ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für
Künstliche Intelligenz

Textuelle Repräsentation von Planerklärungen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Johannes F. Gesell

Gutachter:
Prof. Dr. Susanne Biundo-Stephan

Betreuer:
Bastian Seegebarth

2012

Fassung 12. November 2012

© 2012 Johannes F. Gesell

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/de/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.
Satz: PDF-LaTeX 2ε

Zusammenfassung

Das Ziel dieser Abschlussarbeit war die Entwicklung und Implementierung eines Sprachgenerierungssystems, welches die formalen Erklärungen des *Erklärbär*-Systems in laienverständlicher englischer Sprache auszudrücken vermag. Diese Aufgabe bin ich mittels eines Systems, das sich an den *Natural Language Generation*-Systemen orientiert, angegangen. Diese Ausarbeitung widmet sich der Erläuterung der an die Implementierung gestellten Aufgaben, meinen Ansätzen an ihre Lösung und bietet einen Überblick über die notwendigen Grundlagen der Sprachgenerierung sowie der Handlungsplanung und der Planerklärung sowie eine Systemspezifikation des von mir erstellten Systems.

Inhaltsverzeichnis

1. Einleitung

Die Intelligente Handlungsplanung ist ein Teilgebiet der Künstlichen Intelligenz. Sie befasst sich mit dem Auswählen und Ordnen von als *Tasks* bezeichneten Arbeitsschritten eines Agenten innerhalb einer Planungsdomäne, mit dem Ziel, auf möglichst einfache Art einen zuvor spezifizierten Zielzustand zu etablieren, oder einen Toplevel-Task, der aus mehreren Teilschritten besteht, auszuführen. Ein mögliches Anwendungsgebiet solcher Systeme ist beispielsweise die Planung organisatorischer oder logistischer Abläufe, die selbst von Experten nur mühevoll überblickt werden können. Ein Vertreter eines solchen Systems ist das am Institut für Künstliche Intelligenz an der Universität Ulm erstellte PANDA-System.

Die durch Planungssysteme erstellten Pläne können oftmals nicht die Transparenz liefern, die ein praktischer Einsatz erfordert. Es sind viele Situationen denkbar, in denen ein Nutzer Fragen zu der Notwendigkeit oder der Reihenfolge der zu erledigenden Aufgaben hat. Dies war die Motivation zur Erstellung des Erklärbär-Systems, das für Tasks, Variablengleichungen und temporale Beziehungen zwischen Zeitpunkten formale Erklärungen erstellen kann. Diese formalen Erklärungen entsprechen Ableitungsbäumen, die für Laien unlesbar sind. Da jedoch gerade Laien am meisten von der erhöhten Transparenz profitieren würden, ist es notwendig, die formalen Erklärungen in textuelle, sprachliche oder graphische Erklärungen umzuwandeln.

Diese Arbeit stellt dazu den ersten Schritt dar. Sie untersucht die Verwendung von Natural-Language-Generation Systemen zur Lösung dieses Problems und implementiert ein solches System. Dieses System führt die definierenden Arbeitsschritte eines NLG-Systems aus und ist in der Lage, textuelle Erklärungen über die Notwendigkeit von Tasks in englischer Sprache zu erstellen. Dabei kann es, je nach Bedürfnis, unterschiedlich viel zusätzliches Wissen mit einbeziehen und ist dadurch in der Lage, in einer größeren Bandbreite von Anwendungsfällen eingesetzt und auf verschiedene Planungsdomänen spezialisiert zu werden.

Aufbau der Arbeit

Zuerst widmet sich diese Ausarbeitung in Kapitel 2 den erforderlichen Grundkenntnissen aus den Bereichen Handlungsplanung, Planerklärung und Sprachgenerierung. In Kapitel 3 gehe ich näher auf das von mir erstellte System ein, erkläre seine Funktionsweise, seine Möglichkeiten und biete eine kurze, anschauliche Demonstration der Funktionsmodi. In Kapitel 4 findet sich eine Diskussion meines Systems sowie ein kurzer Ausblick auf weitere Einsatzmöglichkeiten

oder Erweiterungen. Im Anhang finden sich zusätzlich zwei Dokumente, die von meinem System verwendet werden können, um spezialisierte Erklärungen für die Smartphone-Domäne zu erstellen.

2. Grundlagen

2.1. Formale Aspekte der Planerklärung

Die formalen Erklärungen [See11, SMSB12], die später mein System bearbeiten wird, erklären die Notwendigkeit eines Arbeitsschritts, auch *Task* genannt, innerhalb eines Plans, der durch hybrides Planen erstellt wurde. Hybrides Planen vereint [BBG+11] die Mechanismen von *Partial-Order Causal-Link* und *Hybrid Task Network*-Planern. Das bedeutet, dass die Pläne aus Tasks bestehen, zwischen denen zwei Arten von Beziehungen herrschen.

Die erste Art ist der kausale Link. Ein kausaler Link bedeutet, dass der eine Task eine notwendige Vorbedingung für den zweiten liefert. Tasks verfügen über Vor- und Nachbedingungen in Form von logischen Formeln, die ihre Effekte und Vorbedingungen beschreiben. Es könnte beispielsweise ein Literal wie `boxIsIn(`$room_1$`)` negiert in der Vorbedingung eines Tasks vorkommen. In diesem Fall darf die 'Box' nicht in 'room1' stehen, um diesen Task auszuführen. Je nachdem ob eine Relation in einem positiven oder negativen Literal auftritt, werden wir später von dem positiven oder negativen Fall sprechen. Die Argumente von Tasks und Relationen können zuerst Variablen enthalten, sind aber im endgültigen Plan stets mit Konstanten codesigniert.

Die zweite Art von Beziehungen zwischen Tasks ist die Dekomposition. Sie tritt auf, wenn ein Subtask genannter Task ein Teilschritt eines weiteren ist, welcher Supertask genannt wird und mittels einer Dekompositionsmethode expandiert wurde. Alle Pläne münden in einen Toplevel Task oder in einen Goal Task. Ein Toplevel Task ist ein Task, dessen Ausführung ursprünglich gefordert wurde, und von dem die restlichen Arbeitsschritte mittels Dekompositionsmethoden abgeleitet wurden. Ein Goal Task ist ein Task, der einen Zustand etabliert, der als Zielzustand definiert wurde.

Ist nun die Notwendigkeit eines Tasks zu erklären, so gilt es, ihre Rolle als entweder eines notwendigen Vorgängers eines Goal Tasks oder als Teil eines Toplevel Tasks zu beweisen. Dies geschieht unter der Verwendung einer Reihe von Axiomen, die im Wesentlichen auf den Relationen *Necessary*, *Causal Relation* und *Decomposition Relation* arbeiten. Das Literal Necessary($task_1$) bedeutet, dass $task_1$ notwendig für die Ausführung des Plans ist. Zu Beginn der Beweisführung sind Necessary(goalTask) und Necessary(TopTask) die einzigen vorhandenen Literale über die Relation *Necessary*. Kausale Links werden mittels *CausalRelationt(t1,*ϕ*,t2)* modelliert. Hierbei ist $t1$ der produzierende und $t2$ der konsumierende Task, ϕ ist die Bedingung des kausalen Links. Das Gegenstück zu einer Dekompositionsbeziehung ist die *Decom-*

positionRelation. $DecompositionRelation(t1, m, t2)$ drückt aus, das $t1$ ein Subtask von $t2$ ist (und m die zugehörige Expansionsmethode ist).

Um weitere Necessary-Relationen abzuleiten, wird von folgenden Axiomen Gebrauch gemacht:

C-Ch: $\forall t : Task.[Necessary(t) \leftarrow$
$\exists t' : Task; \phi : Formula.[CausalRelation(t, \phi, t') \wedge Necessary(t')]]$

D-Ch: $\forall t : Task.[Necessary(t) \leftarrow$
$\exists t' : Task; m : Method.[DecompositionRelation(t, m, t') \wedge Necessary(t')]]$

Ein Task ist also notwendig, wenn er entweder Teil eines notwendigen Supertask (D-Ch), oder notwendiger Vorgänger eines anderen notwendigen Tasks (C-Ch) ist. Damit ist es nun möglich jeden tatsächlich notwendigen Schritt auf einen Top oder Goal Task zurückzuführen. Zusätzlich werden noch folgende Axiome verwendet:

Con-CR: Um Bedingungen von kausalen Links zu vereinigen.
$\forall t, t' : Task; \phi\phi' : Formula.[CausalRelation(t, conjunction(\phi, \phi'), t') \leftrightarrow$
$CausalRelation(t, \phi, t') \wedge CausalRelation(t, \phi', t')]$

C-Prop: Zum Propagieren kausaler Beziehungen.
$\forall t, t' : Task; \phi : Formula.[CausalRelation(t, \phi, t') \leftarrow$
$\exists t'' : Task.[CausalRelation(t, \phi, t'')] \wedge$
$\exists m : Method.[DecompositionRelation(t'', m, t')]]$

P-Prop: Ebenfalls zum Propagieren kausaler Beziehungen.
$\forall t, t' : Task; \phi : Formula.[CausalRelation(t, \phi, t') \leftarrow$
$\exists t'' : Task.[CausalRelation(t'', \phi, t')] \wedge$
$\exists m : Method.[DecompositionRelation(t'', m, t)]]$

Es werden noch einige weitere Axiome verwendet, die allerdings für die textuellen Erklärungen keine Bedeutung mehr haben. Um nun die Notwendigkeit eines Tasks $t1$ zu erklären wird versucht, $Necessary(t1)$ innerhalb des Axiomsystems, das die oben genannten Axiome enthält, zu beweisen. Der hierbei entstehende Ableitungsbaum wird im folgenden als *formale Erklärung* bezeichnet.

2.2. Sprachgenerierung

Sprachgenerierung bezeichnet das automatisierte Erzeugen von Texten in menschlicher Sprache. Pattern-basierte Systeme beschränken sich darauf, einer zugrundeliegenden Liste von Informationsblöcken vorgefertigte Sätze zuzuordnen und Platzhalter durch die zu repräsentierenden Daten zu ersetzten. Diese Systeme erstellen einen einfachen, formularähnlichen, von Wiederholungen geprägten Text. Wenn erhöhte Lesbarkeit und sprachliche Vielfalt erwünscht sind, rät es sich, auf die Techniken der *Natural Language Generation-Systeme* (im folgendem NLG-Systeme genannt) zurückzugreifen. NLG-Systeme dienen als Oberbegriff für eine große Menge an Systemen, die für zahlreiche unterschiedliche Zwecke entworfen werden und verschiedenste Techniken für einzelne Arbeitsschritte verwenden. Sie haben jedoch alle ein Muster gemeinsam. Dieses Muster ist die Aufteilung des Systems in, zumeist sechs, Arbeitsschritte, die sequentiell ausgeführt am Ende einen Text produzieren, der den Texten Pattern-basierter Systeme in den Punkten Lesbarkeit und sprachlicher Vielfalt überlegen ist. Diese Arbeitsschritte werden von System zu System unterschiedlich kombiniert, sind aber dennoch meist alle vorhanden.

Content Determination

Als erster Schritt ist es offensichtlich nötig, die Informationen, die im Text später enthalten sein sollen, eindeutig zu bestimmen. Auf welche Art und Weise die Informationen aus beispielsweise einer Datenbank extrahiert werden, ist in hohem Maße domänenspezifisch und stellt im allgemeinen einen sehr aufwändig zu realisierenden Schritt dar. Er würde in unserer Anwendung hier die gesamte Herleitung der Notwendigkeit eines Planschritts und mehr umfassen. Da dies aber nur ein randlicher Teilaspekt meiner Arbeit ist, ist es nicht notwendig, auf Techniken der Content Determination näher einzugehen.
Als gemeinsamen Nenner aller möglichen Arten der Content Determination verbleibt die Tatsache, dass die zu vermittelnden Informationen als *Messages* zusammengefasst werden. Ein mögliches Beispiel für eine solche Message wäre:

message-id:	1
relation:	identity
arguments:	arg1: Zucchini
	arg2: Gemüse

Sie würde folgende Information enthalten: Eine Zucchini ist ein Gemüse.

Allerdings ist eine derartige Message nicht als fertiger Satz aufzufassen. Es ist den späteren Arbeitsschritten überlassen, die Messages in Sätze zu gruppieren, zu lexikalisieren etc.

Discourse Planning

Ein Text, der sich durch gute Lesbarkeit auszeichnet, verfügt meist über eine zugrundeliegende Struktur, die die präsentierten Informationen in eine Reihenfolge ordnet, die dem Leser als natürlich erscheint. Während des Discourse Planning wird entschieden, in welcher Reihenfolge die Messages, die aus dem Schritt der Content Determination übergeben wurden, geordnet werden. In den meisten Fällen ergibt sich die Reihenfolge aus den zugrundeliegenden Informationen. Üblicherweise ist das Ergebnis des Discourse Planning ein als *Text Tree* bezeichneter Baum, bei dem die Blätter einzelne Messages enthalten und die inneren Knoten Informationen zur Ordnung ihrer Kinder ausdrücken. Dies können beispielsweise SEQUENZ-Knoten, für einfache, gereihte Informationen oder ELABORATIONEN-Knoten, wenn eine Message ausführende Informationen zum Subjekt einer anderen enthält.

