der condition wurde außerdem bereits abgeprüft, ob zu Beginn des Frühstücks ausreichend Essen zur Verfügung steht (`(at start (>= (vorhandenes_Essen) 8))`. Die im Haushalt vorhandenen Lebensmittel sind innerhalb dieser Modellierung noch relativ abstrakt als nicht weiter definierte Menge dargestellt. Die function `vorhandenes_Essen` soll hier vorwiegend dazu genutzt werden, um später die Person zu veranlassen, einkaufen zu gehen, wenn nicht mehr genügend Essen im Haus vorhanden ist. Aufbauend auf dieser Grundmodellierung sind natürlich diverse Erweiterungen denkbar. So könnte man beispielsweise jedes Lebensmittel als eigene function abbilden, wodurch die Person nur die wirklich benötigten Produkte einkauft.

Im Folgenden sollen nun die beiden Aktionen Mittagessen und Abendessen (vgl. Abbildung 4-2) betrachtet werden. Das Abendessen unterscheidet sich dabei vom Frühstücken lediglich durch Dauer und benötigtes Essen, sowie in der Nutzung des Prädikats `zu_Abend_gegessen ?y`.

```
(:durative-action Mittagessen
:parameters (?y - Person)
:duration (=?duration 45)
:condition (and (at start (hat_Zeit ?y))(over all (wach ?y)) (at start
       (mittagessen_geliefert))(over all (befindet_sich ?y Zu_Hause))
       (at start (not (mittagessen_kalt))))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))(at
       end (zu_Mittag_gegessen ?y)))
)

(:durative-action Abendessen
:parameters (?y - Person)
:duration (=?duration 40)
:condition (and (at start (hat_Zeit ?y))(over all(wach ?y)) (over all
       (moeglich_Abendessen))(at start ( >= (vorhandenes_Essen) 12))
       (over all (befindet_sich ?y Zu_Hause)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))(at
       end (zu_Abend_gegessen ?y))(at end (decrease (vorhandenes_Essen)
       12)))
)
```

Abbildung 4-2: Aktionen Mittagessen und Abendessen

Eine etwas andere Überlegung wurde dann dem Mittagessen zugrunde gelegt. Bei diesem gehen wir davon aus, dass das Mittagessen nicht selbst zubereitet wird, sondern fertig geliefert wird („Essen auf Rädern“). Aus diesem Grund wurde auf ein Zeitfenster, in dem das Mittagessen möglich ist, verzichtet und stattdessen auf das Prädikat `mittagessen_geliefert` zurückgegriffen. Innerhalb des Planungsproblems wird dann der Lieferzeitpunkt des Essens angegeben. Allerdings entstand hierdurch wieder das Problem, dass es ab diesem Zeitpunkt möglich wäre, den ganzen restlichen Tag über das Mittagessen einzunehmen. Eine einfache Lösung des Problems wäre wiederum, nur ein bestimmtes Zeitfenster zu modellieren, in dem `mittagessen_geliefert` gültig ist. Da dies in diesem Zusammenhang allerdings nicht sinnvoll erscheint, da das gelieferte Essen nicht einfach wieder verschwinden kann, wird ein weiteres Prädikat `mittagessen_kalt` eingesetzt. Als Vorbedingung gilt nun, dass das Mittagessen nur zu sich genommen werden kann, wenn es nicht der Fall ist, dass das Mittagessen kalt ist. Aus diesem Grund werden als requirement auch negative preconditions benötigt `(at start (not (mittagessen_kalt)))`. Selbstverständlich wäre es auch leicht denkbar, ohne negative Vorbedingungen zu arbeiten, indem man den Sachverhalten umgekehrt modelliert und ein Prädikat für warmes Mittagessen einsetzt, welches dann in einem bestimmten Zeitintervall gültig ist.

