Inkrementelle linguistische Interpretation inkrementell erzeugter Wortgraphen

Der Technischen Fakultät der
Universität Bielefeld

zur Erlangung des Grades

Doktor-Ingenieur

vorgelegt von

Bernd Seestaedt

Bielefeld 1995

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Die zutreffende Einordnung meiner Arbeit in die Weiterentwicklung der automatischen Sprachverarbeitung soll durch den Titel angeleitet werden. Für meinen Beitrag zur Tagung AAAI'94 verwandte ich noch den Titel "Left-to-Right-Analysis of spoken Language". Leider hat das zu einem Mißverständnis Anlaß gegeben. Dem will ich durch den zweifachen Hinweis auf *Inkrementalität*, man gestatte das Kunstwort, begegnen. Als experimentelle Basis fungierten eine akustische Erkennungs- und eine linguistische Interpretationseinheit. Beide wurden in die Lage versetzt, ein inkrementelles, d.h. schritthaltendes, Verarbeitungsregime einzuhalten. Beide wurden somit offen für Integration und Rückkopplung. In meiner Abhandlung betrachte ich die Voraussetzungen der so entstehenden Innovationsmöglichkeiten aus der Sicht der Interpretationseinheit. Zu ihr habe ich die programmierte Lösung erarbeitet.

Der alte Titel weckte Erwartungen im Sinne einer Modell-Lösung der Computerlinguistik für die Analyse von Sprachsignalen – der LR-Parser [Tom86] in direkter Einbindung in eine Erkennungseinheit. Beschränkt man das Blickfeld auf die Aufgabe "Qualifizierung der Signalerkennung", so hat die hier vorgelegte Kopplung viel Paralleles anzubieten. Sie verfolgt jedoch ein weiterreichendes Endziel mit einem anderen Analyseverfahren.

Im Bereich der Systeme, die sprach-akustische Eingaben interpretieren, setzen der GLR*-Parser von Tomita/Lavie [LA93] sowie der Konzept-Parser PHOENIX von W. Ward [War91a] Vergleichsmaßstäbe. Beide Systeme wurden an der Carnegie Mellon Universität (Pittsburgh, USA) aufgebaut. Durch meine Teilnahme an der AAAI'94 konnte ich ihren aktuellen Entwicklungsstand abschätzen, wozu auch die fortgesetzten persönlichen Kontakte beitrugen. Dem von mir vorgestellten System könnte man das Signet TLR* anheften:
es ist eine **T**iefenstruktur-Beschreibung von Sprachsignalen herzustellen und es sind die Prinzipien der **L**inks-Rechts-Verarbeitung und der **R**echts-Assoziation anzuwenden. Der * soll einzuführende Flexibilisierungen dieser Prinzipien annoncieren, die zur Behandlung spontan-sprachlicher Eingabe notwendig sind.

Durch die Bezeichnung werden konzeptionelle Parallelen zu den aufgeführten Systemen behauptet. In vorsichtiger Annäherung könnte man TLR* als den Versuch ansehen, Vorzüge beider Systeme zu vereinigen. Wie PHOENIX stützt sich TLR* auf ein Konzeptsystem als Grammatik im erweiterten Sinne, eine Art "logische Grammatik" [Wit77] eines eingeschränkten Diskursbereiches und auch Diskursverhaltens. Ein Beispiel sind Zugauskunftsdialoge, in denen gewisse Vereinfachungen im Sinne des Kommunikationserfolges zweckmäßig sind. Weiter verzichten PHOENIX und TLR* auf Satzgrammatik. L.Mayfield, M.Gavalda, W.Ward und A.Waibel haben den Stellenwert des PHOENIX-Parsers im 'Speech-to-Speech'-Translation-Projekt JANUS in dem Artikel "Concept Based Speech Translation" (Dezember 1994, im Druck) so beschrieben:

'PHOENIX parses input into slots in semantic frames, and then uses these frames to generate output in the target language.
Starting from the assumption that semantic units used in a task domain are, unlike individual words, not language specific, we have developed a set of tokens, *representing the different concepts a speaker would use, as the fundamental units of the parser* (1).

A typical temporal token could have as a subtoken a date, which could in turn consist of month and day subtokens. ...
Unlike traditional speech translation systems, ours performs *no syntactic analysis* (2). Speaker utterances, as decoded by the recognizer, are parsed into semantic chunks which can be strung together *without grammatical rules* (3). This approach is particularly well suited to parsing *spontaneous speech, which is often ungrammatical and subject to recognition errors* (4). We feel this approach is more *robust* (5) than systems which require well-formed input and rely upon *markers and syntactic cues provided by short function words such as articles and prepositions* (6).'