Oft ergänzen sich Content Determination und Discourse Planning, so dass sie zu einem einzigen Arbeitsschritt, Text Planning, zusammengefasst werden können.

Sentence Aggregation

Es ist möglich, jede einzelne Message in einem einzelnen Satz auszudrücken. Oft aber bietet es sich an, mehrere Messages in einem Satz zusammenzufassen, um damit den Lesefluss nicht unnötig zu unterbrechen. Die drei Sätze:
“Eine Zucchini ist ein Gemüse. Sie wird meistens gekocht verzehrt. Es gibt sie aber auch als Salat.” könnten auch zusammengefasst werden: “Eine Zucchini ist ein Gemüse. Sie wird meistens gekocht verzehrt, es gibt sie aber auch als Salat.”.
Bei der zweiten Variante erleichtert der Einsatz eines Kommas anstelle eines Satzendes den Lesefluss. Da allerdings nicht alle Informationen willkürlich in Sätze gruppiert werden können, ist darauf zu achten, dass die Informationen der einzelnen Sätze sich ergänzen und die im Discourse Planning festgelegte Ordnung nicht zerstört wird.

Lexicalization

Der Vorgang, die Informationen der Messages in Rahmensätze oder Satzfragmente einzubetten, wird als *Lexicalization* bezeichnet. Normalerweise ist es möglich, jeder Art von Message ein oder mehrere passende Rahmensätze zuzuordnen, ohne dabei spezielle Anforderungen an die Sentence Aggregation zu stellen. Betrachten wir als Beispiel folgende Message:

message-id:	1
relation:	eatingHabit
arguments:	arg1: Lukas
	arg2: gesund

Eine der möglichen Lexikalisierungen wäre: “Lukas isst gesund”. Andere, äquivalente Ausdrücke wären: “Lukas ernährt sich gesund” oder “Lukas legt Wert auf eine gesunde Ernährung”. Wenn nun im Text häufiger Messages vom Typ *eatingHabit* auftreten, erscheint es für den Leser des späteren Textes natürlicher und interessanter, wenn die ähnliche Information auf unterschiedliche Art und Weise ausgedrückt wird. Je mehr äquivalente Ausdrücke für häufig verwendete Messages verwendet werden, desto höher ist die zu erwartende sprachliche Vielfalt die mit dem Ausgabetext erzielt wird.

Referring Expression Generation

Im Schritt der Referring Expression Generation wird versucht, die in der natürlichen Sprache alltägliche Praxis nachzuahmen, Objekte, Personen etc. durch Platzhalter wie Pronomen zu referenzieren. Dabei sind eine ganze Reihe von Faktoren zu beachten, unter anderem auch grammatikalische Feinheiten, wie beispielsweise, ob das zu bezeichnende Objekt erst- oder letztgenannt, Subjekt oder Objekt oder diverses andere im Vorgängersatz war. Auch können sich einzelne Messages in ihrem Inhalt widersprechen, und Worte wie “aber” oder “nichts desto trotz” werden nötig, um den Leser nicht zu verwirren. Diese Aufgabe ist meist sehr aufwändig, kann durch die vorhergehenden Arbeitsschritte, namentlich dem der Lexicalization, jedoch vereinfacht werden.

Linguistic Realization

Linguistic Realization bezeichnet den letzten Schritt eines NLG-Systems, das Anwenden von Grammatikregeln und das Vervollständigen von Sätzen. Je nachdem, ob wärend der Lexicalization den Messages einzelne Wörter oder ganze Satzfragmente zugeordnet wurden, ist es notwendig, die einzelnen Satzfragmente zu korrekten Sätzen zu erweitern. Hierfür wird teilweise von großen Datenbanken mit Grammatikregeln und zugehörigen Ausdrücken Gebrauch gemacht. Es ist aber, wie bei der Referring Expression Generation, möglich, diese Aufgabe durch Vorarbeit zu vereinfachen.

3. Textuelle Repräsentation von Planerklärungen

Diese Kapitel befasst sich mit der Erläuterung des von mir erstellten NLG-Systems. Zuerst wird in Kapitel 3.1 die Zielsetzung präzisiert. Dann werden die einzelnen Arbeitsschritte näher erläutert, das Text Planning in Kapitel 3.2, die Sentence Aggregation in Kapitel 3.3, die Lexicalization in Kapitel 3.4 und die Referring Expression Generation und Linguistic Realization in Kapitel 3.5. Anschließend werden in Kapitel 3.6 die verschiedenen Eingabedokumente näher betrachtet und ihre Handhabung erklärt. Abschließend bietet Kapitel 3.7 eine Demonstration des Systems unter Verwendung unterschiedlicher Betriebsmodi.

3.1. Zielsetzung

Als Vorbild für die textuellen Erklärungen verwende ich folgendes Beispiel [See11]:

1.	Necessary(press_newAppointment)	C-Ch 4,5
2.	CausalRelation(press_newAppointment, ϕ , set_Name)	Basic
3.	DecompositionRelation(set_Name, *m*, configure_Appointment)	Basic
4.	CausalRelation(press_newAppointment, ϕ , configure_Appointment)	C-Prop 2,3
5.	Necessary(configure_Appointment)	C-Ch 8,9
6.	CausalRelation(set_Time, isSet_Time$_{Appointment,Date}$, press_OK)	Basic
7.	DecompositionRelation(set_Time, *m*, configure_Appointment)	Basic
8.	CausalRelation(configure_Appointment, isSet_Time$_{Appointment,Date}$, press_OK)	P-Prop 6,7
9.	Necessary(press_OK)	C-Ch 10,11
10.	CausalRelation(press_OK, created$_{Appointment}$, goalTask)	Basic
11.	Necessary(goalTask)	Basic

Für diese Ableitung wurde folgende textuelle Erklärung vorgeschlagen:

"`press_newAppointment` ist notwendig, weil dieser Task für `configure_Appointment`, dieser für `press_OK` und dieser für das Erreichen des Zielzustands notwendig ist."

Alle Erklärungen, die mit meinem System automatisch generiert werden, werden einen ähnlichen Aufbau haben.

Abgesehen davon wurden eine Reihe von optionalen, zusätzlichen Zielen vorgeschlagen:

- **Schönere Namen für Tasks einbinden**
  Ursprünglich sollten die Namen der Task-Schemas einen Task bezeichnen. Da diese allerdings einen eher technischen Klang haben, wurde eine Option gewünscht, diese Namen durch selbstdefinierte "schönere" Namen zu ersetzen.

- **Ausgewählte Relation mitverwenden**
  Zu Anfangs wurde davon ausgegangen, dass Relationen nie ein Teil der textuellen Erklärung sein müssen. Da es aber in manchen Domänen durchaus möglich sein kann, dass Relationen wichtige Informationen tragen, wurde beschlossen, eine Möglichkeit für das Einbinden von Relationen zu finden.

- **Argumente von Tasks und eventuell Relationen einbinden**
  Ähnlich wie bei den Relationen verhielt es sich bei den Argumenten von Relationen und Tasks. Da sie ebenfalls wichtige und erwähnenswerte Informationen beinhalten können, wurde eine Möglichkeit gewünscht, Argumente miteinzubeziehen. Ebenfalls sollten sie (sprich: die Konstanten) mit schöneren, selbstdefinierten Namen versehen werden können.

- **Den Satzfundus erweiterbar gestalten**
  Die Satzfragmente, die im Zuge der Lexicalization den Messages zugewiesen werden, müssen eher abstrakt und universell verwendbar aufgebaut sein. Da aber ein speziell an eine bestimmte Planungsdomäne angepasster Satzfundus vorteilhaft sein kann, wurde eine Option zum Erweitern desselben gewünscht.

- **Einen möglichst modularen, modifizier- und erweiterbaren Aufbau**
  Mein System sollte für viele verschiedene Planungsdomänen einsetzbar sein. Hierbei kann unter Umständen domänenspezifisches Wissen die Qualität der erzeugten Erklärungen verbessern. Es wurden also Möglichkeiten gewünscht, dieses Wissen miteinzubeziehen. Dennoch sollte das System auch vollkommen ohne weitere Eingaben Erklärungen erzeugen können.

- **Die für die Smartphone-Domäne notwendigen Eingabedateien erstellen**
  Da die einfachsten Erklärungen (die denen aus dem Beispiel am Anfang dieses Kapitels entsprechen) die Möglichkeiten des Systems nicht einmal im Ansatz ausschöpfen, sollten für Demonstrationszwecke und für eine sofortige Anwendbarkeit des Systems für eine oft benutze Planungsdomäne alle erforderlichen Dokumente erstellt werden. Die Wahl fiel auf die Smartphone-Domäne, die auch schon in [See11] verwendet wurde.

3.2. Text Planning

Die Arbeitsschritte der Content Determination und des Discourse Planning werden in meinem System zum Text Planning zusammengefasst. Dies liegt hauptsächlich daran, dass die Content Determination, sprich das Festlegen der zu vermittelnden Informationen, nicht direkt Aufgabe meines Systems, sondern vielmehr des Erklärbär-Systems ist. Die in der Erklärung zu verwendenden Informationen umfassen generell genau die Informationen, die in einer lückenlosen Kette aus *Causal Relations*, *Decomposition Relations* und den jeweiligen *Necessary*-Axiomen, zwischen dem zu erklärenden Planschritt und einem GoalTask oder TopTask enthalten sind. Da der Baum, den das System als Eingabe enthält, jedes Necessary-Axiom mit einem *Causal Relation PlanStep Necessity Deduction* oder einem *Decomposition Relation PlanStep Necessity Deduction* erklärt, die wiederum ein *Necessary* enthalten, ist diese Kette bei jeder formalen Erklärung gegeben. Weil alle weiteren Ableitungen für die textuelle Erklärung nur von sehr geringer Relevanz sind, können wir diese getrost ignorieren und den Text Tree, das Endergebnis des Discourse Planning, aus einer Kette von aufeinander aufbauenden Erklärungen des Vorgängerschritts bilden.

3.2.1. Aufbau des Text Trees

Der Text Tree enthält sowohl die Inhalte des späteren Textes, als auch die Reihenfolge ihres Auftretens. Deshalb besteht er nicht nur aus einfachen *Messages*, sondern bettet diese bereits in eine Struktur ein, aus der sich später sinnvolle Sätze auslesen lassen. Diese Struktur kann sich während der Sentence Aggregation-Phase zwar noch ändern, jedoch ist dies nur optional. Ein Text Tree kann auch ohne anschließende Sentence Aggregation verwendet werden. Um diesen Anforderungen gerecht zu werden, werden eine Reihe von verschiedenen Knotenarten sowie Messages mit verschiedenen Relationen verwendet.

Die verwendeten Knotentypen sind:

Sequence Node: Sequence Nodes können eine beliebige Anzahl weiterer Nodes als Nachkommen haben. Sie bezeichnen eine einfache Sequenz von Tochtersätzen bzw. Satzfragmenten und werden in zwei Arten unterteilt: COMMA und STOP. Sequence Nodes vom Typ COMMA sagen aus, dass ihre Nachkommen in einem einzigen Satz stehen, getrennt durch Kommas. STOP-Knoten drücken aus, dass jeder Nachfolger ein vollständiger und in sich selbst abgeschlossener Satz ist.

Elaboration Node: Ein Elaboration Node besitzt zwei Nachfolger, die beide vom Typ Message Node sind. Seine Bedeutung ist, dass der zweite Nachfolger Informationen beinhaltet, die zur Ausführung des ersten dienen. In meinem System drückt die Message des ersten Nachfolgers stets ein *Necessary* aus, das mit der Message des zweiten Nachfolgers erklärt wird.

Message Node: Ein Message Node hat keine Nachfolgerknoten, sondern dient als ein einfacher Container für eine Message.

Die verwendeten Messages sind von größerer Vielfalt. Im allgemeinen sind alle ihre Argumente Planschritte. Die einzige Ausnahme ist die Phi Message, die im Advanced Discourse Planning verwendet wird.

Necessary: Diese Message enthält einen Planschritt als Argument, für den die Aussage getroffen werden kann, dass er ”notwendig” im Sinne der Ableitung ist. Diese Message kommt ausschließlich als linkes Kind eines Elaboration Node vor.

Causal: Messages vom Typen *Causal* enthalten zwei Planschritte als Argumente. Ihre Bedeutung ist, dass der erste Planschritt die Voraussetzung zur Ausführung des zweiten erfüllt. Sie drücken also eine kausale Beziehung zwischen ihren Argumenten aus.

Decomp: Als Gegenstück zu den Causal Messages ist die Aussage einer Decomp Message, dass der erste ihrer Planschritte ein Subtask des zweiten ist.

Phi: Phi Messages ähneln sehr stark den Causal Messages und basieren ebenfalls auf einer kausalen Beziehung ihrer Planschritte. Zusätzlich besitzen sie aber ein weiteres Argument, die Relation der ursprünglichen *Causal Relation*.

Top: Eine Top Message wird verwendet um auszudrücken, dass ein Planschritt ein Top Level Task ist, also die Erklärung danach endet.

Goal: Ganz ähnlich wie eine Top Message drückt eine Goal Message aus, dass der Task, den sie enthält, eine Bedingung für den Zielzustand etabliert. Auch in diesem Fall kann die Erklärung danach enden.