Als nächsten sollen die Aktionen Einkaufen, Arztbesuch und Apothekengang (vgl. Abbildung 4-3) betrachtet werden, deren Grundaufbau sehr ähnlich ist. Alle besitzen eine feste Dauer, haben als Bedingung, dass die Person wach sein muss, und es kommt das „Ressourcenpattern“ zum Einsatz. Weitere Bedingungen sind, dass sich die Person jeweils an einem bestimmten Ort befinden muss (z.B. `(befindet_sich ?y Arztpraxis)`) und außerdem die jeweiligen Öffnungszeiten beachtet werden müssen, die wiederum über timed initial literals gesteuert werden (z.B. `over all(aerztliche_Sprechstunde)))`.

```
(:durative-action Einkaufen
:parameters (?y - Person)
:duration (=?duration 60)
:condition (and (over all (wach ?y)) (at start (hat_Zeit ?y))
        (over all(geoeffneter_Laden))(over all (befindet_sich
        ?y Laden)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))
        (at end (increase (vorhandenes_Essen) 30)))
)

(:durative-action Apothekengang
:parameters (?y - Person ?z - Medikament)
:duration (=?duration 10)
:condition (and (over all (wach ?y)) (at start (hat_Zeit ?y))
        (over all(geoeffnete_Apotheke))(over all (befindet_sich
        ?y Apotheke)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))
        (at end (increase (vorhandene_Anzahl ?z)20)))
)

(:durative-action Arztbesuch
:parameters (?y - Person)
:duration (=?duration 60)
:condition (and (over all (wach ?y)) (at start (hat_Zeit ?y))
        (at start(fuehlt_sich_unwohl ?y))
        (over all(aerztliche_Sprechstunde))(over all (befindet_sich
        ?y Arztpraxis)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))
        (at end (not (fuehlt_sich_unwohl ?y))))
)
```

Abbildung 4-3: Aktionen Einkaufen, Apothekengang und Arztbesuch

Größere Unterschiede der Aktionen sind lediglich in den Effekten vorhanden. Nach einem Arztbesuch fühlt sich die Person nicht mehr unwohl und dieser kommt in einem Plan nur dann vor, wenn `fuehlt_sich_unwohl ?y` im Initialzustand angegeben wurde. Die Aktion Einkaufen wird ausgeführt, wenn sich für die Mahlzeiten nicht genügend Essen im Haus befindet und erhöht das vorhandene Essen um die willkürlich gewählte Menge 30 (`(increase (vorhandenes_Essen)30)`). Für mögliche alternative Modellierungen sei auf die Ausführungen zur Aktion Frühstücken verwiesen. Schließlich wird die Aktion Apothekengang dazu benötigt, um den Nachschub von Medikamenten sicherzustellen. D.h. durch das Ausführen der Aktion erhöht sich die vorhandene Anzahl von Tabletten eines bestimmten Medikaments, welches nicht mehr ausreichend vorhanden ist.

Oftmals sind ältere Personen gezwungen, verschiedene Medikamente einzunehmen. Dies wird ebenfalls durch eine spezielle Aktion abgebildet, die als Parameter neben der Person ein bestimmtes Medikament benötigt. Damit soll es möglich sein, nicht nur ein einziges Medikament zu verabreichen (vgl. Abbildung 4-4).

```
(:durative-action Medikamente_nehmen
:parameters (?y - Person ?z - Medikament)
:duration (=?duration 3)
:condition (and (over all(wach ?y)) (over all (moeglich_Medikament ?z))
        (at start (>= (vorhandene_Anzahl ?z) 2)))
:effect (and (at end (genommen ?z))(at end (decrease (vorhandene_Anzahl
        ?z) 2)))

)
```