Auf kleinstem Raum sind hier alle Themen versammelt, auf die das vorgelegte System anspricht, was nicht heißt, daß alle Behauptungen geteilt werden. Mit Blick auf die Themen (4) und (5), der 'Robustheit' als Gegenzug zum Phänomen Ungrammatikalität spontaner Äußerungen, ist die automatische Sprachverarbeitung teilweise recht weit vom traditionellen Pfad der Computer-Linguistik und Natural-Language-Philosophie abgewichen. So verfolgte ein Workshop der AAAI'94 zum Thema 'Integration of Natural Language and Speech Processing' geradenwegs das Ziel, zwischen beiden Disziplinen wieder Brücken zu bauen. Manches ist Mißverständnis der Terminologien. Deshalb verwende ich im Kapitel I meiner Arbeit nicht wenig Mühe darauf, mit Hilfe eines teils formalisierten, teils (gegenüber seiner grammatikalischen Version) generalisierten Regel-Begriffs das Scharnier zwischen regel- und netzwerk-basierten grammatikalischen Ableitungssystemen aufzuzeigen. Man kann das als bloßes Vorspiel auffassen. Ich versuche zu zeigen, daß das ERNEST-System allgemein-methodologische Bedeutung besitzt und eine Art NETLOG-Projekt darstellt – eine Fortsetzung von PROLOG Basisideen mit anderen Mitteln (vor allem der Analysekontrolle). Gegenüber einer Implementation in PROLOG wird der Grad ingenieurmäßiger Verwertbarkeit gesteigert, was aber auch erhöhte Anforderungen an die aufgaben-spezifische Ausgestaltung des von ERNEST gelieferten Gerüsts stellt. Der Untersuchung dieser Anforderungen widme ich erhöhte Aufmerksamkeit.

Die Argumentation zu den Themen (2), (3) und (6) deutet auf Schwierigkeiten der Selbstdarstellung von PHOENIX, die ich für meine Anwendung vermeiden möchte. Um durchaus beachtenswerte Besonderheiten herauszustellen, werden Differenzen rhetorisch zu Gegensätzen gesteigert. Faktisch verzichtet das System absichtlich auf die Kontrolle linguistischer Attribute wie Genus, Kasus u.a, was die Behauptung (2) motiviert. Grundelement der Wissensbasis sind Wortketten, die aus Lexemen, sie umfassenden Sammel-Konzepten, die zum Teil der verwendeten Regionalsprache doch recht nahe stehen (siehe Bild 2.41 gemäß Abbildungsverzeichnis) und Subnetzen solcher Ketten bestehen. Sie werden gegen den Suchraum verglichen. Nicht *Ableitung* sondern Matching ist also das Hauptprocedere. Wenn das System allein aufgrund einer derartigen Konzeptualisierung Ansprüche geltend macht, einen Beitrag zu Thema (1) zu leisten (einschließlich der Relevanz für maschinelle Übersetzungssysteme) – dann gilt das erst recht für das homogene semantische Konzeptnetz in meiner Anwendung.

Die der Kettenbildung impliziten Stellungsregeln von PHOENIX – im landläufigen Sinne ist das eine Art von Syntax – lassen auch die Argumentation (2) bedenklich erscheinen. Um Robustheit zu erreichen, werden gewisse Freiheiten beim Matching eingeräumt. Nur Subketten müssen kongruieren. Das kann man wiederum als "Aufweichung" der Stellungsregeln auffassen. Die der Computerlinguistik traditionelle

inkrementelle Denkweise muß systembedingt aufgegeben werden. Gewohnte Maßstäbe scheinen auf ein dezidiert der Applikationssituation zugewandtes System nicht mehr anwendbar.

Die Netzkonstruktion stellt im Gegensatz zur Argumentation (3) im Sinne von Woods [Woo70a] einen speziellen Typ von Grammatik dar. W.Ward selbst identifiziert die reine ATN-Grammatik einerseits und den einfachen Keyword-Spotter andererseits als die Extremalpositionen, zwischen denen seine Konzeption in der “Mitte” anzusiedeln sei. Ein Ableitungssystem auf der Basis von ERNEST dürfte kaum in die Verlegenheit kommen, mit Keyword-Spottern auf eine Stufe gestellt zu werden. Trotzdem wurde meinem Vorgängersystem [Kum92a] solch Mißverständnis noch vor 2 Jahren in einer Projektausarbeitung der vom BMFT geförderten Vorstufe des VERBMOBIL-Projektes zuteil.

Deshalb scheint es mir angebracht, die Selbstdarstellung nochmals zu thematisieren und es nicht bei einem Vorwort über Grundbegriffe des ERNEST-Formalismus zu belassen. Ich versuche, den Formalismus in eine veränderte Perspektive zu rücken: Form und Kontrolle der Wissensbasis als semantisches Netzwerk stehen in keinem Gegensatz zu den von Parsern verwendeten linguistischen Regelbasen. Durch den Netzüberbau der Regelbasis wird der Ablauf der Regelauswahl informiert und effektiviert. Wie beim Parsen kristallisiert sich der regelgestützte Ableitungsprozeß in einer *Stukturbeschreibung* [Eik89]. Zunächst beschränkt sie sich absichtsvoll auf die Bildung syntaktisch korrekter Phrasen aus den Subketten des vom inkrementellen Erkenner produzierten Sub-Wortgraphen. Es handelt sich um einen Graphen, der über dem Sprachsignal instantiierten, sprich regel-abgeleiteten, Prädikate (Konzepte in semantischen Netzen, Nonterminals in Grammatiken). Die wesentliche Neuerung ist nun, daß der Ableitungsprozeß mit unvollständiger Bindung der in Regeln aufgeführten Prädikatsvariablen auskommen kann. Das macht diesen Parsertyp, der bei der Phrasenbildung nicht stehenbleibt, sondern auch deren pragmatische Interpretanten findet, für die inkrementelle Verarbeitung von Wortgraphen so geeignet. Mehr als meine Vorgänger in der ERNEST-Anwendung, untersuche ich die Kontrollfunktion der Strukturbeschreibung. Der terminus technicus *Konzeptgraph* verweist wie der von J.Sowa vorgestellte ’conceptual graph’ [Sow84] auf die Herstellung einer Zwischencodierung nach Maßgabe einer *Interlingua* [Lea94] für Übersetzungssysteme.