3.2.2. Varianten des Text Plannings

Das System bietet zwei Varianten des Text Plannings an, das Simple und das Advanced Text Planning. Das Advanced Text Planning stellt hierbei ein neues Feature zur Verfügung, funktioniert aber im Wesentlichen fast identisch zum Simple Text Planning.

Simple Text Planning

Die Ausführung des Simple Text Planning erfolgt, indem der Ableitungsbaum, der eingelesen wurde, abgearbeitet wird. Zuerst wird ein Sequence Node vom STOP-Typ als Wurzel des Text Trees initialisiert. Danach wird, beginnend mit dem *Necessary*, das als Wurzel des Eingabebaumes fungiert, für jedes *Necessary* eine Reihe von Nodes und Messages, deren Bestimmung im nächsten Abschnitt beschrieben wird, in die Wurzel des Text Trees eingehängt. Danach wird mit dem erklärendem *Necessary* fortgefahren, bis das Ende des Ableitungsbaumes erreicht wurde.

Welche Messages verwendet werden, wird dadurch bestimmt, wie das aktuelle *Necessary* erklärt wurde. Die Struktur der Nodes ist aber stets die selbe:
Für jedes Necessary wird ein Elaboration Node mit zwei Message Nodes verwendet. Die linke Message ist stets vom Typ Necessary und enthält den Planschritt des aktuellen *Necessarys*. Für die rechte Message kann jede andere Art von Message stehen. Wenn der aktuelle Planschritt mit einer *Decomposition Relation PlanStep Necessity Deduction* abgeleitet wurde, wird eine Decomp Message mit den beiden entsprechenden Planschritten erstellt, im Falle einer *Causal Relation PlanStep Necessity Deduction* eine Causal Message. Sollte der zu erklärende Planschritt ein Top oder Goal Task sein, so wird selbstverständlich eine entsprechende Message verwendet. Ausschließlich für das letzte *Necessary* werden gar keine Messages oder Nodes gebildet. Sie enthält stets die Information, dass ein Goal Task oder der erwünschte Top Task notwendig ist. Da sich diese Information aus dem Kontext ergibt, müssen wir sie nicht gesondert erwähnen.

Advanced Text Planning

Als Alternative zum Simple Text Planning ist auch eine erweiterte Variante bereits implementiert. Dieses Advanced Text Planning macht zusätzlich Gebrauch von den Phi Messages. Grund für die Unterscheidung zwischen *Causal Message* und *Phi Message* ist, dass die Relationen, die in CausalRelations zum Einsatz kommen, für das Verständnis des Vorgangs von sehr unterschiedlicher Wichtigkeit sind. Beispielhaft ist diese Anwendung einer CausalRelation:

CausalRelation(leereEimerAus, eimerIstLeer, putzeEimer)

Hier wäre eine Erklärung wie: “Den Eimer auszuleeren ist notwendig für das Putzen des Eimers” angemessen. Dass die Voraussetzung für putzeEimer gerade eimerIstLeer ist, ist nicht weiter wichtig zu erwähnen, da es sich aus dem Kontext ergibt. Ein Gegenbeispiel hierfür wäre:

CausalRelation(pressButton294, MachineIsRunning, loadCAD)

Hier ergibt sich die Tatsache, dass die Maschine nun läuft nicht daraus, dass Knopf 294 gedrückt wurde, denn dazu ist dessen Beschreibung nicht ausreichend. Da die Wichtigkeit der Vorbedingung für eine schlüssige Erklärung nicht von meinem System entschieden werden kann, da sie abhängig von der Anwendungsdomäne und abhängig vom Vorwissen des Lesers ist, ist es dem Anwender des Systems überlassen, für jede Relation die Wichtigkeit selbst zu beurteilen. Sollte er sie für hoch genug erachten, so kann er diese Relation in einer Eingabedatei vermerken und ein zu verwendendes Satzfragment für den positiven wie für den negativen Fall angeben. Sie wird dann in zukünftig generierten Erklärungen mit eingebunden, sofern das Advanced Text Planning verwendet wird. Welche Form ein solcher Eintrag haben muss, ist in Kapitel 3.5 genauer beschrieben.

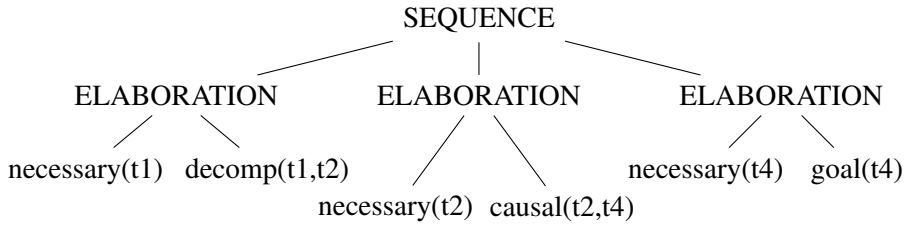
Generell läuft das Advanced Text Planning genauso wie das Simple Text Planning ab. Der ein-

zige Unterschied besteht im Bearbeiten von *Causal Relation PlanStep Necessity Deductions*. In diesem Fall wird geprüft, ob die in der Ableitung verwendete Relation in der Eingabedatei verzeichnet ist. Sollte dies nicht der Fall sein, so wird wie gewohnt eine Causal Message verwendet, falls doch, so wird anstatt der Causal Message eine Phi Message erstellt, die zusätzlich noch die Relation als Argument enthält.

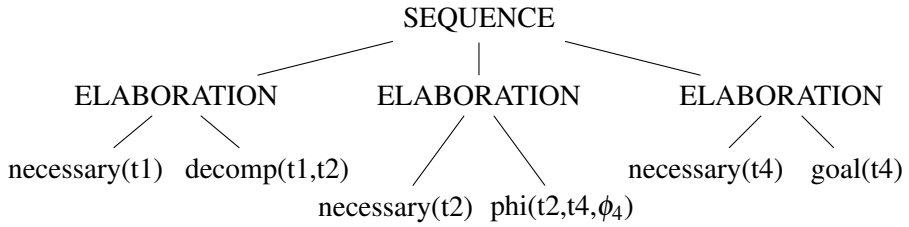
Beispiel Diese formale Erklärung dient uns als Beispiel:

1.	Necessary(t1)	D-Ch 2,3
2.	DecompositionRelation(t1, m , t2)	Basic
3.	Necessary(t2)	C-Ch 6,7
4.	CausalRelation(t3, ϕ_3, t4)	Basic
5.	DecompositionRelation(t3, m, t2)	Basic
6.	CausalRelation(t2, ϕ_4, t4)	P-Prop 4,5
7.	Necessary(t4)	C-Ch 8,9
8.	CausalRelation(t4, ϕ_5, goalTask)	Basic
9.	Necessary(goalTask)	Basic

Wenn keine Relationseinträge vorhanden sind oder das Simple Text Planning verwendet wird, würde folgender Text Tree erzeugt werden.



Angenommen für ϕ_4 wäre ein Eintrag vorhanden. Das Advanced Text Planning würde dann folgenden Baum erzeugen:



3.3. Sentence Aggregation

Durch die Sentence Aggregation wird der aus dem Text Planning stammende Text Tree modifiziert, was einen Sentence Tree zum Ergebnis hat. Dies dient dazu, die Lesbarkeit des Textes zu erhöhen. In meinem System sind die Necessary Messages teilweise redundant, schließlich sind sie stets Schlussfolgerungen ihrer erklärenden, nachfolgenden Message. Dies hat zu Konsequenz, dass sich manche vollständigen Elaboration Nodes auf ihre "rechte Seite", also die erklärende Message in einem Message Node, reduzieren lassen. Dadurch wird die Grenze zwischen Text Planning und Sentence Aggregation zwar leicht verwischt, allerdings auch die Lesbarkeit und Konsistenz des Ergebnistextes drastisch erhöht. Der Arbeitsschritt der Sentence Aggregation bleibt dennoch optional und mein System stellt zusätzlich zu zwei regulären Sentence Aggregation-Varianten eine dritte bereit, die den Text Tree unverändert als Sentence Tree weiterreicht.
Der Sentence Tree hat einen sehr ähnlichen Aufbau wie der Text Tree und verwendet auch dieselben Nodes und Messages. Einzig die COMMA-Variante des Sequence Nodes ist ausschließlich im Sentence Tree zu finden.

3.3.1. Schema der Sentence Aggregation-Varianten

Alle echten Varianten der Sentence Aggregation machen es sich zu Nutze, dass die aufeinanderfolgenden Elaboration Nodes beliebig vereint werden können. In seiner ursprünglichen Form beinhaltet ein Elaboration Node einen Satz von folgendem Aufbau:

	1.Teil	2.Teil	3.Teil
Herkunft:	linke Message	Elaboration Node	rechte Message
Inhalt:	Task1 ist notwendig	denn	Task1 ist Subtask von Task2

Folgen drei solche Sätze aufeinander, können sie vereinigt werden. Das würde folgenden Satz ergeben:

"Task1 ist notwendig, denn Task1 ist ein Subtask von Task2, Task2 ist notwendig, denn Task2 ist ein Subtask von Task3 und Task3 ist notwendig, da Task3 ein Subtask von Task4 ist."

Die häufige Verwendung der Necessary Message stört den Lesefluss. Deshalb wird der mittlere Teilsatz verkürzt. Dadurch ist der Satz besser lesbar:

"Task1 ist notwendig, denn Task1 ist ein Subtask von Task2, Task2 ist ein Subtask von Task3 und Task3 ist notwendig, da er ein Subtask von Task4 ist."

Wenn drei solcher Sätze aufeinanderfolgen, ist also zumindest im mittlerem der drei die linke Seite des Elaboration Nodes und das 'denn' für das Verständnis und die Korrektheit überflüssig und der Satz kann auf die rechte Seite reduziert werden. Dies funktioniert natürlich auch

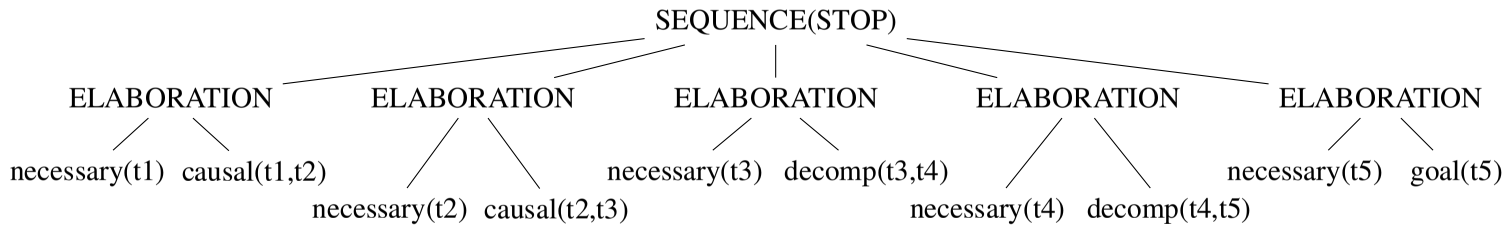
für längere Ketten von Sätzen, allerdings kommen zu lange Sätze dem Lesefluss auch nicht zugute. Es ist also notwendig, die Sequenz von Elaboration Nodes in angemessen lange Untersequenzen zu unterteilen. Meine Implementierung stellt hierfür zwei Möglichkeiten bereit: Die Content-based Sentence Aggregation, die ähnliche Erklärungen gruppiert und die Length-based Sentence Aggregation, die zufallsgenerierte Satzlängen verwendet.