Abbildung 4-4: Aktion Medikamente_nehmen

Es kommen vom Medikament abhängige Zeitfenster zum Einsatz, womit die oftmals vorhandene Aufteilung der Einnahmezeitpunkte in morgens, mittags, abends dargestellt werden soll. Weiterhin wird die Anzahl der vorhandenen Tabletten abhängig vom jeweiligen Medikament mit functions abgebildet. Sind zu wenige Tabletten eines Medikaments vorhanden, so wird ein Apothekengang der Person notwendig. Bei dieser Aktion wurde erstmals auf den Einsatz des „Ressourcenpatterns“ verzichtet, da wir davon ausgehen, dass das Einnehmen von Tabletten eine andere Aktion nicht „stört“. So ist es denkbar, dass beispielsweise das Essen kurz unterbrochen werden kann oder auch die Tabletten an einem anderen Ort als zu Hause eingenommen werden können, da diese problemlos mitzuführen sind. Außerdem gehen wir davon aus, dass die Dauer der Aktion so kurz ist, dass die Dauer einer anderen parallelen Aktion nicht (z.B. Fernsehen) oder zumindest nicht signifikant (z.B. Essen) beeinflusst wird. Für diese Aktion sind wieder verschiedenste Modellierungsalternativen denkbar. Eine erste Erweiterung wäre beispielsweise, dass die Anzahl der genommenen Tabletten nicht mehr als konstant angenommen wird, sondern abhängig vom Medikament modelliert wird.

Für die Aktionen Einkaufen, Arztbesuch und Apothekengang wurde bereits erwähnt, dass eine Vorbedingung ist, dass sich die Person an dem entsprechenden Ort aufhalten muss. Somit muss natürlich eine Aktion existieren, die einen Ortswechsel möglich macht. Für eine ältere Person schien uns hierfür das Fahren mit dem Bus als sehr realitätsnah (vgl. Abbildung 4-5).

```
(:durative-action Busfahren
:parameters (?y - Person ?x - Ort ?z - Ort )
:duration (= ?duration (fahrzeit ?x ?z))
:condition (and (over all (wach ?y)) (at start (hat_Zeit ?y)) (at start
        (befindet_sich ?y ?x)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))
        (at end (befindet_sich ?y ?z)) (at start (not (befindet_sich
        ?y ?x))) (at end (increase gesamtfahrzeit (fahrzeit ?x ?z))))
)
```

Abbildung 4-5: Aktion Busfahren

Dabei wird davon ausgegangen, dass die Zeitdauer für das Busfahren nicht konstant ist, sondern abhängig davon, zwischen welchen Orten der Bus verkehrt. Dadurch ist es möglich, eine Art Busfahrplan in der Problembeschreibung anzugeben, indem die Fahrzeiten abhängig von den verbundenen Orten variieren. Allerdings handelt es sich dabei um eine Vereinfachung, da es zu jedem Zeitpunkt möglich ist, den Bus zu nehmen, d.h. Wartezeiten, bis der Bus kommt, existieren nicht. Hier wäre es wiederum denkbar, mit Hilfe von timed initial literals bestimmte Abfahrtszeitpunkte darzustellen und als Bedingung für die Busfahrt zu verwenden. Zusätzlich erfolgt eine „Speicherung“ der gesamten Zeit, die eine Person an einem Tag mit Busfahren verbringt `(increase gesamtfahrzeit (fahrzeit ?x ?z))`. Dies ist notwendig, damit unnötige Busfahrten und nicht geeignete Routen verhindert werden können. Eine genauere Erläuterung und Begründung dieses Sachverhaltes erfolgt im Abschnitt 4.2.

Bis jetzt handelt es sich bei allen vorgestellten Aktionen um „lebensnotwendige“ Tätigkeiten, die während eines Tages ausgeführt werden müssen. Allerdings sind wir der Meinung, dass ein Tagesplan auch die „Lebensqualität“ einer Person nicht unberücksichtigt lassen sollte. Aus diesem Grund sollen noch die letzten vier Aktionen unserer Daily Life Domäne eingeführt werden (vgl. Abbildung 4-6).