Um die Strukturbeschreibung zum wirksamen Medium der problem-abhängigen und inkrementellen Kontrolle auszugestalten, mußte ich eine Reihe von Änderungen am allgemeinen Schema der ERNEST-Kontrolle vornehmen. Dabei hatte ich spätere Anwendungen in ganz anderen Problemkreisen vor Augen.

Wie GLR* arbeitet der Tiefenstrukturparser inkrementell, von-Links-nach-Rechts. Der Stern (*) signalisiert die Flexibilisierung des traditionellen GLR-Parsers, indem er befähigt wird, unsichere Signalbereiche zu überspringen (SKIP), ohne die erreichte Strukturbeschreibung der Äußerung aufzugeben. Das mit TLR* implementierte SKIP-Verfahren geht sogar noch weiter als die GLR*-Strukturbeschreibung. Der von Lavie/Tomita [LA93] erreichte Stand des Wiederanknüpfens an inaktivierte Knoten des Parser-Suchbaums wurde schärfer gefaßt. TLR* reaktiviert im Falle nicht-parsbarer Subketten genau den Suchbaumknoten (gegebenenfalls rekursiv), der als letzter datengetrieben zur Einfügung der inkrementell nächsten Worthypothese erzeugt wurde. Das Neue an TLR* besteht darin, die in der linguistischen Wissensbasis, insbesondere

im Lexikon vorgezeichneten Ambiguitäten als Trigger zur SKIP–Behandlung anzuwenden. Worthypothesen, sprich Lexemen, ist z.B. eine mehrdeutige Zuordnung von Wortkategorien vorgegeben. In der aktuellen Anwendung fallen darunter Präpositionen, Determinans und eine Reihe Adjektive, die in 2 Phasen rückwirkend integriert werden. Damit wird jedoch nicht der Argumentation (6) gefolgt. Präpositionen z.B. stellen in der betrachteten Domäne unverzichtbare, pragmatisch-wichtige Schlüssel des Satzverstehens dar. Nur muß man schon temporär etwas über die Tiefenstruktur wissen, d.h. zunächst die im pragmatischen Sinne verstandenen Konstituenten genau des Handlungsschemas ermitteln, vor dessen Hintergrund die Äußerung – insofern als eine Art von Handlung – erfolgte.

Dem Leser gebe ich als Hinweis mit auf den Weg, daß alle 3 Kapitel inhaltliche Fragen behandeln. Das 3. Kapitel befaßt sich relativ detailliert mit Fragen der Implementation in Form generalisierter Programm-Codierungen. Das geschieht in der Hoffnung auf Wiederverwendbarkeit in weiteren inkrementell arbeitetenden Anwendungen. Der rote Faden mag dank mancher Nebenfäden schwer festzuhalten sein. Ein Nebenfaden behandelt das in den Grenzen dieser Abhandlung wenigstens im Ansatz zu diskutierende Problem der für Suchprozesse adäquaten *virtuellen Maschinen* ([Mar80]). Ich diskutiere solche Fragen aber nicht, wie in der theoretischen Informatik üblich, unter Aspekten wie Berechenbarkeit und Komplexität, also der Sicht z.B. von Hopcroft/Ullman in [Hop88], sondern unter den Aspekten von Beherrschbarkeit und Überschaubarkeit des Verarbeitungsregimes für einen differenzierten Anwenderkreis. Diese Tendenz sehe ich eher in dem weitverbreiteten Buch über Compiler-Technologie von Aho, Rethi und Ullman, [Aho86] verwirklicht. Die hintergründige Verbindungslinie beider Sichtweisen ist mir dabei wohlbewußt.

Es sollte normal sein, daß eine Arbeit über maschinelle Sprachverarbeitung einzelne neue Lichter auf die Beziehungen von Computer-Maschine und Sprache wirft. Meine erstrangige Aufgabe war es jedoch, eine Anwendung aufzubauen, die nicht nur die Realisierbarkeit des TLR*-Ansatzes bestätigt, sondern z.B. in Signalform codierte spontansprachliche Äußerungen analysieren kann. Dazu wurden in einer Messehalle aufgenommene Äußerungen gewählt. Beantwortung, gegebenenfalls gezielte Nachfrage zu den gesprochenen Anfragen, die reellen Intercity-Angebote der Deutschen Bundesbahn betreffend, sind das Ziel der Anwendung. Da auf sie häufig Bezug zu nehmen ist, gebe ich ihr “für die Lesestunde meines Textes” den Bezugsnamen ICZ = **I**nter**C**ity-**Z**ugauskunft.

