Universität Karlsruhe (TH)
Institut für Algorithmen und
Kognitive Systeme
der Fakultät für Informatik

Automatische Analyse natürlichsprachlicher Texte zur Generierung von synthetischen Bildfolgen

Studienarbeit
von
M. Arens und A. Ottlik

Betreuer: Prof. Dr. H.–H. Nagel, Dipl. Inf. M. Middendorf

Kurzfassung

Am Institut für Algorithmen und Kognitive Systeme der Fakultät für Informatik an der Universität Karlsruhe (TH) wurden zahlreiche Werkzeuge entwickelt, um Videobildfolgen von Straßenverkehrsszenen auszuwerten, die darin erkannten Fahrzeuge zu verfolgen und deren Aktionen sowie deren Verhalten in Form von natürlichsprachlichen Texten auszugeben. Auch der umgekehrte Weg, also die Generierung von – dann synthetischen – Bildfolgen aus natürlichsprachlichen Texten, wurde in [Jeyakumar 98] begonnen. [Nagel et al. 99] zeigen, wie auf der Basis von abstrakten begrifflichen Beschreibungen von Handlungsabläufen in einer Straßenszene eine synthetische Bildfolge generiert werden kann. Eine direkte Ansteuerung dieses Generierungsprozesses durch natürlichsprachliche Texte fehlt jedoch.
Diese Lücke zu schliessen, also aus der natürlichsprachlichen Beschreibung eines Sachverhalts die Semantik dieses Sachverhalts zu erschließen und daraus eine systeminterne begriffliche Repräsentation dessen, was beschrieben wurde, automatisch zu generieren, ist Ziel dieser Arbeit.
Dabei soll auf der Basis von der in [Kamp & Reyle 93] vorgestellten Diskurs-Repräsentationstheorie zunächst ein Werkzeug erstellt werden, mit dem natürlichsprachlicher englischer Text eingelesen und in eine Diskurs-Repräsentationsstruktur überführt werden kann.
Diese Struktur soll in einem weiteren Schritt in eine geeignete Logiksprache übersetzt werden. Hierfür bietet sich die in [Schäfer 96] vorgestellte Unscharfe Metrisch-Temporale Logik (UMTL) an.
In einem letzten Schritt soll dann aus der so gewonnenen UMTL-Repräsentation das von [Jeyakumar 98] geforderte Eingabeformat zur Generierung von Bildfolgen erzeugt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Problemstellung

Immer mehr Untersuchungen im Bereich der Künstlichen Intelligenz beschäftigen sich mit der Frage, wie maschinelle Bildauswertung und maschinelle Sprachverarbeitung miteinander verbunden werden können [MacKevitt 95]. Dabei geht es zum einen darum, die mittels maschineller Bildauswertung erlangte rechnerinterne Repräsentation von Geschehen in einer Bildfolge in Form einer natürlichsprachlichen Beschreibung auszugeben und sie somit für den Menschen verständlicher zu machen [Gerber 95; Gerber 2000]. Ein anderer wünschenswerter Aspekt ist die gegenseitige Unterstützung von Sprach- und Bildeingabedaten zur Erstellung einer internen Repräsentation der in beiden Teilen enthaltenen Informationen. So kann beispielsweise eine textuelle Bildbeschreibung Hinweise auf die Art und den Ort von im Bild zu erkennenden Gegenstände geben, die dann von einem Bildauswertungssystem schneller zu finden sind. Eine dritte Möglichkeit schließlich besteht darin, die in einem natürlichsprachlichen Text enthaltene Information in ein Bild oder eine Bildfolge zu überführen. Diese Art der Integration von Sprache und Bild ist noch eher selten anzutreffen. [Nagel et al. 99] beschreiben, wie aus einer rechnerinternen Repräsentation des Geschehens einer Straßenszene eine synthetische Bildfolge erzeugt werden kann, die eben dieses Geschehen darstellt. Um bewertbare Ergebnisse zu erhalten, wurde dabei auf begriffliche Beschreibungen zurückgegriffen, die aus einer realen Bildfolge mit dem XTRACK-System [Koller 92; Kollnig 95; Haag 98] erzeugt wurden. Die generierte synthetische Bildfolge konnte so mit ihrem realen Vorgänger verglichen werden, um die Güte des gesamten Generierungsprozesses zu überprüfen und diesen gegebenenfalls zu verbessern.
Ziel dieser Studienarbeit soll es nun sein, aus einem englischsprachigen Text die darin beschriebenen Sachverhalte – im weiteren Verlauf Diskurs genannt – zu extrahieren und sie in die von [Jeyakumar 98] verwendete rechnerinterne Darstellung zu überführen. Hierzu müssen Werkzeuge entwickelt werden, die einen Text einlesen und die ihm zugrundeliegende grammatikalische Struktur erfassen und überprüfen können, so daß nur grammatikalisch und syntaktisch korrekte Eingabetexte verarbeitet werden. Dies um-

faßt sowohl die Erstellung geeigneter Wörterbücher als auch die Bereitstellung eines Zerteilers, der den Syntaxbaum des gerade zu behandelnden Satzes ausgibt. Bei der weiteren Verarbeitung soll auf die von [Kamp & Reyle 93] vorgestellte Diskurs-Repräsentationstheorie (DRT) zurückgegriffen werden, da diese erstens schon sehr detaillierte Verfahren zur Behandlung englischsprachiger Texte beinhaltet, und zweitens schon in mehreren Teilsystemen am Institut für Algorithmen und Kognitive Systeme der Fakultät für Informatik an der Universität Karlsruhe (TH) Verwendung findet.
Das Ergebnis dieses ersten Verarbeitungsschritts wird dann eine von [Kamp & Reyle 93] vorgestellte Diskurs-Repräsentationsstruktur (DRS) sein, die in einem weiteren Schritt in eine Formel der von [Schäfer 96] vorgestellten Unscharfen Metrisch-Temporalen Logik überführt werden soll.
Aus dieser UMTL-Repräsentation des Diskurses kann dann die für das von [Jeyakumar 98] erstellte Werkzeug benötigte Darstellung des Geschehens erzeugt werden. Schließlich kann das natürlichsprachlich beschriebene Geschehen als synthetische Bildfolge dargestellt werden.

1.2 Alternative Ansätze

Im folgenden wird zunächst ein Ansatz vorgestellt, aus einer natürlichsprachlichen Beschreibung ein Bild zu generieren. Das zweite vorgestellte System befaßt sich dann mit der Generierung von Bildfolgen. Ein dritter Ansatz schließlich behandelt die Ansteuerung von Animations-Werkzeugen mit natürlichsprachlichen Begriffen.

1.2.1 OVO-GENESYS – Ontology based Virtual Object GENErating SYStem

[Tijerino et al. 95] stellt das an der Universität von Kyoto entwickelte System OVO - GENESYS vor, das es ermöglicht, dreidimensionale Objekte mit natürlichsprachlichen Befehlen zu erzeugen und zu manipulieren. OVO-GENESYS entstand im Rahmen der Entwicklung eines Systems zur Bereitstellung eines rechnerunterstützten kooperativen Arbeitsraumes (*computer supported cooperative workspace*, CSCW). Als Anwendungsgebiet für das Gesamtsystem sind zum Beispiel Telekonferenzen denkbar, bei denen die Teilnehmer gemeinsam an der Konstruktion eines 3D-Objektes (z.B.: Auto) arbeiten. Die Eingabe von Befehlen erfolgt über ein Mikrofon. Somit wird zuerst das Sprachsignal mittels Spracherkennung in einen Text umgewandelt und danach aus diesem Text das betroffene Objekt und die vom Benutzer gewünschte Manipulation extrahiert. Schließlich wird die Manipulation durchgeführt und das Ergebnis angezeigt.
In einer Objektdatenbank wird die Form eines Objektes als Superquadrik abgespeichert. Diese Darstellungsform eignet sich gerade für deformierbare Objekte sehr gut, da sie einerseits mit einer geringen Anzahl von Parametern eine sehr große Formenvielfalt darzustellen gestattet, und andererseits eine Formänderung sehr leicht mit einer

Veränderung der Parameter verknüpft werden kann. Komplexere Objekte setzen sich aus mehreren deformierbaren Grundobjekten zusammen.
Bei der Interpretation der natürlichsprachlichen Eingabe werden Nomen mit Objekten und Verbformen mit Manipulationen assoziiert. So kann der Benutzer beispielsweise eine nach seinen Wünschen geformte Superquadrik erzeugen, indem er dabei den Bezeichner einer bestimmten Grundform benutzt. Grundformen sind dabei Kugel, Würfel, Pyramide, Zylinder, etc.. Die Superquadrik wird dann entsprechend der gewünschten Grundform vorparametrisiert. Zur Unterstützung der Spracheingabe arbeitet der Benutzer gleichzeitig noch mit einem Datenhandschuh, der ihm erlaubt, räumliche Attribute wie ,,hier", ,,dort", ,,in diese Richtung" sowie Vergrößerung und Verkleinerung durch Gesten auszudrücken.

1.2.2 AnimNL - Animation from Natural Language

[Badler et al.93] stellt das an der University of Pennsylvania entwickelte System AnimNL vor, das es erlaubt, aus natürlichsprachlichen Texten ganze Animationen zu generieren. Dabei sollen 3D-Modelle von Menschen mittels textueller Befehle und Befehlsfolgen in einer virtuellen 3D-Umgebung gesteuert werden.
Das Eingabeformat ist dabei vor allem auf direkte Anweisungen an das zu steuernde 3D-Modell hin ausgerichtet. Aus den im Eingabetext enthaltenen Informationen wird zunächst ein initialer Planungsgraph erstellt, der dann – abhängig von der internen Repräsentation der 3D-Umgebung – immer weiter verfeinert wird. Die feinste Stufe des Planungsgraphen sind dann schließlich Abfolgen von Grundoperationen (*low-level-behaviors*) wie Laufen, Greifen, Tragen, etc.. Solche Grundoperationen sprechen dann die Simulation des zu steuernden 3D-Modells an, das diese in konkrete Translationen und Rotationen einzelner Modell-Komponenten umsetzt.

1.2.3 Natürlichsprachliche Ansteuerung von Animations-Werkzeugen

[Zeltzer 82] beschreibt ein System, das es erlaubt, Grafikanimationen von Figuren auf der Basis von natürlichsprachlichen Begriffen zu erstellen. Figuren werden dabei als Skelette – also als dreidimensionale Anordnungen von verbundenen festen Segmenten – verstanden. Die Eingabe des Benutzers besteht aus der Beschreibung einer Bewegungsaufgabe (z.B.: *Go to the door and open it.*), die von dem System zunächst in eine Folge von Grundbewegungen (*skills*) umgesetzt wird. Jede dieser Grundbewegungen (z.B.: *walk, grasp, etc.*) wird von einem dafür vorgesehenen Bewegungsprogramm (*motor program*) gesteuert, das seinerseits wiederum lokale, die einzelnen Skelett-Teile bewegende Unterprogramme steuert. Bewegungsprogramme und deren Unterprogramme werden dabei durch endliche Automaten repräsentiert, in denen ein Zustand jeweils eine konstante Ansteuerung der ihm zugeordneten Freiheitsgrade des Skeletts bewirkt und

ein Übergang zu einem anderen Zustand einen Wechsel zu einer anderen Bewegungsart bedeutet. [Zeltzer 82] gibt als Beispiel hierfür die Geh-Bewegung eines menschlichen Skeletts an, für dessen Steuerung schon fünf Automaten ausreichend sind. Die genaue Parametrisierung der Einzelbewegungen – wie etwa Schrittlänge und Schrittdauer – kann hierbei den Automaten beim Aufruf übergeben werden, wodurch die Vielfalt der erzeugbaren Animationen weiter zunimmt.

Kapitel 2

Zerteilererzeugung

[Kamp & Reyle 93] beschreiben sehr genau, wie aus einer Folge von englischen Sätzen eine Diskurs-Repräsentationsstruktur (DRS) gebildet werden kann. Sie setzen dabei allerdings voraus, das jeder dieser Sätze in Form eines Syntaxbaums vorliegt. Um also auf die von [Kamp & Reyle 93] entwickelten Konstruktionsvorschriften zurückgreifen zu können, ist das erste Ziel zunächst, einen Zerteiler zu entwickeln, der zu einem Satz eines Fragments der englischen Sprache dessen Syntaxbaum erzeugt.

2.1 Zerteiler

Mit Hilfen einer Grammatik kann eine Sprache beschrieben werden. Dabei ist eine Grammatik nach [Schöning 95] folgendermaßen definiert.

Definition 2.1 *Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, das folgende Bedingung erfüllt. V ist eine endliche Menge, die Menge der Variablen. Σ ist eine endliche Menge, das Terminalalphabet. Es muß gelten: $V \cap \Sigma = \emptyset$. P ist die endliche Menge der Regeln oder Produktionen. Formal ist P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$. $S \in V$ ist die Startvariable.*

Die in der Grammatik angegebenen Regeln zeigen also, auf welche Art und Weise die vorhandenen Terminalsymbole der Sprache zu Sätzen kombiniert werden dürfen. Ein Zerteiler überprüft, ob eine Äußerung korrekt im Sinne der angegebenen Grammatik ist. Somit wird ein Zerteiler für eine natürliche Sprache wie Englisch immer nur das Fragment akzeptieren, das durch die Grammatikregeln festgelegt worden ist. Zusätzlich zur Überprüfung erzeugt der Zerteiler noch eine Baumrepräsentation der Struktur der Äußerung, einen Syntaxbaum.
Die Regeln einer einfachen Beispielgrammatik für ein Fragment der englische Sprache sind in Tabelle 2.1 angegeben. Dabei werden Satzzeichen vernachlässigt. Für den weiteren Verlauf wird die Konvention getroffen, Terminale und Variablen in der Schriftart

SENTENCE	→	NOUNPHRASE VERBPHRASE
NOUNPHRASE	→	PROPERNAME | DETERMINER NOUN
VERBPHRASE	→	VERB | VERB NOUNPHRASE
DETERMINER	→	a | the
PROPERNAME	→	john | durlacher-strasse | karlsruhe
NOUN	→	truck | intersection | lane | cars
VERB	→	crosses | brakes | drive | follow

Tabelle 2.1: Beispielgrammatik für ein einfaches Fragment der englischen Sprache

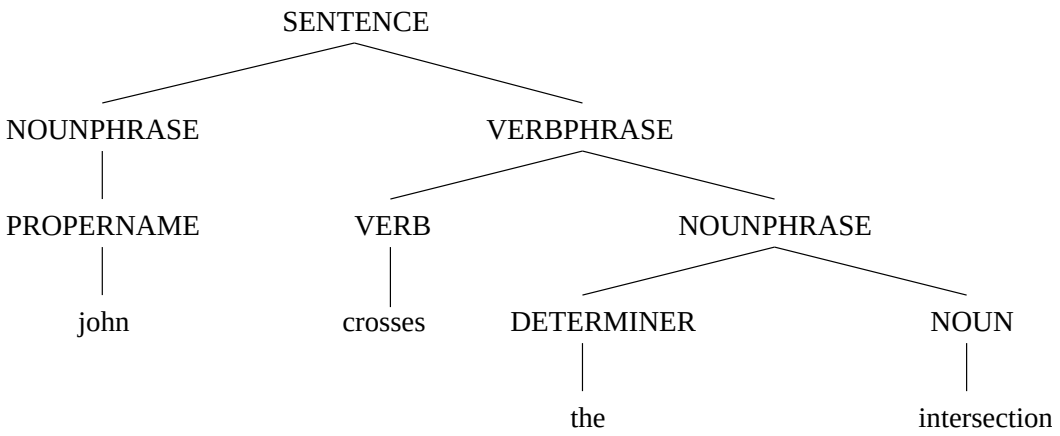


Abbildung 2.1: Syntaxbaum zum Satz `john crosses the intersection`

`TypeWriter` darzustellen. Terminale werden dabei stets mit Kleinbuchstaben, Variablen dagegen mit Großbuchstaben geschrieben. Das Terminalalphabet Σ entspricht {`a, the, john, durlacher-strasse, karlsruhe, cars, truck, lane, intersection, brakes, crosses, drive, follow`}.
Die Menge V der Variablen, die auch Nichtterminale genannt werden, entspricht in diesem Falle {`SENTENCE, NOUNPHRASE, VERBPHRASE, DETERMINER, PROPERNAME, NOUN, VERB`}. Die Startvariable S ist hier `SENTENCE`. Ein Zerteiler für die angegebene Grammatik kann nun einfache englische Sätze, die aus Wörtern des Terminalalphabets Σ bestehen, akzeptieren und dazu Syntaxbäume erstellen. In Abbildung 2.1 sieht man einen Syntaxbaum für den Satz `john crosses the intersection`.

Mit diesem Zerteiler können zum Beispiel folgende Sätze akzeptiert werden.

```
the cars drive
a truck brakes
john crosses the lane
the cars follow a truck
```

Folgende Sätze würden aber allerdings auch akzeptiert werden.

```
the cars crosses the intersection
john drive
```

In den beiden letzteren Sätzen stimmt der Numerus des Subjekts nicht mit dem des Prädikats überein. [Kamp & Reyle 93] führen zur Auflösung dieser Konflikte attributierte Nichtterminale ein. Jedem Nichtterminal wird ein Attribut **number** zugewiesen, das den Numerus des Nichtterminals repräsentiert. **number** kann die Werte **SINGULAR** und **PLURAL** annehmen (Attribute und deren Ausprägungen werden im weiteren Verlauf in der Schriftart **Bold Face** dargestellt. Attributnamen werden dabei in Kleinbuchstaben, Ausprägungen dagegen in Großbuchstaben geschrieben.). Nun kann man fordern, daß ein Satz nur dann akzeptiert wird, wenn die Nichtterminale `NOUNPHRASE` und `VERBPHRASE` im Attribut **number** übereinstimmen. Bei den Regeln, die ein Nichtterminal auf ein Terminal abbilden, muß man dieses Attribut auf eine konkrete Ausprägung setzen (siehe Tabelle 2.2). In den Regeln wird jedem Nichtterminal eine Liste mit Bedingungen hinzugefügt. Dabei kann man entweder ein Attribut mit einer konkreten Ausprägung oder aber mit einer Variablen vergleichen. Um festzustellen, ob eine Regel ausgeführt werden kann, vergleicht der Zerteiler all die Attribute der Nichtterminale auf der rechten Regelseite, die dieselbe Variable besitzen. Ist dieser Vergleich erfolgreich, so weist der Zerteiler den Attributen des Nichtterminals der linken Regelseite die Werte zu.
Mit zunehmender Komplexität der Grammatik kann es nötig sein, weitere Attribute hinzuzufügen. Dabei kann es vorkommen, daß einige Attribute nur von bestimmten Nichtterminalen benutzt werden. So kann das Nichtterminal `VERB` in einer komplexeren Grammatik das Attribut **transitivity** besitzen, welches angibt, ob ein Verb ein direktes Objekt besitzt oder nicht. Dieses Attribut ist für das Nichtterminal `DETERMINER` ohne Bedeutung, da ein Artikel diese Eigenschaft nicht besitzt. Eine komplette Auflistung der Attribute der in dieser Studienarbeit verwendeten Grammatik findet sich im Anhang A.1.2.
Betrachtet man nun die Wortregeln, also all die Regeln, die ein Nichtterminal auf ein Terminal, ein Wort der natürlichen Sprache, abbilden, so stellt man fest, daß mit einer Erweiterung der Attribute die Anzahl dieser Regeln sehr stark zunimmt. Führt man zum Beispiel das Attribut **person** mit den möglichen Ausprägungen **P1ST**, **P2ND** und **P3RD** für die erste, zweite und dritte Person eines Verbs ein, so verdreifacht sich die Anzahl der Wortregeln für das Nichtterminal `VERB`. Um hier die Regelmenge einzuschränken führt man ein Lexikon ein, das alle Terminale beinhaltet. Dabei werden im Lexikon zu jedem Terminal auch eine Liste von allen Attributbelegungen, wie sie

$$\underset{\left[\ number = a\ \right]}{\texttt{SENTENCE}} \rightarrow \underset{\left[\ number = a\ \right]}{\texttt{NOUNPHRASE}}\ \underset{\left[\ number = a\ \right]}{\texttt{VERBPHRASE}}$$

$$\underset{\left[\ number = a\ \right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\ number = a\ \right]}{\texttt{DETERMINER}}\ \underset{\left[\ number = a\ \right]}{\texttt{NOUN}}$$

$$\underset{\left[\ number = a\ \right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\ number = a\ \right]}{\texttt{PROPERNAME}}$$

$$\underset{\left[\ number = a\ \right]}{\texttt{VERBPHRASE}} \rightarrow \underset{\left[\ number = a\ \right]}{\texttt{VERB}}\ \underset{\left[\ number = b\ \right]}{\texttt{NOUNPHRASE}}$$

$$\underset{\left[\ number = a\ \right]}{\texttt{VERBPHRASE}} \rightarrow \underset{\left[\ number = a\ \right]}{\texttt{VERB}}$$

$$\underset{\left[\ number = SINGULAR\ \right]}{\texttt{NOUN}} \rightarrow \texttt{truck}, \texttt{intersection}, \texttt{lane}$$

$$\underset{\left[\ number = PLURAL\ \right]}{\texttt{NOUN}} \rightarrow \texttt{cars}$$

$$\underset{\left[\ number = SINGULAR\ \right]}{\texttt{VERB}} \rightarrow \texttt{crosses}, \texttt{brakes}$$

$$\underset{\left[\ number = PLURAL\ \right]}{\texttt{VERB}} \rightarrow \texttt{drive}, \texttt{follow}$$

$$\underset{\left[\ number = SINGULAR\ \right]}{\texttt{PROPERNAME}} \rightarrow \texttt{john}, \texttt{durlacher} - \texttt{strasse}, \texttt{karlsruhe}$$

$$\underset{\left[\ number = PLURAL\ \right]}{\texttt{PROPERNAME}} \rightarrow$$

$$\underset{\left[\ number = SINGULAR\ \right]}{\texttt{DETERMINER}} \rightarrow \texttt{a}, \texttt{the}$$

$$\underset{\left[\ number = PLURAL\ \right]}{\texttt{DETERMINER}} \rightarrow \texttt{the}$$

Tabelle 2.2: Attributierte Beispielgrammatik für die einfachen Regeln aus Tabelle 2.1

in der bisherigen Grammatik angegeben waren, abgelegt. Zusätzlich erhalten die Terminale das Attribut **kind**, das angibt aus welchem Nichtterminal sie abgeleitet werden können. Die Regeln werden nun in der Form $\texttt{Nichtterminal}_{[\ ]} \rightarrow \texttt{LEX}_{[\ ]}$ geschrieben. Die Wortregel und der Lexikoneintrag des Nichtterminals `VERB` für die in Tabelle 2.2 angegebene Grammatik sind in Tabelle 2.3 exemplarisch für das Wort `follow` dargestellt. Somit wird die Menge der Regeln verringert. Die Menge der Einträge zu einem Wort im Lexikon kann mit Hilfe von Morphologie vermindert werden.

$$VERB_{[\,]} \rightarrow LEX_{[\,]}$$

Lexikoneintrag für die Worte follow und follows

$$\texttt{follow} \begin{bmatrix} \begin{bmatrix} kind = VERB \\ number = SINGULAR \\ person = P1ST \end{bmatrix}, \begin{bmatrix} kind = VERB \\ number = SINGULAR \\ person = P2ND \end{bmatrix} \\ \begin{bmatrix} kind = VERB \\ number = PLURAL \\ person = P1ST \end{bmatrix}, \begin{bmatrix} kind = VERB \\ number = PLURAL \\ person = P2ND \end{bmatrix}, \begin{bmatrix} kind = VERB \\ number = PLURAL \\ person = P3RD \end{bmatrix} \end{bmatrix}$$

$$\texttt{follows} \begin{bmatrix} kind = VERB \\ number = SINGULAR \\ person = P3RD \end{bmatrix}$$

Tabelle 2.3: Wortregel der attributierten Grammatik mit Lexikoneintrag für das Terminal follow

2.1.1 Morphologie

Die Morphologie beschreibt, wie aus einem Wortstamm eine finite Wortform gebildet werden kann. Diese Bildung kann durch Hinzufügen bestimmter Endungen bzw. Voranstellen von Vorsilben bei einem Wort geschehen, oder auch durch die Veränderung des Stammes selbst. Morphologieregeln geben an, wie eine finite Form aus einem Stamm gebildet wird und welchen Einfluß diese hat. [Krovetz 99] unterscheidet zwei Arten von Morphologie, ableitende und beugende. Durch ableitende Morphologie können die Wortart und die Bedeutung eines Wortes verändert werden, wie z.B. im Englischen durch Anhängen von “ly” aus einem Adjektiv ein Adverb werden kann. Beugende Morphologie verändert die syntaktische Rolle eines Wortes. So wird in der englischen Sprache durch Anhängen von “s” an ein Nomen kenntlich gemacht, daß es sich um den Plural des Nomens handelt. Die syntaktische Rolle eines Wortes wird nicht allein durch das Anfügen einer Endung, sondern auch durch die Wortart des Stammes bestimmt. So beschreibt die Endung “s” bei einem Nomen, daß es sich um ein Pluralnomen handelt, bei einem Verb hingegen, daß es in der dritten Person Singular steht. Wörter, deren syntaktische Rolle sich aus den Morphologieregeln ableiten läßt, werden als regelmäßig bezeichnet.
Die Morphologieeigenschaften können nun zur Reduzierung der Einträge im Lexikon verwendet werden. Die syntaktische Rolle einer finiten Form entspricht der Belegung ihrer Attribute. Somit ist es ausreichend für regelmäßige Wörter nur deren Wortstamm einzutragen. Alle weiteren Wortformen und deren Attributbelegungen ergeben sich aus den Morphologieregeln. Eine Morphologieregel für Verben ist das Anhängen des Buchstabens “s” für die dritte Person Singular. Man kann nun eine Funktion $regularForms_{Nichtterminal}(\texttt{Wortstamm})$ definieren, die zu einem Wortstamm alle Attributbelegungen und die zugehörigen Bezeichnungen liefert. Für den Wortstamm `follow`

$$regularForms_{VERB}(\texttt{follow}) = \left\{ \begin{array}{ll} \left(\texttt{follow}, \left[\begin{array}{l} kind = VERB \\ number = SINGULAR \\ person = P1ST \end{array}\right]\right), & \left(\texttt{follow}, \left[\begin{array}{l} kind = VERB \\ number = SINGULAR \\ person = P2ND \end{array}\right]\right) \\ \left(\texttt{follows}, \left[\begin{array}{l} kind = VERB \\ number = SINGULAR \\ person = P3RD \end{array}\right]\right), & \left(\texttt{follow}, \left[\begin{array}{l} kind = VERB \\ number = PLURAL \\ person = P1ST \end{array}\right]\right) \\ \left(\texttt{follow}, \left[\begin{array}{l} kind = VERB \\ number = PLURAL \\ person = P2ND \end{array}\right]\right), & \left(\texttt{follow}, \left[\begin{array}{l} kind = VERB \\ number = PLURAL \\ person = P3RD \end{array}\right]\right) \end{array} \right\}$$

Tabelle 2.4: Die Attributbelegungen und zugehörigen Bezeichnungen zum Wortstamm `follow`

o	→	oes
ss	→	sses
sh	→	shes
ch	→	ches
x	→	xes
Vokal y	→	*Vokal* ys
Konsonant y	→	*Konsonant* ies
ϵ	→	s

Tabelle 2.5: Regeln zur Änderung der Endung eines Wortstammes für die dritte Person Singular Präsens bei Verben

würde die Funktion $regularForms_{VERB}$ das in Tabelle 2.4 dargestellte Ergebnis liefern. Im Lexikon benötigt man dann nur noch einen Eintrag, der das Nichtterminal zum Wortstamm setzt.

$$\texttt{follow}[kind = VERB]$$

Dafür müssen zu jedem Nichtterminal Morphologieregeln angegeben werden, welche die Änderung des Stammes und die zugehörigen Veränderungen der Attributbelegungen beschreiben. Im Englischen verändern die meisten Morphologieregeln die Endung des Wortstammes. Man kann für diese Bildungen nun eine Grammatik angeben, die sich nur auf das Ende eines Wortes bezieht. Die Regeln zur Bildung der dritten Person Singular Präsens von Verben sind in der Tabelle 2.5 angegeben. Besitzt ein Wort jedoch unregelmäßige Bildungsformen, so müssen diese im Lexikon aufgenommen werden, da sie nicht über die Funktion $regularForms_{Nichtterminal}$ erreichbar sind.
Der eingegebene Text, den der Zerteiler verarbeiten soll, enthält allerdings nur finite Wortformen, nicht jedoch Wortstämme. Da im Lexikon nur die Wortstämme eingetragen sind, muß zunächst die finite Wortform mit Hilfe einer Funktion $getStems_{Nichtterminal}$ auf einen Stamm abgebildet werden. Diese Funktion ist die Rückabbildung zur oben beschriebenen Funktion $regularForms_{Nichtterminal}$. Allerdings ist *getStems* keine eindeutige Abbildung. So führen die folgenden beiden Morphologieregeln für die Vergangenheits-

form eines Verbs im Englischen zu Mehrdeutigkeiten.

e	→	ed
ϵ	→	ed

Versucht man zum Beispiel zum Wort `followed` den Stamm zu bilden, so muß man sich für die untere der beiden Morphologieregeln entscheiden, um auf den Wortstamm `follow` zu kommen. Bei dem Wort `braked`, müßte allerdings die obere Morphologieregel benutzt werden, um den Stamm `brake` zu erhalten. Wie man sieht, ist anhand der Endung der Wortform allein nicht entscheidbar, welche der beiden Morphologieregeln angewandt werden soll. Eine Lösung besteht darin, daß die Funktion *getStems* alle möglichen Stämme ermittelt. Auch die finite Wortform kann ein möglicher Stamm sein, da im Englischen der Stamm zur Bildung der finiten Wortform für das Präsens von Verben nicht verändert wird. Zu `followed` wären die möglichen Stämme `followe`, `follow` und `followed`.
Um entscheiden zu können, welche Attributbelegungen die finite Form besitzt, werden zu jedem der möglichen Stämme, die auch im Lexikon vorhanden sind, mit Hilfe der Funktion *getRegularForms* alle regulären Formen ermittelt. Die Attributbelegungen der Wortformen, bei denen die Bezeichnung mit der ursprünglichen Wortform übereinstimmt, werden vom Zerteiler übernommen. Liest der Zerteiler zum Beispiel die Wortform `follows` ein, so bildet er zunächst die möglichen Stämme `follow` und `follows`. Im Lexikon befindet sich allerdings nur ein Eintrag zu dem Stamm `follow`. Zu diesem Stamm werden die in Tabelle 2.4 dargestellten regulären Formen gebildet. Die Attributbelegung mit der Bezeichnung `follows` wird übernommen.
Es wurde die Annahme getroffen, daß *getRegularForms* eine eindeutige Abbildung ist. Würde *getRegularForms* nicht eindeutig abbilden, so würde der Zerteiler falsche Wortformen wie zum Beipiel `crosss` akzeptieren. Die gebildeten Stämme von `crosss` sind `cross` und `crosss`. Im Lexikon ist der Stamm `cross` vorhanden, und es würden als gültige Bezeichnungen für die dritte Person Singular sowohl `crosses` als auch `crosss` vergeben werden, da es zum einen die Morphologieregel `ss` → `sses` als auch die Regel ϵ → `s` gibt. Ein Nachteil dieser eindeutigen Abbildung tritt bei Morphologieregeln auf, die einen Konsonanten am Ende verdoppeln und ein Endung anhängen. Solch ein Fall tritt zum Beispiel bei der Vergangenheitsform von Verben auf.