3.3.2. Length-based und Content-based Sentence Aggregation

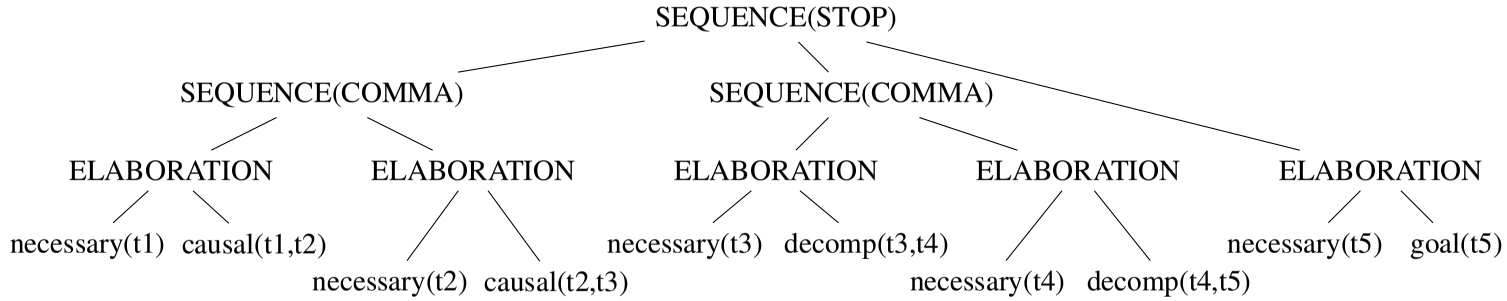
Beide Varianten der Sentence Aggregation sind im Ablauf sehr ähnlich. Es werden die Nachfolger der Wurzel linear durchlaufen und in einen Buffer eingefügt. Sobald das jeweilige Kriterium erreicht ist, wird der Buffer gelehrt und alle beinhalteten Nodes zu einem Satz zusammengefasst. Dazu wird ein neuer Sequence Node vom Typ COMMA erstellt und alle Elaboration Nodes werden eingehängt. Während der erste und letzte in seinem ursprünglichen Zustand belassen wird, werden alle dazwischenliegenden auf ihren rechten Message Node reduziert. Ausnahmen davon sind eventuelle Elaboration Nodes, die eine Phi Message beinhalten. Dies ist nötig, weil sonst keine korrekten Sätze mehr gebildet werden können, da in diesem Fall ein spezielles Satzfragment aus den Eingabedaten verwendet wird. Anschließend wird der Buffer nach dem gleichen Muster neu gefüllt.
Die Abbruchkriterien, die zu einem Flush des Buffers führen, sind bei der Content-based Sentence Aggregation das Auftreten eines Elaboration Nodes, der als rechte Message einen anderen Typ besitzt als der oder die bisherigen Elaboration Nodes im Buffer oder das Ende des Textes. Es werden also Sätze gruppiert, die die gleiche Art der Ableitung beinhalten. Die Abbruchkriterien für die Length-based Sentence Aggregation sind das Erreichen einer maximalen, zuvor per Zufall bestimmten Satzlänge oder ebenfalls das Ende des Textes. Ist der Abbruch aufgrund der maximalen Satzlänge erfolgt, so wird für den neuen Satz eine neue maximale Satzlänge festgelegt. Alle Satzlängen liegen gleichverteilt zwischen drei und fünf. In der Praxis zeigte sich, dass die Length-based Sentence Aggregation im allgemeinen eine viel einfacher lesbare Erklärung produzierte als die Content-based Variante, diese jedoch bei besonders langen Erklärungen mit langen Folgen ähnlicher Ableitungen wiederum vorteilhafter war.

Beispiel Auf der folgenden Seite findet sich ein Beispiel eines Text Trees, der einmal mit der Content-based und einmal mit der Length-based Sentence Aggregation bearbeitet wurde. Bei der Length-based Variante gehen wir davon aus, dass zuerst eine maximale Satzlänge von drei und einmal von fünf gewählt wurde.

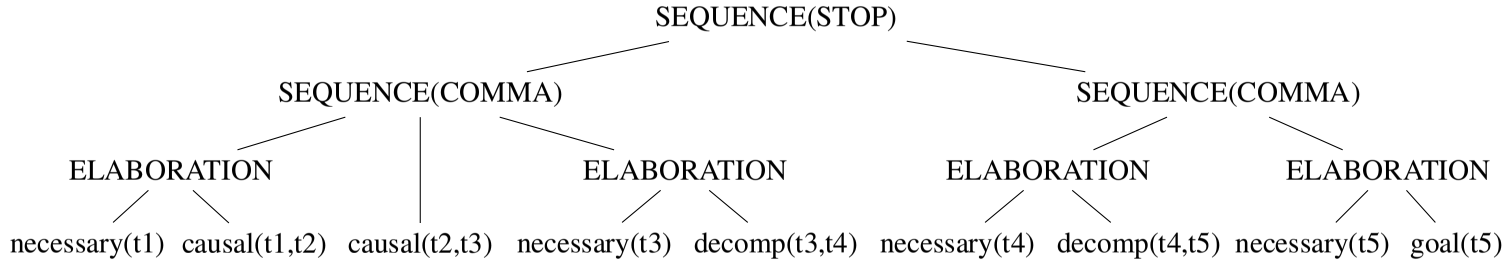
Beispiel Text Tree:



Sentence Tree nach Content-based Sentence Aggregation:



Sentence Tree nach Length-based Sentence Aggregation:



3.4. Lexicalization

Die Arbeitsschritte der Lexicalization, Referring Expression Generation und Linguisitic Realization wurden in meinem System teilweise verzahnt implementiert. Dies geschah, um in den beiden ersten Schritten möglichst viel Vorarbeit für die Linguistic Realization zu leisten. Diese ist ein sehr aufwändiger Prozess, der von einem System mit spezialisierten Anwendungsaufgaben wie dem meinen nicht gerechtfertigt wird. Der von mir verfolgte Ansatz, bei der Lexicalization vorgefertigte Satzfragmente anstatt einzelner Worte auszuwählen, sollte gewährleisten, dass nach der Lexicalization bereits lesbare, grammatikalisch korrekte Sätze stehen, die von den nachfolgenden Schritten nur noch ergänzt werden.

3.4.1. Aufgabe der Lexicalization

Der nach der Sentence Aggregation anstehende Arbeitsschritt, die Lexicalization, weist den einzelnen Messages des Sentence Trees ein Satzfragment zu, mit dem sie später im Ausgabetext repräsentiert werden. Jedes dieser Satzfragmente ist an sich schon ein vollständiger und korrekter Satz, sie werden aber in den allermeisten Fällen gemäß des in der Sentence Aggregation festgelegten Musters zusammengefügt. Um korrekte Sätze zu erhalten, ist es daher notwendig, dass zwischen den rohen Satzfragmenten der Messages kleinere Strings eingefügt werden. Neben den Satzzeichen, die von den verschiedenen Sequence Nodes eingefügt werden, werden auch die Wörter: ‘because’, ‘since’, oder ‘for’ verwendet, da sie den Zusammenhang ausdrücken, der zwischen dem linken und rechten Nachkommen eines Elaboration Nodes besteht. Eine Lexicalization eines Sentence Trees umfasst also nicht nur das Zuweisen von Satzfragmenten an Messages, sondern es müssen auch für Sequence und Elaboration Nodes Textfragmente bestimmt werden. Da letzteres aber unabhängig von den Messages in den Nachkommen getan werden kann, wurde dieser Vorgang aus den wählbaren Lexicalization-Varianten ausgeschlossen, so dass diese ausschließlich Messages behandeln müssen. Die Lexicalization von Nodes wird erst bei dem endgültigen Auflösen der Baumstruktur und der Umwandlung in einen einzigen String vorgenommen. Ähnlich wie die Messages wählen auch hier die Elaboration Nodes für sich aus einer Reihe äquivalenter Ausdrücke.

Allen Varianten der Lexicalization ist gemein, dass sie aus einer Liste äquivalenter Ausdrücke für jede Art von Message einen zufällig auswählen und den Messages zuweisen. Diese Prototypen von Satzfragmenten enthalten mehrere Strings, die den eigentlichen Satz beinhalten und von Platzhaltern für die Ausdrücke der Tasks, die später im fertigen Text verwendet werden, unterbrochen werden. Bei der Zuweisung eines Prototypen zu einer Message werden die Argumente der Message, die mit der Ausnahme von Phi Messages stets Tasks sind, ausgelesen und den Platzhaltern zugewiesen. Diese Platzhalter spielen bei der Referring Expression Generation eine große Rolle, werden aber auch schon bei der Lexicalization für verschiedene Ausdrücke von Tasks gebraucht. Auch enthält jeder Satzprototyp bereits eine Reihe von Daten, die später bei der Referring Expression Generation sehr große Bedeutung haben werden, für die Lexicalization selbst aber nicht weiter wichtig sind.

3.4.2. Die verschiedenen Arten der Lexicalization

Ich habe für mein System bereits drei verschiedene Varianten der Lexicalization implementiert. Sie unterscheiden sich teils recht deutlich im Umfang der zusätzlich nötigen Eingabeinformationen und in der Qualität der erzeugten Sätze. Generell kann gesagt werden, dass je mehr Input eine Lexicalization verlangt, umso ausgefeiltere, interessantere und vielfältigere Sätze kann sie erzeugen. Durch die drei Varianten habe ich die beiden Extreme von sehr wenig Input und eher einfachen Sätzen, von sehr viel Input mit sehr ausgefeilten Sätzen (und großer Erweiterbarkeit und Modifizierbarkeit) sowie einen Kompromiss der beiden zur Verfügung gestellt.

Simple Lexicalization

Die einfachste Variante der Lexicalization besitzt den Vorteil, dass sie keinerlei weitere Eingabedokumente verlangt und damit auch ohne Vorarbeit erste, wenn auch eher einfache, Erklärungen generieren kann. Sie behandelt auch eventuell auftretende Phi Messages wie deren allgemeineren Fall, die Causal Messages, und ist daher mit allen Varianten des Text Planning und der Sentence Aggregation kompatibel. Auch bei der Lexicalization der Namen der Planschritte werden keine weiteren Eingaben beansprucht. Es wird stattdessen der Name des Task-Schemas mit dem Präfix ‘the task’ verwendet. Das Präfix ist nötig, da in manchen Fällen ein Zusatz zu dem eigentlichen Task-Namen nötig ist, um einen korrekten Satz zu bilden. Dieser kann nicht Teil der Satzfragmente des Prototypen sein, da er zusammen mit dem Task-Namen entfernt werden muss, wenn der Task durch ein Pronomen bezeichnet wird (Pronomen werden im Rahmen der Referring Expression Generation vergeben, beschrieben in Kapitel 3.4.3).

Die Simple Lexicalization bietet für die Message Typen Causal, Decomp, Necessary, Goal und Top jeweils eine Reihe äquivalenter Ausdrücke:

Message Type	Satzfragment-Prototyp (Beispiel)
Necessary	<Platzhalter für 1. Argument> is necessary
Causal	<Platzhalter für 1. Argument> is needed for <Platzhalter für 2. Argument>
Phi	*wie Causal*
Decomp	<Platzhalter für 1. Argument> is a substep of <Platzhalter für 2. Argument>
Goal	<Platzhalter für 1. Argument> establishes a goal
Top	<Platzhalter für 1. Argument> is a top task

Ein Beispiel für einen kompletten Elaboration Node verwenden wir:

Elaboration Node			
Message		Message	
relation:	necessary	relation:	decomp
arg1:	t1 (emptyBucket)	arg1:	t1 (emptyBucket)
		arg2:	t2 (cleaningBucket)

Um diesen Elaboration Node zu lexikalisieren, wird nun für jede der Messages und für den Elaboration Node aus der Menge der passenden äquivalenten Ausdrücke einer zufällig gewählt und mit den Argumenten gefüllt. Anschließend kann man den Satz einfach zusammenfügen:

"The task 'emptyBucket' is necessary, since the task 'emptyBucket' is part of the task 'cleaningBucket'."

Advanced Lexicalization

Die Advanced Lexicalization ist als eine Erweiterung der Simple Lexicalization zu verstehen. Es werden nach wie vor nur Sätze verwendet, die im System hinterlegt sind. Allerdings macht die Advanced Lexicalization Gebrauch von detaillierteren und treffenderen Ausdrücken für die Planschritte. Dafür ist eine Eingabedatei nötig, die für jeden verwendeten Planschritt drei weitere Strings enthält: eine Umschreibung des Planschrittes als Namen, in Infinitivform und in Verlaufsform. Jeder Satzprototyp enthält Informationen darüber, an welcher Stelle welche Form des Namens und mit welchem Präfix verwendet werden kann. Es sind bei dieser Art der Lexicalization nun für unterschiedliche Formen der Ausdrücke für Tasks auch unterschiedliche Präfixe gefordert und sie können nicht mehr automatisch vergeben werden wie bei der Simple Lexicalization.

Da die meisten Sätze verschiedene Formen an derselben Stelle verwenden können, ist die Anzahl möglicher Lexicalizations gegenüber der Simple Lexicalization potenziert, was die Vielfalt der Ausgabetexte beträchtlich erhöht.

Von noch größerer Bedeutung ist, dass nun die Argumente der Tasks mit in die Erklärung eingebunden werden können. Dazu muss nur in der Eingabedatei für die verschiedenen Task-Namen ein Platzhalter mit dem jeweiligen Index des Arguments eingefügt werden. Bei der Zuweisung eines Satzprototypen werden nach wie vor die Argumente der Message aufgelöst. Sollte ein Argument für einen Task-Namen gefordert sein, wird zuerst die entsprechende Variable aufgelöst und durch die Konstante ersetzt. Sollte der Name der Konstanten nicht wie erwünscht formatiert sein, kann man in einem weiteren Eingabefile diese Konstante vermerken und ihr einen passenderen Namen zuweisen.

Darüber hinaus können nun auch Phi Messages verwendet werden. Neben dem Eintrag in der

für Relationen zuständigen Eingabedatei werden auch zwei Varianten der Beschreibung der Relation angeboten. Diese können, genauso wie die Task-Namen, Platzhalter für ihre Argumente enthalten, und diese werden auch auf die gleiche Weise aufgelöst. Eine Phi Message wird lexikalisiert, indem die Beschreibung der Relation in einen passenden Rahmensatz eingefügt wird.
Die erforderliche Form der Eingabedokumente ist in Kapitel 3.5 näher ausgeführt.
Im Bezug auf die Zuweisung der Satzprototypen arbeitet sie auf die gleiche Weise wie die Simple Lexicalization.