Kapitel 1 ERNEST als Grundlage der Wissensbasis und Kontrolle

1.1 Regelform der Wissensbasis

1.1.1 Prädikation linguistischer 'Gegenstände'

Im Einleitungsteil zu Kapitel 1 wurde eine Akzentverschiebung in der Sicht auf den Netzformalismus ERNEST angekündigt. Es ist üblich, Formalismen für Regelbasen solchen für semantische Netze in einer Weise gegenüberzustellen, die mehr vom Gegensatz als von den Gemeinsamkeiten beider ausgeht. Es soll nun der Versuch gemacht werden, mit ERNEST aufgebaute Wissensbasen als eine spezielle Art von *Regelbasen* aufzufassen. Der Versuch kann als gelungen betrachtet werden, falls es gelingt, ERNEST-Deklarationen mit einem Regelformalismus adäquat auszudrücken. Dazu benötigt man einen weithin bekannten Vergleichs-Formalismus und eine Anpassung des Regelbegriffs. Es wird ein formeller Gebrauch des Terminus 'Regel' angestrebt. Ein formeller, weil formalisiert anzuwendender Begriff braucht keineswegs formal, sprich 'inhaltslos', zu sein. Eine bestimmte Fassung des Regel-Begriffs ermöglicht es, die besondere Art der Ableitungsvorgänge, die man mit semantischen Netzen durchführen kann, zu präzisieren. Absichtlich formell soll der Begriff sein, um mehr kognitive Fragestellungen herauszuhalten. Fragen derart, ob z.B. die Inhalte von Wissen und Fakten generell unterscheidbar sind und Wissen stets in Regelform notierbar ist, werden ganz ausgespart.

Als Vergleichsformalismus wird PROLOG hergenommen. Zur Analyse der deklarativen Potenzen des Formalismus sei eine für die Domäne linguistischer Verstehenssysteme passende, jedoch möglichst einfache und übersichtliche Anwendung derselbigen gewählt. In einem Übersichtsbuch zur Computerlinguistik notiert Grisham mittels PROLOG die in Bild 1.1 gezeigte (von mir vereinfachte) Modell-Grammatik [Gri86]. Sie ist als Regelbasis aufgebaut und enthält z.B. die Regel (2)

```
np(X) :- nomen(X).
```

Mögliche Lesarten der Regel sind:

```
für alle 'Gegenstände' X gilt:
X ist np,    wenn X ein nomen ist;
X ist nomen, also auch np.
```

Man beachte, daß Variablen vom PROLOG-Compiler an den großgeschriebenen Anfangsbuchstaben erkannt werden, Prädikate und Konstanten dagegen am Kleinbuchstaben. Das Prädikat np() steht für Nominalphrase, nomen() für die linguistische Kategorie Nomen.

Clocksin/Mellish fassen Regeln als komprimierte Darstellung des Tatbestandes, daß "Fakten von Fakten abhängen" [Clo90]. Formeller gedacht, werden mittels Regeln Abhängigkeiten zwischen Prädikationen (Prädikat-Zusprechungen) bestimmt. PROLOG-Fakten werden als Prädikationen zu Konstanten notiert. Die zunächst zu betrachtenden Konstanten sind genau die Identifikatoren der Objekte, welche als Basiselemente der Wissensdomäne auftreten. In einer linguistischen Wissensbasis sind das die Wörter eines Lexikons.

Als Basis weiterer Bestimmungen wird gewählt:

Allgemeine Form für Prädikationsregeln

Ein Prädikat p heiße bezüglich einer Variablenfolge (X_i), (i=1,..,m), **abhängig** von einer Folge von Prädikaten p_j, (j=1,..,n),
formal notiert durch:
$p(Arg) :- p_1(Arg_1), ..., p_n(Arg_n)$, mit Argumentfolgen Arg,Arg_j, (i=1,..,n), falls gilt:

(1) (X_i) ist eine Teilfolge von Arg,

(2) die p_j gewährleisten die *Bindung* der Variablen[1] X_i ; formell: indem jede dieser Variablen mindestens in einer Argumentfolge $Arg_j(p_j)$ vorkommt.

Die X_i heißen die *Gegenstandsvariablen* von p. Das zur Linken stehende Prädikat heiße auch das *abhängige* Prädikat, die rechts stehenden heißen die *unabhängigen* Prädikate der Regel. Mit Blick auf den Modellgehalt semantischer Netze heißen die p_i die *Elementarprädikate* oder auch *Konzepte* von p.

Die Regelform wird z.B. befriedigt durch

```
(1) a(X,Y,Z) :- b(X),c(Y,Z).
(2) a(X,Y)   :- b(X,V),C(Y,V).
(3) a(X,Y,W) :- b(X),c(Y), W=wert.
(4) a(X,Y)   :- b(X),c(Y), bewertung(X,Y) > 0.
```

Regel (1) zeigt an, daß nicht alle Arg_j stellengleich sein müssen. Für semantische Netze gilt sogar die Disjunktheit der Argumentmengen $\{Arg_j\}$.

Die Regeln (2),(3) verwenden außer den Gegenstandsvariablen noch Hilfsvariable V und W, die zusätzlich zu binden sind – übereinstimmend in (2), wertgenau in (3).

Regel (4) wird durch die Verwendung von Hilfsprädikaten neben den Konzepten gekennzeichnet. Das können Bewertungsfunktionen sein, die dem Gegenstand der Prädikation Zahlenwerte zuordnen.

Gegenstandsbildung läuft in der Domäne der linguistischen Datenverarbeitung auf die Bildung von Wortketten aus Einzelwörtern und Substrings hinaus.