Konsonant1 Vokal Konsonant2	→	*Konsonant1 Vokal Konsonant2 Konsonant2* ed
ϵ	→	ed

Die Vergangenheitsform von `enter` wird mit dieser Morphologieregel zu `enterred`. Läßt man jedoch diese Morphologieregel weg, so würde zu `stop` die Vergangenheitsform `stoped` lauten. Allein anhand der Schreibweise der Wörter läßt sich nicht entscheiden, ob der letzte Konsonant verdoppelt werden muß. Diese Entscheidung kann nur mit Hilfe einer Lautschrift erfolgen, da der letzte Konsonant verdoppelt wird, wenn der Vokal davor kurz ausgesprochen wird. Somit muß die Wortform `entered` als irreguläre Wortform eingetragen werden, wenn die obige Morphologieregel benutzt wird. Wird

diese Regel hingegen weggelassen, so muß die Wortform `stopped` als irregulär ins Lexikon eingetragen werden.

2.2 Erzeugung des Zerteilers

Im letzten Abschnitt wurde vorgestellt, wie mit Hilfe einer attributierten Grammatik und eines Lexikons die Struktur von englischsprachigen Sätzen formal beschrieben werden kann. Im folgenden wird gezeigt, wie aus einer solchen formalen Beschreibung ein Zerteiler in Form eines Programms automatisch generiert werden kann.

2.2.1 JavaCC \ JJTree

JavaCC, Java Compiler Compiler von MetaMata, ist ein Zerteilererzeuger für kontextfreie Grammatiken. Erzeugt wird der Java Quelltext für den Zerteiler zur angegebenen Grammatik. Die JavaCC Syntax ist eine Erweiterung der Java Syntax. Grammatikalische Regeln werden in Methodenform geschrieben, wobei der Name einer Methode dem Nichtterminal der linken Seite einer Regel entspricht. Die Nichtterminale der rechten Seite der Regel werden dabei als Methodenaufrufe geschrieben. Die Syntax erlaubt es auch, Java Anweisungen in die Regeln mit einzubinden. Ohne zusätzliche Java Anweisungen kann der erzeugte Zerteiler nur entscheiden, ob eine eingegebene Zeichenfolge ein Element der Sprache ist. JJTree ist ein Vorverarbeitungswerkzeug für JavaCC, das die Grammatikregeln mit Java Anweisungen anreichert, so daß der dann mit JavaCC erzeugte Zerteiler zu jeder akzeptierten Zeichenfolge auch deren Syntaxbaum angibt.
JavaCC ist nicht in der Lage, attributierte Nichtterminale zu verwenden. Durch Hinzufügen von Nebenbedingungen in Form von Java Anweisungen in die Regelmethoden kann jedoch auch eine attributierte Grammatik auf JavaCC Quelltext abgebildet werden. Diese Nebenbedingungen direkt in den JavaCC Quelltext zu schreiben ist jedoch sehr fehleranfällig und aufwendig, da auch bei nur kleinen Änderungen der Grammatik der Aufwand für die nötigen Änderungen im JavaCC Quelltext schon relativ groß ist. Deshalb empfiehlt es sich die Abbildung der attributierten Grammatik automatisch durchführen zu lassen.

2.2.2 Erweiterung auf attributierte Grammatiken

Zur automatischen Abbildung von attributierten Grammatiken auf JJTree Quelltext wurde ein Werkzeug entwickelt. Als Eingabe fordert dieses Werkzeug die Spezifikation einer attributierten Grammatik in Form einer Textdatei. Die Spezifikation der attributierten Grammatik innerhalb dieser Textdatei besteht aus vier Abschnitten.

- **NONTERMINALS** In diesem Abschnitt werden alle in der Grammatik verwendeten Nichtterminale aufgeführt. Das Nichtterminal `STARTSYMBOL` muß hierbei stets vorkommen.

- **ATTRIBUTES** Hier werden alle Attribute mit ihren möglichen Ausprägungen aufgeführt.

- **WORDPROPERTIES** Dieser Abschnitt enthält diejenigen Nichtterminale, die direkt auf das Lexikon abgebildet werden. Zusätzlich wird zu jedem dieser Nichtterminale eine Liste von Attributen angegeben, die es besitzt.

- **RULES** Hier werden die Regeln der attributierten Grammatik angegeben. Dabei muß es stets mindestens eine Ableitungsregel für das Nichtterminal `STARTSYMBOL` geben. Bei Regeln, die ein Nichtterminal auf das Lexikon abbilden, kann zusätzlich noch ein Abschnitt für die Morphologie der Wortart, die durch dieses Nichtterminal repräsentiert wird, folgen.

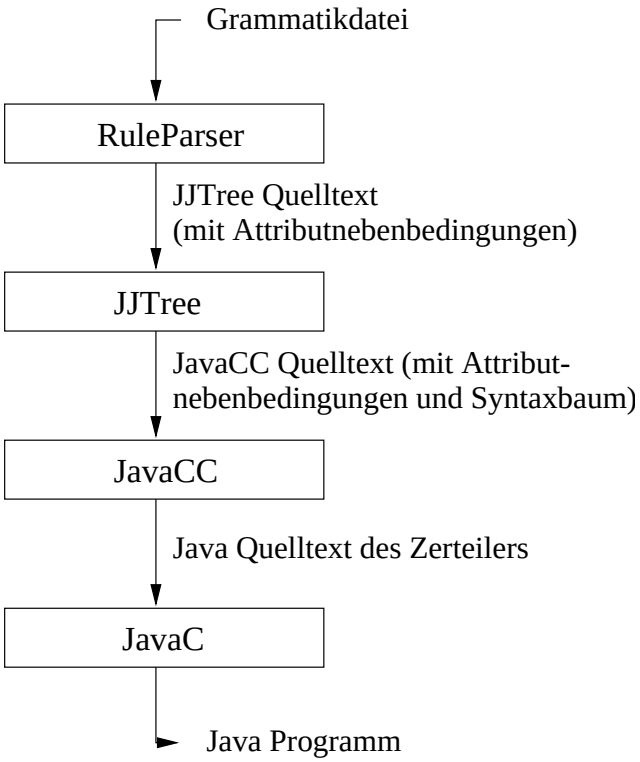


Abbildung 2.2: Übersetzungsabfolge von einer Grammatikdatei bis zu dem zugehörigen Java Zerteilerprogramm

Um aus einer solchen Grammatikdatei einen Zerteiler zu erzeugen, übersetzt man mittels des entwickelten Werkzeuges die Grammatikdatei in JJTree Quelltext, anschließend mittels JJTree in JavaCC Quelltext, mittels JavaCC in Java Quelltext und schließlich mittels JavaC in Binärtext, der von der Java Virtual Machine ausgeführt werden kann. Diese Übersetzungsabfolge wird in Abbildung 2.2 dargestellt.
So läßt sich nur durch Angabe der Grammatikdatei in der oben beschriebenen Form ein Zerteiler erzeugen. Dies bietet zum einen den Vorteil, daß die Grammatik leicht um neue Regeln und Attribute erweitert werden kann. Zum anderen können Fehler bei der Eingabe der Grammatik schon bei der Verarbeitung der Grammatikdatei entdeckt werden. Schließlich lehnt sich die Regelschreibweise der Grammatikdatei sehr stark an die Notation von [Kamp & Reyle 93] an, was eine Auseinandersetzung mit der Implementierung dieser attributierten Grammatik direkt in JJTree Quelltext überflüssig macht. Somit ist es einfacher eine Grammatik zu entwickeln und zu testen.

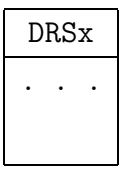
Kapitel 3

Diskurs-Repräsentationstheorie

Im vorhergehenden Kapitel wurde gezeigt, wie mit Hilfe einer attributierten Grammatik und eines dazu erstellten Zerteilers der Syntaxbaum eines englischsprachigen Satzes, der von der Grammatik erfaßt wird, erzeugt werden kann. Im folgenden soll beschrieben werden, wie die von [Kamp & Reyle 93] eingeführte Diskurs-Repräsentationsstruktur (DRS) aus solch einem Syntaxbaum gebildet werden kann.

3.1 DRS

Eine DRS ist definiert als eine abstrakte Struktur, die die Interpretation eines Textes beinhaltet. Eine solche Struktur besteht aus einer Menge von Diskursbezugsträgern (*discourse referent*, DR) und einer Menge von Bedingungen. Die Menge der Diskursbezugsträger wird das Universum der DRS genannt, und beinhaltet Bezugsträger für Individuen, für Mengen von Individuen sowie für Ereignisse. Die Bezugsträger sind die Variablen der DRS. Die Bedingungen einer DRS repräsentieren Eigenschaften, die ein Bezugsträger besitzt, oder die Beziehungen, die zwischen den Bezugsträgern bestehen. Eine DRS sei im folgenden stets mit `DRSx` bezeichnet, wobei `x` die eindeutige Nummer der DRS sei. Insgesamt wird eine DRS dann in der Form



dargestellt.

3.1.1 Individuen-Bezugsträger

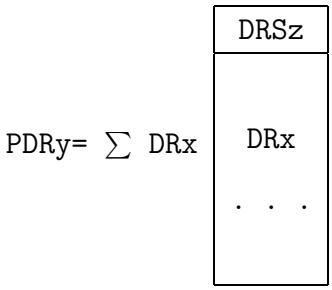
Als Individuen-Bezugsträger werden die Bezugsträger bezeichnet, die als Platzhalter für ein Individuum des zu repräsentierenden Diskursbereichs in die DRS eingeführt werden. Individuen-Bezugsträger seien im folgenden mit `DRx` bezeichnet.

3.1.2 Plural-Bezugsträger

Plural-Bezugsträger sind Platzhalter für eine Menge von Individuen des Diskursbereichs, die im folgenden stets mit `PDRx` bezeichnet werden. Sie können als explizite Zusammenfassung von Individuen-Bezugsträgern und/oder Plural-Bezugsträgern definiert werden, was mit

$$\texttt{PDRy} = \bigoplus \texttt{(DRu, ... DRv, PDRw, ..., PDRx)}$$

bezeichnet werden soll. Außerdem ist es auch möglich, eine implizite Definition eines Plural-Bezugsträgers vorzunehmen, indem die Eigenschaften der Elemente der Menge mittels einer DRS beschrieben werden. Dies soll mit



bezeichnet werden. Die dritte Möglichkeit der Definition eines Plural-Bezugsträgers besteht in der Quantifizierung eines anderen Plural-Bezugsträgers. Beispiele für Quantoren sind hierbei Zahlwörter (wie `two, three ...`), aber auch allgemeine Quantoren (wie `some, most, all, ...`). Dies wird durch

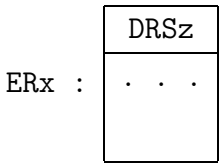
$$\texttt{PDRy = <quantor> (PDRx)}$$

ausgedrückt und bedeutet, daß die neu definierte Menge Teilmenge der quantorisierten Menge ist, und zwar in genau dem Teilmengenverhältnis, das der Quantor angibt.

3.1.3 Ereignis-Bezugsträger

Ereignisbezugsträger sind Platzhalter für den Zeitpunkt oder das Zeitintervall, an oder in dem ein bestimmtes Ereignis des Diskurses stattfand. Sie werden stets mit `ERx`

bezeichnet. Eine mögliche Definition eines Ereignis-Bezugsträgers ist die explizite Bindung an eine DRS, was mit



bezeichnet werden soll. Eine weitere Möglichkeit besteht in der Zusammenfassung mehrerer schon bestehender Ereignis-Bezugsträger zu einem neuen, zeitlich die anderen umfaßenden Bezugsträger. Dies sei durch

```
ERx = [ERy, ..., ERz]
```

dargestellt. Die letzte mögliche Definition eines Ereignis-Bezugsträgers wird für die Darstellung von explizit gegebenen Zeiträumen (etwa `two minutes`) benötigt. Dies sei mit

```
ERx = DURATION(PDRy)
```

bezeichnet.

3.1.4 Bedingungen

Die Bedingungen einer DRS stellen sowohl Eigenschaften von Bezugsträgern als auch Beziehungen dieser untereinander dar. Hierbei lassen sich zunächst drei Arten von Bedingungen unterscheiden. Zum einen können Bedingungen in Form von Attributen, dargestellt durch `<Attribut>(DRx)` oder `<Attribut>(PDRx)`, gegeben sein, die eine Eigenschaft eines Individuen- bzw. Plural-Bezugsträgers repräsentieren. Eine zweite Form der Bedingungen sind Prädikate in der Form `<Prädikat>(DRu, ..., DRv, PDRw, ..., PDRx)`, die eine Beziehung zwischen den enthaltenen Bezugsträgern ausdrücken. Die dritte Form von Bedingungen sind schließlich Namensbedingungen, dargestellt durch `<Name>(DRx)`, die stets einem Individuen-Bezugsträger einen Eigennamen zuweisen. Von diesen drei Bedingungsarten unterscheidet sich eine vierte, die als Ereignisrelation bezeichnet wird. Diese drückt die zeitliche Beziehung zwischen zwei Ereignis-Bezugsträgern aus und wird durch `<Relation>(ERx, ERy)` dargestellt.

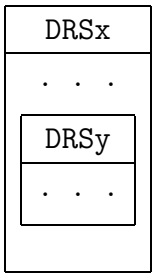
3.1.5 Duplex-Bedingungen

Eine Sonderform der Bedingung ist die Duplex-Bedingung. Sie besteht aus zwei DRSn, die durch einen Quantor verbunden sind. Dies sei durch

dargestellt. Die linke DRS der Duplex-Bedingung ist dabei als Mengendefinition aufzufassen. Aus dieser Menge wird dann für eine durch den Quantor (hier allgemein `Q`) bestimmte Teilmenge von Elementen der durch die rechte DRS repräsentierte Diskurs gefolgert.

3.1.6 Bedingungen in DRS-Form

Eine weitere Sonderstellung nehmen Bedingungen ein, die ihrerseits wieder DRSn sind. Diese Unter-DRSn können negiert oder nicht-negiert auftreten, je nachdem, ob sie einen negierten oder nicht-negierten Teildiskurs repräsentieren, verhalten sich aber ansonsten wie jede andere DRS. Eine Unter-DRS `DRSy` innerhalb einer DRS `DRSx` wird im folgenden stets durch



dargestellt.

3.2 Konstruktionsregeln

Abbildung 3.1 zeigt eine DRS für den Satz `john crosses the intersection`. Diese DRS besitzt wegen der Einfachheit des vorliegenden Satzes noch keine Bedingungen, die selbst wieder DRSn enthalten. Im folgenden soll nun gezeigt werden, wie aus dem Syntaxbaum eines englischen Satzes mit Hilfe von Konstruktionsregeln eine DRS erzeugt werden kann.
Die Umwandlung eines Syntaxbaumes in eine DRS beginnt mit der Erzeugung einer zunächst leeren DRS, in die der Syntaxbaum eingefügt wird. Danach werden schrittweise sogenannte Konstruktionsregeln (*Construction Rule*, CR) auf den sich nun innerhalb einer DRS befindenden Syntaxbaum angewandt. Eine solche Konstruktionsregel

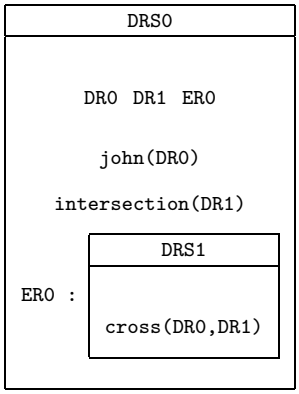


Abbildung 3.1: Beispiel einer einfachen DRS für den Satz : `john crosses the intersection`. `DR0` und `DR1` sind Individuenbezugsträger, `ER0` bezeichnet einen Ereignisbezugsträger. `john(DR0)` weist dem Bezugsträger `DR0` den Namen `john` zu, während `intersection(DR1)` als einstelliges Prädikat die Eigenschaft von `DR1` ausdrückt, `intersection` zu sein. Die letzte Zeile der DRS stellt die Ereignisbedingung für den Ereignisbezugsträger `ER0` dar.

besteht aus einer auslösenden Struktur (*Triggering Structure*, TS) und einem Aktionsteil. Die auslösende Struktur wird als Baumstruktur definiert. Wird diese Struktur im zu bearbeitenden Syntaxbaum gefunden, ist die Konstruktionsregel an dieser Stelle des Syntaxbaumes anwendbar und der Aktionsteil der Regel wird ausgeführt. Der Aktionsteil einer Konstruktionsregel kann sowohl aus Aktionen bestehen, die den Syntaxbaum verändern, als auch Aktionen enthalten, die die DRS erweitern. Veränderungen des Syntaxbaumes sind das Löschen, das Einfügen, aber auch das Umhängen von Knoten bzw. Teilbäumen. Erweiterungen der DRS sind das Hinzufügen neuer Bezugsträger und Bedingungen, wobei auf schon bestehende Bezugsträger Bezug genommen werden kann, etwa um den Rückbezug eines Pronomens herzustellen. Werden bei der Anwendung einer Konstruktionsregel DRS-Bedingungen erzeugt, die ihrerseits wieder DRSn enthalten, werden zunächst diese neuen Strukturen auf die Anwendbarkeit einer Konstruktionsregel hin untersucht. Ist in einer DRS ohne weitere Unter-DRSn keine Konstruktionsregel mehr anwendbar und besitzt sie keinen Syntaxbaum mehr, so gilt sie als reduziert. Eine DRS mit Unter-DRSn oder Bedingungen, die DRSn enthalten, gilt als reduziert, wenn alle ihre Unter-DRSn reduziert sind, sie selbst keinen Syntaxbaum mehr besitzt und in ihr keine Konstruktionsregel mehr anwendbar ist.

Um zum Beispiel aus dem Syntaxbaum des Satzes `john crosses the intersection` die in Abbildung 3.1 gezeigte DRS zu erzeugen, ist zunächst eine Konstruktionsregel nötig, die zu dem Eigennamen `john` einen neuen Individuenbezugsträger `DR0` und die Namensbedingung `john(DR0)` erzeugt. Eine solche Konstruktionsregel ist in Abbildung 3.2 zu sehen.

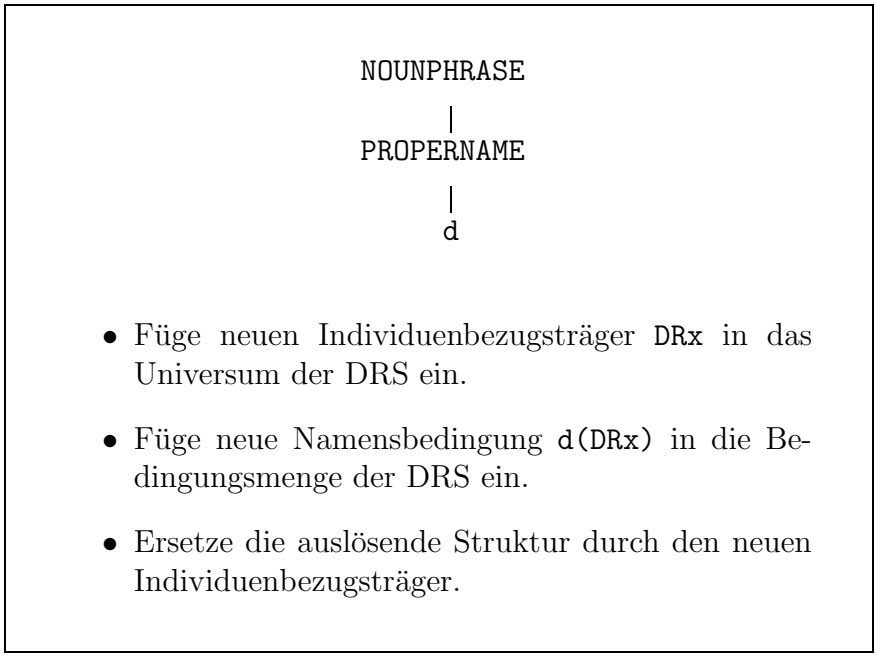
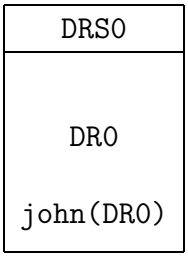


Abbildung 3.2: Eine erste Konstruktionsregel zur Behandlung von Eigennamen. Im oberen Teil der Regel ist die auslösende Struktur zu sehen. Im unteren Teil schließt sich der Aktionsteil der Regel an. d ist ein Platzhalter für ein Terminal der Sprache.

Die Anwendung dieser Regel ergibt die vorläufige DRS:



und den modifizierten Syntaxbaum:

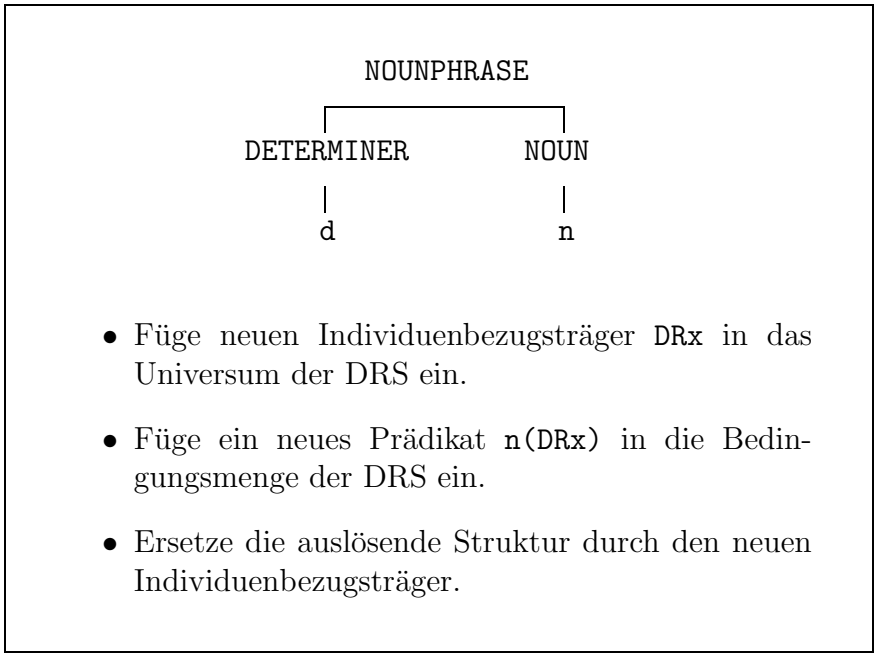
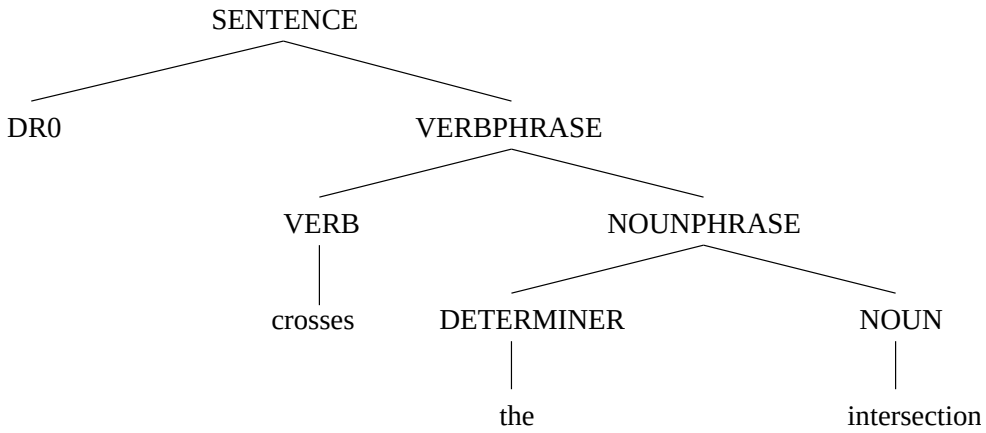
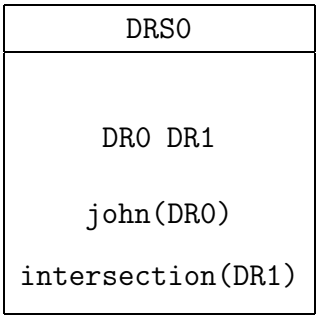


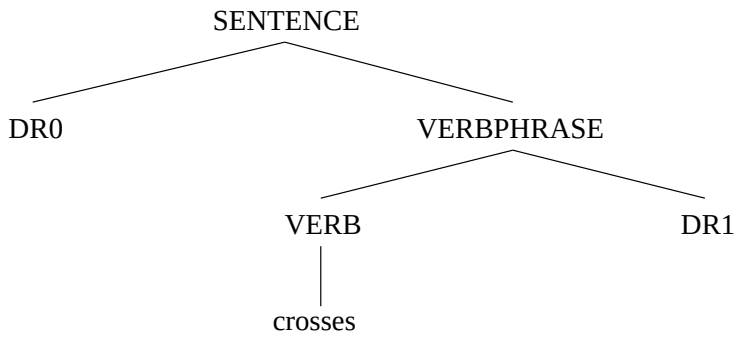
Abbildung 3.3: Diese Konstruktionsregel behandelt Nominalphrasen, die aus einem Artikel und einem Nomen bestehen. `d` und `n` sind Platzhalter für Terminale der Sprache.



Eine weitere Konstruktionregel behandelt Nominalphrasen, die aus einem Artikel und einem Nomen bestehen. Sie ist in Abbildung 3.3 zu sehen. Die Anwendung dieser Regel führt zu der neuen DRS:



und zu dem nun noch weiter reduzierten Syntaxbaum:

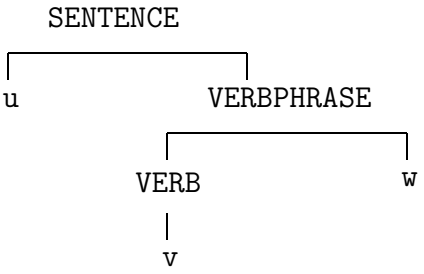


Dieser jetzt noch verbleibende Syntaxbaum läßt sich mit nur einer weiteren Konstruktionsregel, die in Abbildung 3.4 zu sehen ist, vollständig abarbeiten. Nach diesem Arbeitsschritt ist schließlich die endgültige DRS aus Abbildung 3.1 für den Ausgangssatz erstellt.