Für eine Lexicalization der Beispiel-Message aus dem vorherigem Kapitel wären folgende zusätzlichen Informationen aus Eingabedokumenten nötig:

Beschreibungen der Tasks:

Task	Name	Infinitiv	Verlaufsform
emptyBucket	empty Bucket No. %1%	empty the bucket No. %1%	emptying the bucket No. %1%
cleanBucket	clean Bucket No. %1%	clean the bucket No. %1%	cleaning the bucket No. %1%

(% Zeichen markieren Anfang und Ende eines Argumentenindex)

Alternative Konstantennamen (optional):

Ursprünglicher Name	zugewiesener Name
bucket#12	12

Es werden wieder aus der Menge der passenden äquivalenten Ausdrücke für jede Message einer zufällig ausgewählt:

Message	Satzfragment-Prototyp
Necessary	<Platzhalter:1;continuous;””> is necessary
Decomp	<Platzhalter:1;infinitive;”to”> is part of <Platzhalter:2;continuous;””>

(Für die Platzhalter gilt: <Platzhalter: Nummer des Arguments; Form; Präfix>)

Nun können die Variablen der Task-Argumente aufgelöst und alle Satzteile zusammengeführt werden:

“Emptying the Bucket No. 12 is necessary, because to empty the Bucket No. 12 is part of cleaning the Bucket No. 12.”

Bei der Lexicalization von Phi Messages wird, je nachdem ob die verlangte Relation positiv oder negativ gefordert wird, der entsprechende Ausdruck der Relation verwendet. Der endgültige Satz besteht aus eben jenem Ausdruck und dem zweiten Task (in allen drei Formen möglich), getrennt durch ein ‘for’.

Angenommen die Relation, die in dem kausalen Link zwischen emptyBucket und cleanBucket

verwendet wird, würde den Name ‘isEmpty’ tragen, als Argument ebenfalls die Nummer des Eimers verwenden und folgender Eintrag wäre aus der Eingabedatei eingelesen worden:

Relationsname	true-Variante	false-Variante
isEmpty	you need the Bucket to be empty	the Bucket No. %1% must be filled

Dann würde anstatt der Causal Message eine Phi Message in der rechten Seite des Elaboration Nodes stehen, die auf folgende Weise lexikalisiert wreden würde:

“Emptying the Bucket No. 12 is necessary, because you need the Bucket to be empty for cleaning the Bucket No. 12.”

Input Supporting Lexicalization

Die Input Supporting Lexicalization trägt diesen Namen, weil sie noch mehr zusätzliche Eingaben verarbeiten kann als die Advanced Lexicalization. Während auch bei der Advanced Lexicalization die Rahmensätze vorgegeben waren, liest die Input Supporting Lexicalization diese aus einer Eingabedatei ein. Auf diese Art kann die sprachliche Vielfalt des Ausgabetextes weiter erhöht oder es können domänen- oder anwendungsfallspezifischere Rahmensätze erstellt werden. Ihre Arbeitsweise, was das Zuordnen der Sätze zu den Messages und das Füllen der Prototypen mit detaillierteren Informationen betrifft, ist identisch zur Advanced Lexicalization und sie verfügt somit über alle deren Features. Mit Hilfe der Input Supporting Lexicalization ist es nun möglich, nahezu alle Teile des Ausgabetextes zu modifizieren oder zu ergänzen, ohne Änderungen am tatsächlichen Code vornehmen zu müssen, womit ein Maximum der gewollten Erweiterbarkeit des Systems erreicht wird. Wenn die Informationen der neuen Satzprototypen bezüglich der Referring Expression Generation und der zu verwendenden Task-Namen vollständig und korrekt sind, kann auch eine selbst geschriebene Lexicalization mit jeder Variante von Text Planning und Sentence Aggregation verwendet werden.

Die Form, welche die Input Supporting Lexicalization in den Eingabedokumenten verlangt, ist in Kapitel 3.5 näher beschrieben.

3.5. Referring Expression Generation und Linguistic Realization

Zu dem Zeitpunkt der Systemausführung an dem die Referring Expression Generation beginnt, ist der bisher erstellte Text bereits gut lesbar und (bis auf die Groß-, Kleinschreibung) grammatikalisch korrekt. Die nun folgenden Arbeitsschritte dienen dazu, den Text soweit weiter zu bearbeiten, dass er möglichst natürlich erscheint.

3.5.1. Referring Expression Generation

Bisher wurden die Task-Namen recht häufig verwendet. Dies ist nicht nur redundant, es stört auch den Lesefluss und erscheint dem Leser als künstlich. Der Arbeitsschritt der Referring Expression Generation dient dazu, unnötige Namensnennungen durch die Verwendung von Pronomen zu reduzieren. Das System verwendet hierfür die Pronomen ‘that’, ‘it’ und ‘which’. Die grammatikalischen Regeln für Pronomen sind zum Teil recht komplex und erfordern zusätzliches Wissen, das meinem System nicht zur Verfügung steht. Auch muss für ihre Vergabe meist mehr als nur ein einziger Satz betrachtet werden. Die Referring Expression Generation verwendet Informationen, die von den Sätzen angeboten werden, sowie Informationen über die Position eines Satzfragmentes innerhalb des Satzes (sprich: ob das Satzfragment eventuell den Satzanfang bildet oder nicht), um den Gebrauch der drei Pronomen im natürlichen Sprachgebrauch zu imitieren.

Im Folgendem dient uns dieser Satz als Beispiel:

“Pressing the [new eMail]-button is needed, because to press the [new eMail]-button is a substep of sending an eMail to Dave, to send Dave an eMail is part of the task ’contact Dave’ and to contact Dave is crucial, since contacting Dave establishes a goal.”

Dieser Satz würde aus einer lexikalisierten Sequenz vom COMMA-Typ bestehen, die eine Elaboration Node, dann eine Message Node und dann wiederum eine Elaboration Node enthält. Dieser Satz hätte von meinem System ohne Referring Expression Generation in genau dieser Form erstellt werden können und bietet ein gutes Beispiel dafür, wie sehr die wiederholte Nennung von ‘contacting Dave’ trotz der unterschiedlichen Formulierungen des Tasks als sehr unnatürlich empfunden wird.

Mein Ansatz zur Referring Expression Generation geht nach folgendem Muster vor:

Zuerst werden alle Messages samt ihren Satzteilen in eine lineare Liste überführt (die Baumstruktur wird aber noch nicht verworfen, die Referring Expression Generation arbeitet in place). Jeder dieser Sätze stellt Informationen darüber zur Verfügung, welcher seiner Tasks als ‘it’, ‘which’ oder ‘that’ im darauf folgendem Satz verwendet werden darf, und Informationen darüber, welchen Task er als ‘it’, ‘which’ oder ‘that’ selbst akzeptieren würde. Diese Informationen wurden bei der Erstellung des Satzprototypen festgelegt, bzw. in der Eingabedatei der

Input Supporting Lexicalization angegeben. Auf diese Weise ist bereits gesichert, dass niemals ein Pronomen an einer Stelle im Satz steht, an der es nicht zum Aufbau des restlichen Satzes passen würde.

In unserem Beispiel würden die Sätze folgende Tasks zur Verwendung anbieten:

Satz	it	which	that
Pressing the [new eMail] button is needed	0	-	-
to press the [new eMail] button is a substep of sending an eMail to Dave	-	1	1
to send Dave an eMail is part of the task 'contact Dave'	-	1	1
to contact Dave is crucial	0	-	-
contacting Dave establishes a goal	-	-	-

Der Eintrag der Zeilen 'it', 'which' und 'that' ist jeweils der Index des Tasks in der Argumentenliste der zugrundeliegenden Message. So bedeutet die 0 in der Spalte 'it' der ersten Zeile, dass der nachfolgende Satz sich durch Verwendung des Pronomens 'it' auf das Argument mit Index 0 des aktuellen Satzes beziehen kann.

Folgende Tasks würde sie akzeptieren:

Satz	it	which	that
Pressing the [new eMail] button is needed	-	0	0
to press the [new eMail] button is a substep of sending an eMail to Dave	0	0	-
to send Dave an eMail is part of the task 'contact Dave'	0	0	-
to contact Dave is crucial	-	0	0
contacting Dave establishes a goal	0	-	-

In diesem Fall wird zwar stets derselbe Task für die meisten Pronomen angeboten, allerdings kann dies nicht generalisiert werden, da exotischere Sätze, die eventuell durch die Input Supporting Lexicalization zu dem System hinzugefügt werden, dies nicht tun müssen. Der Satz der Goal Message bietet keine Tasks an, da er stets am Ende der Erklärung steht und deswegen keinen Folgesatz hat. Die Sätze der Decomp Messages wiederum bieten kein 'it' an, weil auf sie stets ein Satzende, ein Necessary oder eine weitere Causal oder Decomp Message folgt, was eine Verwendung von 'it' ausschließt. Die Teilsätze, die aus Necessary Messages gebildet wurden, akzeptieren ebenfalls kein 'it', da sie stets am Satzanfang stehen oder auf eine Causal oder Decomp Message folgen; an dieser Stelle wäre ein 'it' nicht angebracht. Ähnlich verhält es sich bei den Decomp Sätzen, die kein 'that' akzeptieren.

Bevor die Pronomen zugewiesen werden, muss beachtet werden, dass 'and' oder ein Satzende keine Verwendung von 'it' oder 'which' im darauf folgenden Teilsatz zulassen, da sie die Satzteile zu stark voneinander trennen. Deshalb wird der Sentence Tree durchlaufen und jene Sätze, die als erstes in einem Sequence Node vom STOP-Typ, also am Satzanfang, oder als letztes in einem Sequence Node vom COMMA-Typ stehen, bzw. im linken Blatt des entsprechenden

Teilbaums des Sentence Tree stehen, markiert und das jeweilige Feld, das die Akzeptanz von ‘it’ oder ‘which’ bestimmt, resetted.

Anschließend ist nichts weiter zu tun, als die Liste von Sätzen zu durchlaufen und jeweils die als Pronomen angebotenen Tasks des Vorgängersatzes mit den Tasks des aktuellen Satzes zu vergleichen. Im Falle einer Übereinstimmung wird im aktuellen Satz der Ausdruck des Tasks mit seinem Prefix, falls es einen gibt, durch das passende Pronomen ersetzt. Wenn mehrere Pronomen verwendet werden können, wird ‘that’ vor ‘which’ und ‘which’ vor ‘it’ bevorzugt. Der Beispielsatz würde danach wie folgt aussehen:

“Pressing the [new eMail]-button is needed, because it is a substep of sending an eMail to Dave, which is part of the task ’contact Dave’ and that is crucial, since it establishes a goal.”

In diesem Satz wird jeder Task nur einmal tatsächlich genannt und in allen anderen Fällen stets mit einem passendem Pronomen bezeichnet. Auch ist er kürzer und kompakter, was der Lesbarkeit zugute kommt.

3.5.2. Nachbearbeitung des Textes

Am Ende eines Natural Language Generation-Systems steht normalerweise die Linguistic Realization, also das Anpassen des Textes an Grammatikregeln. Durch die Vorarbeit, die in meinem System von den vorhergehenden Arbeitsschritten geleistet wurde, und durch die Verwendung von Satzfragmenten anstatt eines kennzeichnenden Wortes bei der Lexicalization reduziert sich der Arbeitsschritt hier nahezu ausschließlich auf das Auswerten des Baumes und das Zusammenfügen der Strings. Einzig die Groß- und Kleinschreibung am Satzanfang wurde noch nicht beachtet. Die endgültigen, vollständigen Sätze, bei denen der erste Buchstabe großgeschrieben werden muss, entsprechen jeweils einem Teilbaum, dessen Wurzel ein Nachfolger eines Sequence Nodes vom STOP-Typ ist. Da die Wurzel des gesamten Baumes der einzige dieser Sequence Nodes ist, muss nur jeweils der erste Buchstabe des am weitesten links stehenden Blatts seiner Tochterbäume ersetzt werden.

Ein weiteres Detail dieses Arbeitsschritts ist das korrekte Einfügen von Leerzeichen. An jeder Stelle, an der zwei Teilstrings zusammengefügt werden, muss ein Leerzeichen eingefügt werden. Ausgenommen davon sind jene Nahtstellen, auf die ein Satzzeichen folgt. Dies entspricht allerdings nicht den Nahtstellen von Teilbäumen von Sequence Nodes. Zwar sind diese die einzigen, die Satzzeichen an Nahtstellen einfügen, allerdings werden diese teilweise durch ‘and’ ersetzt und ein Leerzeichen ist nötig.

Das Auflösen der Baumstruktur geschieht gemeinsam mit dem Verbinden der Strings. Jeder Node-Typ besitzt eine Methode, die, unter Verwendung eben jener Methode seiner Nachkommen, einen String generiert und zurückgibt. Sequence Nodes vom Typ STOP setzen Punkte zwischen die Strings der Nachkommen, Sequence Nodes vom Typ COMMA setzen Kommas

bzw. ein ‘and’ zwischen den letzten und vorletzten Nachkommen. Elaboration Nodes wählen aus einer Liste äquivalenter Ausdrücke wie because, since, etc. und setzen diesen Ausdruck (und ein Komma) zwischen den Strings ihrer Nachkommen. Message Nodes liefern den Ausdruck ihrer Messages zurück.