Dem hier bezogenen Standpunkt der **Prädikation** liegt die Vorstellung einer hierarchischen Ordnung der Prädikate zugrunde. Man konzentriert sich auf den Übergang von hierarchisch tieferstehenden zu höheren (abstrakteren) Prädikaten. Prädikationen setzen allerdings zu prädizierende Gegenstände voraus.

’Prädikate’ oder ’Konzepte’ gehen nicht ’an sich’ Beziehungen ein, sondern nur als Zusprechungen über Gegenstandsbildungen einer Domäne.

Vom Standpunkt der **Ableitung** konzentriert man sich auf den Aspekt der Bindung der Gegenstandsvariablen durch die konsistente Belegung aus den Werten der unabhängigen Prädikate bei vorausgesetzt konsistenter Belegung der Attributvariablen.

---

[1] Variablen werden als defaultmäßig nicht-initialisierte (sprich ungebundene) Datentypen gedacht

Formell läßt sich das Fungieren der Variablen X als **Gegenstandsvariable** versus **Attributvariable** so festmachen:

1. Prädikat präd: präd ist ableitbar für X
2. Attribut(-funktion) attr: X —> {werte} .

Prädikation geht von Gegenständen aus und fragt nach den zusprechbaren Prädikaten. Ableitung ist stets eingeschlossen, indem das gewählte Prädikat einer Konsistenzprüfung zu unterziehen ist.

Die von Grisham im Anschluß an Perreira und Warren [Per80] formulierte Regelbasis ist vom Typ der 'Definite Clause Grammar' (DCG). Sie besitzt ein hierarchisch höchstes Prädikat s(), mit dem alle Sätze über einem Mini-Lexikon geprüft werden können. Sie heiße darum ein *Satzakzeptor*[2].

Die Regelbasis verwendet Wortketten zur Bindung der Variablen. Die zugehörige Backus-Nauer-Formel von s() lautet:

<S> ::= <NP> <VP>.

Mit Blick auf Variablentupel (Testkette, Restkette) ergibt sich die Codierung:

s(X,Y) :- np(X,Z), vp(Z,Y).,

– d.h., gegeben seien eine feste Wortkette für X (zu prüfender Satz) und eine Variable Y zur Aufnahme der Restkette. Teste X zuerst mit der np-Regel und binde die Zwischenvariable Z mit der Restkette des Tests. Teste danach Z mit der vp-Regel und sieh nach, ob Y als Restkette entsteht.

```
(1)  s(X,Y) :- np(X,Z), vp(Z,Y).
(2) np(X,Y) :- nomen(X,Y).
(3) np(X,Y) :- det(X,Z), nomen(Z,Y).
(4) vp(X,Y) :- verb(X,Z), np(Z,Y).

det([the|X],X).
det([a|X],X).
nomen([goose|X],X).
nomen([geese|X],X).
nomen([fox|X],X).
nomen([foxes|X],X).
verb([eats|X],X).
verb([eat|X],X).
```

Bild 1.1 PROLOG-Wissensbasis für den Satzakzeptor s()[3]

Die untenstehende Anfrage 1 fordert mittels der leeren Kette [] die Prüfung der Akzeptanz – Linguisten bezeichnen es als "Wohlgeformtheit" – der angegebenen Wortkette an. Leerheit der Kette steht für vollständige Reduzierbarkeit, sprich Ersetzung, der Fakten durch Prädikate.

---

[2] in der Computerlinguistik nennt man dies einen Recognizer und meint damit die Vorstufe eines Parsers

[3] die Zählklammern vor den Regeln gehören nicht zur eigentlichen PROLOG-Notation

```
1 ?- s([foxes,eat,goose],[]).
     Yes

% Syntaxtest:
2 ?- s([goose,eat,a,foxes],[]).
No

% Semantiktest:
3 ?- s([foxes,eat,a,goose],[]).
No

% Pragmatiktest:
4 ?- s([geese,eat,foxes],[]).
Yes
5 ?- s([foxes,eat,geese],[]).
Yes
```

Bild 1.2 Anfragen zur Wissenbasis

1.1.2 Ziele in Regelbasen

Eine Anfrage der im obenstehenden Bild 1.2 formulierten Art heißt ein **abzuleitendes Ziel** der Regelbasis. Ziel-Anfragen stellen das PROLOG-typische Mittel der Wissensnutzung dar. In PROLOG liefern die Prädikate der Regelbasis das Schema einer Zielformel, dessen Variablen dann teilweise durch Konstanten fixiert oder ungebunden angegeben werden. Die untenstehende Anfrage 6 kann z.B. dazu benutzt werden, die zur Bildung von Verbalphrasen vp() zulässigen Verben des Lexikons abzufragen.

```
6 ?- vp([Y,foxes],X).
     X = []
     Y = eats  ->user-eingabe: ;
     Y = eat                   ;
 No
```

Der Nutzer einer solchen Wissensbasis (in diesem Kontext der PROLOG-*User*) betätigt sich als Zielformulierer, indem er passenden Gebrauch von Variablen und Konstanten macht. Durch mehrfache Verwendung des Semikolons[4] in Anfrage 6 hat der User die Möglichkeit, die Liste der passenden Verben zu duchlaufen, falls er sich dafür interessiert. Der PROLOG-Formalismus bietet ihm einfache Interaktionen an, um Wissen abzufragen.