Die Aktionen Spaziergang, Fernsehen und MP3_Player erhöhen jeweils die Zufriedenheit einer Person um einen bestimmten Wert `(at end (increase (zufriedenheit ?y) [Wert]))`. Auf diese Weise ist es möglich, als Ziel eines Tages eine bestimmte Mindestzufriedenheit der Person festzulegen, die erreicht werden muss. Diese Mindestzufriedenheit kann der Planer abhängig vom Initialzustand und den Vorbedingungen der Aktionen auf unterschiedliche Art und Weise erreichen. So ist es beispielsweise denkbar, dass eine Zufriedenheit von 10 durch das einmalige Sehen eines interessanten Fernsehprogramms erreicht wird oder durch das zehnmalige Nutzen des MP3-Players. Auf die Einsatzweise der function `zufriedenheit ?y` wird ebenfalls im Abschnitt 4.2 noch gezielter eingegangen.

```
(:durative-action Spaziergang
:parameters (?y - Person)
:duration (=?duration 45)
:condition (and (over all (wach ?y)) (at start (hat_Zeit ?y)))
:effect (and (at start (not (hat_Zeit ?y))) (at end (hat_Zeit ?y))
        (at end (increase (zufriedenheit ?y)15)))
)

(:durative-action Fernsehen
:parameters (?y - Person ?z - Fernseher)
:duration (=?duration 60)
:condition (and (at start (verfuegbar ?z))(over all (wach ?y))
        (over all (interessantes_Programm))(over all (befindet_sich
        ?y Zu_Hause)))
:effect (and (at start (not (verfuegbar ?z)))
        (at end (verfuegbar ?z))(at end (increase (zufriedenheit
        ?y)10)))
)

(:durative-action MP3_Player_nutzen
:parameters (?y - Person ?z - MP3_Player)
:duration (=?duration 10)
:condition (and (over all (wach ?y))  (at start (>=
        (akku_Kapazitaet ?z) 10)))
:effect (and (at end (increase (zufriedenheit ?y)1)) (at end (decrease
        (akku_Kapazitaet ?z) 10)))
)

(:durative-action Akku_Aufladen
:parameters (?y - Person ?z - MP3_Player)
:duration (=?duration 120)
:condition (and (at start (befindet_sich ?y Zu_Hause)))
:effect (and (at end (increase (akku_Kapazitaet ?z)100)))
)
```

Abbildung 4-6: Aktionen Spaziergang, Fernsehen, MP3_Player_nutzen, Akku_Aufladen

Zum Abschluss der Ausführungen zur Daily Life Domäne sollen noch einige kurze Anmerkungen gegeben werden. Die Aktionen MP3_Player_nutzen und Fernsehen verzichten auf das „Ressourcenpattern“ für die Person. Das Hören von Musik kann folglich parallel zu jeder Aktion ausgeführt werden, wenn der Akku des Gerätes ausreichend geladen ist. Fernsehen ist dagegen nur möglich, wenn die Person sich zu

Hause befindet, also beispielsweise parallel zum Essen, aber nicht parallel zu einem Arztbesuch. Außerdem wurde für das Fernsehen ein zusätzliches Ressourcenpattern (Einsatz von `verfuegbar ?z`) notwendig, da ansonsten Pläne entstehen könnten, die paralleles Fernsehen im Tagesablauf vorsehen könnten. Dies ergibt allerdings keinen Sinn, da zu einem Zeitpunkt nur ein einziges Programm verfolgt werden kann. Alle bisherigen Aktionen hatten als Vorbedingung, dass die Person wach sein muss. Dass dies nicht notwendigerweise für alle Aktionen gelten muss, zeigt die Aktion, mit der der Akku des MP3-Players aufgeladen werden kann, da hier die einzige Vorbedingung ist, dass die Person zu Beginn des Ladevorgangs zu Hause sein muss.

4.2 Eine mögliches Planungsproblem für die Daily Life Domäne

Im Folgenden soll nun ein mögliches Planungsproblem für die vorher betrachtete Daily Life Domäne kurz charakterisiert werden.

```
(define (problem Tag1)(:domain DailyLife)

(:objects      Oma - Person
               Blutverduenner Knoblauch - Medikament
               TV - Fernseher
               IPod - MP3_Player
               Zu_Hause - Ort
               Arztpraxis - Ort
               Apotheke - Ort
               Laden - Ort
)
```