3.6. Erläuterungen zu den Eingabedokumenten

Das System verwendet neben der ursprünglichen formalen Erklärung drei weitere Eingabedokumente, die teils mehreren Zwecken dienen und nur bei einzelnen Betriebsmodi verlangt werden. Dieses Kapitel dient dazu, ihre Handhabung zu erläutern.

Die Dreiteilung der Eingabedokumente dient dazu, den Aufwand bei einer Umstellung des Systems auf eine andere Domäne oder einen anderen Anwendungsfall zu verringern, indem Daten, die unverändert weiterverwendet werden können, von jenen isoliert werden, die ausgetauscht werden müssen. Das erste Dokument enthält die Namenszuweisungen für die Konstanten, welche als anwendungsspezifische Daten am häufigsten ausgetauscht werden müssen. Das zweite enthält alle domänenspezifischen Daten, die Ausdrücke für Tasks und Relationen. Das dritte enthält die für die Input Supporting Lexicalization nötigen Satzfragmente und muss nur angepasst werden, wenn die normale Advanced Lexicalization unzureichende Ergebnisse liefert.

Alle diese Dokumente wurden so gestaltet, dass sie möglichst einfach und schnell angepasst werden können. Sie sind einfache Textdokumente, werden zeilenweise geparsed und besitzen eine Kommentarfunktion (voranstehende ‘//’ bezeichnen den Rest der Zeile als Kommentar).

Namenszuweisungen für Konstanten

Dieses Dokument dient dazu, Konstanten, deren Namen von Task- oder Relationsbeschreibungen verlangt werden, einen neuen Namen zuzuweisen, sollte der ursprüngliche nicht die gewünschte Form haben. Jeder Eintrag ist optional. Sollte eine Konstante im Text erwähnt werden, die nicht in diesem Dokument verzeichnet ist, wird ihr regulärer Name verwendet. Eine Zeile dieser Datei sollte folgendes Format haben:

```
ursprünglicherName<=>neuer Name // Kommentar
```

Dieses Dokument ist nur nötig, wenn in dem Dokument mit domänenspezifischen Ausdrücken Argumente von Tasks und Relationen erwähnt werden. Sollte auch dieses Dokument von den gewählten Betriebsmodi nicht verlangt werden, sind Konstantennamen nie Teil der textuellen Erklärung und somit auch ihr Dokument nicht erfordert.

Domänenspezifische Daten

Dieses Dokument spielt eine zentrale Rolle bei der Generierung von ansprechenden Erklärungen. Sowohl das Advanced Text Planning als auch die Advanced Lexicalization verwenden dieses Dokument. Auch die Input Supporting Lexicalization kann dieses Dokument verwenden, wenn die vom Nutzer hinzugefügten Satzfragmente dies erfordern. Grundsätzlich enthält das Dokument zwei Arten von Einträgen. Die einen beziehen sich auf Tasks, die anderen auf Relationen. Durch eine Zeile der Form

```
& Tasks
```

beziehungsweise

```
& Relations
```

werden die nachfolgenden Einträge einer dieser Kategorien zugeordnet. Es ist problemlos möglich, diese Kategorien mehrmals im Dokument zu wechseln.

Einträge, die sich auf Tasks beziehen, müssen neben dem ursprünglichen Namen des Task-Schemas drei weitere Beschreibungen enthalten. Einen in Form eines einfach lesbaren Task-Namens, eine als Beschreibung des Tasks im Infinitiv und einmal als Verlaufsform. Diese werden durch '<=>' voneinander getrennt. Dieses Trennzeichen kann problemlos Teil des ursprünglichen Namens des Task-Schemas sein, darf aber nicht in den drei folgenden Teilen verwendet werden. Wenn ein Argument des Tasks in die Beschreibung oder den Namen eingefügt werden soll, so muss der Index dieses Arguments aus dem entsprechenden Task-Schema inklusive vorausgehendem als auch folgendem '%' Zeichen an die entsprechende Stelle eingefügt werden. Eine mögliche Zeile hätte folgende Form:

```
sendMail<=>"send Mail to %1%"<=>send an eMail to %1%<=>
        sending %1% an eMail
```

Für Einträge für Relationen gilt, dass nach dem Relationsnamen zwei Beschreibungen der Relation folgen müssen. Eine sollte den True-, die andere den False-Fall abdecken. Es können auf die gleiche Weise wie bei Tasks auch die Argumente von Relationen mit eingebunden werden. Ein mögliche Zeile hätte folgendes Aussehen:

```
AlarmSet<=>the alarm must be set to %1% o'clock<=>
        the alarm must not be set
```

In Anhang A.1 findet sich ein vorgefertigtes domänenspezifisches Dokument, das die Tasks der Smartphone-Domäne mit ihrer Lexicalization enthält.

Selbstdefinierte Lexicalizations

Die Input Supporting Lexicalization erlaubt es, selbst definierte Satzfragmente anstelle den von mir vorgefertigten zu verwenden. Um solch eine Lexicalization zu erstellen, muss eine Eingabedatei mit allen nötigen Informationen angegeben werden. Diese Informationen umfassen neben den eigentlichen Strings auch eine Auflistung, in welcher Reihenfolge welche Argumente eingebunden werden, in welcher Form sie dargestellt werden sollen, ihre Präfixe, welche von ihnen durch 'it', 'which' oder 'that' abgekürzt werden können und welche Tasks für den Folgesatz als Pronomen angeboten werden. Bei der Angabe all dieser Informationen sollte unbedingt darauf geachtet werden, welchen Platz der spätere Satz innerhalb des endgültigen Textes einnimmt. Oft können beispielsweise Pronomen in einem einfachen Satz an einer bestimmten Stelle vorkommen. Wird derselbe Satz aber in einen Rahmensatz eingefügt, macht dieselbe Verwendung desselben Pronomens keinen Sinn oder ist sogar schlicht falsch. Es ist daher angebracht, eher zu wenige als zu viele akzeptierte Pronomen einzufügen und nur bei Bedarf nachzukorrigieren. Von großer Bedeutung ist, dass für jede Message-Art, außer Phi, zumindest ein Satz angegeben ist. Ansonsten wäre es unmöglich, eine textuelle Erklärung zu erstellen.

Die einzelnen Sektionen des Dokuments werden, wie bei den domänenspezifischen Daten, durch einzeilige Einträge begonnen, die stets mit einem '`&`' beginnen. Die Schlüsselworte `Decomposition Sentences`, `Causal Sentences`, `Necessary Sentences`, `Top Sentences` und `Goal Sentences` markieren die folgenden Sätze als der jeweiligen Message-Art zugehörig.

Die Einträge für die Sätze selbst bestehen jeweils aus drei Teilen, die durch '<=>' voneinander getrennt werden. Der erste Teil enthält die Strings, der zweite die Indizes, Präfixe und Ausdrucksarten der Argumente und der dritte Teil die angebotenen und akzeptierten Pronomen. Oft ist es möglich, Sätze zu bilden, die bis auf die Ausdrucksarten der Tasks identisch sind. In diesem Fall müssen dennoch mehrere Einträge erstellt werden. Beim Erstellen eines Satzes aus dieser Datei werden, beginnend mit einem String, alternierend ein String aus dem ersten Teil und ein Task aus dem zweiten Teil aneinandergefügt.

Im ersten Teil werden die einzelnen Strings durch '`;`'-Zeichen getrennt. Für den zweiten Teil muss für jeden Task, der eingebunden werden soll, folgendes angegeben werden: Erst der Index in der Argumentenliste der Message (bei 0 beginnend), gefolgt von einem '`,`'-Zeichen als Trennzeichen, dann ein einzelner Character für die jeweilige Form: 'i' für die Infinitiv-Form, 'c' für die Continuous-Form, 'n', wenn der Task-Name verwendet werden soll oder 's', wenn die aus der Simple Lexicalization stammende einfache Form verwendet werden soll. Nach einem weiteren '`,`'-Zeichen kann ein beliebiger String als Präfix angegeben werden. Wenn weitere Tasks eingebunden werden sollen, muss mit einem '`;`'-Zeichen ein weiterer Task begonnen werden. Für den dritten Teil gilt, dass jeder Index bei 0 beginnt und, wenn keine Verwendung irgendeines Tasks als akzeptiertes oder angebotenes Pronomen vorgesehen ist, -1 als Index anzugeben ist. Die ersten drei durch '`;`'-Zeichen getrennten Ganzzahlen stehen für die

akzeptierten, die nächsten drei für die angebotenen Tasks. Bei allen sechs Zahlen gilt die Reihenfolge: it, which, that.

Ein Satz der Causal Sentences-Sektion könnte beispielsweise folgende Form haben:

```
;is required for;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
```

Er würde folgenden Satzprototypen darstellen:

+ erster String (in diesem Fall leer) + ”to” + Task Nr.0 in Infinitiv-Form + ”is required for” + Präfix des zweiten Tasks (in diesem Fall leer) + Task Nr.1 in Continuous-Form.

Des Weiteren würde er den Task Nr.0 durch ‘it’ oder ‘which’ ersetzen und dem nachfolgendem Satz den Task Nr.1 als ‘that’ anbieten.

In Anhang A.2 findet sich ein vorgefertigtes Dokument für die Input Supporting Lexicalization.

3.7. Demonstration

Um das Zusammenspiel der einzelnen Varianten der Zwischenschritte zu verdeutlichen, enthält dieses Kapitel ein ausführliches Beispiel einer kompletten Umsetzung einer formalen Erklärung in eine textuelle Erklärung. Es werden die Ergebnisse unterschiedlicher Text Planning-, Sentence Aggregation- und Lexicalization-Varianten gezeigt. Es wurden die Eingabedateien aus dem Anhang verwendet, aber in den *Relations*-Abschnitt des domänenspezifische Dokuments ein weiterer Eintrag eingefügt:

```
inMode_EMail<=>you have to be in eMail mode<=>you must not be in
eMail mode
```

Die verwendete formale Erklärung hatte folgende Form:

1	Necessary(pressHome.EMail)	D-Ch 2,3
2	DecompositionRelation(pressHome.EMail, m, enterMode.EMail)	[*]
3	Necessary(enterMode.EMail)	C-CH 4,5
4	CausalRelation(enterMode.EMail, inMode_EMail, pressEMail.NewEMail)	[*]
5	Necessary(pressEMail.NewEMail)	C-Ch 6,7
6	CausalRelation(pressEMail.NewEMail, ϕ_1, pressEMail.NewEMail.Send)	[*]
7	Necessary(pressEMail.NewEMail.Send)	C-Ch 8,9
8	CausalRelation(pressEMail.NewEMail.Send, ϕ_2, goalTask)	[*]
9	Necessary(goalTask)	Basic

Hierbei bedeutet [*], dass dieser Schritt entweder mit Basic, oder mit für die textuelle Erklärung nicht relevanten Axiomen erklärt wurde

Nachfolgend werden nun die verwendeten Text Planning, Sentence Aggregation und Lexicalization Varianten zusammen mit der textuellen Erklärung, die das System unter ihrer Verwendung erstellt, genannt.

Simple Text Planning, Simple Sentence Aggregation, Simple Lexicalization

The task press_Home.EMail is necessary because it belongs to the task enterMode_EMail. That is obligatory since it is needed for the task press_EMail.NewEMail. That is crucial as it is needed for the task press_EMail.NewEMail.Send. That is necessary since it establishes a goal.

Simple Text Planning, Content-based Sentence Aggregation, Simple Lexicalization

The task press_Home.EMail is necessary because it belongs to the task enterMode_EMail. That is obligatory since it is needed for the task press_EMail.NewEMail and that is crucial as it is needed for the task press_EMail.NewEMail.Send. That is necessary since it establishes a goal.

Simple Text Planning, Length-based Sentence Aggregation, Simple Lexicalization

The task press_Home.EMail is obligatory since it belongs to the task enterMode_EMail, which is needed for the task press_EMail.NewEMail and that is crucial as it is needed for the task press_EMail.NewEMail.Send. That is obligatory since it establishes a goal.

Simple Text Planning, Length-based Sentence Aggregation, Advanced Lexicalization

Pressing the [eMail] button is crucial since it is part of the task “enter the eMail menu”, which is needed for pressing [New eMail] and that is necessary as it is needed for pressing [Send]. That is crucial since it establishes a goal.

Advanced Text Planning, Length-based Sentence Aggregation, Advanced Lexicalization

Pressing the [eMail] button is crucial since it is part of the task “enter the eMail menu”, that is obligatory since you have to be in eMail mode for pressing [New eMail] and that is necessary since it is needed for pressing [Send]. That is crucial because it establishes a goal.

Simple Text Planning, Length-based Sentence Aggregation, Input Supporting Lexicalization

Pressing the [eMail] button is crucial since it is part of the task “enter the eMail menu”, which is needed for pressing [New eMail] and that is necessary as it is needed for pressing [Send]. That is crucial since it establishes a goal.

Advanced Text Planning, Length-based Sentence Aggregation, Input Supporting Lexicalization

Pressing the [eMail] button is crucial since it is part of the task “enter the eMail menu”, that is obligatory since you have to be in eMail mode for pressing [New eMail] and that is necessary since it is needed for pressing [Send]. That is crucial because it establishes a goal.