3.2.1 Erzeugung der Konstruktionsregeln

Im letzten Abschnitt wurde gezeigt, wie ein Syntaxbaum mit Hilfe von Konstruktionsregeln in eine Diskurs-Repräsentationsstruktur überführt werden kann. Um den Aufwand bei der Erstellung und Überprüfung der Konstruktionsregeln niedrig zu halten, bietet es sich an, aus der formalen Beschreibung der Konstruktionsregeln automatisch den entsprechenden Java Quelltext zu erzeugen. Zu diesem Zweck wurde die in Kapitel 2.2 beschriebene Grammatikdatei um Konstruktionsregeln erweitert.
Der Konstruktionsregelteil der Grammatikdatei wird durch das reservierte Wort **CONSTRUCTION_RULES** eingeleitet. Anschließend folgen die einzelnen Konstruktionsregeln. Eine Konstruktionsregel besteht aus drei Abschnitten.

- **TS** (Triggering Structure) In diesem Abschnitt wird die auslösende Struktur der Konstruktionsregel durch geklammerte Ausdrücke angegeben. Knoten eines Baumes können aus Nichtterminalen bestehen, wobei gewünschte Attributbelegungen



- Erzeuge eine neue DRS `DRSx`.
- Füge ein neues Prädikat `v(u, w)` in die Bedingungsmenge von `DRSx` ein.
- Füge einen neuen Ereignisbezugsträger `ERx`, der durch `DRSx` definiert wird, in das Universum der DRS ein.
- Lösche die auslösende Struktur.

Abbildung 3.4: Diese Konstruktionsregel behandelt Syntaxbäume von Sätzen, in denen Subjekt und Objekt schon von anderen Konstruktionsregeln reduziert wurden. `u` und `w` stehen deshalb schon für Diskursbezugsträger. `v` hingegen steht für ein Terminal der Sprache und bezeichnet das Prädikat des Satzes.

dieser Nichtterminale wie in den Syntaxregeln angegeben werden können. Ferner können anstelle von konkreten Attributbelegungen auch Variablen eingeführt werden, die, falls die auslösende Struktur im Syntaxbaum gefunden wurde, mit den entsprechenden Werten belegt werden. Der Typ solcher Variablen wird dabei nicht explizit definiert, sondern ergibt sich aus der Stelle, an der die Variable das erste Mal verwendet wird. Anstelle von Terminalen der Sprache können ebenfalls Variablen eingesetzt werden, denen später die Terminale aus den entsprechenden Knoten des Syntaxbaumes zugewiesen werden.

- **HELP_STRUCTURES** Hier werden die im Aktionsteil benötigten Hilfsstrukturen beschrieben. Die in der auslösenden Struktur eingeführten Variablen können hier nur benutzt, es können aber keine neuen Variablen mehr eingeführt werden. Ferner ist es möglich sich auch auf einen Knoten bzw. einen Teilbaum aus dem der auslösenden Struktur entsprechenden Syntaxbaum zu beziehen.

- **ACTION** Dieser Abschnitt beinhaltet die Aktionen, die ausgeführt werden, falls die auslösende Struktur im Syntaxbaum gefunden wurde. Es können die in Kapitel A.1.8 beschriebenen vordefinierten Methoden benutzt werden. Hier kann auch auf die schon im **TS**–Abschnitt eingeführten Variablen und in **HELP_STRUCTURES** definierten Hilfsstrukturen zurückgegriffen werden. Es können auch neue Variablen eingeführt werden. Der Typ einer Variable ergibt sich wie im Abschnitt **TS** aus der Stelle ihres erste Vorkommens.

Ist die Beschreibung der Konstruktionsregeln abgeschlossen, so muß noch die Reihenfolge ihrer Anwendung angegeben werden. Diese Reihenfolge wird durch das reservierte Wort **CONSTRUCTIONRULE_ORDER** eingeleitet.
Die Anbindung an das bisher erzeugte Zerteilerprogramm geschieht automatisch. Nach Erstellen des Syntaxbaumes wird die Diskurs-Repräsentationsstruktur aus den angegebenen Konstruktionsregeln berechnet. Somit ist es also möglich Konstruktionsregeln einfach zu erstellen und zu modifizieren, ohne sich mit der rechnerinternen Repräsentation auseinandersetzen zu müssen.

Kapitel 4

Zusammenfassung und Ausblick

4.1 Zusammenfassung

Ziel dieser Arbeit war es, ein Werkzeug zu entwickeln, mit dem natürlichsprachlicher Text eingelesen, die darin enthaltene Semantik erschlossen und in die von [Kamp & Reyle 93] vorgestellte Logikstruktur DRS überführt werden kann. In einem zweiten Schritt sollte diese Logikstruktur in die von [Schäfer 96] entwickelte Logiksprache UMTL überführt werden. Aus dieser Logik-Repräsentation sollte danach eine Repräsentation in dem von [Jeyakumar 98] geforderten Eingabeformat abgeleitet werden, die es schließlich erlaubt, die in dem natürlichsprachlichen Text beschriebenen Sachverhalte in synthetische Bildfolgen zu überführen.
Um das erste Ziel zu erreichen, wurde ein Java-Werkzeug entwickelt, das eine attributierte Grammatik und die zu dieser Grammatik gehörenden Konstruktionsregeln in Form einer Textdatei entgegen nimmt und daraus einen Sprach-Zerteiler und alle zur DRS-Erstellung notwendigen Java-Klassen erzeugt. Dabei wurde versucht, die Möglichkeiten zur Erweiterung der Grammatik weitestgehend offen zu halten. Beispielsweise können sehr leicht neue Wortarten in die Grammatik aufgenommen werden, die dann auch mit eigenen Morphologieregeln versehen werden können. Da die in dieser Arbeit verwendete Grammatik ein Fragment der englischen Sprache erzeugt und dort fast jede Morphologie sich in der Endung eines Wortes ausdrückt, wurde jedoch nur diese Form der Morphologie betrachtet. Eine von den Ausdrucksmöglichkeiten her vergleichbare Grammatik für das Deutsche würde deshalb eine Erweiterung der Morphologie um die Behandlung von Vorsilben voraussetzen. Zur Angabe von Konstruktionsregeln wurde eine Bibliothek von Grundbefehlen erstellt, aus denen sich die Konstruktionsregeln zusammensetzen lassen. Auf der Seite der DRS wurde versucht, viele möglichst allgemeine Strukturen bereit zu stellen, so daß erwartet werden kann, daß die daraus resultierende DRS weitaus mächtiger ist, als dies von der in dieser Arbeit verwendeten Grammatik verlangt wurde.
Um diesen ersten Schritt der DRS-Erzeugung auf den abschließenden Stand zu bringen, war es notwendig, die schon bestehenden Werkzeuge um weitgehende Anzeige-

und Testfähigkeiten zu erweitern. Aus diesem Grund wurde der anfangs geplante Verlauf der Arbeit mehrmals verzögert, wodurch die anderen beiden Ziele, nämlich die Überführung in UMTL und die anschließende Konvertierung zur Erstellung von synthetischen Bildfolgen, nicht erreicht werden konnten.

4.2 Ausblick

Aufbauend auf den Ergebnissen dieser Arbeit wäre es nun wünschenswert, zunächst die Überführung einer DRS in UMTL zu ermöglichen. Dazu sollten zunächst die Mächtigkeiten der beiden Logikdarstellungen DRS und UMTL verglichen werden, um so zu klären, ob und wie eine Übersetzung einzelner DRS-Bedingungen möglich ist bzw. zu geschehen hat, und ob die vorhandenen DRS-Bedingungen erweitert werden müssen. Ist dies erfolgreich, sollte es auch möglich sein, die gewonnenen UMTL-Formeln sowohl zu der in dieser Arbeit angestrebten Erzeugung von synthetischen Bildfolgen, als auch allgemeiner zur Repräsentation von Wissen aus einer DRS und damit aus natürlicher Sprache zu erzeugen. Ferner wäre es interessant zu untersuchen, wie schon erzeugte UMTL-Formeln als Hintergrundwissen für die Verarbeitung weiterer natürlichsprachlicher Aussagen genutzt werden können.

Anhang A

Programm-Dokumentation

A.1 Grammatik

Die in der Studienarbeit entwickelte Grammatik wird in den folgenden Kapiteln beschrieben. In Kapitel A.1.1 ist die zur Erzeugung des Zerteilers und der Konstruktionsregeln benutzte Grammatikdatei abgebildet. Die darauf folgenden Kapitel stellen die in der Datei beschriebenen Sachverhalte in einer übersichtlichen Form dar und wurden von dem Werkzeug zur Erzeugung des Zerteilers automatisch erstellt.
Im folgenden wird das von der Grammatik verarbeitbare Fragment der englischen Sprache grob beschrieben. Der genaue Umfang des Fragments ist aus der in den nächsten Kapiteln dargestellten Grammatik zu ersehen.
Das mit dieser Grammatik abgedeckte Fragment der englischen Sprache beinhaltet Texte, die aus durch Punkt getrennte Aussagesätzen bestehen können. Sätze wiederum können aus mehreren mit durch `and` verknüpften Teilsätzen bestehen. Ein Teilsatz kann mit einer Adverbialphrase beginnen und enthält ein Subjekt und eine Verbphrase. Der erste Teilsatz muß ein Subjekt besitzen. Fehlt das Subjekt in einem der weiteren Teilsätze, so wird das Subjekt des vorherigen Teilsatzes auch als Subjekt dieses Teilsatzes angenommen. Die Verbphrase setzt sich aus einem Verb, einem Objekt und beliebig vielen Adverbialphrasen zusammen. Subjekte und Objekte können aus einem Nomen mit einem Artikel, einem Quantor, einem Adjektiv und einem Relativsatz oder deren sinnvoller Zusammensetzung bestehen. Adverbialphrasen können aus einem Adverb oder einem Adverbialsatz bestehen. Es wird dabei zwischen Adverbialphrasen der Zeit, des Ortes und der Art und Weise unterschieden.