Abbildung 4-7: Objekte

Um die einzelnen Aktionen nutzen zu können, ist eine Reihe von Objekten notwendig (vgl. Abbildung 4-7). Als Nächstes erfolgt die Definition des Initialzustandes. Dort werden zuerst die benötigten timed initial literals (vgl. Abbildung 4-8) festgelegt, d.h. die möglichen Zeitfenster für bestimmte Aktionen.

```
(:init  (at 420 (wach Oma))
        (at 1290 (not (wach Oma)))
        (at 420 (moeglich_Fruehstueck))
        (at 540 (not (moeglich_Fruehstueck)))
        (at 720 (mittagessen_geliefert))
        (at 735 (mittagessen_kalt))
        (at 1110 (moeglich_Abendessen))
        (at 1200 (not (moeglich_Abendessen)))
        (at 480 (geoeffneter_Laden))
        (at 1200 (not (geoeffneter_Laden)))
        (at 480 (moeglich_Medikament Blutverduenner))
        (at 540 (not (moeglich_Medikament Blutverduenner)))
        (at 1020 (moeglich_Medikament Knoblauch))
        (at 1080 (not (moeglich_Medikament Knoblauch)))
        (at 540 (geoeffnete_Apotheke))
        (at 1230 (not (geoeffnete_Apotheke)))
        (at 860 (geoeffnete_Apotheke))
        (at 1080 (not (geoeffnete_Apotheke)))
        (at 480 (aerztliche_Sprechstunde))
        (at 840 (not (aerztliche_Sprechstunde)))
        (at 1200 (interessantes_Programm))
        (at 1260 (not (interessantes_Programm)))
        (at 1380 (interessantes_Programm))
        (at 1440 (not (interessantes_Programm)))
```

Abbildung 4-8: Timed Initial Literals

Danach erfolgt die „Initialisierung“ der eingeführten functions (vgl. Abbildung 4-9). Die Werte wurden dabei so gewählt, dass die davon abhängigen Aktionen in einem gültigen Plan zumindest einmal ausgeführt werden müssen.

```
(= (vorhandene_Anzahl Blutverduenner) 3)
       (= (vorhandene_Anzahl Knoblauch) 1)
       (= (vorhandenes_Essen) 18)
       (= (zufriedenheit Oma) 0)
       (= (akku_Kapazitaet IPod) 20)
```

Abbildung 4-9: functions

In vorherigen Abschnitt wurde bereits erwähnt, dass für die Aktion Busfahren eine Art vereinfachter Busfahrplan festgelegt werden kann. Dies geschieht hier durch die Angabe der Verbindungszeiten zwischen den verschiedenen Orten. Außerdem wird die Gesamtfahrzeit zu Beginn eines Tages mit 0 festgelegt (vgl. Abbildung 4-10).

```
(= (fahrzeit Zu_Hause Arztpraxis) 15)
(= (fahrzeit Zu_Hause Laden) 18)
(= (fahrzeit Zu_Hause Apotheke) 20)
(= (fahrzeit Arztpraxis Zu_Hause) 15)
(= (fahrzeit Arztpraxis Apotheke) 12)
(= (fahrzeit Arztpraxis Laden) 16)
(= (fahrzeit Laden Zu_Hause) 18)
(= (fahrzeit Laden Apotheke) 22)
(= (fahrzeit Laden Arztpraxis) 16)
(= (fahrzeit Apotheke Zu_Hause) 20)
(= (fahrzeit Apotheke Arztpraxis) 12)
(= (fahrzeit Apotheke Laden) 22)
(= (gesamtfahrzeit) 0)
```

Abbildung 4-10: Busfahrzeiten

Der letzte Teil des Initialzustandes besteht dann aus einer Reihe von weiterhin notwendigen Ausgangsprädikaten, damit ein gültiger Plan erstellt werden kann.

```
(hat_Zeit Oma)
(fuehlt_sich_unwohl Oma)
(befindet_sich Oma Zu_Hause)
(verfuegbar TV)
```

Abbildung 4-11: weitere Bestandteile des Initialzustandes

Die Definition des Zielzustandes (vgl. Abbildung 4-12) erfolgt dann wiederum aufgrund der Überlegung, dass zumindest jede Aktion in einem möglichen Tagesablauf vorkommen soll.

```
(:goal (and
               (gefruehstueckt Oma)
               (zu_Mittag_gegessen Oma)
               (zu_Abend_gegessen Oma)
               (genommen Blutverduenner)
               (genommen Knoblauch)
               (not (fuehlt_sich_unwohl Oma))
               (>= (zufriedenheit Oma) 60)
       )
)
```