Es wird eine Mindest-Kenntnis des Users über die Reihenfolge der Regelabarbeitung, sprich die *prozedurale Semantik*, vorausgesetzt. Gemeint sind die 2 Metaregeln:

Die Auswahl der Regeln verläuft horizontal von Oben-nach-Unten,

die Auswahl der unabhängigen Prädikate von-Links-nach-Rechts.

[4] 'No' hat er als Ende der Liste der zulässigen Verben zu deuten

Bild 1.2 verweist auf verschiedene linguistische Testvarianten, die mit dem gezeigten Anfragetyp realisiert werden können. In Frage 2 wird eine im Englischen syntaktisch unzulässige Wortgruppe ’a foxes’ gewählt, um das zu fordernde Resultat der Nicht-Akzeptanz (Systemantwort ’No’) abzuprüfen.

In Frage 3 wird ein grammatisch korrekter Satz gewählt, der ein semantisches Problem aufwirft. Es bedarf theoretischer Mittel, um die in ihm enthaltene Unsicherheit des Verstehens des Satzes einzukreisen. Eine Möglichkeit liefert die Theorie der *semantischen Rollen* von Satzbestandteilen von Fillmore [Fil71]. Wir spüren beim Hören des Satzes, daß sich die Worte, die für die Rollenträger einer Realhandlung stehen sollen, nicht so miteinander kombinieren lassen, wie es im Satz geschieht. Dabei hat man gewisse Einschränkungen der Diskurswelt vor Augen, die der Satz referiert – etwa Ausschluß des Satzes “foxes eat a goose” als Aussage über Realhandlungen. Stattdessen will man nur verallgemeinernde Sätze — etwa solche, denen nicht unterstellt werden muß, deiktisch auf sinnlich-beobachtbare Situationen zu verweisen[5]. Das trifft im Unterschied zu dem Satz von Test 3 sicher auf den Satz “foxes eat geese” zu, sofern er im Diskursfeld der Zoologie eine Dominanz-Relation aussagen soll.

Der vorliegende Satzakzeptor ist blind für solche Unterscheidungen. Er kann jedoch auf verschiedenen Wegen diesbezüglich ’sensibilisiert’ werden:

1. indem globale Attributvariablen[6] eingeführt und durchgängige Propagierungen von Werten (z.B. syntaktische Merkmale wie Numerus) durchgeführt werden[7],
2. durch Erweiterung des Prädikationssystems nach-oben, indem Regeln zur Bildung von Rollenträgern für semantische Rollen formuliert und lokale Attribute eingesetzt werden.

Um den Fall zu skizzieren, werden der AGENT ([foxes]) und der EXPERIENCER[8] ([geese]) als die relevanten Rollen/Rollenträger eines durch das Verb ’to eat’ vorher-bestimmten Rollenensembles für den betrachteten Satz hergenommen.

Linguistische ’Gegenstände’ (Verben und Nomen) definieren Ensembles semantischer Rollen.

Verzichtet man einmal auf die Listenschreibweise, so könnten geeignete Regeln mittels des lokalen Attributs numerus, wie in Bild 1.3 dargestellt, definiert werden. Die Erweiterung des Akzeptors nach-oben wirft allerdings Fragen der Änderung des Grammatik-Modells auf, die erst im 2. Kapitel, das sich mit den linguistischen Phänomenen der speziellen Domäne ’Intercity-Zugauskunft’ (ICZ) befaßt, behandelt werden.

Mit Anfragen des Typs 4,5 wird die pragmatische Zulässigkeit des Satzes, d.h. eine weitere Einschränkung mit Bezug auf die gewählte Domäne, getestet. Man spricht von *pragmatischen (Rollen-)* Bestimmungen von Satzteilen und meint damit gewisse Anforderungen an die Rollenträger.

---

5 die Einordnung der Quantor-ambigen Übersetzung “jeder frißt seine (eigene) ...” lasse ich beiseite.

6 Variablen, die sich durch alle Regeln hindurchziehen

7 dieser Weg wird im Abschnitt ’Attribute’ des ersten Kapitels erläutert

8 die Rollennamen verweisen auf eine Rollenverteilung nach dem Schema ’aktiv/passiv’

```
experiencer(X):-np(X),numerus(X)=plural.
agent(X)       :-np(X).
-------------------------
numerus(geese)=plural.
numerus(goose)=singular.
...
```

Bild 1.3 Regeln für semantische Rollen

Wie das Ergebnis der Anfragen 4,5 zeigt, ist der Akzeptor blind für die pragmatischen Bestimmungen der Relation fressen().

Diese Einschätzung setzt allerdings die Bereitschaft voraus, die linguistische Rollentheorie als tragfähige Basis der Satzanalyse anzuerkennen. Beim Übergang zu pragmatischen Bestimmungen rekurriert man unmittelbar auf die relevanten Handlungszusammenhänge der Domäne. Es werden schematisierbare Handlungsmuster betrachtet.

Die geeigneten Träger pragmatischer Rollen findet man häufig anhand von lexikalisch festgeschriebenen Klassenzuordnungen in Bezug auf eine domänen-relevante Interpretation des den Handlungszusammenhang bestimmenden Verbs.