A.1.1 Eingabedatei zur Sprachzerteilererzeugung

```
NONTERMINALS

STARTSYMBOL #
TEXT #
CONJOFSENTENCE #
SENTENCE #
NOUNPHRASE #
VERBPHRASE #
DETERMINER #
PREPRENOUN #
PRENOUN #
PROPERNAME #
RELATIVECLAUSE #
NOUN #
ADJECTIVE #
VERB #
BE #
PREVERBPHRASE #
PREPREVERBPHRASE #
PREADVERBIALPHRASE #
AND #
DOT #
PRESENTENCE #
RELATIVEPRONOUN #
ADVERBIALPHRASE #
PREPOSITION #
PRONOUN #
EMPTYWORD #
ADVERB #
CONJUNCTION #
CONJOFNOUNPHRASE #
QUANTIFIER #

ATTRIBUTES

number          [SINGULAR, PLURAL] #
gender          [MALE, FEMALE, NOTHUMAN] #
casus           [NOMINATIVE, NOTNOMINATIVE] #
gap             [HASGAP, HASNOGAP] #
transitivity    [TRANSITIVE, INTRANSITIVE] #
finitivity      [FINITE,INFINITE,GERUND] #
countability    [COUNTABLE,UNCOUNTABLE] #
adverbial_kind  [MANNER,PLACE,TIME] #
processed       [NONE,SYNTAX,DUPLEX] #
time_relation   [NONE,AFTER,BEFORE,WHILE] #

WORDPROPERTIES

VERB [number,transitivity,finitivity] #
NOUN [number,gender,casus,adverbial_kind,countability] #
DETERMINER [number] #
PRONOUN [gender,casus] #
PREPOSITION [adverbial_kind,time_relation] #
PROPERNAME [number,gender,casus,adverbial_kind] #
ADJECTIVE [adverbial_kind] #
BE [finitivity, number] #
RELATIVEPRONOUN [gender, number] #
AND [_] #
DOT [_] #
ADVERB [adverbial_kind] #
CONJUNCTION [adverbial_kind,time_relation] #
QUANTIFIER [_] #

RULES

  STARTSYMBOL [_]
  ->
    TEXT [_]
  #

  TEXT [_]
  ->
    CONJOFSENTENCE [gap = HASNOGAP]
    DOT [_]
    TEXT [_]
  #

  TEXT [_]
  ->
    CONJOFSENTENCE [gap = HASNOGAP]
    DOT [_]
  #

  CONJOFSENTENCE [gap = c]
  ->
```

```
  SENTENCE [gap = c]
  AND [_]
  CONJOFSENTENCE [_]
#

CONJOFSENTENCE [gap = c]
->
  SENTENCE [gap = c]
#

SENTENCE [gap = c, processed = NONE]
->
  ADVERBIALPHRASE [_]
  PRESENTENCE [gap = c]
#

SENTENCE [gap = c, processed = NONE]
->
  PRESENTENCE [gap = c]
#

PRESENTENCE [number=n, gap = c, finitivity = FINITE]
->
  CONJOFNOUNPHRASE [number = n, gap = c, casus = NOMINATIVE]
  PREPREVERBPHRASE [number = n, finitivity = FINITE]
#

CONJOFNOUNPHRASE [number = PLURAL, gap = HASNOGAP, casus = c]
->
  NOUNPHRASE [gap = HASNOGAP, casus = c]
  AND [_]
  CONJOFNOUNPHRASE [gap = HASNOGAP, casus = c]
#

CONJOFNOUNPHRASE [number = n, gap = g, casus = c]
->
  NOUNPHRASE [number = n, gap = g, casus = c]
#

NOUNPHRASE [number = n, gap = HASNOGAP, adverbial_kind = k,casus = c]
->
  DETERMINER [number = n]
  PREPRENOUN [number = n, countability = COUNTABLE, adverbial_kind = k, casus = c]
#

NOUNPHRASE [number = n, gap = HASNOGAP, adverbial_kind = k, casus = c]
->
  PROPERNAME [number = n, adverbial_kind = k, casus = c]
#

NOUNPHRASE [number = PLURAL, gap = HASNOGAP, adverbial_kind = k, casus = c]
->
  PREPRENOUN [number = PLURAL, countability = COUNTABLE, adverbial_kind = k, casus = c]
#

NOUNPHRASE [number = n, gap = HASNOGAP, adverbial_kind = k, casus = c]
->
  PREPRENOUN [number = n, countability = UNCOUNTABLE, adverbial_kind = k, casus = c]
#

NOUNPHRASE [number = SINGULAR,gap = HASNOGAP, casus = c]
->
  PRONOUN [casus = c]
#

NOUNPHRASE [number = n, gap = HASGAP, casus = c]
->
  EMPTYWORD [number = n, casus = c]
#

PREPRENOUN[number = PLURAL, countability = c,adverbial_kind = k, gender =g, casus = f]
->
  QUANTIFIER[_]
  PRENOUN[number = PLURAL, countability = c,adverbial_kind = k, gender =g, casus = f]
#

PREPRENOUN[number = n, countability = c,adverbial_kind = k, gender =g, casus = f]
->
  PRENOUN[number = n, countability = c,adverbial_kind = k, gender =g, casus = f]
#

PRENOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
->
  ADJECTIVE [_]
  NOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
  RELATIVECLAUSE [_]
#
```

```
PRENOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
->
  NOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
  RELATIVECLAUSE [_]
#

PRENOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
->
  ADJECTIVE [_]
  NOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
#

PRENOUN [number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
->
  NOUN[number = n, countability = c, adverbial_kind = k, gender = g, casus = f]
#

VERBPHRASE [number = n, transitivity = a,finitivity = FINITE]
->
  BE [number = n, finitivity = FINITE]
  VERB [transitivity = a,finitivity = GERUND]
#

VERBPHRASE [number = n, transitivity = TRANSITIVE,finitivity = a]
->
  VERB [number = n, transitivity = TRANSITIVE,finitivity = a]
  CONJOFNOUNPHRASE [casus = NOTNOMINATIVE]
#

VERBPHRASE [number = n, transitivity = INTRANSITIVE,finitivity = a]
->
  VERB [number = n, transitivity = INTRANSITIVE,finitivity = a]
#

PREVERBPHRASE [number = n, transitivity = a, finitivity = b]
->
  VERBPHRASE [number = n, transitivity = a, finitivity = b]
#

PREPREVERBPHRASE [number = n, transitivity = a, finitivity = b]
->
  PREVERBPHRASE [number = n, transitivity = a, finitivity = b]
  PREADVERBIALPHRASE [_]
#

PREPREVERBPHRASE [number = n, transitivity = a, finitivity = b]
->
  PREVERBPHRASE [number = n, transitivity = a, finitivity = b]
#

RELATIVECLAUSE [_]
->
  RELATIVEPRONOUN [_]
  PRESENTENCE [gap = HASGAP]
#

ADVERBIALPHRASE [adverbial_kind = k]
->
  ADVERB [adverbial_kind = k]
#

ADVERBIALPHRASE [adverbial_kind = k]
->
  CONJUNCTION [adverbial_kind = k]
  PRESENTENCE [_]
#

ADVERBIALPHRASE [adverbial_kind = k]
->
  PREPOSITION [adverbial_kind = k]
  NOUNPHRASE [gap = HASNOGAP,adverbial_kind = k]
#

ADVERBIALPHRASE [adverbial_kind = k]
->
  CONJUNCTION [adverbial_kind = k]
  PREPREVERBPHRASE [finitivity = GERUND]
#

ADVERBIALPHRASE [adverbial_kind = MANNER]
->
  PREPREVERBPHRASE [finitivity = GERUND]
#

PREADVERBIALPHRASE [_]
->
  ADVERBIALPHRASE [_]
  PREADVERBIALPHRASE [_]
```

```
  #

  PREADVERBIALPHRASE [_]
  ->
    ADVERBIALPHRASE [_]
  #

  NOUN [_] -> LEX
  { [number = PLURAL]
    "o"                         -> "oes";
    "ss"                        -> "sses";
    "sh"                        -> "shes";
    "ch"                        -> "ches";
    "x"                         -> "xes";
    <vowel>"y"                  -> <vowel>"ys";
    <cons>"y"                   -> <cons>"ies";
    <vowel>"s"                  -> <vowel>"sses";
    "f"                         -> "ves";
    "fe"                        -> "ves";
    _                           -> "s";
  }
  #

  PROPERNAME [_] -> LEX {
  }
  #

  VERB [_] -> LEX
  { [finitivity = FINITE]
    <vowel>"y"                  -> <vowel>"yed";
    <cons>"y"                   -> <cons>"ied";
    <cons1><vowel><cons2>       -> <cons1><vowel><cons2><cons2>"ed";
    "e"                         -> "ed";
     _                          -> "ed";
  }
  { [finitivity = GERUND]
    "e"                         -> "ing";
    "ie"                        -> "ying";
    <cons1><vowel><cons2>       -> <cons1><vowel><cons2><cons2>"ing";
    _                           -> "ing";
  }
  #

  DETERMINER [_] -> LEX {
  }
  #

  PRONOUN [_] -> LEX {
  }
  #

  RELATIVEPRONOUN [_] -> LEX {
  }
  #

  PREPOSITION [_] -> LEX {
  }
  #

  ADVERB [_] -> LEX {
  }
  #

  CONJUNCTION [_] -> LEX {
  }
  #

  ADJECTIVE [_] -> LEX {
  }
  #

  QUANTIFIER [_] -> LEX {
  }
  #

  BE [_] -> LEX {
  }
  #

  AND [_] -> LEX {
  }
  #

  DOT [_] -> LEX {
  }
  #

CONSTRUCTION_RULES
```

```
CR_START
  /* This CR finally reduces the Syntaxtree to an empty tree.
   *
   *        STARTSYMBOL
   *            |          =>    Null
   *          getWord(erx)
   */
  TS
    1:STARTSYMBOL [_]{
      2:getWord(erx)
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    removeTS();
  #
# // end of CR_START


CR_TEXT1
  /* This CR reduces a TEXT consisting of one event referent and a dot. It simply replaces the tree with the node
   * of the event referent name.
   *
   *             TEXT               getWord(ERx)
   *            /    \
   *           /      \       =>
   *          /        \
   *    getWord(ERx)    DOT
   *
   */
  TS
    1:TEXT [_] {
      2:getWord(e)
      3:DOT [_]
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    substitute(TS.2,TS.1);
  #
# // end of CR_TEXT1

CR_TEXT2
  /* This CR deals with a TEXT consisting of two event referents seperated by a DOT. It introduces a new event referent
   * which is a event referent set of the two events.
   *
   *               TEXT (a)                getWord(ERz)
   *              /  | \
   *             /   |  \
   *            /    |   \         =>
   *           /     |    \
   *   getWord(ERx)  DOT   getWord(ERy) (b)
   *
   */
  TS
    1:TEXT [_] {
      2:getWord(erx)
      3:DOT [_]
      4:getWord(ery)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(er_neu)
    #
  #
  ACTION
    er_neu = addNewLocalEventReferentSet(erx,ery);
    substitute(HS_1, TS.1);
  #
# // end of CR_TEXT2


CR_CONJOFSENTENCE1
  /* This CR deals with CONFOFSENTENCEs which consist only of one single event and replaces the root node of this tree
   * with the node of the event referent's name.
   *
   *          CONJOFSENTENCE                 getWord(erx)
   *                |              =>
   *                |
   *             getWord(erx)
   */
  TS
    1:CONJOFSENTENCE [_] {
      2:getWord(erx)
```

```
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    substitute(TS.2,TS.1);
  #
# // end of CR_CONJOFSENTENCE1


CR_CONJOFSENTENCE2
  /* This CR deals with CONFOFSENTENCEs which consist of two events. It creates a new event referent set of the two
   * event referents.
   *
   *           CONJOFSENTENCE (a)                  getWord(erneu)
   *             /   |   \                =>
   *            /    |    \
   *           /     |     \
   *          /      |      \
   *  getWord(erx)  AND   getWord(ery)
   *
   */
  TS
    1:CONJOFSENTENCE [_] {
      2:getWord(erx)
      3:AND [_]
      4:getWord(ery)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(er_neu)
    #
  #
  ACTION
    er_neu = addNewLocalEventReferentSet(erx,ery);
    substitute(HS_1, TS.1);
  #
# // end of CR_CONJOFSENTENCE2

CR_NOUNPHRASE1
  /* This CR deals with propernames. It introduces a new DR, add a new Name to the global DRS
   * and substitutes the given TS as follows:
   *
   *    NOUNPHRASE        NOUNPHRASE
   *        |                 |
   *    PROPERNAME   =>  getWord(DRx)
   *        |
   *    getWord(a)
   *
   */
  TS
    1:NOUNPHRASE [_] {
      2:PROPERNAME [_] {
        3:getWord(a)
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1 {
        2:getWord(d)
      }
    #
  #
  ACTION
    d = addNewGlobalDR(TS.2);
    addNewName(a,d);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE1

CR_NOUNPHRASE2
  /* This CR deals with a sinuglar noun. It introduces a new DR to the local DRS and
   * adds one new local constraint according to the used noun.
   * The TS and its substitution is as follows:
   *
   *     PRENOUN                getWord(DRx)
   *        |
   *       NOUN
   *        |
   *    getWord(n)
   *
   */
  TS
      1:PRENOUN [number = SINGULAR] {
        2:NOUN [_] {
          3:getWord(n)
```

```
        }
      }
  #
  HELP_STRUCTURES
    HS_1
        1:getWord(d)
    #
  #
  ACTION
    d = addNewLocalDR(TS.1);
    addNewLocalAttribute(n,d);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE2


CR_NOUNPHRASE3
  /* This CR deals with a singular noun with one adjective. It introduces a new DR to the local DRS,
   * adds one new local constraint according to the used noun, and another for the used adjective.
   * The TS and its substitution is as follows:
   *
   *     PRENOUN                 getWord(DRx)
   *      /   \
   *     /     \             =>
   *    /       \
   * ADJECTIVE  NOUN
   *   |          |
   * getWord(a)  getWord(n)
   *
   */
  TS
      1:PRENOUN [number = SINGULAR] {
        2:ADJECTIVE [_] {
          3:getWord(a)
        }
        4:NOUN [_] {
          5:getWord(n)
        }
      }

  #
  HELP_STRUCTURES
    HS_1
        1:getWord(d)
    #
  #
  ACTION
    d = addNewLocalDR(TS.1);
    addNewLocalAttribute(n,d);
    addNewLocalAttribute(a,d);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE3


CR_NOUNPHRASE4
  /* CR for preprocessing a singular NOUNPHRASE which has an additional RELATIVECLAUSE.
   *
   *           PRENOUN                              PRENOUN
   *          /       \                =>          /     \
   *         /         \                 getWord(DRx)  RELATIVECLAUSE
   *       NOUN   RELATIVECLAUSE
   *        |
   *    getWord(n)
   *
   */
  TS
     1:PRENOUN [number=SINGULAR] {
       2:NOUN [_] {
         3:getWord(n)
      }
      4:RELATIVECLAUSE [_]
     }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1 {
        2:getWord(d)
        3:TS.4.copy()
      }
    #
  #
  ACTION
    d = addNewLocalDR(TS.2);
    addNewLocalAttribute(n,d);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE4
```

```
CR_NOUNPHRASE5
  /* CR for preprocessing a singular Nounphrase with an ADJECTIVE and a RELATIVECLAUSE.
   *
   *           PRENOUN                                  PRENOUN
   *         /   |   \                 =>             /     \
   *        /    |    \                          getWord(DRx)  RELATIVECLAUSE
   *  ADJECTIVE  NOUN  RELATIVECLAUSE
   *     |        |
   * getWord(a)  getWord(n)
   *
   */
  TS
      1:PRENOUN [number=SINGULAR] {
        2:ADJECTIVE [_] {
          3:getWord(a)
        }
        4:NOUN [_] {
          5:getWord(n)
        }
        6:RELATIVECLAUSE [_]
      }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1 {
        2:getWord(d)
        3:TS.6.copy()
      }
    #
  #
  ACTION
    d = addNewLocalDR(TS.5);
    addNewLocalAttribute(n,d);
    addNewLocalAttribute(a,d);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE5


CR_NOUNPHRASE6
  /* This CR deals with a plural noun. It introduces a new PDR to the local DRS and
   * adds a new Set description (in form of a subDRS) for the PDR.
   *
   *     PRENOUN[PLURAL]                 getWord(PDRx)
   *        |
   *       NOUN                 =>
   *        |
   *    getWord(n)
   *
   */
  TS
      1:PRENOUN [number = PLURAL] {
        2:NOUN [_] {
          3:getWord(n)
        }
      }
  #
  HELP_STRUCTURES
    HS_1
        1:getWord(pdr)
    #
  #
  ACTION
    drs = buildEmptyDRS(POSITIVE);
    dr = getNextFreeDR(drs);
    a = buildAttribute(n,dr);
    addAttributeToDRS(a,drs);
    pdr = addNewLocalReferentSet(dr,drs);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE6


CR_NOUNPHRASE7
  /* This CR deals with a plural noun with one adjective. It introduces a new PDR to the local DRS and
   * adds a new Set description (in form of a subDRS) for the PDR.
   *
   *     PRENOUN
   *      /   \
   *     /     \             =>   getWord(PDRx)
   *    /       \
   * ADJECTIVE  NOUN
   *   |          |
   * getWord(a)  getWord(n)
   *
   */
```

```
  TS
      1:PRENOUN [number = PLURAL] {
        2:ADJECTIVE [_] {
          3:getWord(a)
        }
        4:NOUN [_] {
          5:getWord(n)
        }
      }

  #
  HELP_STRUCTURES
    HS_1
        1:getWord(pdr)
    #
  #
  ACTION
    drs = buildEmptyDRS(POSITIVE);
    dr = getNextFreeDR(drs);
    attr = buildAttribute(n,dr);
    addAttributeToDRS(attr,drs);
    attr = buildAttribute(a,dr);
    addAttributeToDRS(attr,drs);
    pdr = addNewLocalReferentSet(dr,drs);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE7

CR_NOUNPHRASE8
  /* This CR deals with a plural noun which has a relativeclause. It introduces a new PDR to the local DRS and
   * adds a new set description (in form of a subDRS) for the PDR. The relative clause syntax tree is moved
   * into the set description.
   *
   *     PRENOUN[PLURAL]
   *        |       \
   *       NOUN    RELATIVECLAUSE____            =>     getWord(PDRx)
   *        |          |             \
   *    getWord(n)  RELATIVEPRONOUN  PRESENTENCE
   *                                   /     \
   *                      CONJOFNOUNPHRASE  PREPREVERBPHRASE
   */
  TS
      1:PRENOUN [number = PLURAL] {
        2:NOUN [_] {
          3:getWord(n)
        }
        4:RELATIVECLAUSE[_]{
          5:RELATIVEPRONOUN[_]
          6:PRESENTENCE[_]{
            7:CONJOFNOUNPHRASE[_]
            8:PREPREVERBPHRASE[_]
          }
        }
      }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(pdr)
    #
    HS_2
      0:STARTSYMBOL[_]{
        1:getWord(er){
          2:SENTENCE[_]{
            3:PRESENTENCE[number = SINGULAR]{
              4:CONJOFNOUNPHRASE[_]{
                5:NOUNPHRASE[number = SINGULAR]{
                  6:EMPTYWORD[_]
                }
              }
              7:TS.8.copy()
            }
          }
        }
      }
    #
    HS_3
      1:EMPTYWORD[_]
    #
    HS_4
      1:getWord(dr)
    #
  #
  ACTION
    er = getNextFreeEventReferent();
    g = findNodeOfType(HS_3,HS_2);
    drs = buildDRS(HS_2,POSITIVE);
    dr = getNextFreeDR(drs);
    substitute(HS_4,g);
```

```
    a = buildAttribute(n,dr);
    addAttributeToDRS(a,drs);
    pdr = addNewLocalReferentSet(dr,drs);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE8


CR_NOUNPHRASE9
  /* This CR deals with a plural noun with one adjective. It introduces a new PDR to the local DRS and
   * adds a new Set description (in form of a subDRS) for the PDR.
   *
   *     PRENOUN___
   *      /   \    \
   *     /     \    \                                    =>   getWord(PDRx)
   *    /       \    \
   * ADJECTIVE  NOUN  RELATIVECLAUSE_______________
   *   |          |              |                  \
   * getWord(a)  getWord(n)   RELATIVEPRONOUN   PRESENTENCE
   *                                             /     \
   *                                  CONJOFNOUNPHRASE  PREPREVERBPHRASE
   *
   */
  TS
      1:PRENOUN [number = PLURAL] {
        2:ADJECTIVE [_] {
          3:getWord(a)
        }
        4:NOUN [_] {
          5:getWord(n)
        }
        6:RELATIVECLAUSE[_]{
          7:RELATIVEPRONOUN[_]
          8:PRESENTENCE[_]{
            9:CONJOFNOUNPHRASE[_]
            10:PREPREVERBPHRASE[_]
          }
        }
      }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(pdr)
    #
    HS_2
      0:STARTSYMBOL[_]{
        1:getWord(er){
          2:SENTENCE[_]{
            3:PRESENTENCE[number = SINGULAR]{
              4:CONJOFNOUNPHRASE[_]{
                5:NOUNPHRASE[number = SINGULAR]{
                  6:EMPTYWORD[_]
                }
              }
              7:TS.10.copy()
            }
          }
        }
      }
    #
    HS_3
      1:EMPTYWORD[_]
    #
    HS_4
      1:getWord(dr)
    #
  #
  ACTION
    er = getNextFreeEventReferent();
    g = findNodeOfType(HS_3,HS_2);
    drs = buildDRS(HS_2,POSITIVE);
    dr = getNextFreeDR(drs);
    substitute(HS_4,g);
    attr = buildAttribute(n,dr);
    addAttributeToDRS(attr,drs);
    attr = buildAttribute(a,dr);
    addAttributeToDRS(attr,drs);
    pdr = addNewLocalReferentSet(dr,drs);
    substitute(HS_1, TS.1);
  #
# // end of CR_NOUNPHRASE9

CR_DETERMINER1
  /* This CR deals with determiners in front of a prenounphrase which has already been reduced.
   *
   *         NOUNPHRASE
   *            /  \                  NOUNPHRASE
   *           /    \          =>        |
```

```
   *          /      \               getWord(DRx)
   *    DETERMINER getWord(DRx)
   *
   */
  TS
    1:NOUNPHRASE[_]{
      2:DETERMINER[_]
      3:getWord(d)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1 {
        2:getWord(d)
      }
    #
  #
  ACTION
    substitute(HS_1,TS.1);
  #
# // end of CR_DETERMINER1


CR_PRONOUN1
  /* This CR processes NOUNPHRASES which consist of a PRONOUN. The Pronoun must refer to an already
   * introduced DR. The CR will try to find this so called antecedent and will substitute the TS
   * for further processing.
   *
   *        NOUNPHRASE          NOUNPHRASE
   *            |         =>         |
   *         PRONOUN           getWord(DRx)
   *
   */
  TS
    1:NOUNPHRASE [_] {
      2:PRONOUN [_]
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1 {
        2:getWord(d)
      }
    #
  #
  ACTION
    d = getAntecedent(TS.2);
    substitute(HS_1, TS.1);
  #
#  // end of CR_PRONOUN1


CR_RELATIVECLAUSE1
  /* This CR reduces PRENOUNs which consist of an ADJECTIVE, A NOUN and an additional RELATIVECLAUSE.
   * A new SubDRS is built here.
   *
   *         PRENOUN
   *         /       \
   *        /         \                                  getWord(DRx)
   *   getWord(DRx) RELATIVECLAUSE          =>
   *                  /       \
   *                 /         \
   *          RELATIVEPRONOUN  PRESENTENCE
   *
   */
  TS
    1:PRENOUN [_]{
      2:getWord(r)
      3:RELATIVECLAUSE [_] {
        4:RELATIVEPRONOUN [_]
        5:PRESENTENCE [_]
      }
    }
  #
  HELP_STRUCTURES
    HS_3
      1:EMPTYWORD [_]
    #
    HS_4
      0:STARTSYMBOL[_]{
        1:getWord(er){
          2:SENTENCE[processed = NONE]{
            3:TS.5.copy()
          }
        }
      }
    #
  #
```

```
  ACTION
    er = getNextFreeEventReferent();
    g = findNodeOfType(HS_3, HS_4);
    substitute(TS.2, g);
    d = buildDRS(HS_4, POSITIVE);
    addNewGlobalSubDRS(d);
    substitute(TS.2, TS.1);
  #
# // END OF CR_RELATIVECLAUSE1

CR_CONJOFNOUNPHRASE1
  /* This CR deals with simple nounphrase-conjunctions which consist of only one NOUNPHRASE.
   * In fact it is just a simple reduction. Note that this CR can only be applied if the NOUNPHRASE
   * was already processed by one of the NOUNPHRASE-CRs.
   *
   *   CONJOFNOUNPHRASE          NOUNPHRASE
   *          |                      |
   *     NOUNPHRASE      =>     getWord(DRx)
   *          |
   *    getWord(DRx)
   */
  TS
    1:CONJOFNOUNPHRASE [_] {
      2:NOUNPHRASE [_] {
        3:getWord(d)
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.2.copy()
    #
  #
  ACTION
    substitute(HS_1, TS.1);
  #
# // end of CR_CONJOFNOUNPHRASE1

CR_CONJOFNOUNPHRASE2
  /* This CR deals with conjunction of two nounphrases. A new ReferentSum is introduced.
   *
   *        CONJOFNOUNPHRASE                    NOUNPHRASE
   *          /      |      \                       |
   *   NOUNPHRASE   AND   NOUNPHRASE        =>     getWord(DRx)
   *        |                 |
   *    getWord(DRx)      getWord(DRx)
   */
  TS
    1:CONJOFNOUNPHRASE [_] {
      2:NOUNPHRASE [_] {
        3:getWord(left)
      }
      4:AND[_]
      5:NOUNPHRASE [_] {
        6:getWord(right)
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:NOUNPHRASE[number = PLURAL]{
        2:getWord(pdr_neu)
      }
    #
  #
  ACTION
    pdr_neu = addNewLocalReferentSum(left,right);
    substitute(HS_1, TS.1);
  #
# // end of CR_CONJOFNOUNPHRASE2

CR_SENTENCE1
  /* This CR deals with a SENTENCE which has no ADVERBIALPHRASES at the beginning as first child. It introduces an event
   * referent with the name er and with the presentence sub tree.
   *
   *      getWord(er)
   *           |
   *       SENTENCE
   *          |
   *      PRESENTENCE                  =>      getWord(er)
   *       /       \
   *      /         \
   * NOUNPHRASE  PREPREVERBPHRASE
   *     |
   * getWord(DRx)
   */
  TS
    0:getWord(er){
```

```
      1:SENTENCE[_]{
        2:PRESENTENCE[_]{
          3:NOUNPHRASE[_]{
            5:getWord(w)
          }
          4:PREPREVERBPHRASE[_]
        }
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      0:STARTSYMBOL[_]{
        1:getWord(er){
          2:TS.2.copy()
        }
      }
    #
    HS_2
      1:getWord(er)
    #
  #
  ACTION
    d = buildDRS(HS_1,POSITIVE);
    addLocalEventReferent(er,d);
    substitute(HS_2,TS.0);
  #
# // end of CR_SENTENCE1

CR_ADVERBIALPHRASE0
  /* This CR deals with ADVERBIALPHRASES of place, consisting of a PREPOSITION and an already processed NOUNPHRASE.
   * A new local predicate is created and the structure is thrown away.
   *
   *      ADVERBIALPHRASE[place]
   *        /       \
   *       /         \
   *      /           \              =>        0
   * PREPOSITION    NOUNPHRASE
   *     |              |
   *  getWord(p)    getWord(DRx)
   *
   */
  TS
    1:ADVERBIALPHRASE [adverbial_kind = PLACE] {
      2:PREPOSITION [_] {
        3:getWord(p)
      }
      4:NOUNPHRASE [_] {
        5:getWord(d)
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.5
    #
  #
  ACTION
    addNewLocalPredicate(p, d);
    removeTS();
  #
# // end of CR_ADVERBIALPHRASE0


CR_ADVERBIALPHRASE1
  /* This CR deals with ADVERBIALPHRASES of manner, consisting of a PREPOSITION and an already processed NOUNPHRASE.
   * A new local predicate is created and the structure is thrown away.
   *
   *      ADVERBIALPHRASE[manner]
   *        /       \
   *       /         \
   *      /           \              =>        0
   * PREPOSITION    NOUNPHRASE
   *     |              |
   *  getWord(p)    getWord(DRx)
   *
   */
  TS
    1:ADVERBIALPHRASE [adverbial_kind = MANNER] {
      2:PREPOSITION [_] {
        3:getWord(p)
      }
      4:NOUNPHRASE [_] {
        5:getWord(d)
      }
    }
  #
  HELP_STRUCTURES
```

```
    HS_1
      1:TS.5
    #
  #
  ACTION
    addNewLocalPredicate(p, d);
    removeTS();
  #
# // end of CR_ADVERBIALPHRASE1

CR_ADVERBIALPHRASE2
  /* This CR deals with an ADVERBIALPHRASE at the beginning of a sentence and no other adverbial phrase.
   * It takes the subtree and moves it from the beginning behind the VERBPHRASE of the sentence.
   *
   *              SENTENCE                                          SENTENCE
   *              /   \                                                |
   *             /     \                                               |
   * ADVERBIALPHRASE   PRESENTENCE                                PRESENTENCE
   *                     /     \                    =>              /     \
   *                    /       \                                 /       \
   *        CONJOFNOUNPHRASE   PREPREVERBPHRASE     CONJOFNOUNPHRASE   PREPREVERBPHRASE
   *                                 |                                    /      \
   *                           PREVERBPHRASE                             /        \
   *                                                                PREVERBPHRASE  PREADVERBIALPHRASE
   *                                                                                     |
   *                                                                                ADVERBIALPHRASE
   */
   TS
      1:SENTENCE[_]{
        2:ADVERBIALPHRASE[_]
        3:PRESENTENCE[_]{
          4:CONJOFNOUNPHRASE[_]
          5:PREPREVERBPHRASE[_]{
            6:PREVERBPHRASE[_]
          }
        }
      }
   #
   HELP_STRUCTURES
     HS_1
       1:TS.5{
         2:TS.6.copy()
         3:PREADVERBIALPHRASE[_]{
           4:TS.2.copy()
         }
       }
     #
   #
   ACTION
     substitute(HS_1,TS.5);
     remove(TS.2);
   #
# // end of CR_ADVERBIALPHRASE2

CR_ADVERBIALPHRASE3
  /* This CR deals with an ADVERBIALPHRASE at the beginning of a sentence and other adverbial phrases.
   * It takes the subtree and moves it from the beginning behind the verb phrase of the sentence.
   *
   *              SENTENCE                                                SENTENCE
   *              /   \                                                      |
   *             /     \                                                     |
   * ADVERBIALPHRASE   PRESENTENCE                                      PRESENTENCE
   *                     /     \                    =>                    /     \
   *                    /       \                                       /       \
   *        CONJOFNOUNPHRASE   PREPREVERBPHRASE            CONJOFNOUNPHRASE   PREPREVERBPHRASE
   *                             /       \                                        /      \
   *                  PREVERBPHRASE    PREADVERBIALPHRASE(*)                    /        \
   *                                                                      PREVERBPHRASE  PREADVERBIALPHRASE
   *                                                                                       /    \
   *                                                                                      /      \
   *                                                                            ADVERBIALPHRASE   PREADVERBIALPHRASE(*)
   */
   TS
      1:SENTENCE[_]{
        2:ADVERBIALPHRASE[_]
        3:PRESENTENCE[_]{
          4:CONJOFNOUNPHRASE[_]
          5:PREPREVERBPHRASE[_]{
            6:PREVERBPHRASE[_]
            7:PREADVERBIALPHRASE[_]
          }
        }
      }
   #
   HELP_STRUCTURES
     HS_1
       1:TS.5{
         2:TS.6.copy()
```

```
        3:TS.7{
          4:TS.2.copy()
          5:TS.7.copy()
        }
      }
    #
  #
  ACTION
    substitute(HS_1,TS.5);
    remove(TS.2);
  #
#// end of CR_ADVERBIALPHRASE3

CR_ADVERBIALPHRASE4
  /* This CR deals with an adverbial sentence phrase. It creates a new event referent with the sentence as the defining DRS
   * and replaces the PRESENTENCE node with the event referents name.
   *
   *              ADVERBIALPHRASE                 ADVERBIALPHRASE
   *                /      \            =>          /       \
   *               /        \                       /         \
   *          CONJUNCTION   PRESENTENCE      CONJUNCTION    getWord(er_neu)
   *
   */
  TS
    1:ADVERBIALPHRASE[_]{
      2:CONJUNCTION[_]
      3:PRESENTENCE[_]
    }
  #
  HELP_STRUCTURES
    HS_1
      0:STARTSYMBOL[_]{
        1:getWord(er_neu){
          2:SENTENCE[_]{
            3:TS.3.copy()
          }
        }
      }
    #
    HS_2
      1:getWord(er_neu)
    #
  #
  ACTION
    er_neu = getNextFreeEventReferent();
    d = buildDRS(HS_1,POSITIVE);
    addNewGlobalSubDRS(d);
    substitute(HS_2,TS.3);
  #
# // end of CR_ADVERBIALPHRASE4

CR_ADVERBIALPHRASE5
  /* This CR deals with an adverbial phrase of time which is specified by a conjunction and an event referent. It creates an global
   * event relation predicate of the form conjunction(my_current_event_referent, event_referent)
   *
   *        PREADVERBIALPHRASE
   *         /            \
   *  getWord(er)  PREADVERBIALPHRASE
   *                       |
   *                 ADVERBIALPHRASE
   *                 /      \            =>        getWord(er)
   *                /        \
   *          CONJUNCTION   getWord(er_conj)
   *               |
   *           getWord(c)
   */
 TS
    0:PREADVERBIALPHRASE[_]{
      1:getWord(er)
      2:PREADVERBIALPHRASE[_]{
        3:ADVERBIALPHRASE[adverbial_kind = TIME]{
          4:CONJUNCTION[_]{
            5:getWord(c)
          }
          6:getWord(er_conj)
        }
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(er)
    #
  #
  ACTION
    addNewGlobalEventRelation(c,er_conj,er);
    substitute(HS_1,TS.0);
  #
```

```
# // end of CR_ADVERBIALPHRASE5

CR_ADVERBIALPHRASE6
  /* This CR deals with an adverbial phrase of time which is specified by a conjunction and an event referent. It creates an global
   * event relation predicate of the form conjunction(my_current_event_referent, event_referent)
   *
   *        PREADVERBIALPHRASE
   *         /            \
   *  getWord(er)  PREADVERBIALPHRASE__
   *                       |           \                                PREADVERBIALPHRASE
   *                 ADVERBIALPHRASE   PREADVERBIALPHRASE(*)             /          \
   *                /      \                                  =>  getWord(er)      PREADVERBIALPHRASE(*)
   *               /        \
   *          CONJUNCTION   getWord(er_conj)
   *              |
   *           getWord(c)
   */
 TS
    0:PREADVERBIALPHRASE[_]{
      1:getWord(er)
      2:PREADVERBIALPHRASE[_]{
        3:ADVERBIALPHRASE[adverbial_kind = TIME]{
          4:CONJUNCTION[_]{
            5:getWord(c)
          }
          6:getWord(er_conj)
        }
        7:PREADVERBIALPHRASE[_]
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:PREADVERBIALPHRASE[_]{
        2:getWord(er)
        3:TS.7.copy()
      }
    #
  #
  ACTION
    addNewGlobalEventRelation(c,er_conj,er);
    substitute(HS_1,TS.0);
  #
# // end of CR_ADVERBIALPHRASE6

CR_ADVERBIALPHRASE7
  /* This CR deals with adverbial nounphrases. They are interpreted as durations. The NOUNPHRASE is replaced with the
   * event referent for the duration.
   *
   *         ADVERBIALPHRASE
   *           /       \
   *    PREPOSITON  NOUNPHRASE[time]
   *        |               |
   *     getWord(p)    getWord(dr/pdr)
   */
  TS
    1:ADVERBIALPHRASE[_]{
      2:PREPOSITION[_]{
        0:getWord(p)
      }
      3:NOUNPHRASE[adverbial_kind = TIME]{
        4:getWord(dr)
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.1{
        2:TS.2{
          3:TS.0
        }
        4:getWord(er)
      }
    #
  #
  ACTION
    er = addNewGlobalDuration(dr);
    substitute(HS_1,TS.1);
  #
#  // end of CR_ADVERBIALPHRASE7

CR_PREADVERBIALPHRASE1
  /* This CR reduces simple PREADVERBIALPHRASES consisting of only one PREADVERBIALPHRASE as generated with the rule CR_VERBPHRASE3.
   *
   *    PREADVERBIALPHRASE
   *            |             =>       PREADVBERBIALPHRASE(*)
   *    PREADVERBIALPHRASE(*)
   */
```

```
  TS
    1:PREADVERBIALPHRASE [_] {
       2:PREADVERBIALPHRASE [_]
    }
  #
  HELP_STRUCTURES
    HS_2
      1:TS.2.copy()
    #
  #
  ACTION
    substitute(HS_2,TS.1);
  #
# // end of CR_PREADVERBIALPHRASE1

CR_PREADVERBIALPHRASE2
  /* This CR reduces simple PREADVERBIALPHRASES consisting of one PREADVERBIALPHRASE and the name of the event referent.
   * It cleans up the structure, by replacing  the top with the name node.
   *    PREADVERBIALPHRASE
   *         /         \              =>       getWord(er)
   *  getWord(er)  PREADVERBIALPHRASE(*)
   */
  TS
    1:PREADVERBIALPHRASE [_] {
      2:getWord(er)
      3:PREADVERBIALPHRASE [_]
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    substitute(TS.2,TS.1);
  #
# // end of CR_PREADVERBIALPHRASE2

CR_PREPREVERBPHRASE1
  /* This CR reduces simple PREPREVERBPHRASES consisting of only one PREVERBPHRASE
   *
   *
   *      PREPREVERBPHRASE
   *            |             =>       PREVERBPHRASE
   *       PREVERBPHRASE
   */
  TS
    1:PREPREVERBPHRASE [_] {
      2:PREVERBPHRASE [_]
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.2.copy()
    #
  #
  ACTION
    substitute(HS_1,TS.1);
  #
# // end of CR_PREPREVERBPHRASE1


CR_BE1
  /* This CR reduces a syntax tree which consists of a Subject and a Predicate. The Predicate should
   * be a construction of BE and a VERB.
   *
   *      SENTENCE
   *          |
   *     PRESENTENCE
   *        /   \
   *       /     \
   *      /       \
   * NOUNPHRASE   PREVERBPHRASE     =>       0
   *     |              |
   * getWord(n)    VERBPHRASE
   *                 /    \
   *                BE    VERB
   *                        |
   *                    getWord(v)
   */
  TS
    1:PRESENTENCE [_]{
      2:NOUNPHRASE [_]{
        3:getWord(n)
      }
      4:PREVERBPHRASE [_]{
        5:VERBPHRASE [_] {
          6:BE [_]
          7:VERB [_]{
            8:getWord(v)
          }
```

```
        }
      }
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    addNewLocalPredicate(v,n);
    removeTS();
  #
# // end of CR_BE1


CR_VERBPHRASE1
  /* This CR reduces a syntax tree which consists of a Subject and a Predicate
   *
   *      SENTENCE
   *          |
   *     PRESENTENCE
   *        /   \
   *       /     \
   *      /       \
   * NOUNPHRASE   PREVERBPHRASE     =>       0
   *     |              |
   * getWord(n)    VERBPHRASE
   *                   |
   *                  VERB
   *                   |
   *                getWord(v)
   */
  TS
    1:PRESENTENCE [_]{
      2:NOUNPHRASE [_]{
        3:getWord(n)
      }
      4:PREVERBPHRASE [_]{
        5:VERBPHRASE [_] {
          6:VERB [_]{
            7:getWord(v)
          }
        }
      }
    }
  #
  HELP_STRUCTURES
  #
  ACTION
    addNewLocalPredicate(v,n);
    removeTS();
  #
# // end of CR_VERBPHRASE1


CR_VERBPHRASE2
  /* This CR reduces a syntax tree which consists of a Subject, a Predicate and the Object
   * of the transitive Verb
   *
   *     PRESENTENCE
   *        /   \
   *       /     \
   *      /       \
   * NOUNPHRASE   PREVERBPHRASE     =>       0
   *     |              |
   * getWord(n)    VERBPHRASE
   *                 /   \
   *                /     \
   *             VERB    NOUNPHRASE
   *              |         |
   *         getWord(v)   getWord(w)
   */
  TS
    1:PRESENTENCE [_]{
      2:NOUNPHRASE [_]{
        3:getWord(n)
      }
      4:PREVERBPHRASE [_]{
        5:VERBPHRASE [_] {
          6:VERB [_]{
            7:getWord(v)
          }
          8:NOUNPHRASE [_] {
          9:getWord(w)
          }
        }
      }
    }
  #
  HELP_STRUCTURES
```

```
  #
  ACTION
    addNewLocalPredicate(v,n,w);
    removeTS();
  #
# // end of CR_VERBPHRASE2

CR_VERBPHRASE3
  /* This CR restructures the syntax tree for a verbphrase with adverbialphrases.
   * It moves the adverbial phrases to the top of the tree, such that they can be processed later.
   *
   *     getWord(er)
   *         |
   *     PRESENTENCE                                                  PREADVERBIALPHRASE
   *        /   \                                                        /             \
   *       /     \                                                getWord(er)          PREADVERBIALPHRASE(*)
   *      /       \                                               and its subtree
   * NOUNPHRASE   PREPREVERBPHRASE     =>                         without PREADBERBIALPHRASE
   *                   |          \
   *              PREVERBPHRASE   PREADVERBIALPHRASE(*)
   *
   */
  TS
    0:getWord(er){
      1:PRESENTENCE [_]{
        2:NOUNPHRASE [_]
        3:PREPREVERBPHRASE [_]{
          4:PREVERBPHRASE [_]
          5:PREADVERBIALPHRASE[_]
        }
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:PREADVERBIALPHRASE[_]{
        2:TS.0{
          3:TS.1{
            4:TS.2.copy()
            5:TS.3{
              6:TS.4.copy()
            }
          }
        }
        7:TS.5.copy()
      }
    #
  #
  ACTION
    substitute(HS_1,TS.0);
  #
# // end of CR_VERBPHRASE3


CR_CONJOFSENTENCE3
 /* This CR deals with a conjuction of two sentences where the second sentence does not have an explicit subject
  * but uses the subject from the first sentence.
  *
  *
  *                CONJOFSENTENCE
  *                /      |      \
  *               /       |       \
  *              /        |        \
  *           SENTENCE    AND       CONJOFSENTENCE
  *              |                         |
  *        PRESENTENCE                  SENTENCE
  *          /     \                        |
  *         /       \                       |
  *    NOUNPHRASE PREPREVERBPHRASE     PRESENTENCE             =>    everything remains the same but EMPTYWORD becomes getWord(d)
  *        |                             /     \
  *    getWord(d)                       /       \
  *                           CONJOFNOUNPHRASE  PREPREVERBPHRASE
  *                                   |
  *                              NOUNPHRASE
  *                                  |
  *                              EMPTYWORD
  */
  TS
    1:CONJOFSENTENCE[_]{
      2:SENTENCE[_]{
        3:PRESENTENCE[_]{
          4:NOUNPHRASE[_]{
            5:getWord(d)
          }
          6:PREPREVERBPHRASE[_]
        }
      }
      7:AND[_]
```

```
      8:CONJOFSENTENCE[gap=HASGAP]{
        9:SENTENCE[_]{
          10:PRESENTENCE[_]{
            11:CONJOFNOUNPHRASE[_]{
              12:NOUNPHRASE[_]{
                13:EMPTYWORD[_]
              }
            }
            14:PREPREVERBPHRASE[_]
          }
        }
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:CONJOFSENTENCE[gap=HASNOGAP]{
        2:TS.9{
          3:TS.10{
            4:TS.4.copy()
            5:TS.14.copy()
          }
        }
      }
    #
  #
  ACTION
    substitute(HS_1,TS.8);
  #
# // end of CR_CONJOFSENTENCE3

CR_CONJOFSENTENCE4
 /* This CR deals with a conjuction of two sentences where the second sentence does not have an explicit subject
  * but uses the subject from the first sentence.
  *
  *
  *                 CONJOFSENTENCE
  *                /      |      \
  *               /       |       \
  *              /        |        \
  *           SENTENCE    AND       CONJOFSENTENCE_________
  *             |                         |      \        \
  *        PRESENTENCE                 SENTENCE    AND     CONJOFSENTENCE
  *          /      \                      |
  *         /        \                     |
  *    NOUNPHRASE PREPREVERBPHRASE     PRESENTENCE                    =>    everything remains the same but EMPTYWORD becomes getWord(d)
  *        |                             /     \
  *    getWord(d)                       /       \
  *                            CONJOFNOUNPHRASE  PREPREVERBPHRASE
  *                                   |
  *                               NOUNPHRASE
  *                                   |
  *                               EMPTYWORD
  */
  TS
    1:CONJOFSENTENCE[_]{
      2:SENTENCE[_]{
        3:PRESENTENCE[_]{
          4:NOUNPHRASE[_]{
            5:getWord(d)
          }
          6:PREPREVERBPHRASE[_]
        }
      }
      7:AND[_]
      8:CONJOFSENTENCE[gap = HASGAP]{
        9:SENTENCE[_]{
          10:PRESENTENCE[_]{
            11:CONJOFNOUNPHRASE[_]{
              12:NOUNPHRASE[_]{
                13:EMPTYWORD[_]
              }
            }
            14:PREPREVERBPHRASE[_]
          }
        }
        15:AND[_]
        16:CONJOFSENTENCE[_]
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      1:CONJOFSENTENCE[gap=HASNOGAP]{
        2:TS.9{
          3:TS.10{
            4:TS.4.copy()
            5:TS.14.copy()
```

```
        }
       }
       6:TS.15
       7:TS.16
      }
    #
  #
  ACTION
    substitute(HS_1,TS.8);
  #
# // end of CR_CONJOFSENTENCE4

CR_PREPRENOUN1
 /* This CR deals with a PREPRENOUN in case that there is no quantifier. The Preprenoun node is not needed here,
  * and thus the PRENOUN node below will be moved to its position.
  *
  *         PREPRENOUN
  *            |        =>     PRENOUN
  *          PRENOUN
  *
  */
  TS
    1:PREPRENOUN[_]{
      2:PRENOUN[_]
    }
  #
  HELP_STRUCTURES
    HS_1
      1:TS.2.copy()
    #
  #
  ACTION
    substitute(HS_1,TS.1);
  #
#

CR_PREPRENOUN2
 /* This CR deals with a PREPRENOUN in case of a plural nounphrase. The Prenoun node contains the name of a PDR. Now a new PDR
  * is created with the quantifier.
  *
  *         PREPRENOUN
  *           /     \            =>     PRENOUN
  *   QUANTIFIER    getWord(pdr)
  *       |
  *    getWord(q)
  */
  TS
    1:PREPRENOUN[number = PLURAL]{
      2:QUANTIFIER[_]{
        3:getWord(q)
      }
      4:getWord(pdr)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(new_pdr)
    #
  #
  ACTION
    new_pdr = addNewLocalReferentSubSet(q,pdr);
    substitute(HS_1,TS.1);
  #
# // end of CR_PREPRENOUN2

CR_SINGULAR1
 /* This CR deals with a singular discourse referent as the subject of the sentence. It puts the name of the event referent for
  * this sentence at the top of the tree.
  *
  *              SENTENCE
  *                 |
  *             PRESENTENCE
  *               /    \
  *              /      \                 =>     getWord(er_neu)
  *             /        \                              |
  *       NOUNPHRASE    PREPREVERBPHRASE           Rest of the tree
  *           |
  *      GetWord(dr)
  *
  */
  TS
    1:SENTENCE[processed = NONE]{
      2:PRESENTENCE[number = SINGULAR]{
        3:NOUNPHRASE[number = SINGULAR]{
          4:getWord(dr)
        }
        5:PREPREVERBPHRASE[_]
      }
```

```
    }
  #
  HELP_STRUCTURES
    HS_1
      1:getWord(er_neu){
        2:SENTENCE[processed = SYNTAX]{
          3:TS.2.copy()
        }
      }
    #
  #
  ACTION
    er_neu = getNextFreeEventReferent();
    substitute(HS_1,TS.1);
  #
# // end of CR_SINGULAR1

CR_PLURAL1
 /* This CR deals with a plural discourse referent as the subject of the sentence.
  * It creates a DuplexCondition of the PDR with the quantifier all. Thus the PREVERBPHRASE is valid for all
  * elements of the plural discourse referent. The triggering structure is replaced with the Name of the event referent of this sentence.
  *
  *             getWord(er)
  *                 |
  *              SENTENCE
  *                 |
  *             PRESENTENCE
  *               /    \
  *              /      \                => getWord(er)
  *             /        \
  *       NOUNPHRASE    PREPREVERBPHRASE
  *           |
  *      GetWord(pdr)
  *
  */
  TS
    0:getWord(er){
      1:SENTENCE[_]{
        2:PRESENTENCE[number = PLURAL]{
          3:NOUNPHRASE[number = PLURAL]{
            4:getWord(pdr)
          }
          5:PREPREVERBPHRASE[_]
        }
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      0:STARTSYMBOL[_]{
        1:getWord(er){
          2:SENTENCE[processed = DUPLEX]{
            3:PRESENTENCE[number = SINGULAR]{
              4:NOUNPHRASE[number = SINGULAR]{
                5:getWord(d)
              }
              6:TS.5.copy()
            }
          }
        }
      }
    #
    HS_2
      1:getWord(er)
    #
  #
  ACTION
    left = buildEmptyDRS(POSITIVE);
    d = getNextFreeDR(left);
    pred = buildPredicate(ISELEM,d,pdr);
    addPredicateToDRS(pred,left);
    right = buildDRS(HS_1,POSITIVE);
    addNewLocalDuplexCondition(left,ALL,d,right);
    substitute(HS_2,TS.1);
  #
# // end of CR_PLURAL1


CR_PLURAL2
 /* This CR deals with a plural discourse referent as the subject of the sentence.
  * It creates a DuplexCondition of the PDR with the quantifier all. Thus the PREVERBPHRASE is valid for all
  * elements of the plural discourse referent. The triggering structure is replaced with the Name of the event referent of this sentence.
  *
  *              SENTENCE
  *                 |
  *             PRESENTENCE
  *               /    \
  *              /      \                => getWord(er_neu)
```

```
  *            /       \
  *       NOUNPHRASE    PREPREVERBPHRASE
  *          |
  *      GetWord(pdr)
  *
  */
  TS
    1:SENTENCE[_]{
      2:PRESENTENCE[number = PLURAL]{
        3:NOUNPHRASE[number = PLURAL]{
          4:getWord(pdr)
        }
        5:PREPREVERBPHRASE[_]
      }
    }
  #
  HELP_STRUCTURES
    HS_1
      0:STARTSYMBOL[_]{
        1:getWord(er_neu){
          2:SENTENCE[processed = DUPLEX]{
            3:PRESENTENCE[number = SINGULAR]{
              4:NOUNPHRASE[number = SINGULAR]{
                5:getWord(d)
              }
              6:TS.5.copy()
            }
          }
        }
      }
    #
    HS_2
      1:getWord(er_neu)
    #
  #
  ACTION
    left = buildEmptyDRS(POSITIVE);
    d = getNextFreeDR(left);
    pred = buildPredicate(ISELEM,d,pdr);
    er_neu = getNextFreeEventReferent();
    addPredicateToDRS(pred,left);
    right = buildDRS(HS_1,POSITIVE);
    addNewLocalDuplexCondition(left,ALL,d,right);
    substitute(HS_2,TS.1);
  #
# // end of CR_PLURAL2

CR_PLURAL3
  /* This CR deals with a quantified plural discourse referent as an object.
   * The Prepreverbphrase does not have an ADVERBIALPHRASE. The name of the event referent is already given.
   *
   *          getWord(er)
   *              |
   *           SENTENCE
   *              |
   *          PRESENTENCE
   *            /    \
   *           /      \
   *    NOUNPHRASE   PREPREVERBPHRASE
   *       |                |
   *    getWord(DRx)   PREVERBPHRASE
   *                       |
   *                     VERBPHRASE              => getWord(er)
   *                     /     \
   *                  VERB    NOUNPHRASE
   *                             |
   *                          getWord(PDRy)
   */
  TS
    0:getWord(er){
      1:SENTENCE[_]{
        2:PRESENTENCE[_]{
          3:NOUNPHRASE[_]{
            4:getWord(d_subjekt)
          }
          5:PREPREVERBPHRASE[_]{
            6:PREVERBPHRASE[_]{
              7:VERBPHRASE[_]{
                8:VERB[_]
                9:NOUNPHRASE[number = PLURAL]{
                  10:getWord(pdr)
                }
              }
            }
          }
        }
      }
    }
```

```
  #
  HELP_STRUCTURES
    HS_1
      0:STARTSYMBOL[_]{
        1:TS.0{
          2:SENTENCE[processed = DUPLEX]{
            3:TS.2{
              4:TS.3.copy()
              5:TS.5{
                6:TS.6{
                  7:TS.7{
                    8:TS.8.copy()
                    9:NOUNPHRASE[number = SINGULAR]{
                      10:getWord(d)
                    }
                  }
                }
              }
            }
          }
        }
      }
    #
    HS_2
      1:getWord(er)
    #
  #
  ACTION
    left = buildEmptyDRS(POSITIVE);
    d = getNextFreeDR(left);
    pred = buildPredicate(ISELEM,d,pdr);
    addPredicateToDRS(pred,left);
    right = buildDRS(HS_1,POSITIVE);
    addNewLocalDuplexCondition(left,ALL,d,right);
    substitute(HS_2,TS.0);
  #
# //  end of CR_PLURAL3


CR_PREPOSITION1
  /* replaces a preposition of time with a conjunction which has the same time relation.
   *
   *     PREPOSITION[adverbial_kind = TIME, time_relation = t]        CONJUNCTION[time_relation = t]
   *          |                                                  =>          |
   *      getWord(p)                                                       getWord(p)
   */
  TS
    1:PREPOSITION[adverbial_kind=TIME,time_relation=t]{
      2:getWord(p)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:CONJUNCTION[time_relation = t]{
        2:getWord(p)
      }
    #
  #
  ACTION
    substitute(HS_1,TS.1);
  #
# // end of CR_PREPOSITION1



CR_CONJUNCTION1
  /* replaces the string of the word which is a conjunction of time, and time relation AFTER by the reserved name AFTER
   *
   *      CONJUNCTION        CONJUCNTION
   *          |          =>      |
   *        getWord(c)        getWord(AFTER)
   */
  TS
    1:CONJUNCTION[time_relation = AFTER]{
      2:getWord(c)
    }
  #
  HELP_STRUCTURES
    HS_1
      1:CONJUNCTION[time_relation = NONE]{
        2:getWord(r)
      }
    #
  #
  ACTION
    r = getString(AFTER);
    substitute(HS_1,TS.1);
  #
```

```
  # // end of CR_CONJUNCTION1

  CR_CONJUNCTION2
    /* replaces the string of the word which is a conjunction of time, and time relation BEFORE by the reserved name BEFORE
     *
     *      CONJUNCTION        CONJUCNTION
     *          |          =>      |
     *        getWord(c)        getWord(BEFORE)
     */
    TS
      1:CONJUNCTION[time_relation = BEFORE]{
        2:getWord(c)
      }
    #
    HELP_STRUCTURES
      HS_1
        1:CONJUNCTION[time_relation = NONE]{
          2:getWord(r)
        }
      #
    #
    ACTION
      r = getString(BEFORE);
      substitute(HS_1,TS.1);
    #
  # // end of CR_CONJUNCTION2

  CR_CONJUNCTION3
    /* replaces the string of the word which is a conjunction of time, and time relation WHILE by the reserved name WHILE
     *
     *      CONJUNCTION        CONJUCNTION
     *          |          =>      |
     *        getWord(c)        getWord(WHILE)
     */
    TS
      1:CONJUNCTION[time_relation = WHILE]{
        2:getWord(c)
      }
    #
    HELP_STRUCTURES
      HS_1
        1:CONJUNCTION[time_relation = NONE]{
          2:getWord(r)
        }
      #
    #
    ACTION
      r = getString(WHILE);
      substitute(HS_1,TS.1);
    #
  # // end of CR_CONJUNCTION3


  CONSTRUCTIONRULE_ORDER
    CR_ADVERBIALPHRASE2,CR_ADVERBIALPHRASE3,
    CR_PREPRENOUN1,
    CR_NOUNPHRASE1,CR_NOUNPHRASE2,CR_NOUNPHRASE3,CR_NOUNPHRASE4,
    CR_NOUNPHRASE5,CR_NOUNPHRASE6,CR_NOUNPHRASE7,CR_NOUNPHRASE8,CR_NOUNPHRASE9,
    CR_PREPRENOUN2,
    CR_PRONOUN1,
    CR_RELATIVECLAUSE1,
    CR_ADVERBIALPHRASE4,
    CR_DETERMINER1,
    CR_CONJOFNOUNPHRASE1,
    CR_CONJOFNOUNPHRASE2,
    CR_CONJOFSENTENCE4,CR_CONJOFSENTENCE3,
    CR_CONJOFSENTENCE2, CR_CONJOFSENTENCE1,
    CR_SINGULAR1,CR_PLURAL1,CR_PLURAL2,CR_PLURAL3,
    CR_SENTENCE1,
    CR_VERBPHRASE3,
    CR_PREPREVERBPHRASE1,
    CR_BE1,
    CR_VERBPHRASE1, CR_VERBPHRASE2,
    CR_ADVERBIALPHRASE0,CR_ADVERBIALPHRASE1,
    CR_PREADVERBIALPHRASE1,
    CR_ADVERBIALPHRASE7,
    CR_PREPOSITION1,
    CR_CONJUNCTION1,CR_CONJUNCTION2,CR_CONJUNCTION3,
    CR_ADVERBIALPHRASE6,CR_ADVERBIALPHRASE5,
    CR_PREADVERBIALPHRASE2,
    CR_START,
    CR_TEXT2, CR_TEXT1
  #