Abbildung 4-12: Zielzustand

Den interessantesten Teil des Planungsproblems stellt die von uns verwendete Metrik dar.

```
(:metric       minimize (gesamtfahrzeit))
```

Abbildung 4-13: verwendete Metrik

Hier war eine erste Überlegung, die in vielen IPC-Domänen verwendete Metrik total-time zu verwenden, die versucht, die Gesamtdauer des erstellten Plans zu minimieren. Allerdings erscheint die Anwendung in diesem Kontext nur bedingt sinnvoll, da es nicht das Ziel ist, die notwendigen Tätigkeiten eines Tages in möglichst schneller Abfolge hinter sich zu bringen. Das Ziel ist lediglich, einen möglichen Tagesplan zu erstellen. Eine weitere Überlegung daraufhin war, als Metrik das Maximieren der Zufriedenheit der Person als Ziel auszugeben. Dies hätte bedeutet, das der Planer einen Tagesplan ausgeben soll, der zum einen alle notwendigen Tätigkeiten (z.B. essen) enthält und gleichzeitig so oft wie möglich versucht, die Aktionen Fernsehen, MP3_Player_nutzen und Spaziergang einzubauen, um eine hohe Zufriedenheit der Person zu erreichen. Dies war allerdings bei der Nutzung von LPG-td nicht möglich, da dieser die Werte für die Zufriedenheit als Kosten betrachtet, die dann maximiert werden sollen. Dies ist selbstverständlich nicht logisch und führte somit auch zu nicht sinnvoll interpretierbaren Plänen. Aus diesem Grund haben wir uns dafür entschieden, innerhalb des Zielzustandes eine Art Mindestzufriedenheit der Person anzugeben, die variiert werden kann `(>= (zufriedenheit Oma) 60)`.
Bei Testläufen mit der Metrik minimize total-time war auffällig, dass zum Teil Pläne vorgeschlagen wurden, in denen unnötige Busfahrten vorkamen. (z.B. Busfahrt zum Arzt ohne Arztbesuch und danach Weiterfahrt zum Laden). Um dies zu verhindern, wird als Metrik nun (minimize (gesamtfahrzeit)) eingesetzt. Auf diese Weise werden diese unnötigen Busfahrten nicht nur verhindert, sondern zusätzlich ein Tagesplan entwickelt, der eine möglichst optimale Abfolge der zu besuchenden Orte gewährleistet.

4.3 Planungsergebnisse von LPG-td

Zum Abschluss der Vorstellung der selbst modellierten Daily Life Domäne soll kurz ein Planungsergebnis des Planers LPG-td aufgezeigt werden. Den in der Abbildung 4-14 sichtbaren Plan erhält man im inkrementellen Modus als fünfte Lösung.