Die zum Verb 'to eat' korrespondierende Relation fressen(X,Y) wird *asymmetrisch* aufgefaßt, d.h. für den Fall der schon erwiesenen Geltung von fressen(x,y) ist die Geltung von fressen(y,x) auszuschließen. Die pragmatische Klasse geeigneter Rollenträger A,B für die Relation fressen() kann zur Regeldefinition als lokales Attribut eingesetzt werden:

```
executor(X)
     :- agent(X), pragmatik_klasse(X)= dominant.
-----------------------------------------------
  pragmatik_klasse(goose)= nicht_dominant.
  pragmatik_klasse(fox)= dominant.
  ...
```

Bild 1.4 Regeln für pragmatische Rollen

1.1.3 Sätze mit Handlungsbedarf

Allgemein reicht die Analyse der pragmatischen Bestimmungen eines Satzes von der gesicherten Zuordnung desselben zur Domäne bis zum Erkennen der dem Satz zugrunde liegenden *Handlungs-Intention* des Satzbildners. Überhöhte kognitive Ansprüche werden vermieden, indem man sich auf die Wiedererkennung eines von mehreren möglichen Handlungsschemata beschränkt:

```
hierarchisch-höchstes Prädikat
-> Ziel der Satzanalyse
-----------------------------
abstraktes Handlungsschema der Äusserung
```

Man betrachte die folgenden Sätze, von denen der erstere als Beispiel einer fehlerhaften Einzelwort-Zuweisung ('dort' statt 'Dortmund') durch den Worterkenner gewählt wurde.

```
Ich möchte nach dort
---------------> ?
 .     .      .    Dortmund
```

Beide Sätze sind syntaktisch korrekt und semantisch sinnvoll. Trotzdem wäre die Akzeptanz der oberen Äußerung als Fehlleistung zu werten, wenn z.B. ein Zugauskunftssystem zu realisieren ist, da dieser Satz keinen pragmatischen Zweck in der Domäne erfüllen kann. Formell erkennt man dies am Verlust des Trägers einer entscheidend wichtigen pragmatischen Rolle des betreffenden Handlungsbereichs, dem Ankunftsort. Man kann es dann als Problem der Systemkonzeption fassen, ob die Wortkette [nach,dort] in einer Mehrsatz-Äußerung elliptisch auf eine semantische Rolle GOAL im vorher gesprochenen Satz 'zeigen' kann.

Unser System soll Äußerungen beantworten können, die ein abstraktes Handlungsschema ausdrücken, z.B.

```
Schema(X,Y,A,B,<C>):
fahren ( [Reisender,
          (am X),(um Y),
          (von A <,über C>, nach B)
         ])).
```

Bild 1.5 Handlungsschema der Äußerung

Das Schema deutet den Versuch an, eine Notation[9] für eine ganze Klasse pragmatisch äquivalenter Sätze zu finden. Man sagt im Anschluß an Wittgenstein [Wit77], Sätze hätten trotz ihrer unterschiedlichen Oberflächengestalt eine identische *Tiefenstruktur*. Palme hat in [Fin71] schon vor mehr als 20 Jahren die Parallele gezogen:

'Das Tiefenstrukturkonzept (wie es von Linguisten gebraucht wird) und das semantische Netzwerk-Konzept (wie es von Autoren der Frage-Antwort-Systeme herangezogen wird) sind in der Tat sehr ähnliche Konzepte: bei beiden besteht das Ziel im Auffinden einer logischen Struktur, die die zugrundeliegenden Basisideen im Satz andeuten' [Pal75] (S.226).

Die in ERNEST formierte Struktur der Wissensbasis des ICZ ist ein Beispiel für ein Prädikationssystem, das ganz auf die Analyse pragmatisch-zentrierter Tiefenstruktur ausgerichtet ist. Es wird vorausgesetzt, daß Äußerungen ihr zugrundeliegendes Handlungsschema "preisgeben". Analyse bedeutet Zerlegen und Weglassen dessen, was nicht zum Schema gehört[10]. Es wird vorerst davon ausgegangen, daß Sprecher ihre Handlungsabsichten in

Ein-Satz-Äusserungen ohne Relativsatz-Konstruktionen

mitteilen können. Die gewählte Einschränkung geht nicht als vereinfachende Prämisse in die allgemeine, über das Applikationssystem ICZ hinausgehende Konzeption ein.

---

[9] <..> steht für optional mögliche Strukturlemente, () für Elementbündel

[10] z.B. Ausschmückungen aller Art

1.2 Wege der Ableitung

1.2.1 Expandierungen der UP-Maschine

Der Mechanismus der Ableitung von Prädikaten in PROLOG hat modelltypische Züge, die jetzt herausgearbeitet werden sollen. PROLOG nimmt die **Regelauswahl** konsequent nach der Oben-Unten-Folge in der Regelbasis und der Links-Rechts-Folge innerhalb der Liste der unabhängigen Prädikate je Regel vor. Bild 1.6 zeigt die Folge der Ableitungsversuche mittels der Regeln (1)-(4) des Satzakzeptors.