*
```

A.1.2 Attribut-Verzeichnis

number Werte: [**SINGULAR, PLURAL**]
gibt den Numerus eines Nichtterminals (NT) an.

gender Werte: [**MALE, FEMALE, NOTHUMAN**]
gibt den Genus eines NT an.

casus Werte : [**NOMINATIVE, NOTNOMINATIVE**]
gibt den Kasus eines NT an.

gap Werte : [**HASGAP, HASNOGAP**]
Wird zur Behandlung von Relativsätzen und Abverbialphrasen benötigt und gibt an, ob in dem entsprechenden Satzteil eine Nominalphrase eines anderen Satzteils Subjekt oder Objekt ist.

transitivity Werte: [**TRANSITIVE, INTRANSITIVE**]
gibt die Transitivität eines NT an.

finitivity Werte: [**FINITE, INFINITE, GERUND**]
gibt an, für welche Form eines Verbes ein NT steht.

countability Werte: [**COUNTABLE, UNCOUNTABLE**]
gibt an, ob ein Nomen bzw. eine Nomenphrase zählbar ist.

adverbial_kind Werte: [**MANNER, PLACE, TIME**]
gibt an, ob eine Adverbialphrase die Art und Weise, eine örtliche oder eine zeitliche Eigenschaft einer Verbphrase genauer beschreibt.

time_relation Werte: [**NONE, AFTER, BEFORE, WHILE**]
gibt an, in welche zeitliche Relation zwei Ereignisreferenten, die durch ein Nichtterminal mit diesem Attribut verknüpft sind, gestellt werden sollen.

A.1.3 Nichtterminale und ihre Attribute

Nichtterminal	Attribute
VERB	[**number, finitivity, transitivity**]
NOUN	[**number, gender, casus, adverbial_kind, countability**]
DETERMINER	[**number**]
PRONOUN	[**number, gender, casus**]
PREPOSITION	[**adverbial_kind, time_relation**]
PROPERNAME	[**number, gender, casus, adverbial_kind**]
ADJECTIVE	[**adverbial_kind**]
BE	[**number, finitivity**]
RELATIVEPRONOUN	[**number, gender**]
AND	[ ]
DOT	[ ]
ADVERB	[**adverbial_kind**]
CONJUNCTION	[**adverbial_kind, time_relation**]
QUANTIFIER	[ ]

A.1.4 Morphologie

Im folgenden sind die Morphologieregeln der Grammatik aufgeführt. Dabei steht `cons` für einen Konsonaten und `vowel` für einen Vokal.

Morphologie für Wörter aus der Wortart VERB

Endungen auf “ed” für die FINITE Form

`vowel` “y”	⟶	`vowel` “yed”
`cons` “y”	⟶	`cons` “ied”
`cons1 vowel cons2`	⟶	`cons1 vowel cons2 cons2` “ed”
“e”	⟶	“ed”

Endungen auf “ing” für die GERUND Form

“e”	⟶	“ing”
“ie”	⟶	“ying”
`cons1 vowel cons2`	⟶	`cons1 vowel cons2 cons2` “ing”

Morphologie für Wörter aus der Wortart NOUN

Endungen auf “s” für die PLURAL Form

“o”	⟶	“oes”
“ss”	⟶	“sses”
“sh”	⟶	“shes”
“ch”	⟶	“ches”
“x”	⟶	“xes”
`vowel` “y”	⟶	`vowel` “ys”
`cons` “y”	⟶	`cons` “ies”
`vowel` “s”	⟶	`vowel` “sses”
“f”	⟶	“ves”
“fe”	⟶	“ves”

A.1.5 Grammatikregeln

$$\underset{\left[\ \right]}{\texttt{STARTSYMBOL}} \rightarrow \underset{\left[\ \right]}{\texttt{TEXT}}$$

$$\underset{\left[\ \right]}{\texttt{TEXT}} \rightarrow \underset{\left[\ gap = HASNOGAP\ \right]}{\texttt{CONJOFSENTENCE}}\ \underset{\left[\ \right]}{\texttt{DOT}}\ \underset{\left[\ \right]}{\texttt{TEXT}}$$

$$\underset{\left[\ \right]}{\texttt{TEXT}} \rightarrow \underset{\left[\ gap = HASNOGAP\ \right]}{\texttt{CONJOFSENTENCE}}\ \underset{\left[\ \right]}{\texttt{DOT}}$$

$$\underset{\left[\ gap = c\ \right]}{\texttt{CONJOFSENTENCE}} \rightarrow \underset{\left[\ gap = c\ \right]}{\texttt{SENTENCE}}\ \underset{\left[\ \right]}{\texttt{AND}}\ \underset{\left[\ \right]}{\texttt{CONJOFSENTENCE}}$$

$$\underset{\left[\ gap = c\ \right]}{\texttt{CONJOFSENTENCE}} \rightarrow \underset{\left[\ gap = c\ \right]}{\texttt{SENTENCE}}$$

$$\underset{\left[\ gap = c\ \right]}{\texttt{SENTENCE}} \rightarrow \underset{\left[\ \right]}{\texttt{ADVERBIALPHRASE}}\ \underset{\left[\ gap = c\ \right]}{\texttt{PRESENTENCE}}$$

$$\underset{\left[\ gap = c\ \right]}{\texttt{SENTENCE}} \rightarrow \underset{\left[\ gap = c\ \right]}{\texttt{PRESENTENCE}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = c \\ finitivity = FINITE \end{array}\right]}{\texttt{PRESENTENCE}} \rightarrow \underset{\left[\begin{array}{l} number = n \\ gap = c \end{array}\right]}{\texttt{CONJOFNOUNPHRASE}}\ \underset{\left[\begin{array}{l} number = n \\ finitivity = FINITE \end{array}\right]}{\texttt{PREPREVERBPHRASE}}$$

$$\underset{\left[\begin{array}{l} number = PLURAL \\ gap = HASNOGAP \end{array}\right]}{\texttt{CONJOFNOUNPHRASE}} \rightarrow \underset{\left[\ gap = HASNOGAP\ \right]}{\texttt{NOUNPHRASE}}\ \underset{\left[\ \right]}{\texttt{AND}}\ \underset{\left[\ gap = HASNOGAP\ \right]}{\texttt{CONJOFNOUNPHRASE}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = c \end{array}\right]}{\texttt{CONJOFNOUNPHRASE}} \rightarrow \underset{\left[\begin{array}{l} number = n \\ gap = c \end{array}\right]}{\texttt{NOUNPHRASE}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = HASNOGAP \\ adverbial\_kind = k \end{array}\right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\ number = n\ \right]}{\texttt{DETERMINER}}\ \underset{\left[\begin{array}{l} number = n \\ countability = COUNTABLE \\ adverbial\_kind = k \end{array}\right]}{\texttt{PREPRENOUN}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = HASNOGAP \\ adverbial\_kind = k \end{array}\right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\begin{array}{l} number = n \\ adverbial\_kind = k \end{array}\right]}{\texttt{PROPERNAME}}$$

$$\underset{\left[\begin{array}{l} number = PLURAL \\ gap = HASNOGAP \\ adverbial\_kind = k \end{array}\right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\begin{array}{l} number = PLURAL \\ countability = COUNTABLE \\ adverbial\_kind = k \end{array}\right]}{\texttt{PREPRENOUN}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = HASNOGAP \\ adverbial\_kind = k \end{array}\right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\begin{array}{l} number = n \\ countability = UNCOUNTABLE \\ adverbial\_kind = k \end{array}\right]}{\texttt{PREPRENOUN}}$$

$$\underset{\left[\begin{array}{l} number = n \\ gap = HASNOGAP \end{array}\right]}{\texttt{NOUNPHRASE}} \rightarrow \underset{\left[\ number = n\ \right]}{\texttt{PRONOUN}}$$

$$\underset{\text{NOUNPHRASE}}{\begin{bmatrix} number = n \\ gap = HASGAP \end{bmatrix}} \rightarrow \underset{\text{EMPTYWORD}}{\begin{bmatrix} number = n \end{bmatrix}}$$

$$\underset{\text{PREPRENOUN}}{\begin{bmatrix} number = PLURAL \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{QUANTIFIER}}{\begin{bmatrix} \end{bmatrix}} \underset{\text{PRENOUN}}{\begin{bmatrix} number = PLURAL \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}}$$

$$\underset{\text{PREPRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{PRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}}$$

$$\underset{\text{PRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{ADJECTIVE}}{\begin{bmatrix} \end{bmatrix}} \underset{\text{NOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \underset{\text{RELATIVECLAUSE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{NOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \underset{\text{RELATIVECLAUSE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{ADJECTIVE}}{\begin{bmatrix} \end{bmatrix}} \underset{\text{NOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}}$$

$$\underset{\text{PRENOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}} \rightarrow \underset{\text{NOUN}}{\begin{bmatrix} number = n \\ countability = c \\ adverbial\_kind = k \\ gender = g \\ casus = f \end{bmatrix}}$$

$$\underset{\text{VERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = FINITE \end{bmatrix}} \rightarrow \underset{\text{BE}}{\begin{bmatrix} number = n \\ finitivity = FINITE \end{bmatrix}} \underset{\text{VERB}}{\begin{bmatrix} transitivity = a \\ finitivity = GERUND \end{bmatrix}}$$

$$\underset{\text{VERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = TRANSITIVE \\ finitivity = a \end{bmatrix}} \rightarrow \underset{\text{VERB}}{\begin{bmatrix} number = n \\ transitivity = TRANSITIVE \\ finitivity = a \end{bmatrix}} \underset{\text{CONJOFNOUNPHRASE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{VERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = INTRANSITIVE \\ finitivity = a \end{bmatrix}} \rightarrow \underset{\text{VERB}}{\begin{bmatrix} number = n \\ transitivity = INTRANSITIVE \\ finitivity = a \end{bmatrix}}$$

$$\underset{\text{PREVERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}} \rightarrow \underset{\text{VERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}}$$

$$\underset{\text{PREPREVERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}} \rightarrow \underset{\text{PREVERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}} \underset{\text{PREADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PREPREVERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}} \rightarrow \underset{\text{PREVERBPHRASE}}{\begin{bmatrix} number = n \\ transitivity = a \\ finitivity = b \end{bmatrix}}$$

$$\underset{\text{RELATIVECLAUSE}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{RELATIVEPRONOUN}}{\begin{bmatrix} \end{bmatrix}} \underset{\text{PRESENTENCE}}{\begin{bmatrix} gap = HASGAP \end{bmatrix}}$$

$$\underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \rightarrow \underset{\text{ADVERB}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}}$$

$$\underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \rightarrow \underset{\text{CONJUNCTION}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \underset{\text{PRESENTENCE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \rightarrow \underset{\text{PREPOSITION}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \underset{\text{NOUNPHRASE}}{\begin{bmatrix} gap = HASNOGAP \\ adverbial\_kind = k \end{bmatrix}}$$

$$\underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \rightarrow \underset{\text{CONJUNCTION}}{\begin{bmatrix} adverbial\_kind = k \end{bmatrix}} \underset{\text{PREPREVERBPHRASE}}{\begin{bmatrix} finitivity = GERUND \end{bmatrix}}$$

$$\underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} adverbial\_kind = MANNER \end{bmatrix}} \rightarrow \underset{\text{PREPREVERBPHRASE}}{\begin{bmatrix} finitivity = GERUND \end{bmatrix}}$$

$$\underset{\text{PREADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}} \underset{\text{PREADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PREADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{ADVERBIALPHRASE}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{NOUN}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PROPERNAME}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

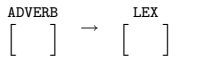
$$\underset{\text{VERB}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

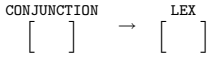
$$\underset{\text{DETERMINER}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

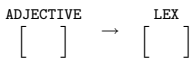
$$\underset{\text{PRONOUN}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

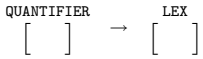
$$\underset{\text{RELATIVEPRONOUN}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$

$$\underset{\text{PREPOSITION}}{\begin{bmatrix} \end{bmatrix}} \rightarrow \underset{\text{LEX}}{\begin{bmatrix} \end{bmatrix}}$$


ADVERB
LEX


CONJUNCTION
LEX


ADJECTIVE
LEX


QUANTIFIER
LEX


BE
LEX


AND
LEX

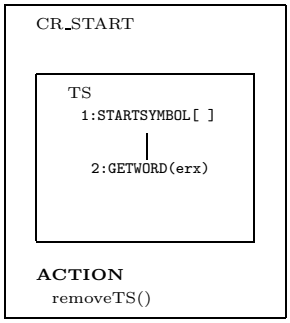

DOT
LEX

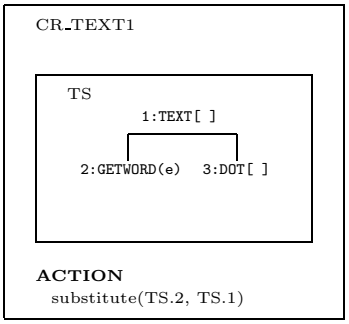
A.1.6 Reihenfolge der Konstruktionsregeln

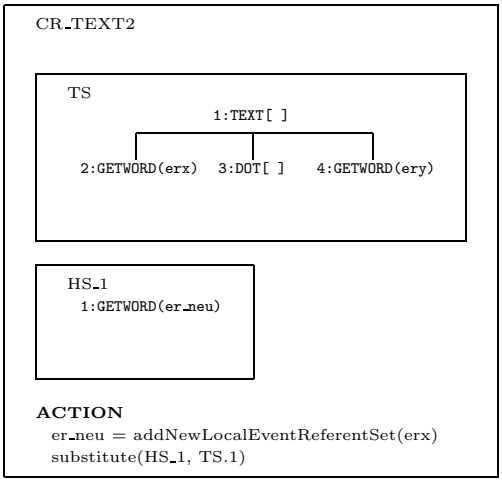
1. CR_ADVERBIALPHRASE2
2. CR_ADVERBIALPHRASE3
3. CR_PREPRENOUN1
4. CR_NOUNPHRASE1
5. CR_NOUNPHRASE2
6. CR_NOUNPHRASE3
7. CR_NOUNPHRASE4
8. CR_NOUNPHRASE5
9. CR_NOUNPHRASE6
10. CR_NOUNPHRASE7
11. CR_NOUNPHRASE8
12. CR_NOUNPHRASE9
13. CR_PREPRENOUN2
14. CR_PRONOUN1
15. CR_RELATIVECLAUSE1
16. CR_ADVERBIALPHRASE4
17. CR_DETERMINER1
18. CR_CONJOFNOUNPHRASE1
19. CR_CONJOFNOUNPHRASE2
20. CR_CONJOFSENTENCE4
21. CR_CONJOFSENTENCE3
22. CR_CONJOFSENTENCE2
23. CR_CONJOFSENTENCE1
24. CR_SINGULAR1
25. CR_PLURAL1
26. CR_PLURAL2
27. CR_PLURAL3
28. CR_SENTENCE1
29. CR_VERBPHRASE3
30. CR_PREPREVERBPHRASE1
31. CR_BE1
32. CR_VERBPHRASE1
33. CR_VERBPHRASE2
34. CR_ADVERBIALPHRASE0
35. CR_ADVERBIALPHRASE1
36. CR_PREADVERBIALPHRASE1

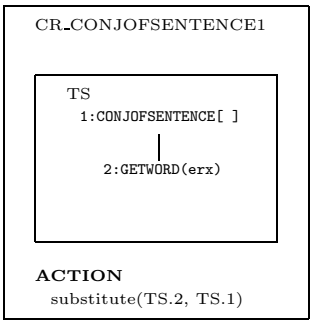
37. CR_ADVERBIALPHRASE7
38. CR_PREPOSITION1
39. CR_CONJUNCTION1
40. CR_CONJUNCTION2
41. CR_CONJUNCTION3
42. CR_ADVERBIALPHRASE6
43. CR_ADVERBIALPHRASE5
44. CR_PREADVERBIALPHRASE2
45. CR_START
46. CR_TEXT2
47. CR_TEXT1