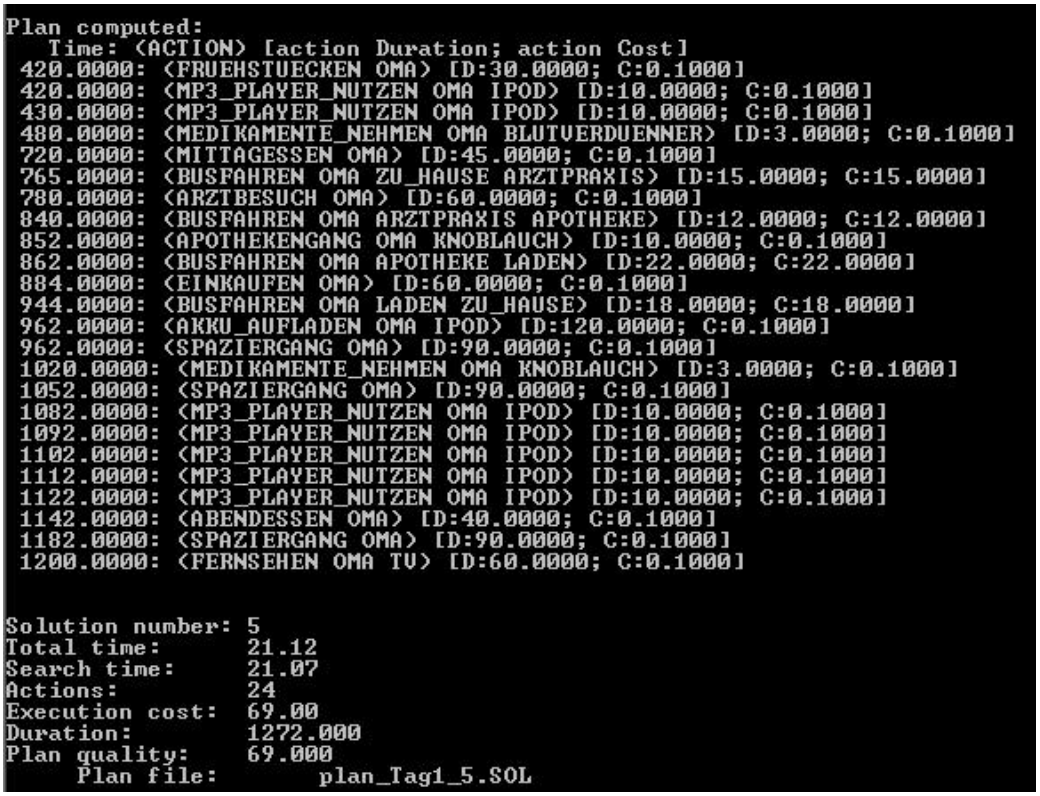

Abbildung 4-14: Planungsergebnis LPG-td 1.0

Wie man erkennen kann, werden alle Aktionen zumindest einmal ausgeführt und die vorgegebenen Zeitintervalle für die einzelnen Aktionen eingehalten. Außerdem ist ersichtlich, dass die Aktionen in der Regel sequentiell durchgeführt werden, mit Ausnahme derjenigen, die parallel ausgeführt werden dürfen. Ein Beispiel hierfür ist, dass gleichzeitig zum Frühstück bereits Musik gehört wird (Zeitpunkt 420). Als Ergebnis wird eine „Plan quality“ von 69 angegeben. Dabei handelt es sich im Grunde um die Summe über die Fahrzeiten der einzelnen Busfahrten (15+12+22+18 = 67). Die zu erkennende Differenz von 2 zwischen der plan quality und der Summe der Fahrzeiten ist damit zu begründen, dass für jede weitere Aktion pauschale Kosten von 0,1 anfallen.

5. Fazit

Als Ergebnis des Empirischen Vergleiches von HSP* und LPG-TD auf Basis der künstlichen Domänen, lässt sich sagen, dass HSP* in allen in diesem Vergleich verwendeten Domänen schneller war als LPG-TD. HSP* benötigte grundsätzlich weniger als 10 Sekunden, um die gestellten Planungsprobleme zu lösen. LPG-TD benötigte bei den komplexeren Problemen fast das Zwanzigfache der Zeit, die HSP* aufwenden musste, um das gleiche Problem zu planen.

Während HSP* alle gestellten Probleme in endlicher Zeit lösen konnte, überschritt LPG-TD bei vielen Problemen mit erhöhter Komplexität die in der Programmierung vorgesehene maximale Zeitgrenze. Zum Beispiel in der „sequentiellen und parallelen Domäne terminierte LPG-TD bei Problemen mit 30 sequentiellen und 30 parallelen Aktionen vorzeitig, ohne eine Lösung zu generieren. Darüber hinaus ist LPG-TD nicht in der Lage , Planungsprobleme mit mehr als 250 parallelen Aktionen zu lösen. Grund hierfür ist ebenfalls die in der Programmierung vorgesehene Obergrenze.