Fazit:

Es ist leicht festzustellen, dass die größte Verbesserung in Sachen Lesbarkeit und Natürlichkeit durch die Verwendung der Advanced (bzw. Input Supporting) Lexicalization anstelle der Simple Lexicalization erzielt wird. Es erscheint also lohnend für häufig verwendete Planungsdomänen, eine Eingabedatei mit domänenspezifischen Daten anzufertigen. Die Advanced Lexicalization bietet mit der gezielten Nennung von Relationen eine Möglichkeit, einzelne Details innerhalb des Textes hervorzuheben oder näher auszuführen und ist daher für das Erläutern besonders wichtiger oder schwer verständlicher Teilschritte nützlich. Die Length-based Sentence Aggregation hat in diesem Fall einen Text mit besserem Lesefluss als die Content-based Sentence Aggregation erzielt, was bei einer solch kurzen Erklärung mit stark gemischten Causal und Decomp Messages zu erwarten war. Bei Erklärungen mit langen Ketten von derselben Art von Ableitungen hatte sich zuvor gezeigt, dass die Content-based Sentence Aggregation dennoch ein bessseres Ergebnis erzielt. Da die für die Input Supporting Lexicalization verwendete Eingabedatei exakt dieselben Satzfragmente in der gleichen Reihenfolge enthält wie die Advanced Lexicalization, unterscheidet sich die mit ihr erstellte Erklärung in keinster Weise der, die mit der Advanced Lexicalization erstellt wurde.

4. Diskussion und Ausblick

Auch wenn das System über alle weiterführenden Features, die im Verlauf seines Entstehens vorgeschlagen wurden, verfügt, so ergeben sich doch einige Punkte, die im Rahmen meiner Bachelorarbeit nicht implementiert wurden. Dies liegt meistens daran, dass sie weitreichende Änderungen in den Eingabedokumenten nötig gemacht hätten, was sehr zulasten der Bedienungsfreundlichkeit ginge. Dennoch sollen diese Vorschläge nicht unerwähnt bleiben.

Kontextspezifische Informationen in der Referring Expression Generation

Im Moment verfügt die Referring Expression Generation über fast keine Informationen bezüglich der Position des aktuell zu bearbeitenden Satzes innerhalb des Sentence Trees. Aus diesem Grund ist es nötig, bei der Zuweisung möglicher Pronomen bei der Input Supporting Lexicalization viel Zeit darauf zu verwenden, die späteren Verwendungsmöglichkeiten des Satzes und seiner möglichen Vorgänger und Nachfolger abzuschätzen. Dies könnte eventuell verbessert werden, indem ein leicht abgewandelter Algorithmus verwendet wird, dem aber mehr Daten bezüglich des Kontextes eines Satzes zur Verfügung gestellt werden. In diesem Fall könnte man im Eingabedokument dazu übergehen, für verschiedene Fälle, wie Anfang, Mittelteil oder Ende eines Satzes, unterschiedliche Pronomen festzulegen. Der Algorithmus könnte dann verschiedene Fälle unterscheiden, um mögliche Pronomen vorzuschlagen. Da es sich aber in der praktischen Anwendung gezeigt hat, dass es nur sehr selten vorkommt, dass verschiedene Pronomen an derselben Stelle häufiger verwendet werden können, habe ich mich entschieden, diesen Ansatz zu verwerfen und damit den Aufwand für das Erstellen von eigenen Lexicalizations verringert.

Seperate Referring Expression Generation für Argumente von Tasks und Relationen

Es ist nur sehr selten der Fall, dass in demselben Text der gleiche Task zweimal ausführlich genannt wird, nachdem er von der Referring Expression Generation bearbeitet wurde. Dies trifft aber nicht auf die Argumente der Tasks zu. Tatsächlich werden manche Konstantennamen recht häufig genannt. Um den Lesefluss des Textes zu unterstützen, könnte man nach der eigentlichen Referring Expression Generation eine zweite durchführen, die anstelle der Tasks nur ihre Argumente betrachtet. Dies hätte aber zur Folge, dass in dem Eingabedokument für domänenspezifische Daten viele neuen Informationen nötig geworden wären, die deren Verwendung, ähnlich wie bei der Input Supporting Lexicalization, steuern. Da das Erstellen dieses

Dokuments ohnehin bereits einen nicht unbeträchtlichen Aufwand für den Endnutzer darstellt, habe ich mich entschlossen, darauf zu verzichten.

Gewichtetes Roulette Wheel-Verfahren oder Stochastic Universal Sampling-Verfahren bei der Lexicalization

Im Moment wird zwischen allen möglichen äquivalenten Ausdrücken bei der Auswahl von Ausdrücken von Tasks und Elaboration Nodes mittels einer einzigen Pseudozufallsvariable entschieden. In seltenen Fällen bedeutet dies, dass derselbe Ausdruck gehäuft auftritt, während andere gar keine Verwendung finden. Dies ist zwar kein großes Problem, aber es wäre denkbar, statt dieser einfachen Auswahl das gewichtete Roulette Wheel-Verfahren zu verwenden und bei jeder Verwendung eines Ausdruckes dessen Wahrscheinlichkeit für ein weiteres Auftreten zu verringern. Eine andere Möglichkeit wäre das aus den Genetischen Algorithmen stammende Stochastic Universal Sampling. Da dies, im Hinblick auf die Determiniertheit des Systems, weitreichende Änderungen nötig gemacht und nur eine äußerst geringe Verbesserung erzielt hätte, habe ich darauf verzichtet.

Weitere Arten der Erklärungen und Repräsentationen

Nicht Teil meiner Bachelorarbeit waren die Erklärungen von Variablengleichungen und Temporalen Beziehungen, die das Erklärbär-System anbietet. Dennoch könnten manche Teile meines Systems auch in einem System Verwendung finden, welches sich dieser Erklärungen annimmt, bzw. könnte mein System dazu erweitert werden. Ebenfalls nicht Teil meiner Bachelorarbeit waren sprachliche oder grafische Repräsentationen der Erklärung. Allerdings stellt mein System für beide eine gute Vorarbeit dar und könnte gegebenenfalls in diese Richtung erweitert werden.