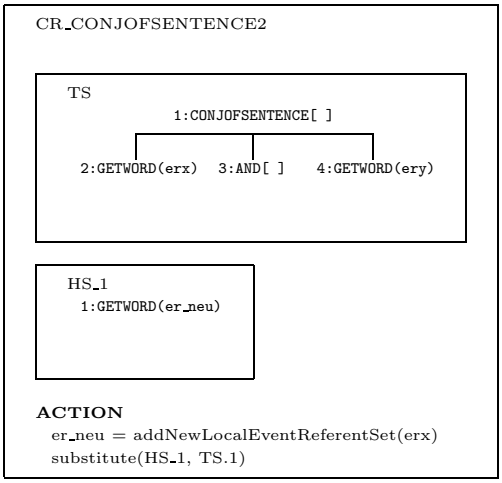
A.1.7 Konstruktionsregeln

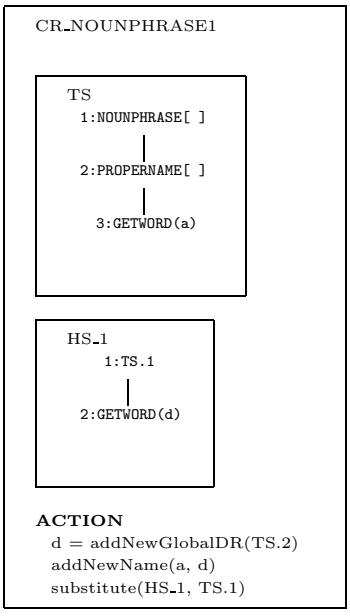


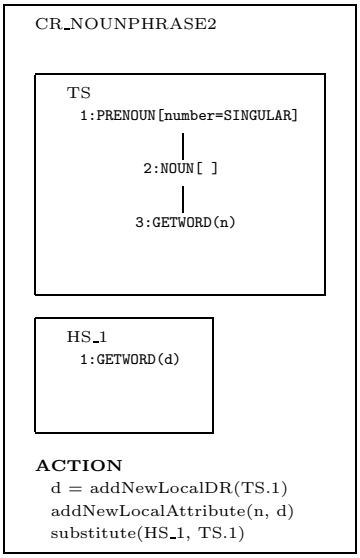


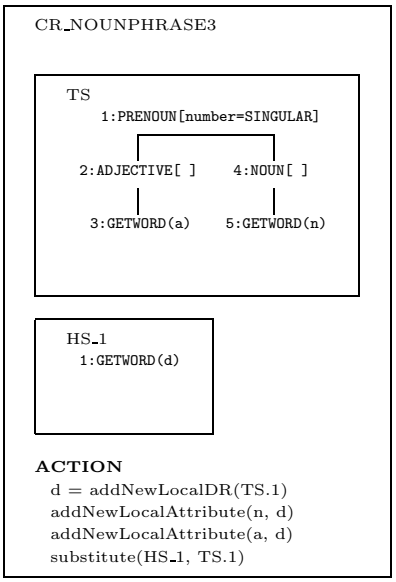


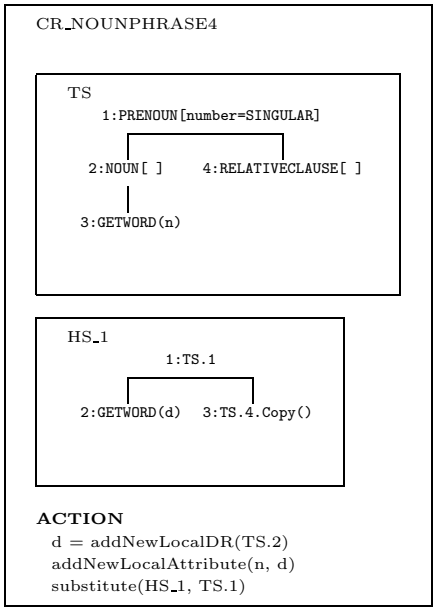


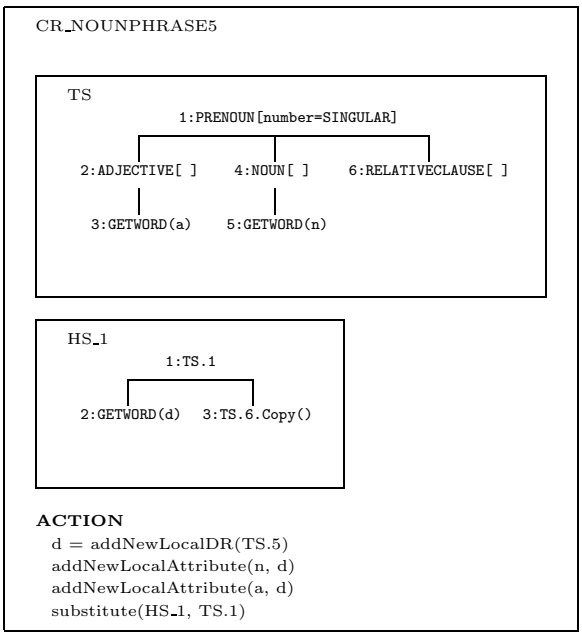

CR_CONJOFSENTENCE2
TS
1:CONJOFSENTENCE[ ]
2:GETWORD(erx)
3:AND[ ]
4:GETWORD(ery)
HS_1
1:GETWORD(er_neu)
ACTION
er_neu = addNewLocalEventReferentSet(erx)
substitute(HS_1, TS.1)


CR_NOUNPHRASE1
TS
1:NOUNPHRASE[ ]
2:PROPERNAME[ ]
3:GETWORD(a)
HS_1
1:TS.1
2:GETWORD(d)
ACTION
d = addNewGlobalDR(TS.2)
addNewName(a, d)
substitute(HS_1, TS.1)

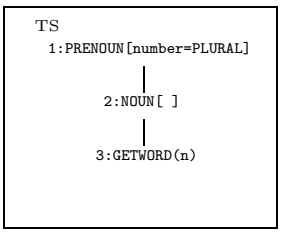

CR_NOUNPHRASE2
TS
1:PRENOUN[number=SINGULAR]
2:NOUN[ ]
3:GETWORD(n)
HS_1
1:GETWORD(d)
ACTION
d = addNewLocalDR(TS.1)
addNewLocalAttribute(n, d)
substitute(HS_1, TS.1)

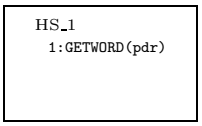

CR_NOUNPHRASE3
TS
1:PRENOUN[number=SINGULAR]
2:ADJECTIVE[ ]
4:NOUN[ ]
3:GETWORD(a)
5:GETWORD(n)
HS_1
1:GETWORD(d)
ACTION
d = addNewLocalDR(TS.1)
addNewLocalAttribute(n, d)
addNewLocalAttribute(a, d)
substitute(HS_1, TS.1)


CR_NOUNPHRASE4
TS
1:PRENOUN[number=SINGULAR]
2:NOUN[ ]
4:RELATIVECLAUSE[ ]
3:GETWORD(n)
HS_1
1:TS.1
2:GETWORD(d)
3:TS.4.Copy()
ACTION
d = addNewLocalDR(TS.2)
addNewLocalAttribute(n, d)
substitute(HS_1, TS.1)


CR_NOUNPHRASE5
TS
1:PRENOUN[number=SINGULAR]
2:ADJECTIVE[ ]
4:NOUN[ ]
6:RELATIVECLAUSE[ ]
3:GETWORD(a)
5:GETWORD(n)
HS_1
1:TS.1
2:GETWORD(d)
3:TS.6.Copy()
ACTION
d = addNewLocalDR(TS.5)
addNewLocalAttribute(n, d)
addNewLocalAttribute(a, d)
substitute(HS_1, TS.1)

CR_NOUNPHRASE6

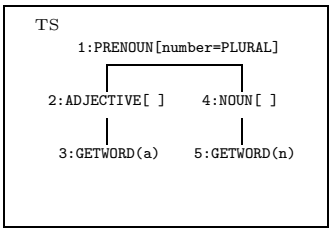


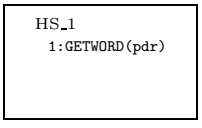


ACTION

drs = buildEmptyDRS(POSITIVE)
dr = getNextFreeDR(drs)
a = buildAttribute(n, dr)
addAttributeToDRS(a, drs)
pdr = addNewLocalReferentSet(dr, drs)
substitute(HS_1, TS.1)

CR_NOUNPHRASE7



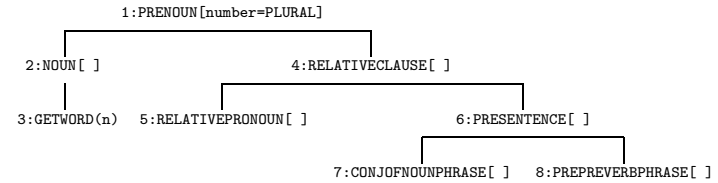


ACTION

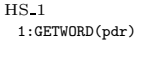
drs = buildEmptyDRS(POSITIVE)
dr = getNextFreeDR(drs)
attr = buildAttribute(n, dr)
addAttributeToDRS(attr, drs)
attr = buildAttribute(a, dr)
addAttributeToDRS(attr, drs)
pdr = addNewLocalReferentSet(dr, drs)
substitute(HS_1, TS.1)

CR_NOUNPHRASE8

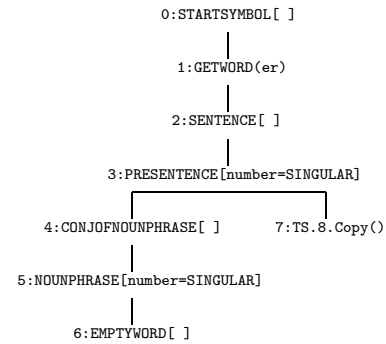
TS



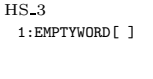
HS_1



HS_2



HS_3



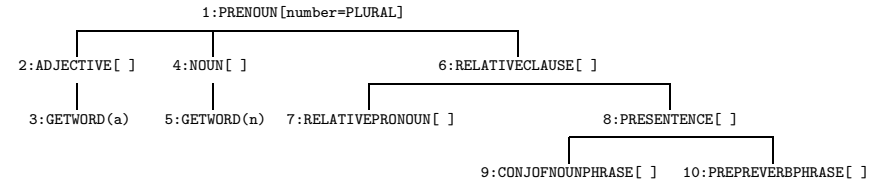
HS_4

ACTION

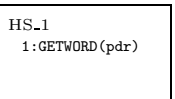
```
er = getNextFreeEventReferent()
g = findNodeOfType(HS_3, HS_2)
drs = buildDRS(HS_2, POSITIVE)
dr = getNextFreeDR(drs)
substitute(HS_4, g)
a = buildAttribute(n, dr)
addAttributeToDRS(a, drs)
pdr = addNewLocalReferentSet(dr, drs)
substitute(HS_1, TS.1)
```

CR_NOUNPHRASE9

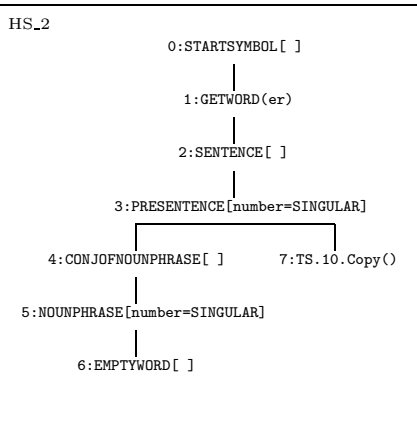
TS



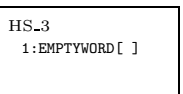
HS_1



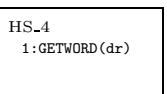
HS_2



HS_3

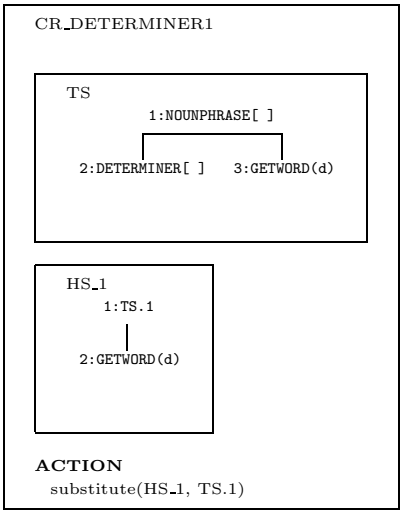


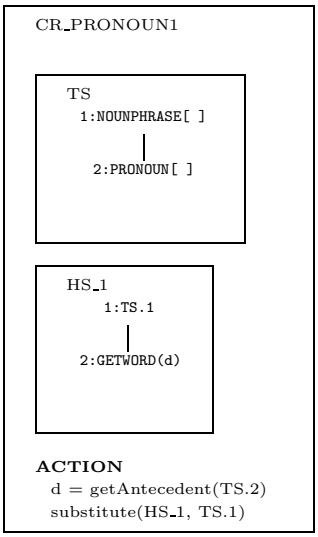
HS_4

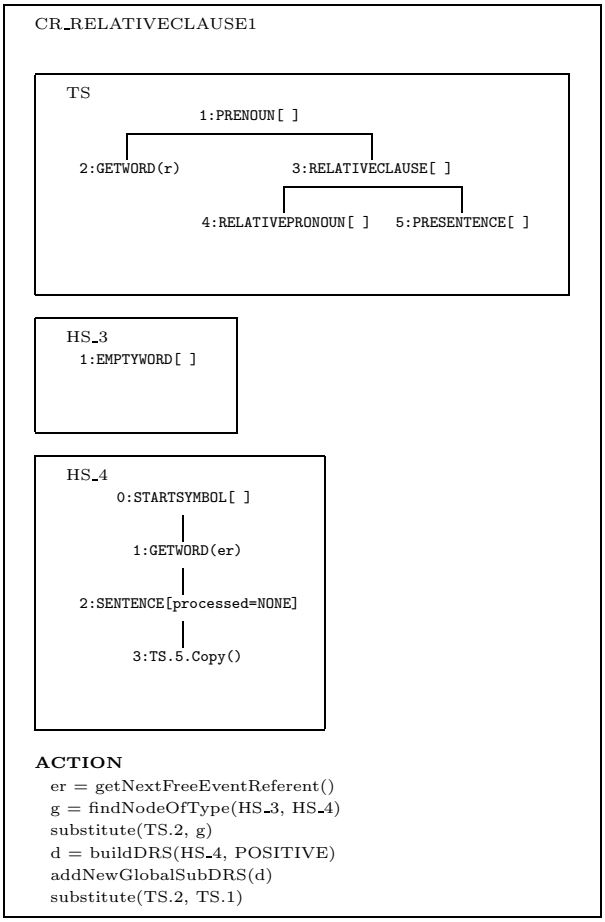


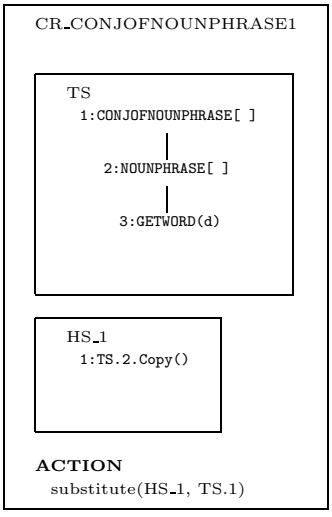
ACTION

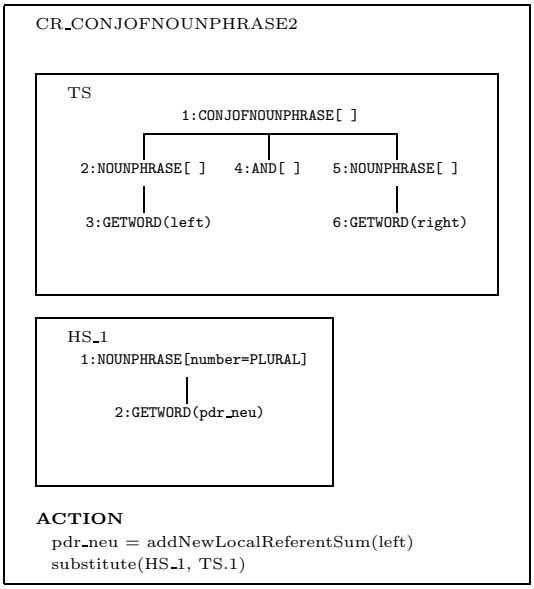
```
er = getNextFreeEventReferent()
g = findNodeOfType(HS_3, HS_2)
drs = buildDRS(HS_2, POSITIVE)
dr = getNextFreeDR(drs)
substitute(HS_4, g)
attr = buildAttribute(n, dr)
addAttributeToDRS(attr, drs)
attr = buildAttribute(a, dr)
addAttributeToDRS(attr, drs)
pdr = addNewLocalReferentSet(dr, drs)
substitute(HS_1, TS.1)
```

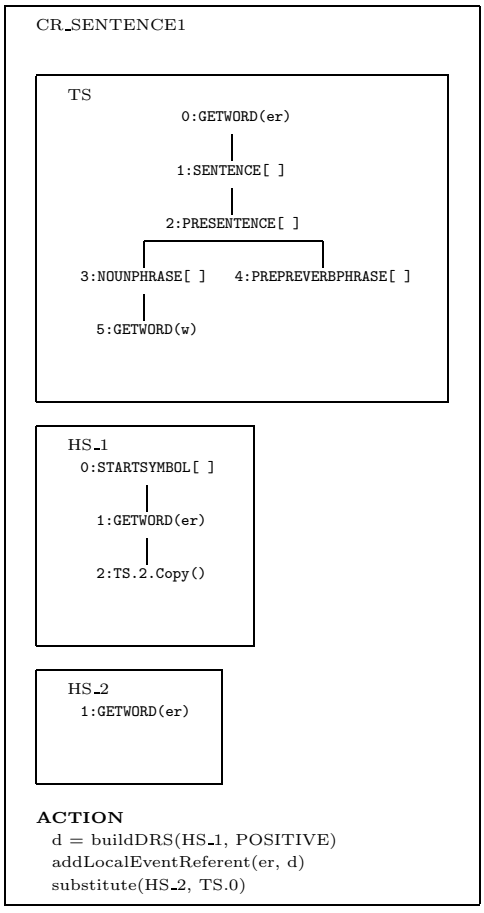

CR_DETERMINER1
TS
1:NOUNPHRASE[ ]
2:DETERMINER[ ]
3:GETWORD(d)
HS_1
1:TS.1
2:GETWORD(d)
ACTION
substitute(HS_1, TS.1)

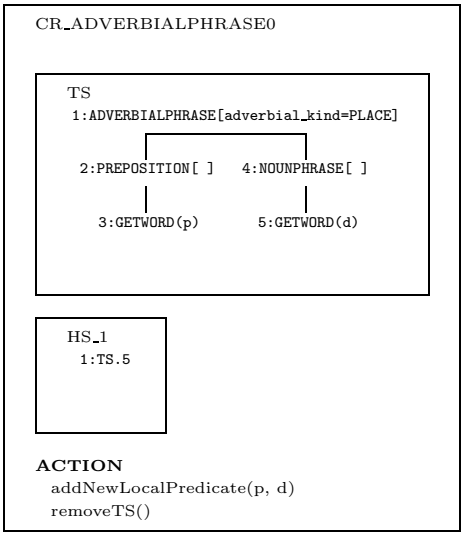

CR_PRONOUN1
TS
1:NOUNPHRASE[ ]
2:PRONOUN[ ]
HS_1
1:TS.1
2:GETWORD(d)
ACTION
d = getAntecedent(TS.2)
substitute(HS_1, TS.1)

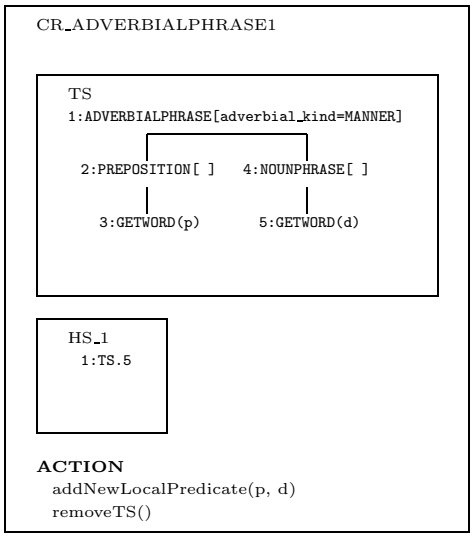

CR_RELATIVECLAUSE1
TS
1:PRENOUN[ ]
2:GETWORD(r)
3:RELATIVECLAUSE[ ]
4:RELATIVEPRONOUN[ ]
5:PRESENTENCE[ ]
HS_3
1:EMPTYWORD[ ]
HS_4
0:STARTSYMBOL[ ]
1:GETWORD(er)
2:SENTENCE[processed=NONE]
3:TS.5.Copy()
ACTION
er = getNextFreeEventReferent()
g = findNodeOfType(HS_3, HS_4)
substitute(TS.2, g)
d = buildDRS(HS_4, POSITIVE)
addNewGlobalSubDRS(d)
substitute(TS.2, TS.1)


CR_CONJOFNOUNPHRASE1
TS
1:CONJOFNOUNPHRASE[ ]
2:NOUNPHRASE[ ]
3:GETWORD(d)
HS_1
1:TS.2.Copy()
ACTION
substitute(HS_1, TS.1)


CR_CONJOFNOUNPHRASE2
TS
1:CONJOFNOUNPHRASE[ ]
2:NOUNPHRASE[ ]
4:AND[ ]
5:NOUNPHRASE[ ]
3:GETWORD(left)
6:GETWORD(right)
HS_1
1:NOUNPHRASE[number=PLURAL]
2:GETWORD(pdr_neu)
ACTION
pdr_neu = addNewLocalReferentSum(left)
substitute(HS_1, TS.1)

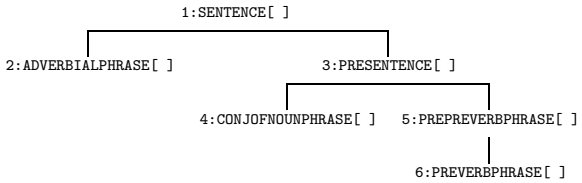

CR_SENTENCE1
TS
0:GETWORD(er)
1:SENTENCE[ ]
2:PRESENTENCE[ ]
3:NOUNPHRASE[ ]
4:PREPREVERBPHRASE[ ]
5:GETWORD(w)
HS_1
0:STARTSYMBOL[ ]
1:GETWORD(er)
2:TS.2.Copy()
HS_2
1:GETWORD(er)
ACTION
d = buildDRS(HS_1, POSITIVE)
addLocalEventReferent(er, d)
substitute(HS_2, TS.0)


CR_ADVERBIALPHRASE0
TS
1:ADVERBIALPHRASE[adverbial_kind=PLACE]
2:PREPOSITION[ ]
4:NOUNPHRASE[ ]
3:GETWORD(p)
5:GETWORD(d)
HS_1
1:TS.5
ACTION
addNewLocalPredicate(p, d)
removeTS()

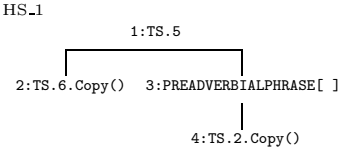

CR_ADVERBIALPHRASE1
TS
1:ADVERBIALPHRASE[adverbial_kind=MANNER]
2:PREPOSITION[ ]
4:NOUNPHRASE[ ]
3:GETWORD(p)
5:GETWORD(d)
HS_1
1:TS.5
ACTION
addNewLocalPredicate(p, d)
removeTS()

CR_ADVERBIALPHRASE2

TS



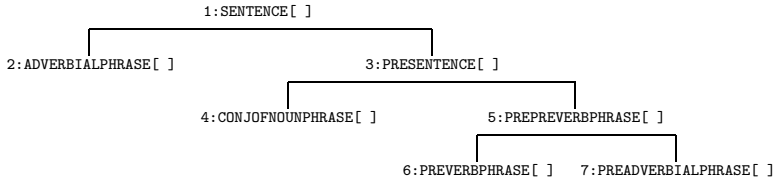
HS_1



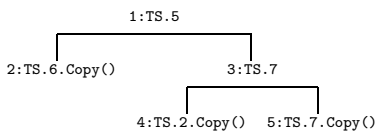
ACTION
substitute(HS_1, TS.5)
remove(TS.2)

CR_ADVERBIALPHRASE3

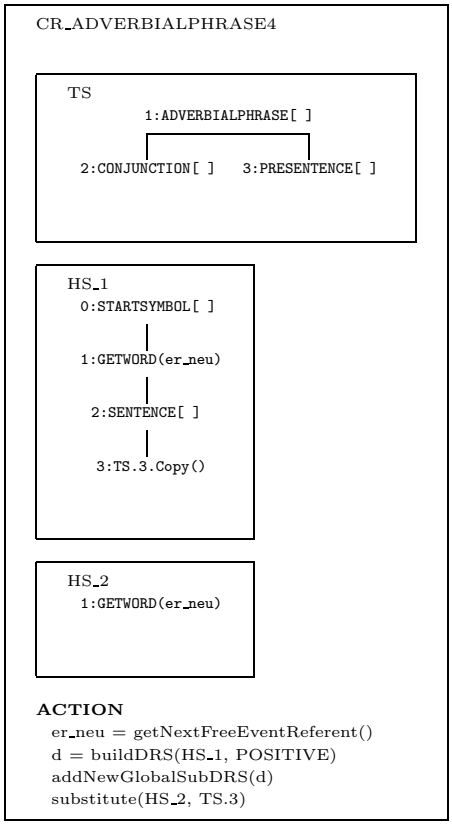
TS

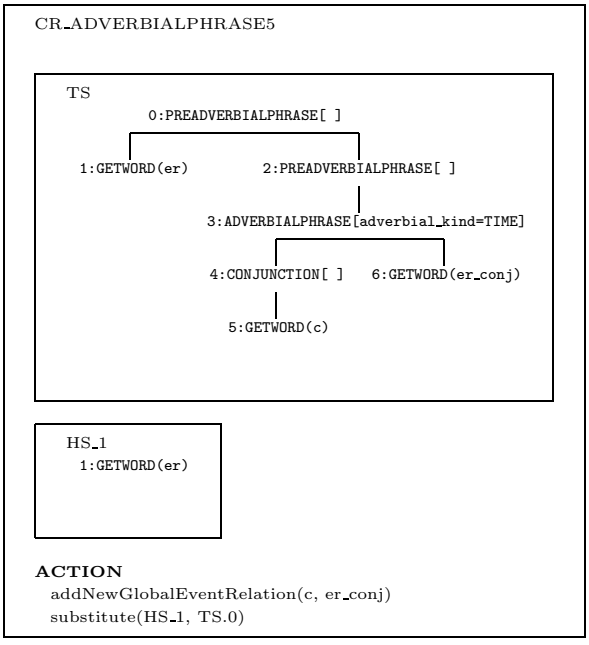


HS_1

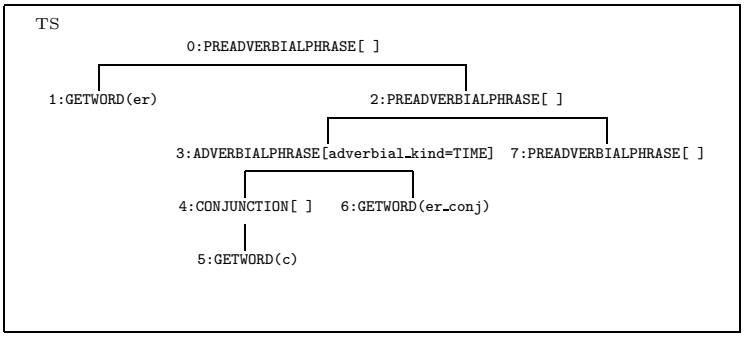


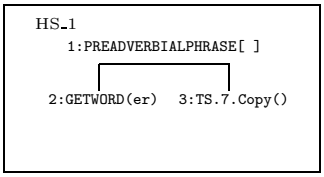
ACTION
substitute(HS_1, TS.5)
remove(TS.2)


CR_ADVERBIALPHRASE4
TS
1:ADVERBIALPHRASE[ ]
2:CONJUNCTION[ ]
3:PRESENTENCE[ ]
HS_1
0:STARTSYMBOL[ ]
1:GETWORD(er_neu)
2:SENTENCE[ ]
3:TS.3.Copy()
HS_2
1:GETWORD(er_neu)
ACTION
er_neu = getNextFreeEventReferent()
d = buildDRS(HS_1, POSITIVE)
addNewGlobalSubDRS(d)
substitute(HS_2, TS.3)


CR_ADVERBIALPHRASE5
TS
0:PREADVERBIALPHRASE[ ]
1:GETWORD(er)
2:PREADVERBIALPHRASE[ ]
3:ADVERBIALPHRASE[adverbial_kind=TIME]
4:CONJUNCTION[ ]
6:GETWORD(er_conj)
5:GETWORD(c)
HS_1
1:GETWORD(er)
ACTION
addNewGlobalEventRelation(c, er_conj)
substitute(HS_1, TS.0)

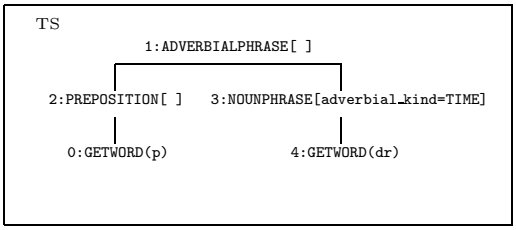
CR_ADVERBIALPHRASE6

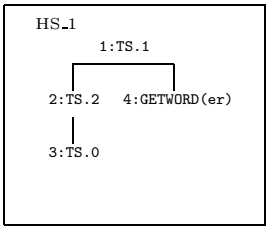




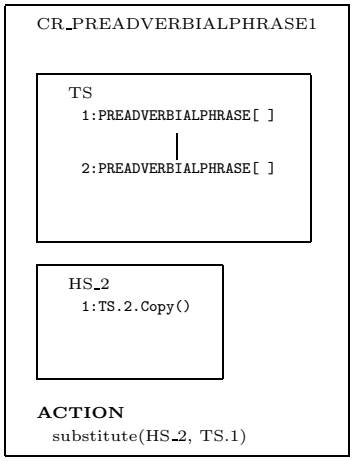
ACTION
addNewGlobalEventRelation(c, er_conj)
substitute(HS_1, TS.0)