Diese Erkenntnisse stehen im direkten Widerspruch zu den Ergebnissen der letzten International Planning Competition. Bei den dort gestellten Planungsproblemen schnitt LPG-TD im Vergleich zu HSP* wesentlich besser ab. Der Grund hierfür liegt im der Struktur der Planungsprobleme. Während bei der IPC die Komplexität der Probleme über den Initialzustand und die Anzahl der definierten Prädikate erhöht wurde, erhöhte sich die Komplexität in den hier verwendeten Problemen über die Anzahl der auszuführenden Aktionen. Dieser Unterschied in den Problemklassen rührt daher, dass sich die Ergebnisse der Domänen der IPC nicht auf eine Eignung von Planungssystemen für die Anwendung in Daily Life Domänen übertragen lassen. Für diese Domänen ist HSP* wesentlich performanter und vielseitiger einsetzbar.

Allgemein lässt sich sagen, dass eine Modellierung von Daily Life Domänen mit PDDL grundsätzlich möglich ist. HSP* ist dabei der zu präferierende Planer. Die Art der Modellierung ist allerdings sehr subjektiv; bereits für die relativ einfache Aktion wie das Nehmen von Medikamenten gibt es eine nahezu unendliche Anzahl von Möglichkeiten der Umsetzung in PDDL. Im Zuge der Vorüberlegungen zu dieser Arbeit wurde eine Vielzahl an Möglichkeiten, Aktionen des täglichen Lebens zu modellieren, erörtert. Einige davon wurden exemplarisch in Kapitel 4 vorgestellt, allerdings lässt sich keine grundsätzliche Empfehlung für die Art der Modellierung geben. Auf der anderen Seite resultiert die Variabilität in der Modellierung in einer flexiblen Anpassbarkeit von Planungsproblemen an unterschiedliche Gegebenheiten (andere Personen, andere Tage, Modellierung von längeren Zeiträumen, etc.).

Eine der wenigen Einschränkungen der Verwendbarkeit von PDDL-basierten Planungssystemen für Daily Life Domänen ist die Problematik, dass wenn bei der Person, für die ein Plan erstellt wird, Verzögerungen in der Ausführung des Planes entstehen (z.B. Bus verspätet sich, etc.), jeweils ein komplett neuer Plan erstellt werden muss. Das Gleiche gilt für den Fall, dass Planschritte aufgrund von Gegebenheiten in der Umwelt der Person obsolet werden, bzw. überarbeitet werden müssen. Allerdings sollten bei entsprechender Rechenleistung des Systems, auf dem der Planer ausgeführt wird, die Auswirkungen marginal sein.

Literaturverzeichnis

[Ba00] Bacchus, F.: Subset of PDDL for the AIPS2000 Planning Competition - Draft 1, 2000.

[EH04] Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. In Technical Report 195, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany, 2004.

[FL03] Fox, M.; Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. In Journal of Artificial Intelligence Research. Special issue on the 3rd International Planning Competition, 2003.

[GS02] Gerevini, A.; Serina, I.: LPG: a Planner based on Local Search for Planning Graphs. In Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02), AAAI Press, 2002.

[GSS04]Gerevini,A.; Saetti, A.; Serina, I.: Planning in PDDL2.2 Domains with LPG-TD in International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), abstract booklet of the competing planners, 2004

[Ha04] Haslum, P.: TP4'04 and hsp*a. In 4th IPC systems descriptions booklet (verfügbar auf http://ipc.icaps-conference.org), 2004.

[HG00] Haslum, P.; Geffner, H.: Admissible Heuristics for Optimal Planning. In Proc. International Conference on AI Planning and Scheduling, 2000

[HSP04]HSP* - Homepage: http://www.ida.liu.se/~pahas/hsps/#ipc04

[IPC02] Homepage IPC 2002: http://planning.cis.strath.ac.uk/competition/

[IPC04] Homepage IPC 2004: http://ipc.icaps-conference.org/

[LPG04]LPG - Homepage: http://zeus.ing.unibs.it/lpg/

[Mc98] McDermott, D. et al.: PDDL – The Planning Domain Definition Language Version 1.2. The AIPS-98 Planning Competition Comitee,1998.

[SB03] Statistisches Bundesamt Deutschland 2003 – Alterpyramiden

http://www.destatis.de/basis/d/bevoe/bevoegra2.php

[TP03] I. Tsamardinos and M. E. Pollack, "Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems," 2003