A. Vorgefertigte Eingabedokumente

A.1. Domänenspezifische Daten für die Smartphone-Domäne

Dies ist der Inhalt einer Eingabedatei, die für jeden Task der Smartphone Domäne eine passende Lexikalisierung enthält. Teilweise verwenden diese Lexikalisierungen die Argumente ihrer Planschritte, desshalb ist es eventuell nötig, in dem Dokument für Konstantennamen weitere Einträge zu erstellen. Es ist zu beachten, dass die hier eingerückten Zeilenumbrüche im Eingabedokument ersetzt werden müssen, da das Dokument zeilenweise geparsed wird.

```
// ##### Smartphone Domain #####

& Relations

// Hier Relationen eintragen

& Tasks

select_People.Favourite_NoOp<=>"not select a favourite"<=>not
    select the favourite %1%<=>not selecting the favourite %1%
enterMode_Alarm<=>"enter the alarm menu"<=>enter the alarm
    menu<=>entering the alarm menu
configure_Alarm<=>"configure alarm"<=>configure the alarm to
    go off at the %2%<=>configuring the alarm to go of at the
    %2%
configure_Contact<=>"configure contact"<=>configure a contact
    for %2%<=>configuring a contact for %2%
set_Contact.Picture<=>"set picture for contact"<=>set the
    picture for contact %1%<=>setting the picture for contact
    %1%
select_EMail.EMailAccount<=>"select an e-mail
    account"<=>select an e-mail account<=>selecting an e-mail
    account
```

```
press_Calendar.NewAppointment.OK<=>"press [OK]"<=>press
    [OK]<=> pressing [OK]
create_Appointment<=>"create appointment"<=>create an
    appointment for %1%<=>creating an appointment for %1%
select_Contacts.Details.Mobile<=>"select mobile"<=>select to
    call on the mobile<=>select calling on the mobile
add_NewFavourite<=>"add new favourite"<=>add %1% as
    favourite<=>adding %1% as favourite
select_People.FavouriteSelected.Office<=>"call favourite in
    office"<=>select to call in the office<=>selecting call in
    the office
set_Task.Name<=>"set name for task"<=>set the name for the
    task<=>setting the name for the task
select_People.AddFavourite.Contact_withMobile<=>"select
    contact"<=>select contact %1%<=>selecting contact %1%
press_Home_From_Telephone_Office<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
set_Task.Time<=>"set time for task"<=>set the time for the
    task<=>setting the time for the task
select_Contacts.Details.Office<=>"select office"<=>select to
    call in the office<=>select calling in the office
press_Telephone.Call_Office<=>"press [Call]"<=>press
    [Call]<=>pressing [Call]
attachContactableInformationToEMail<=>"attach contactable
    information to eMail"<=>attach the information %2% to the
    eMail<=>attaching the information %2% to the eMail
enter_Number_ForSMS<=>"enter number"<=>enter %1%'s
    number<=>entering %1%'s number
set_Contact.Email<=>"set eMail address for contact"<=>set the
    eMail address for %2%<=>setting the eMail address for %2%
press_Tasks.Menu.NewTask<=>"press [New Task]"<=>press [New
    Task]<=>pressing [New Task]
attachRegularInformationToSMS<=>"attach regular information
    to SMS"<=>attach the information %2% to the
    SMS<=>attaching the information %2% to the SMS
press_Home.Calendar<=>"press [Calendar]"<=>press
    [Calendar]<=>pressing [Calendar]
extractsInformation_Contactable<=>"extract contactable
    information<=>extract the information %3%<=>extracting the
    information %3%
select_Contacts.Details.Number<=>"select number"<=>select to
    call<=>selecting to call
```

```
set_Contact.Name<=>"set name for contact"<=>set the name for
    the contact<=>setting the name for the contact
press_People.More<=>"press [More]"<=>press [More]<=>pressing
    [More]
attachMultipleInformation<=>"attach multiple
    information"<=>attach multiple information<=>attaching
    multiple information
press_People.Smallplus_FavouriteSelected<=>"press
    [+]"<=>press [+]<=>pressing [+]
select_People.FavouriteSelected.Number<=>"call
    favourite"<=>select a number to call<=>selecting a number
    to call
set_Appointment.Location<=>"set location for
    appointment"<=>set the location for the
    appointment<=>setting the location for the appointment
press_Contacts.NewContact.SelectPicture<=>"press [Select
    Picture]"<=>press [Select Picture]<=>pressing [Select
    Picture]
set_Appointment.Name<=>"set name for appointment"<=>set the
    name for the appointment<=>setting the name for the
    appointment
press_Home_From_Add_Favourite<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
enter_Number_ForCall_Mobile<=>"enter mobile number"<=>enter
    %1%'s mobile number<=>entering %1%'s mobile number
enterMode_Home<=>"enter the home menu"<=>enter the home
    menu<=>entering the home menu
enter_Number_ForCall<=>"enter number"<=>enter %1%'s
    number<=>entering %1%'s number
contact<=>"contact"<=>contact %1%<=>contacting %1%
press_Home.Telephone<=>"press [Telephone]"<=>press
    [Telephone]<=>pressing [Telephone]
enterMode_People<=>"enter people menu"<=>enter the people
    menu<=>entering the people menu
select_People.AddFavourite.Contact_withEmail<=>"select
    contact"<=>select contact %1%<=>selecting contact %1%
press_Home.People<=>"press [People]"<=>press
    [People]<=>pressing [People]
create_Contact<=>"create a contact"<=>create a contact for
    %2%<=>creating a contact for %2%
press_Home.Messages<=>"press [Messages]"<=>press
    [Messages]<=>pressing [Messages]
```

```
select_AccountSelection.Account<=>"select an eMail
    account"<=>select the eMail account %1%<=>selecting the
    eMail account %1%
press_People.Menu<=>"press [Menu]"<=>press [Menu]<=>pressing
    [Menu]
create_Task<=>"create task"<=>create the task %1%<=>creating
    the task %1%
press_Home<=>"press [Home]"<=>press [Home]<=>pressing [Home]
create_NewContact<=>"create contact"<=>create a contact for
    %2%<=>creating a contact for %2%
activate_Alarm<=>"activate alarm"<=>activate alarm
    %1%<=>activating alarm %1%
select_People.FavouriteSelected.SMS<=>"send SMS to
    favourite"<=>select to send an SMS<=>selecting to send an
    SMS
press_Tasks.Menu<=>"press [Menu]"<=>press [Menu]<=>pressing
    [Menu]
set_Contact.Connection<=>"set telephone number or
    eMail"<=>set a telephone number or eMail<=>setting a
    telephone number or eMail
press_Programs.Tasks<=>"press [Tasks]<=>press
    [Tasks]<=>pressing [Tasks]
press_EMail.NewEMail.Send<=>"press [Send]"<=>press
    [Send]<=>pressing [Send]
extractsInformation_Regular<=>"extract regular
    information<=>extract the information %3%<=>extracting the
    information %3%
set_Task.Reminder<=>"set reminder for task"<=>set the
    reminder for the task<=>setting the reminder for the task
enter_Number_ForSMS_Office<=>"enter number"<=>enter %1%'s
    office number<=>entering %1%'s office number
set_Contact.NumberOffice<=>"set office number for
    contact"<=>set the office number for %2%<=>setting the
    office number for %2%
configure_Task<=>"configure task"<=>configure task
    %1%<=>configuring task %1%
press_People.Smallplus_AlreadyPressed<=>"press [+]<=>press
    [+]<=>pressing [+]
enter_Number_ForCall_Office<=>"enter office number"<=>enter
    %1%'s office number<=>entering %1%'s office number
select_People.FavouriteSelected.EMail<=>"send eMail to
    favourite"<=>select to send an eMail<=>selecting to send
    an eMail
```

```
attachContactableInformationToSMS<=>"attach contactable
    information to SMS"<=>attach the information %2% to the
    SMS<=>attaching the information %2% to the SMS
press_Calendar.Menu.NewAppointment<=>"press [New
    Appointment]"<=>press [New Appointment] button<=>pressing
    [New Appointment]
send_EMail<=>"send eMail"<=>send an eMail to %1%<=>sending an
    eMail to %1%
select_PictureSelection.PictureForNewContact<=>"Select
    Picture"<=>select a picture<=>selecting a picture
press_Home_From_SMS_Mobile<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
press_EMail.NewEMail<=>"press [New eMail]"<=>press [New
    eMail]<=>pressing [New eMail]
press_Home.EMail<=>"press [eMail]"<=>press the [eMail]
    button<=>pressing the [eMail] button
set_Alarm.Days<=>"set days for alarm"<=>set the days for the
    alarm<=>setting the days for the alarm
set_Alarm.Time<=>"set alarm time"<=>set the time for the
    alarm<=>setting the time for the alarm
extractsInformation<=>"extract information<=>extract the
    information %3%<=>extracting the information %3%
press_Home_From_SMS_Office<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
press_NewMessage.Send_Mobile<=>"press [Send]<=>press
    [Send]<=>pressing [Send]
attachContactableInformationToCall<=>"attach contactable
    information to call"<=>attach the information %2% to the
    call<=>attaching the information %2% to the call
enterMode_Contacts<=>"enter contacts menu"<=>enter the
    contacts menu<=>entering the contacts menu
extractsMultipleInformation<=>"extract information<=>extract
    information<=>extracting information
press_People.Smallplus<=>"press [+]<=>press [+]<=>pressing [+]
select_People.FavouriteSelected.Mobile<=>"call favourite on
    mobile"<=>select to call on mobile<=>select calling on
    mobile
press_NewMessage.Send_Office<=>"press [Send]<=>press
    [Send]<=>pressing [Send]
press_Home.Clock<=>"press [Clock]"<=>press [Clock]<=>pressing
    [Clock]
press_Tasks.NewTask.OK<=>"press [OK]"<=>press the [OK]
    button<=>pressing the [OK] button
```

```
attachRegularInformationToEMail<=>"attach regular information
    to eMail"<=>attach regular information to the
    eMail<=>attaching regular information to the eMail
attachInformationToSMS<=>"attach information to SMS"<=>attach
    information to the SMS<=>attaching information to the SMS
enterMode_EMail<=>"enter the eMail menu"<=>enter the eMail
    menu<=>entering the eMail menu
press_Home_From_SMS<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
select_People.Favourite_FromReadyToAddFavourite<=>"select
    favourite"<=>select %2% from the list of
    favourites<=>selecting %2% from the list of favourites
attachInformationToEMail<=>"attach information to
    eMail"<=>attach information to the eMail<=>attaching
    information to the eMail
create_NewAppointment<=>"create new appointment"<=>create a
    new appointment<=>creating a new appointment
set_Appointment.Time<=>"set time for appointment"<=>set the
    time for the appointment<=>setting the time for the
    appointment
select_People.Favourite_WithPreviousSelection<=>"select
    favourite"<=>select %3% from the list of
    favourites<=>selecting %3% from the list of favourites
set_Alarm<=>"set alarm"<=>set the alarm<=>setting the alarm
select_People.AddFavourite.Contact<=>"select
    contact"<=>select contact %1%<=>selecting contact %1%
enter_Number_ForSMS_Mobile<=>"enter mobile number"<=>enter
    %1%'s mobile number<=>entering %1%'s mobile number
set_Task.Name_Light<=>"set name for task"<=>set the name for
    the task<=>setting the name for the task
send_SMS<=>"send SMS"<=>send an SMS to %1%<=>sending an SMS
    to %1%
set_Contact.NumberMobile<=>"set mobile number for
    contact"<=>set mobile number for %2%<=>setting the mobile
    number for %2%
press_Telephone.Call_Mobile<=>"press [Mobile]<=>press
    [Mobile]<=>pressing [Mobile]
select_EMail.EMailAccount_NoOp<=>"not select an eMail
    account"<=>not select the eMail account %1%<=>not
    selecting the eMail account %1%
call<=>"call"<=>call %1%<=>calling %1%
enterMode_Tasks<=>"enter task menu"<=>enter the task
    menu<=>entering the task menu
```

```
attachInformation<=>"attach information"<=>attach
    information<=>attaching information
set_Alarm.Description<=>"set alarm description"<=>set the
    description for the alarm<=>setting the description for
    the alarm
press_Home_From_Telephone_Mobile<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
select_People.AddFavourite.Picture<=>"select
    picture"<=>select a picture<=>selecting a picture
press_Messages.NewMessage<=>"press [New Message]"<=>press
    [New Message]<=>pressing [New Message]
enterMode_People.AddFavourite<=>"enter the add favourite
    menu"<=>enter the add favourite menu<=>entering the add
    favourite menu
press_Clock.Alarm<=>"press [Alarm]"<=>press
    [Alarm]<=>pressing [Alarm]
attachInformationToCall<=>"attach information to
    call"<=>attach information to call<=>attaching information
    to call
press_Home_Default<=>"press [Home]<=>press [Home]<=>pressing
    [Home]
press_Contacts.New<=>"press [New]"<=>press [New]<=>pressing
    [New]
transferMessage<=>"transfer message"<=>transfer the
    message<=>transferring the message
press_NewMessage.Send<=>"press [Send]"<=>press
    [Send]<=>pressing [Send]
press_People.Menu.AddFavourite<=>"press [Add
    Favourite]<=>press [Add Favourite]<=>pressing [Add
    Favourite]
configure_Appointment<=>"configure appointment"<=>configure
    the appointment %1%<=>configuring the appointment %1%
create_NewTask<=>"create new task"<=>create a new
    task<=>creating a new task
select_People.AddFavourite.FavouriteNumber_WithPicture<=>"select
    favourite number"<=>select your favourite
    number<=>selecting the favourite number
set_Appointment.Reminder<=>"set reminder for
    appointment"<=>set the reminder for the
    appointment<=>setting the reminder for the appointment
attachRegularInformationToCall<=>"attach regular information
    to call"<=>attach regular information to call<=>attaching
    regular information to call
```

```
enter_EMailAddress<=>"enter eMail address"<=>enter the eMail
    address of %1%<=>entering the eMail address of %1%
press_Home.Programs<=>"press [Programs]"<=>press
    [Programs]<=>pressing [Programs]
select_People.AddFavourite.FavouriteNumber_WithoutPicture<=>"select
    favourite number"<=>select your favourite
    number<=>selecting the favourite number
press_Home_From_Telephone<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
press_Calendar.Menu<=>"press [Menu]"<=>press
    [Menu]<=>pressing [Menu]
press_Home_From_Contacts.Details<=>"press [Home]"<=>press
    [Home]<=>pressing [Home]
select_People.AddFavourite.Contact_withOffice<=>"select
    contact"<=>select contact %1%<=>selecting contact %1%
press_People.LargePlus<=>"press [+]"<=>press [+]<=>pressing
    [+]
press_Contacts.NewContact.OK<=>"press [OK]"<=>press
    [OK]<=>pressing [OK]
add_Favourite<=>"add favourite"<=>add contact %1% as
    favourite<=>adding contact %1% as favourite
enterMode_Messages<=>"enter message menu"<=>enter the add
    favourite menu<=>entering the add favourite menu
select_Contacts.ContactForContactable<=>"select
    contact"<=>select the contact for %2%<=>selecting the
    contact for %2%
enterMode_Calendar<=>"enter the calendar menu"<=>enter the
    calender menu<=>entering the calender menu
select_People.Favourite<=>"select favourite"<=>select the
    favourite for %2%<=>selecting the favourite for %2%
press_Telephone.Call<=>"press [Call]"<=>press
    [Call]<=>pressing [Call]
enterMode_Telephone<=>"enter telephone menu"<=>enter the
    telephone menu<=>entering the telephone menu
select_EMail.EMailAccount_WithPreviousSelection<=>"select
    eMail-account<=>select the eMail-Account %2%<=>selecting
    the eMail-Account %2%
press_Telephone.Contacts<=>"press [Contacts]"<=>press
    [Contacts]<=>pressing [Contacts]
```

A.2. Input Supporting Lexicalization-Dokument der Advanced Lexicalization

Für die Input Supporting Lexicalization wurde ein Prototypen-Dokument erstellt. Dieses enthält eine Lexikalisierung, die das genau identische Ergebnis wie die Advanced Lexicalization erzielt. Mit diesem Dokument können auf schnelle und einfache Art und Weise neue Sätze hinzugefügt werden.

```
& Causal Sentences

;is needed for;<=>0,c,;1,c,;<=>0;0;-1;-1;1;1
;is needed for;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
;is needed for;<=>0,n,the task;1,c,;<=>0;0;-1;-1;1;1
;is needed for;<=>0,c,;1,n,the task;<=>0;0;-1;-1;1;1
;is needed for;<=>0,i,to;1,n,the task;<=>0;0;-1;-1;1;1
;is needed for;<=>0,n,the task;1,n,the task;<=>0;0;-1;-1;1;1
;is required for;<=>0,c,;1,c,;<=>0;0;-1;-1;1;1
;is required for;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
;is required for;<=>0,n,the task;1,c,;<=>0;0;-1;-1;1;1
;is required for;<=>0,c,;1,n,the task;<=>0;0;-1;-1;1;1
;is required for;<=>0,i,to;1,n,the task;<=>0;0;-1;-1;1;1
;is required for;<=>0,n,the task;1,n,the task;<=>0;0;-1;-1;1;1

& Decomposition Sentences

;is part of;<=>0,c,;1,c,;<=>0;0;-1;-1;1;1
;is part of;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
;is part of;<=>0,n,the task;1,c,;<=>0;0;-1;-1;1;1
;is part of;<=>0,c,;1,n,the task;<=>0;0;-1;-1;1;1
;is part of;<=>0,i,to;1,n,the task;<=>0;0;-1;-1;1;1
;is part of;<=>0,n,the task;1,n,the task;<=>0;0;-1;-1;1;1
;belongs to;<=>0,c,;1,c,;<=>0;0;-1;-1;1;1
;belongs to;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
;belongs to;<=>0,n,the task;1,c,;<=>0;0;-1;-1;1;1
;belongs to;<=>0,c,;1,n,the task;<=>0;0;-1;-1;1;1
;belongs to;<=>0,i,to;1,n,the task;<=>0;0;-1;-1;1;1
;belongs to;<=>0,n,the task;1,n,the task;<=>0;0;-1;-1;1;1
;is a substep of;<=>0,c,;1,c,;<=>0;0;-1;-1;1;1
;is a substep of;<=>0,i,to;1,c,;<=>0;0;-1;-1;1;1
;is a substep of;<=>0,n,the task;1,c,;<=>0;0;-1;-1;1;1
;is a substep of;<=>0,c,;1,n,the task;<=>0;0;-1;-1;1;1
```

```
;is a substep of;<=>0,i,to;1,n,the task;<=>0;0;-1;-1;1;1
;is a substep of;<=>0,n,the task;1,n,the task;<=>0;0;-1;-1;1;1

& Top Sentences

it is a top task;<=><=>-1;-1;-1;-1;-1;-1

& Goal Sentences

it establishes a goal;<=><=>-1;-1;-1;-1;-1;-1

& Necessary Sentences

;is necessary;<=>0,c,;<=>0;0;0;0;-1;-1
;is necessary;<=>0,n,the task;<=>0;0;0;0;-1;-1
;is necessary;<=>0,i,to;<=>0;0;0;0;-1;-1
;is obligatory;<=>0,c,;<=>0;0;0;0;-1;-1
;is obligatory;<=>0,n,the task;<=>0;0;0;0;-1;-1
;is obligatory;<=>0,i,to;<=>0;0;0;0;-1;-1
;is crucial;<=>0,c,;<=>0;0;0;0;-1;-1
;is crucial;<=>0,n,the task;<=>0;0;0;0;-1;-1
;is crucial;<=>0,i,to;<=>0;0;0;0;-1;-1
you have to;<=>0,i,;<=>-1;-1;-1;0;-1;-1
you must to;<=>0,i,;<=>-1;-1;-1;0;-1;-1
```

Literaturverzeichnis

[BBG+11] Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3-4):219–236, 2011. Special Issue on Complex Cognition.

[RD97] E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[See11] Bastian Seegebarth. Formale aspekte der erklärung hybrider pläne. Master thesis, Ulm University, 2011. Diese Abschlussarbeit wurde mit dem eXXellence-Award 2012 ausgezeichnet.

[SMSB12] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 225–233. AAAI Press, 6 2012.

Eidesstattliche Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbstständig angefertigt zu haben. Dabei wurden nur die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ulm, den 12. November 2012 _____

(Johannes F. Gesell)