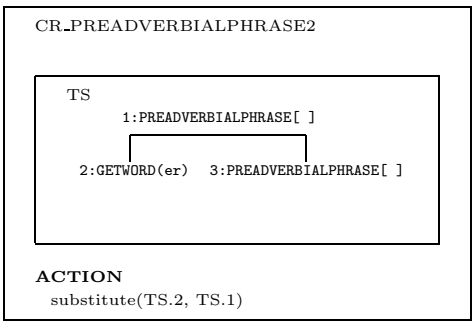
CR_ADVERBIALPHRASE7

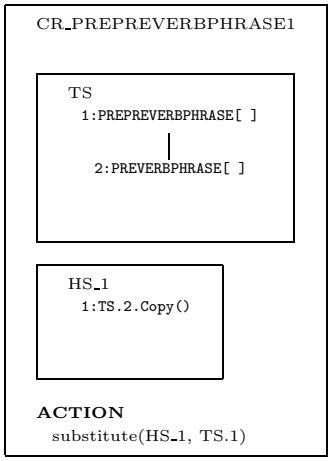


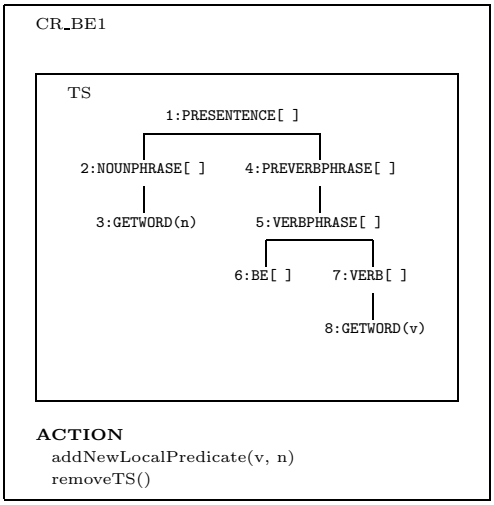


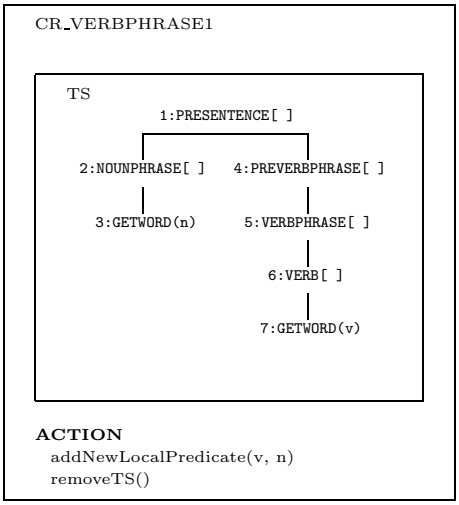
ACTION
er = addNewGlobalDuration(dr)
substitute(HS_1, TS.1)

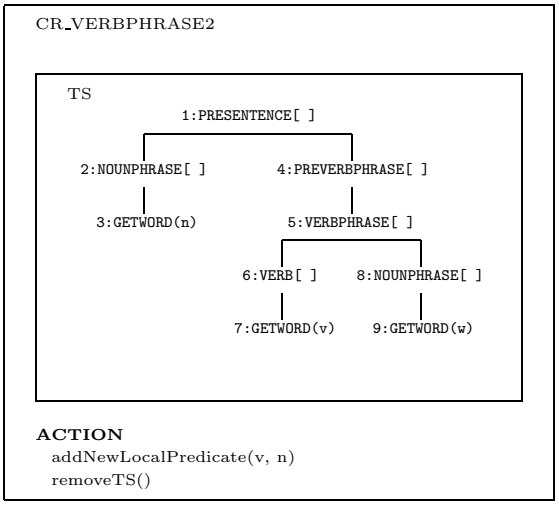

CR_PREADVERBIALPHRASE1
TS
1:PREADVERBIALPHRASE[ ]
2:PREADVERBIALPHRASE[ ]
HS_2
1:TS.2.Copy()
ACTION
substitute(HS_2, TS.1)

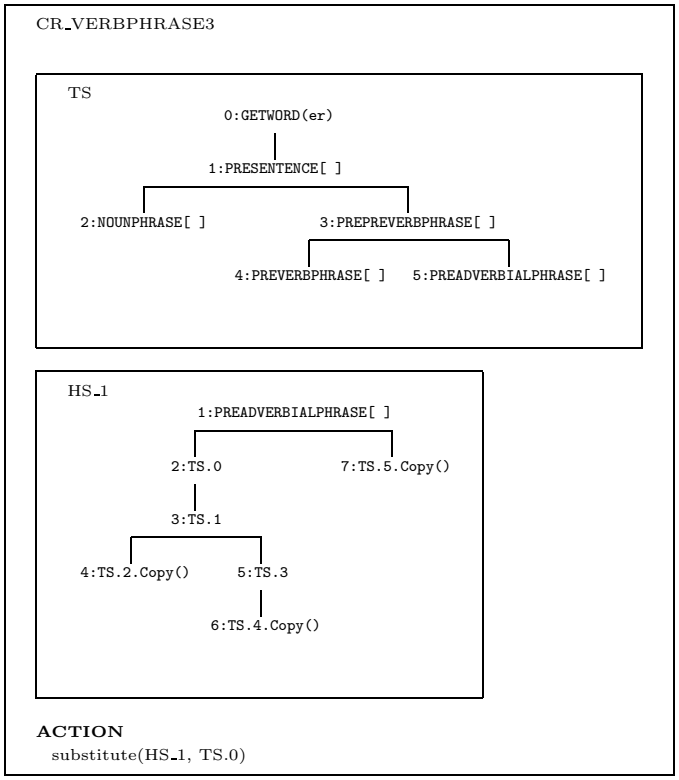

CR_PREADVERBIALPHRASE2
TS
1:PREADVERBIALPHRASE[ ]
2:GETWORD(er)
3:PREADVERBIALPHRASE[ ]
ACTION
substitute(TS.2, TS.1)

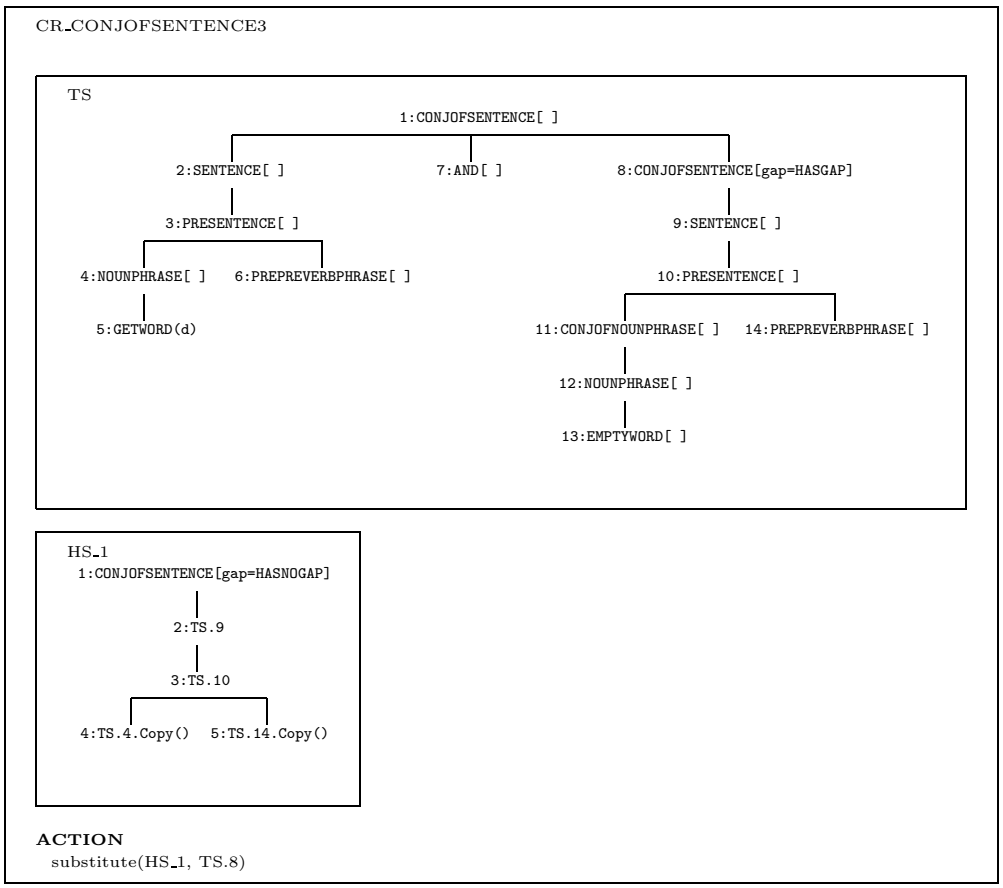

CR_PREPREVERBPHRASE1
TS
1:PREPREVERBPHRASE[ ]
2:PREVERBPHRASE[ ]
HS_1
1:TS.2.Copy()
ACTION
substitute(HS_1, TS.1)

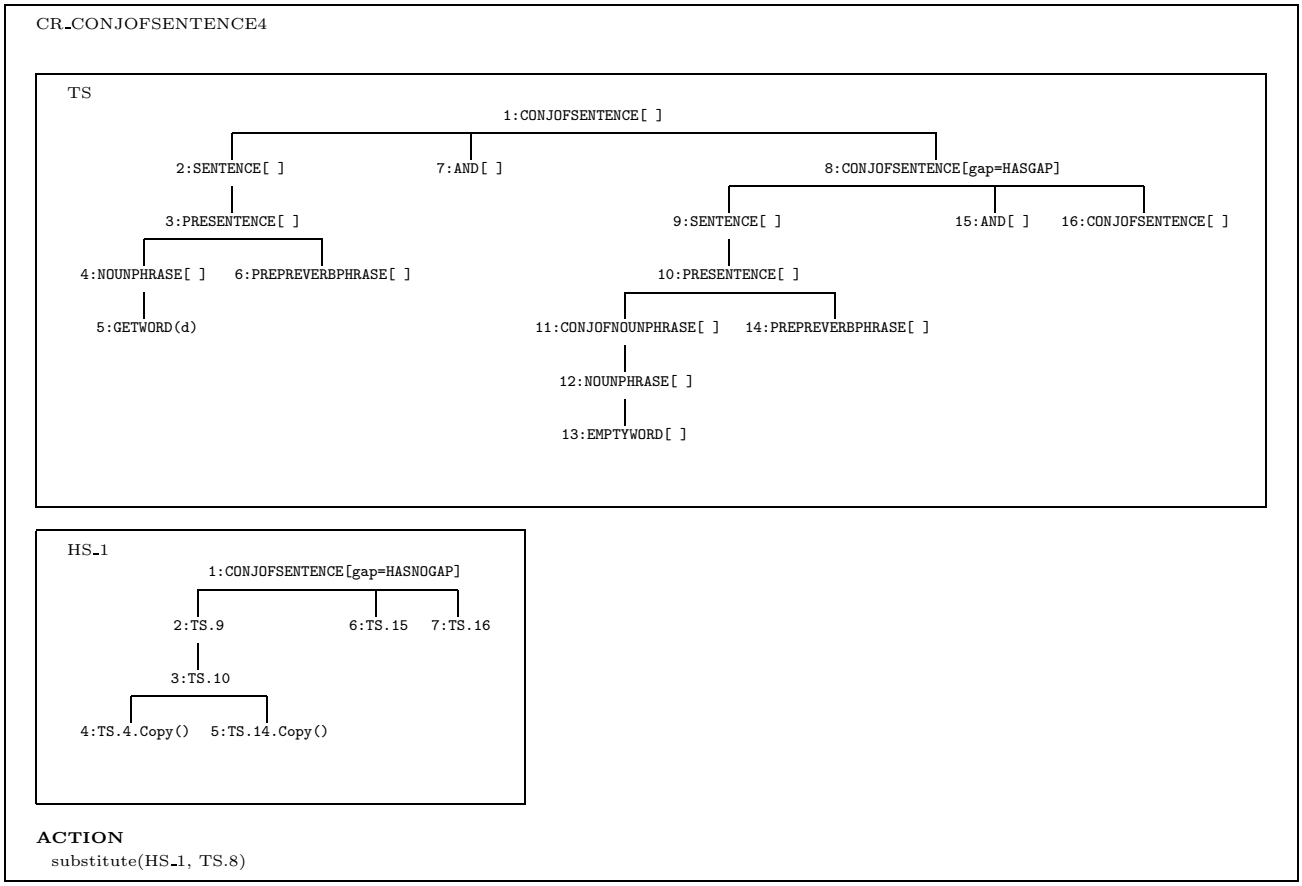

CR_BE1
TS
1:PRESENTENCE[ ]
2:NOUNPHRASE[ ]
4:PREVERBPHRASE[ ]
3:GETWORD(n)
5:VERBPHRASE[ ]
6:BE[ ]
7:VERB[ ]
8:GETWORD(v)
ACTION
addNewLocalPredicate(v, n)
removeTS()

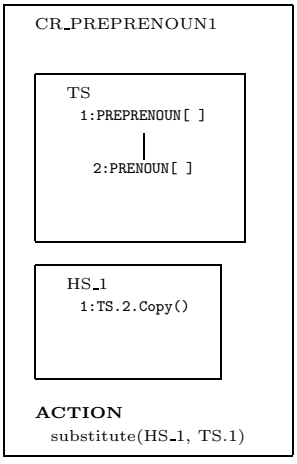

CR_VERBPHRASE1
TS
1:PRESENTENCE[ ]
2:NOUNPHRASE[ ]
4:PREVERBPHRASE[ ]
3:GETWORD(n)
5:VERBPHRASE[ ]
6:VERB[ ]
7:GETWORD(v)
ACTION
addNewLocalPredicate(v, n)
removeTS()

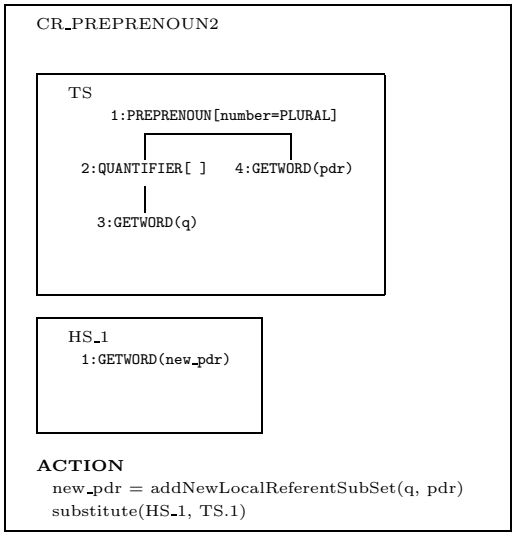

CR_VERBPHRASE2
TS
1:PRESENTENCE[ ]
2:NOUNPHRASE[ ]
4:PREVERBPHRASE[ ]
3:GETWORD(n)
5:VERBPHRASE[ ]
6:VERB[ ]
8:NOUNPHRASE[ ]
7:GETWORD(v)
9:GETWORD(w)
ACTION
addNewLocalPredicate(v, n)
removeTS()

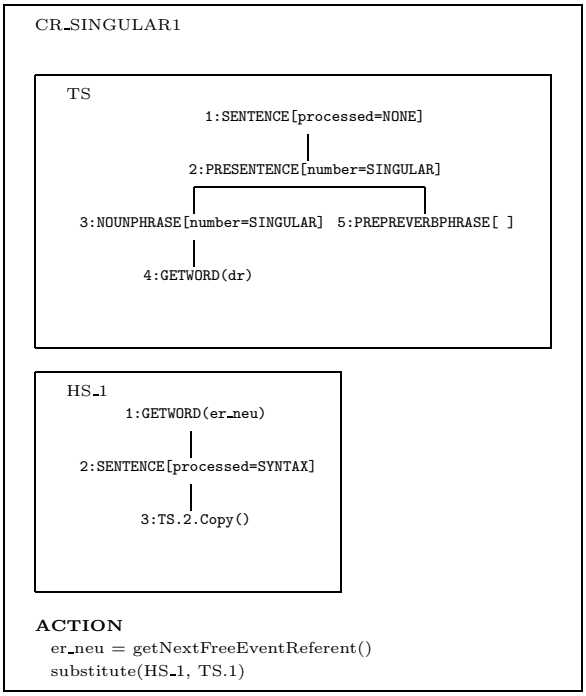

CR_VERBPHRASE3
TS
0:GETWORD(er)
1:PRESENTENCE[ ]
2:NOUNPHRASE[ ]
3:PREPREVERBPHRASE[ ]
4:PREVERBPHRASE[ ]
5:PREADVERBIALPHRASE[ ]
HS_1
1:PREADVERBIALPHRASE[ ]
2:TS.0
7:TS.5.Copy()
3:TS.1
4:TS.2.Copy()
5:TS.3
6:TS.4.Copy()
ACTION
substitute(HS_1, TS.0)

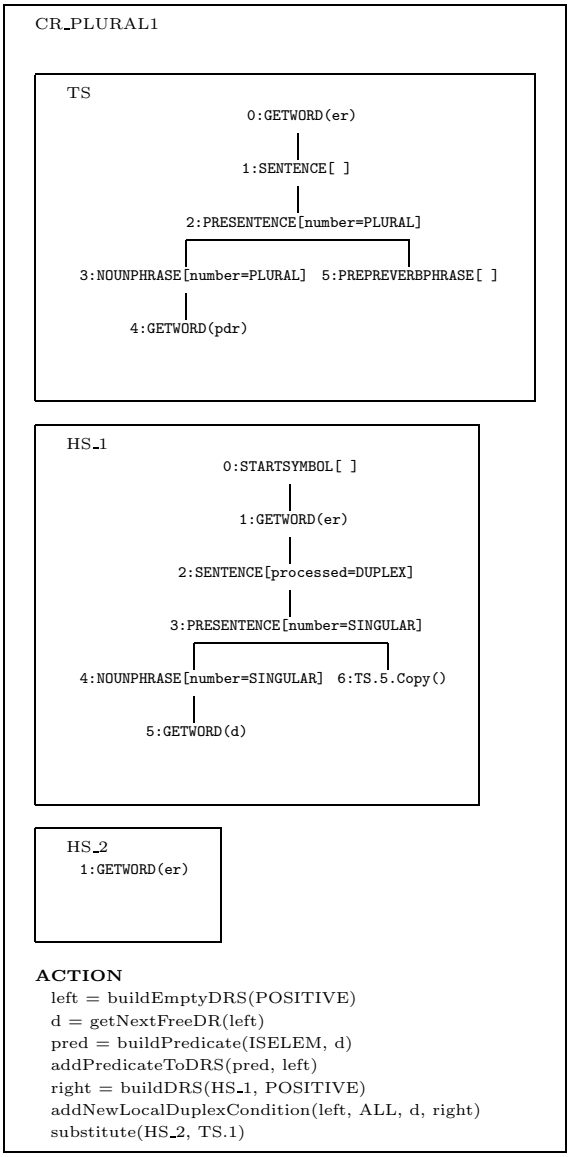

CR_CONJOFSENTENCE3
TS
1:CONJOFSENTENCE[ ]
2:SENTENCE[ ]
7:AND[ ]
8:CONJOFSENTENCE[gap=HASGAP]
3:PRESENTENCE[ ]
9:SENTENCE[ ]
4:NOUNPHRASE[ ]
6:PREPREVERBPHRASE[ ]
10:PRESENTENCE[ ]
5:GETWORD(d)
11:CONJOFNOUNPHRASE[ ]
14:PREPREVERBPHRASE[ ]
12:NOUNPHRASE[ ]
13:EMPTYWORD[ ]
HS_1
1:CONJOFSENTENCE[gap=HASNOGAP]
2:TS.9
3:TS.10
4:TS.4.Copy()
5:TS.14.Copy()
ACTION
substitute(HS_1, TS.8)

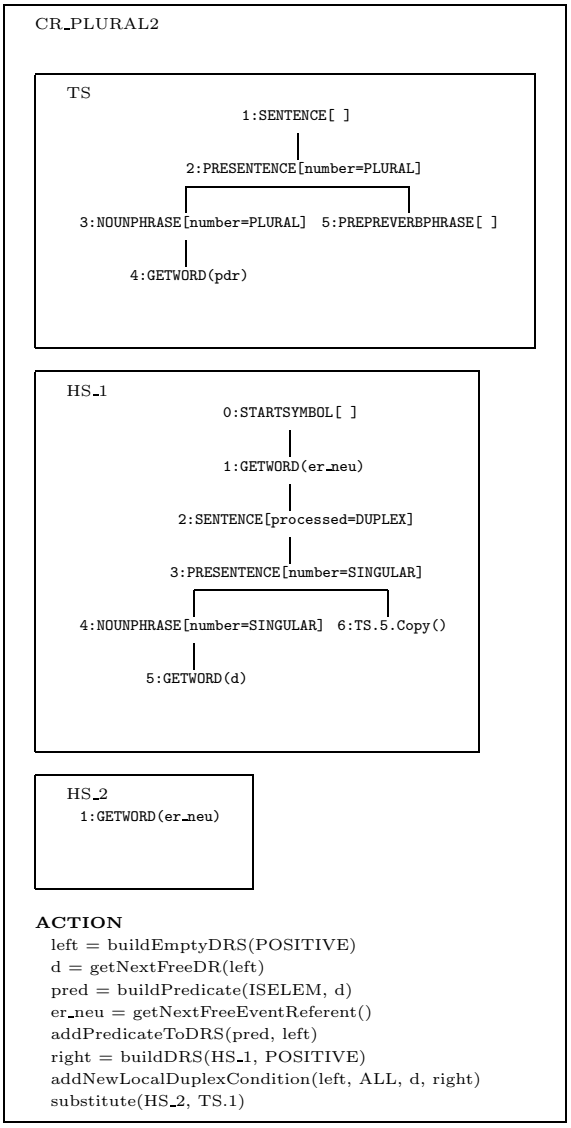

CR_CONJOFSENTENCE4
TS
1:CONJOFSENTENCE[ ]
2:SENTENCE[ ]
7:AND[ ]
8:CONJOFSENTENCE[gap=HASGAP]
3:PRESENTENCE[ ]
9:SENTENCE[ ]
15:AND[ ]
16:CONJOFSENTENCE[ ]
4:NOUNPHRASE[ ]
6:PREPREVERBPHRASE[ ]
10:PRESENTENCE[ ]
5:GETWORD(d)
11:CONJOFNOUNPHRASE[ ]
14:PREPREVERBPHRASE[ ]
12:NOUNPHRASE[ ]
13:EMPTYWORD[ ]
HS_1
1:CONJOFSENTENCE[gap=HASNOGAP]
2:TS.9
6:TS.15
7:TS.16
3:TS.10
4:TS.4.Copy()
5:TS.14.Copy()
ACTION
substitute(HS_1, TS.8)


CR_PREPRENOUN1
TS
1:PREPRENOUN[ ]
2:PRENOUN[ ]
HS_1
1:TS.2.Copy()
ACTION
substitute(HS_1, TS.1)


CR_PREPRENOUN2
TS
1:PREPRENOUN[number=PLURAL]
2:QUANTIFIER[ ]
4:GETWORD(pdr)
3:GETWORD(q)
HS_1
1:GETWORD(new_pdr)
ACTION
new_pdr = addNewLocalReferentSubSet(q, pdr)
substitute(HS_1, TS.1)

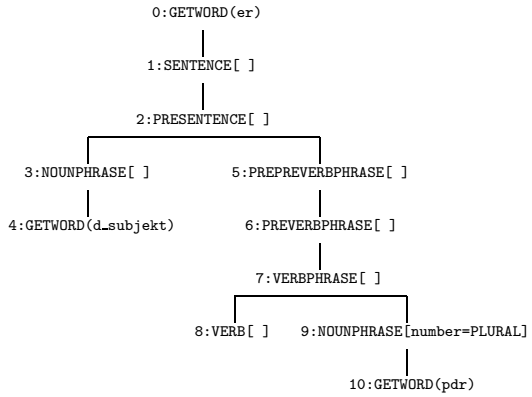

CR_SINGULAR1
TS
1:SENTENCE[processed=NONE]
2:PRESENTENCE[number=SINGULAR]
3:NOUNPHRASE[number=SINGULAR]
5:PREPREVERBPHRASE[ ]
4:GETWORD(dr)
HS_1
1:GETWORD(er_neu)
2:SENTENCE[processed=SYNTAX]
3:TS.2.Copy()
ACTION
er_neu = getNextFreeEventReferent()
substitute(HS_1, TS.1)


CR_PLURAL1
TS
0:GETWORD(er)
1:SENTENCE[ ]
2:PRESENTENCE[number=PLURAL]
3:NOUNPHRASE[number=PLURAL]
5:PREPREVERBPHRASE[ ]
4:GETWORD(pdr)
HS_1
0:STARTSYMBOL[ ]
1:GETWORD(er)
2:SENTENCE[processed=DUPLEX]
3:PRESENTENCE[number=SINGULAR]
4:NOUNPHRASE[number=SINGULAR]
6:TS.5.Copy()
5:GETWORD(d)
HS_2
1:GETWORD(er)
ACTION
left = buildEmptyDRS(POSITIVE)
d = getNextFreeDR(left)
pred = buildPredicate(ISELEM, d)
addPredicateToDRS(pred, left)
right = buildDRS(HS_1, POSITIVE)
addNewLocalDuplexCondition(left, ALL, d, right)
substitute(HS_2, TS.1)

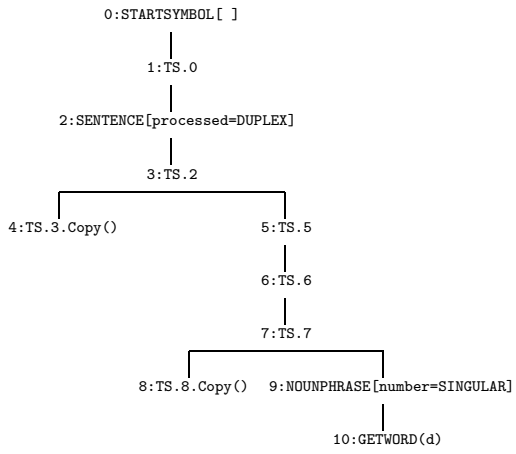

CR_PLURAL2
TS
1:SENTENCE[ ]
2:PRESENTENCE[number=PLURAL]
3:NOUNPHRASE[number=PLURAL]
5:PREPREVERBPHRASE[ ]
4:GETWORD(pdr)
HS_1
0:STARTSYMBOL[ ]
1:GETWORD(er_neu)
2:SENTENCE[processed=DUPLEX]
3:PRESENTENCE[number=SINGULAR]
4:NOUNPHRASE[number=SINGULAR]
6:TS.5.Copy()
5:GETWORD(d)
HS_2
1:GETWORD(er_neu)
ACTION
left = buildEmptyDRS(POSITIVE)
d = getNextFreeDR(left)
pred = buildPredicate(ISELEM, d)
er_neu = getNextFreeEventReferent()
addPredicateToDRS(pred, left)
right = buildDRS(HS_1, POSITIVE)
addNewLocalDuplexCondition(left, ALL, d, right)
substitute(HS_2, TS.1)

CR_PLURAL3

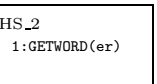
TS



HS_1

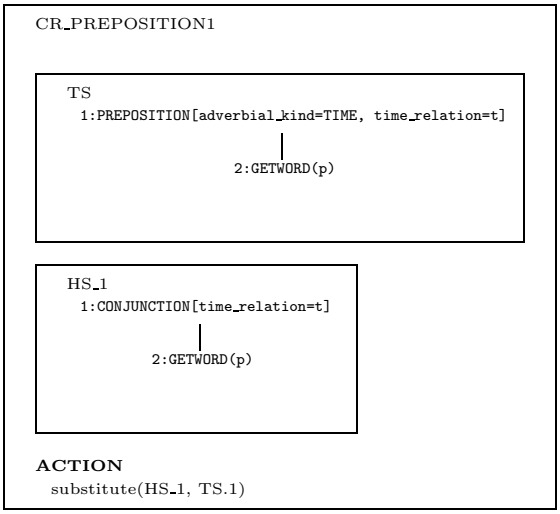


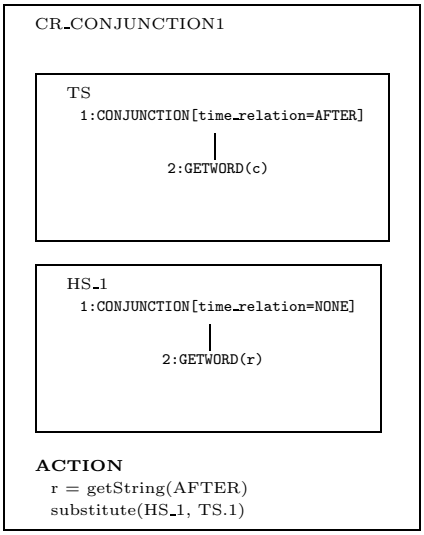
HS_2

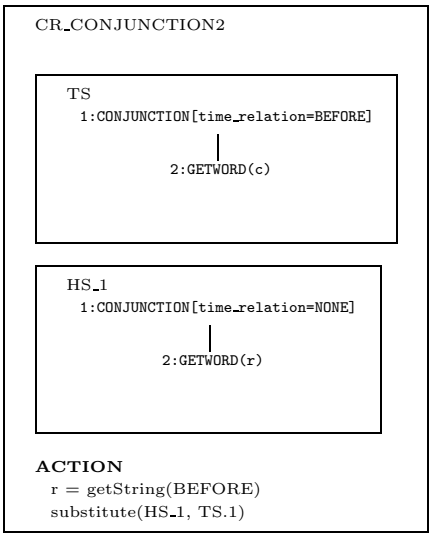


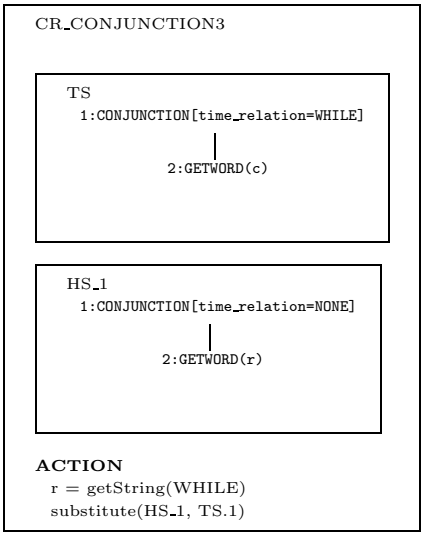
ACTION

left = buildEmptyDRS(POSITIVE)
d = getNextFreeDR(left)
pred = buildPredicate(ISELEM, d)
addPredicateToDRS(pred, left)
right = buildDRS(HS_1, POSITIVE)
addNewLocalDuplexCondition(left, ALL, d, right)
substitute(HS_2, TS.0)


CR_PREPOSITION1
TS
1:PREPOSITION[adverbial_kind=TIME, time_relation=t]
2:GETWORD(p)
HS_1
1:CONJUNCTION[time_relation=t]
2:GETWORD(p)
ACTION
substitute(HS_1, TS.1)


CR_CONJUNCTION1
TS
1:CONJUNCTION[time_relation=AFTER]
2:GETWORD(c)
HS_1
1:CONJUNCTION[time_relation=NONE]
2:GETWORD(r)
ACTION
r = getString(AFTER)
substitute(HS_1, TS.1)


CR_CONJUNCTION2
TS
1:CONJUNCTION[time_relation=BEFORE]
2:GETWORD(c)
HS_1
1:CONJUNCTION[time_relation=NONE]
2:GETWORD(r)
ACTION
r = getString(BEFORE)
substitute(HS_1, TS.1)


CR_CONJUNCTION3
TS
1:CONJUNCTION[time_relation=WHILE]
2:GETWORD(c)
HS_1
1:CONJUNCTION[time_relation=NONE]
2:GETWORD(r)
ACTION
r = getString(WHILE)
substitute(HS_1, TS.1)

A.1.8 Methoden für den Aktionsteil einer Konstruktionsregel

String addNewGlobalDR(SimpleNode defNode)
This method introduces a new *DR* to the global *DRS*. It takes a *SimpleNode*, extracts its list of *Description*s and takes this to define the new *DR*. The return value is the identifier of the new *DR*.

String addNewLocalDR(SimpleNode defNode)
This method introduces a new *DR* to the local *DRS*. It takes the *Description* list of the given *SimpleNode* and returns the identifier of the newly built *DR*.

String getNextFreeDR(DRS drs)
This method returns the identifier of the next free *DR*. This *DR* will be added to *Universe* of the given *DRS*. This method should be used, whenever a *DRS* is built to which a *DR* has to be added from the outside within one *ConstructionRule*. (e.g. in *ConstructionRule*s which build a *ReferentSet*.)

String addNewGlobalReferentSet(String defDR_ID, DRS defDRS)
This method introduces a new *ReferentSet* to the global *DRS*. It takes the given *DRS* as the set constraint of the new *ReferentSet* and returns its identifier as a String.

String addNewLocalReferentSet(String defDR_ID, DRS defDRS)
This method introduces a new *ReferentSet* to the local *DRS*. It takes the given *DRS* as the set constraint of the new *ReferentSet* and returns its identifier as a String.

String addNewGlobalReferentSubSet(String quant, String whole_id)
This method introduces a new *ReferentSubSet* `PDRx` of the form `PDRx = quant(whole_id)` to the global *DRS*. The identifier of the new *ReferentSubSet* is returned as a String.

String addNewLocalReferentSubSet(String quant, String whole_id)
This method introduces a new *ReferentSubSet* `PDRx` of the form `PDRx = quant(whole_id)` to the local *DRS*. The identifier of the new *ReferentSubSet* is returned as a String.

String addNewGlobalReferentSum(Vector refs)
This method introduces a new *ReferentSum* to the global *DRS*. The new *ReferentSum* will consist of the discourse referents given as the Vector `refs` of their identifiers. The identifier of the new *ReferentSum* is returned as a String.

String addNewLocalReferentSum(Vector refs)
This method introduces a new *ReferentSum* to the local *DRS*. The new *ReferentSum* will consist of the discourse referents given as the Vector *refs* of their identifiers. The identifier of the new *ReferentSum* is returned as a String.

void addNewGlobalAttribute(String attr, String var)
This method adds a new *Attribute* of the form `attr(var)` to the global *DRS*.

void addNewLocalAttribute(String attr, String var)
This method adds a new *Attribute* of the form `attr(var)` to the local *DRS*.

void addNewGlobalPredicate(String predicate, Vector argvec)
This method adds a new *Predicate* to the global *DRS*. The new *Predicate* is then of the form `predicate(argvec)`. `argvec` is supposed to be a Vector of Strings.

void addNewLocalPredicate(String predicate, Vector argvec)
This method adds a new *Predicate* to the local *DRS*. The new *Predicate* is then of the form `predicate(argvec)`. `argvec` is supposed to be a Vector of Strings.

void addNewGlobalEventRelation(String rel, Vector argvec)
This method adds a new *EventRelation* to the global *DRS*. The new *EventRelation* is then of the form `rel(argvec)`. `argvec` is supposed to be a Vector of Strings.

void addNewLocalEventRelation(String rel, Vector argvec)
This method adds a new *EventRelation* to the global *DRS*. The new *EventRelation* is then of the form `rel(argvec)`. `argvec` is supposed to be a Vector of Strings.

void addNewName(String _name, String DR_ID)
This method adds a new *Name* to the global *DRS*. It is not possible to add a *Name* to a local *DRS*, so there is only one add-method for *Name*s here.

void addNewGlobalDuplexCondition(DRS left, String quant, String sing_ref, DRS right)
This method adds a new *DuplexCondition* to the global *DRS*. It takes the left and the right side of the new *DuplexCondition* as *DRS*s and the *Quantifier* and the identifier of the quantified *DR* as Strings.

void addNewLocalDuplexCondition(DRS left, String quant, String sing_ref, DRS right)
This method adds a new *DuplexCondition* to the local *DRS*. It takes the left and the right side of the new *DuplexCondition* as *DRS*s and the *Quantifier* and the identifier of the quantified *DR* as Strings.

void addNewGlobalSubDRS(DRS subDRS)
This method adds a new Sub*DRS* to the global *DRS*.

void addNewLocalSubDRS(DRS subDRS)
This method adds a new Sub*DRS* to the local *DRS*.

void substitute(SimpleNode new_node, SimpleNode old_node)
This method substitutes the given *SimpleNode* `new_node` for the *SimpleNode* `old_node` in the actual syntax tree.

void addNewGlobalEquivalence(String ref_id1, String ref_id2)
This method takes the two referent identifiers `ref_id1` and `ref_id2` as Strings and adds a new equivalence consisting of these referents to the global *DRS*.

void addNewLocalEquivalence(String ref_id1, String ref_id2)
This method takes the two referent identifiers `ref_id1` and `ref_id2` as Strings and adds a new equivalence consisting of these referents to the local *DRS*.

String addLocalEventReferent(String name, DRS drs)
This method introduces an *EventReferent* with the specified name `name` to the local *DRS*. `drs` is used as the event-*DRS* for the new *ER*.

String addNewGlobalEventReferent(DRS drs)
This method adds a new *EventReferent* to the global *DRS*, taking the given *DRS* `drs` as the event definition. The return value is the identifier of the newly built *EventReferent*.

String addNewLocalEventReferent(DRS drs)
This method adds a new EventReferent to the local *DRS*, taking the given *DRS* `drs` as the event definition. The return value is the identifier of the newly built *EventReferent*.

String addNewGlobalEventReferentSet(Vector refs)
This method adds a new *EventReferentSet* to the global *DRS*, which will be the set of the *EventReferent*s given as the Vector `refs` of their identifiers. The return value is the identifier of the newly built *EventReferentSet*..

String addNewLocalEventReferentSet(Vector refs)
This method adds a new *EventReferentSet* to the local *DRS*, which will be the set of the *EventReferent*s given as the Vector `refs` of their identifiers. The return value is the identifier of the newly built *EventReferentSet*..

String addNewGlobalDuration(String ref)
This method adds a new *Duration* to the global *DRS*, which will be of the form `ERx = duration(ref)`. The newly built *Duration* is an *EventReferent*, so its identifier `ERx` is returned.

String addNewLocalDuration(String ref)
This method adds a new *Duration* to the local *DRS*, which will be of the form `ERx = duration(ref)`. The newly built *Duration* is an *EventReferent*, so its identifier `ERx` is returned.

String getAntecedent(SimpleNode defNode)
This method searches all accessible *DR*s for a *DR* which fits the *Description*s of the given *SimpleNode*. NOTE : *SimpleNode*s can have more than one *Description*. In addition there could be more than one suitable antecedent for even only one *Description*. See Class *DRS* and its method *getAntecedent* for further information.

void makeReferenceFor(SimpleNode defNode)
This method searches all accessible *DR*s for one which fits the *Description*s AND the Predciates of the *DR* given by `definingNode`. Here the given *DR* should be yet inserted to the *DRS*, and so its *Predicate*s should. Note that if no reference can be made, nothing at all is done and the method simply returns.

DRS buildDRS(SimpleNode new_tree, String sign)
This method builds a new *DRS* with the given syntax tree. The new *DRS* then has the sign `sign` which is only allowed to be "positive" or "negative". The return value is the newly built *DRS*.

DRS buildEmptyDRS(String sign)
This method builds a new *DRS* with no syntax tree. The new *DRS* then has the sign `sign` which is only allowed to be "positive" or "negative". The return value is the newly built *DRS*, which will be set to reduced.

void remove(SimpleNode rem_node)
This method removes the given *SimpleNode* `rem_node` of the current syntax tree.

void removeTS()
This method simply removes the complete Triggering-Structure from the actual syntax tree.

Predicate buildPredciate(String pred, Vector argvec)
This method builds a new *Predicate* with the given predicate name `pred` and the argument Vector `argvec`. The return value is the newly built *Predicate*.

Attribute buildAttribute(String attr, String arg)
This method builds a new *Attribute* with the given attribute name `attr` and the argument `arg`. The return value is the newly built *Attribute*.

Name buildName(String name, String arg)
This method builds a new *Name* with the given name `name` and the argument `arg`. The return value is the newly built *Name*.

EventRelation buildEventRelation(String rel, Vector argvec)
This method builds a new *EventRelation* with the given relation name `rel` and the argument Vector `argvec`. The return value is the newly built *EventRelation*.

void addPredicateToDRS(Predicate pred, DRS drs)
This method adds the given *Predicate* `pred` to the also given *DRS* `drs`. This method is used to build a *DRS* and then add a *Predicate* to that newly built *DRS* in one *ConstructionRule*.

void addAttributeToDRS(Attribute attr, DRS drs)
This method adds the given *Attribute* `attr` to the also given *DRS* `drs`. This method is used to build a *DRS* and then add a *Attribute* to that newly built *DRS* in one *ConstructionRule*.

void addNameToDRS(Name name, DRS drs)
This method adds the given *Name* `name` to the also given *DRS* `drs`. This method is used to build a *DRS* and then add a *Name* to that newly built *DRS* in one *ConstructionRule*. Note that *Name*s are supposed to be added only to the global *DRS*, so this method should not be used at all.

void addEventRelationToDRS(EventRelation e_rel, DRS drs)
This method adds the given *EventRelation* `e_rel` to the also given *DRS* `drs`. This method is used to build a *DRS* and then add a *EventRelation* to that newly built *DRS* in one *ConstructionRule*.

SimpleNode findNodeOfType(SimpleNode type, SimpleNode searchTree)
This method searches the syntax tree `searchTree` for a *SimpleNode* which has the same type as the given *SimpleNode* `type`. The last one found is returned here, according to normal pre-order.

String getString(String const)
This method simply returns a String generated from the given String constant `const`.

A.2 Übersicht über die Klassenstruktur

A.2.1 Klassen-Hierarchie

Dieser Abschnitt zeigt die Klassenhierarchie des erstellten Systems. “drt” bezeichnet hierbei ein neu erstelltes Paket und ist die Abkürzung für **D**iskurs-**R**epräsentations-**T**heorie.

class java.lang.Object

- class drt.ASCII_UCodeESC_CharStream
- class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
  - class java.awt.Canvas
    - class AttributeCanvas
    - class DRSCanvas
    - class TreeCanvas
  - class java.awt.Container
    - class java.awt.Window
      - class java.awt.Dialog
        - class AboutDialog
        - class QuitDialog
        - class ReduceDialog (implements drt.ReduceEventListener)
      - class java.awt.Frame (implements java.awt.MenuContainer)
        - class DRSViewer (implements drt.ReduceEventListener)
        - class HauptFrame
        - class LexikonWindow (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.awt.event.WindowListener)
        - class TreeViewer (implements java.awt.event.MouseListener, java.awt.event.ActionListener, drt.ReduceEventListener)
        - class WordWindow (implements java.awt.event.ItemListener, java.awt.event.TextListener, java.awt.event.ActionListener)
  - class java.awt.List (implements java.awt.ItemSelectable)
    - class SortedList
- class drt.ConstructionRule
  - class drt.CR_ADVERBIALPHRASE0
  - class drt.CR_ADVERBIALPHRASE1
  - class drt.CR_ADVERBIALPHRASE2
  - class drt.CR_ADVERBIALPHRASE3
  - class drt.CR_ADVERBIALPHRASE4
  - class drt.CR_ADVERBIALPHRASE5
  - class drt.CR_ADVERBIALPHRASE6
  - class drt.CR_ADVERBIALPHRASE7
  - class drt.CR_BE1
  - class drt.CR_CONJOFNOUNPHRASE1
  - class drt.CR_CONJOFNOUNPHRASE2
  - class drt.CR_CONJOFSENTENCE1
  - class drt.CR_CONJOFSENTENCE2
  - class drt.CR_CONJOFSENTENCE3
  - class drt.CR_CONJOFSENTENCE4
  - class drt.CR_CONJUNCTION1
  - class drt.CR_CONJUNCTION2
  - class drt.CR_CONJUNCTION3
  - class drt.CR_DETERMINER1
  - class drt.CR_NOUNPHRASE1
  - class drt.CR_NOUNPHRASE2
  - class drt.CR_NOUNPHRASE3
  - class drt.CR_NOUNPHRASE4
  - class drt.CR_NOUNPHRASE5
  - class drt.CR_NOUNPHRASE6
  - class drt.CR_NOUNPHRASE7
  - class drt.CR_NOUNPHRASE8

class drt.CR_NOUNPHRASE9
class drt.CR_PLURAL1
class drt.CR_PLURAL2
class drt.CR_PLURAL3
class drt.CR_PREADVERBIALPHRASE1
class drt.CR_PREADVERBIALPHRASE2
class drt.CR_PREPOSITION1
class drt.CR_PREPRENOUN1
class drt.CR_PREPRENOUN2
class drt.CR_PREPREVERBPHRASE1
class drt.CR_PRONOUN1
class drt.CR_RELATIVECLAUSE1
class drt.CR_SENTENCE1
class drt.CR_SINGULAR1
class drt.CR_START
class drt.CR_TEXT1
class drt.CR_TEXT2
class drt.CR_VERBPHRASE1
class drt.CR_VERBPHRASE2
class drt.CR_VERBPHRASE3

class drt.DR

class drt.DRS

class drt.Description (implements java.io.Serializable)

class drt.ADJECTIVE
class drt.ADVERB
class drt.AND
class drt.BE
class drt.CONJUNCTION
class drt.DETERMINER
class drt.DOT
class drt.NOUN
class drt.PREPOSITION
class drt.PRONOUN
class drt.PROPERNAME
class drt.QUANTIFIER
class drt.RELATIVEPRONOUN
class drt.VERB

class java.util.Dictionary

class java.util.Hashtable (implements java.lang.Cloneable, java.io.Serializable)
class drt.EventUniverse
class drt.Lexikon (implements java.io.Serializable)
class drt.PluralUniverse
class drt.PredicateLookUpTable
class drt.Universe

class DrawTree

class drt.DuplexCondition

class drt.ER

class drt.Duration
class drt.EventReferent
class drt.EventReferentSet

class java.util.EventObject (implements java.io.Serializable)

class drt.ReduceEvent

class drt.Implication

class java.io.InputStream

class TextStream

interface drt.Node

class java.io.OutputStream

class java.io.FilterOutputStream
class java.io.PrintStream
class WindowPrintStream

class drt.PDR

class drt.ReferentSet
class drt.ReferentSubSet
class drt.ReferentSum

class drt.Parser (implements drt.ParserTreeConstants, drt.ParserConstants)

interface drt.ParserConstants

class drt.ParserTokenManager (implements drt.ParserConstants)

interface drt.ParserTreeConstants

class drt.Predicate

class drt.Attribute
class drt.Name
class drt.EventRelation

class drt.Quantifier

interface drt.ReduceEventListener (extends java.util.EventListener)

class drt.SimpleNode (implements drt.Node)

class drt.TraceNode

class java.lang.Throwable (implements java.io.Serializable)

class java.lang.Error
class drt.TokenMgrError
class java.lang.Exception
class drt.GrammarException
class drt.ParseException
class drt.WordException

class drt.Token

class drt.Word (implements java.io.Serializable)

A.2.2 Class drt.Lexicon

Hier soll am Beispiel der Klasse Lexicon die automatische Konvertierung der HTML-Dokumentation des Systems nach TEX gezeigt werden. Das zu diesem Zweck verwendete Werkzeug wies allerdings solch große Mängel auf, daß zur Dokumentation aller übriger Klassen auf automatisch generierte HTML-Seiten verwiesen wird.

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                   |
                   +----drt.Lexicon
```

public class Lexicon

extends Hashtable

implements Serializable

This class represents the Lexicon used by the language parser. Each entry in the Lexicon is of the type Word, which holds the string - and any description that can be connected to it - of a language-word.

Lexicon() This is the constructor of the class Lexicon.

getDescriptions

public synchronized Vector getDescriptions(String _origin) This method returns a Vector of known Descriptions of the given language-word and an empty Vector if the word is unknown

getComplexDescriptions

public synchronized Vector getComplexDescriptions(String _origin) This method returns a Vector of known Descriptions of the given language-word and its word-forms, an empty Vector if the word is unknown

getWord

public synchronized Word getWord(String _origin)

getComplexWord

public synchronized Word getComplexWord(String _origin) This method returns an Object of class Word for the given language-word _origin. It first builds the stem of the word and then tries to derive all forms from this stem. These forms and their Descriptions are then combined to the Word to be returned.

insertWord

public synchronized void insertWord(Word word) This method is used to insert new entries into the Lexicon. New Entries must be of type Word.

Anhang B

Beispieltext

Der folgende Text wird von dem zu der in Anhang A.1 angegebenen Grammatik erzeugten Zerteiler akzeptiert. Zu diesem Text wird auch ein Syntaxbaum erzeugt, der aufgrund seiner Größe hier allerdings nicht aufgeführt werden kann.

A car entered the scene from KarlWilhelmStrasse at regular speed. It stopped on the left lane at the intersection. While it was standing another car crossed the intersection on DurlacherStrasse. The car which was driving changed from the left lane to the right lane. Another car approached the intersection on KarlWilhelmStrasse on the right lane. After a while the two cars which were standing accelerated and crossed the intersection. Then the car which was driving on the left lane overtook the other car and changed to the right lane. Then a car and a truck drove on DurlacherStrasse. The car followed the truck. Approaching the intersection the truck braked. Then it stopped at the intersection. The car waited for two minutes. After two minutes the truck accelerated and crossed the intersection. The car turned into KarlWilhelmStrasse. As it left the scene it drove at regular speed. Two trucks came from Durlacherstrasse. The two trucks drove on the left lane. The first truck stopped at the intersection. The other truck turned into KarlWilhelmStrasse. After the truck turned it followed a car and left the scene on KarlWilhelmStrasse on the left lane. At the same time a car stopped behind the truck which was waiting at the intersection. As the truck accelerated the car followed the truck and left the scene. In the meantime another car approached on KarlWilhelmStrasse. It was driving behind a bus. After a while it overtook the bus and stopped at the intersection on the left lane. The bus stopped beside the car. While the bus and the car were standing a truck and another car stopped behind the bus and two cars stopped behind the car which stood at the intersection in the first row. Then the bus accelerated before the car accelerated. The other cars followed the bus and the car and left the scene.

Anhang C

Zeitplan

C.1 Geplanter Verlauf

Zeitraum	Arbeitsschritt
26. 02. — 03. 03. (1 Woche)	Einarbeitung in die Problemstellung, Literaturstudium
04. 03. — 17. 03. (2 Wochen)	Planung u. Implementierung eines Grammatik-Zerteilers für die in [Kamp & Reyle 93] verwendeten Grammatik-Regeln. Erstellung eines Sprach-Zerteilers auf Basis des Grammatik-Zerteilers. Implementierung und Test der in [Kamp & Reyle 93] aufgeführten Grammatik-Regeln.
18. 03. — 31. 03. (2 Wochen)	Erstellung eines Zerteilers für die in [Kamp & Reyle 93] genannten Konstruktions-Regeln (CR-Zerteiler). Implementierung und Test der in [Kamp & Reyle 93] aufgeführten Konstruktions-Regeln.
01. 04. — 21. 04. (3 Wochen)	Einarbeitung in Syntax und Sprachumfang von UMTL [Schäfer 96]. Einarbeitung in den Umgang mit F-Limette. Implementierung der nötigen Methoden für die Umwandlung einer DRS in UMTL-Formeln. Test dieser Methoden.
22. 04. — 05. 05. (2 Wochen)	Erarbeitung von Methoden zur Generierung des von dem durch [Jeyakumar 98] erstellten SIS-Werkzeug benötigten Eingabeformats aus UMTL-Formeln. Implementierung der Methoden zur Umwandlung von UMTL-Formeln in das SIS-Eingabeformat.
06. 05. — 19. 05. (2 Wochen)	Test des gesamten Systems mit der von XTrack erzeugten natürlichsprachlichen Beschreibung einiger Bildfolgen. Verbesserung der Benutzer-Oberfläche des Systems. Ggf. Erstellung einer Applet-Version.
20. 05. — 25. 05. (1 Woche)	Abschluß der Arbeit. Ggf. Vortrags-Vorbereitung.

C.2 Tatsächlicher Verlauf

Zeitraum	Arbeitsschritt	Dokumentation
26. 02. — 03. 03. (1 Woche)	Einarbeitung in die Problemstellung, Literaturstudium	Kapitel 1: Einleitung
04. 03. — 17. 03. (2 Woche)	Planung u. Implementierung eines Grammatik-Zerteilers für die in [Kamp & Reyle 93] verwendeten Grammatik-Regeln. Erstellung eines Sprach-Zerteilers auf Basis des Grammatik-Zerteilers. Erstellung von Werzeugen zur grafischen Anzeige von Syntax-Bäumen.	Kapitel 2: Zerteilererzeugung, Anhang A: Programm-Dokumentation
18. 03. — 31. 03. (2 Wochen)	Überarbeitung des Grammatik-Zerteilers zum leichteren Auffinden von Fehlern in der Grammatikdatei. Erarbeitung einer vereinfachten Grammatik anhand von Beispieltexten zur Beschreibung von Verkehrsgeschehen.	Überarbeitung von Kapitel 2, Erweiterung von Anhang A
01. 04. — 21. 04. (3 Wochen)	Planung und Implementierung eines Zerteilers für Konstruktionsregeln. Erstellung einfacher Konstruktionsregeln.	Kapitel 3: Diskurs-Repräsentationstheorie, Anhang A.1.8: bereitgestellte Aktionsmethoden, Anhang B: Beispieltext
22. 04. — 05. 05. (2 Wochen)	Erstellung weiterer Konstruktionsregeln. Erstellung von Werkzeugen zur grafischen Anzeige von DRSn und schrittweisen Verfolgung der Konstruktionsregelanwendung. Einarbeitung in die Behandlung von Pluraldiskursbezugsträgern.	Erweiterung des Anhangs A um Konstruktionsregel-verzeichnis.
06. 05. — 19. 05. (2 Wochen)	Erstellung weiterer Konstruktionsregeln. Erstellung von Werkzeugen zur grafischen Anzeige des Zerteilungsprozesses.	Erweiterung des Kapitels 2 um Morphologie, Änderung des Kapitels 3.1 bei den Erklärungen zu Bezugsträgern und Bedingungen. Kapitel 4.
19. 05. — 25. 05. (1 Woche)	Abschließende Arbeiten an den erstellten Werkzeugen. Abschluß der Arbeit.	Letzte Änderungen.

Literaturverzeichnis

[Badler et al.93] N. Badler, C. Phillips, and B. Webber: *Simulating Humans: Computer Graphics, Animation and Control.* Oxford University Press, Oxford/UK, 1993.

[Gerber 95] R. Gerber: *Diskurs-Repräsentationstheorie zur Situationsanalyse in Straßenverkehrsszenen.* Diplomarbeit, Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik der Universität Karlsruhe (TH), Juli 1995.

[Gerber 2000] R. Gerber: *Natürlichsprachliche Beschreibung von Straßenverkehrsszenen durch Bildfolgenauswertung.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, 2000.

[Haag 98] M. Haag: *Bildfolgenauswertung zur Erkennung der Absichten von Straßenverkehrsteilnehmern.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juli 1998. Erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **193**, infix-Verlag Sankt Augustin 1998.

[Jeyakumar 98] V. Jeyakumar: *Generation of Synthetic Image Sequences for Car Maneouvers Extracted by Image Sequence Evaluation.* Master Thesis, Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik der Universität Karlsruhe (TH), Dezember 1998.

[Kamp & Reyle 93] H. Kamp and U. Reyle: *From Discourse to Logic.* Kluwer Academic Publishers, Dordrecht, The Netherlands 1993.

[Koller 92] D. Koller: *Verfolgung und Klassifikation bewegter Objekte in monokularen Bildfolgen am Beispiel von Straßenverkehrsszenen.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juni 1992. Erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **13**, infix-Verlag Sankt Augustin 1992.

[Kollnig 95] H. Kollnig: *Ermittlung von Verkehrsgeschehen durch Bildfolgenauswertung.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Februar 1995. Erschienen

in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **88**, infix-Verlag Sankt Augustin 1995.

[Krovetz 99] R. Krovetz : *Viewing morphology as an inference process.* Artificial Intelligence 118 (2000) 277-294.

[MacKevitt 95] P. MacKevitt (ed.): *Integration of Natural Language and Vision Processing.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.

[Nagel et al. 99] H.-H. Nagel, M. Haag, V. Jeyakumar and A. Mukerjee: *Visualisation of Conceptual Descriptions Derived from Image Sequences.* Mustererkennung 1999, 21. DAGM-Symposium, Bonn, 15.-17. September 1999, Springer-Verlag, Berlin, Heidelberg u.a., pp. 364-371.

[Schäfer 96] K. H. Schäfer: *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in Bildfolgenauswertung und Robotik.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Juli 1996; DISKI **135**, infix-Verlag Sankt Augustin 1996.

[Schöning 95] U. Schöning: *Theoretische Informatik - kurzgefaßt.* 2. überarbeitete Auflage, Spektrum Akademischer Verlag, Heidelberg·Berlin·Oxford 1995.

[Tijerino et al. 95] Y. A. Tijerino, S. Abe, T. Miyasato and F. Kishino : *What You Say Is What You See – Interactive Generation, Manipulation and Modification of 3-D Shapes Based On Verbal Descriptions* in [MacKevitt 95]

[Zeltzer 82] D. Zeltzer : *Motor Control Techniques for Figure Animation.* IEEE Computer Graphics Applications, Vol. 2, No. 9 (Nov. 1982), pp. 53-59