In Bild 1.6 soll für das Ziel

```
s([the,fox,eats,goose],[]).
```

der Vorgang der Bindung der temporär angelegten Variablen V_i, (i=1,..,n) mittels Top-Down-**Propagierung** der Variablenwerte aufgezeigt werden. Die nach der Auswahl einer Regel gegebenfalls aufzubauenden lokalen[11] Variablen werden nach-unten durchgereicht.

```
step 1
(1)  s=np+vp:        np([the,fox,eats,goose],V1).
(2) np=nomen:        nomen([the,fox,eats,goose],V1).
    => FALSE
step 2
(1)  s=np+vp:        np([the,fox,eats,goose],V1).
(3) np=det+nomen:    det([the,fox,eats,goose],V2).
=>  V2=[fox,eats,goose]

(3) np=det+nomen:    nomen([fox,eats,goose],V1).
=>  V1=[eats,goose]
step 3 ----------------------------------------
(1)  s=np+vp:        vp([eats,goose],[]).
(4) vp=verb+np:      verb([eats,goose],V3).
=>  V3=[goose]

(4) vp=verb+np:      np([goose],[]).
(2) np=nomen:        nomen([goose],[]).
```

Bild 1.6 vollständige Ableitungskette eines Zielprädikats

Die Propagierung führt in step1 zum Widerspruch mit einem (lexikalischen) Fakt. Insbesondere in step2,3 wird der Mechanismus der Regelauswahl sichtbar. Es wird versucht, die Ableitung des Ziels sukzessive durch vollständige Ableitung je eines seiner Elementarprädikate zu erreichen, d.h. für (1) s() ist zunächst np() vollständig abzuleiten. Die hierarchische Taxonomie des Akzeptors weist genau eine untere Ebene zur np/vp-Ebene auf.

Allgemein steigt die Ableitung bis zu den Blättern des Hierarchie-Baums ab. Eine Subroutine- oder UP[12]-Maschine zur Realisierung von CALL/RETURN-Absprüngen

[11] lokal innerhalb der Ableitungskette

[12] Abk. für Unterprogramm

bietet ein gutes Vergleichsmodell dieses Vorgehens. Eine UP-Maschine erlaubt es, aus Unterprogrammen mit beliebiger Schachtelungstiefe weitere Unterprogramme aufzurufen und organisiert einen Rückkehrfaden, der nach Beendigung der vorausliegenden Aufrufebene[13] bis zum Hauptprogramm zurückführt.

Seit dem Aufkommen der strukturierten Sprachen ist das, was ich UP-Maschine nennen will, jedem mit Sprachvergleichen befaßten Programmierer vertraut, hat aber nichtsdestoweniger keinen landläufigen Namen. Dazu findet man in dem Buch von Aho, Rethi und Ullman im Kapitel 'Run-Time Environments' (auf S.391f.) die Stichworte[14]. Ausgangspunkt ist der Begriff *Programm i.A.*[15], d.h. zunächst eine Teilung in Code–Kopf und –Körper.

Definition der UP-Maschine (nach [Aho86]):

'The **lifetime** of an activation of a procedure p is a sequence of steps between the first and the last steps in the execution of the procedure body.
...
In an **activation tree**,

1. each node represents an activation of a procedure,
2. the root represents the activation of the main program,
3. the node for *a* is the parent of the node for *b* if and only if control flows from activation *a* to *b*, and
4. the node for *a* is to the left of the node for *b* if and only if the lifetime of a occurs before the lifetime of b.

...
Control stacks
The flow of control in a program corresponds to a depth-first traversal of the activation tree that starts at the root, visits a node before its children, and recursively visits its children at each node in left-to-right-order. ...
We can use a stack, called *control stack*, to keep track of live procedure activations. The idea is to push the node for an activation onto the control stack as the activation begins and to pop the node when the activation ends. ... When node *n* is on the top of the control stack, the stack contains the nodes along the path from *n* to the root.' ([Aho86])[16]

An der Oberfläche stellt sich das für den PROLOG-User so dar, als würden keine Prozedur-Köpfe, sondern Regel-Köpfe, sprich Prädikate, aktiviert. Gegenstück zum zeitlich ersten UP-Aufruf ist das am weitesten links stehende abhängige Prädikat. Es wird zuerst aktiviert. Der Aufbau von Bild 1.7 hebt 5 Ketten von *Elementarableitungen* hervor, die eine *Top-down-Propagierung* von Variablenwerten realisieren, bis eine zulässige Prädikation über Konstanten erreicht wird. Jede der Ableitungsketten, deren Numerierung die zeitliche Abfolge indiziert, heiße eine **Expandierung**.

---

[13] W: erfolgreiche Ableitung, F: Widerspruch zu Fakten

[14] noch näher an die Arbeit realer Prozessoren, die sie mittels Befehls- und Stackzeiger lösen, rückt Klaeren [Kla91]

[15] in [Hop88] spielt der Begriff nur eine Nebenrolle, es dominiert die Zuordnung abstrakte Maschinen/formale Sprachen.

[16] in [Vac90] werden Varianten der UP-Maschine, sprich 'control stack', zur *Fadencode*-Compilation behandelt

Prädikationen (Prädikatsableitungen) sind in Expandierungen zerlegbar.

In Bild 1.7 wird aus den Expandierungen

`s –> np –> det,  np –> nomen` (in step 2),

`s –> vp –> verb,  vp –> np –> nomen` (in step 3)

die – in PROLOG eindeutig bestimmte – Zerlegung für eine vollständige Ableitung zusammengebaut. Die PROLOG-UP-Maschine führt im Normalfall nur soviele Expandierungen durch, als zur erstmaligen Ableitung des Ziels notwendig sind.

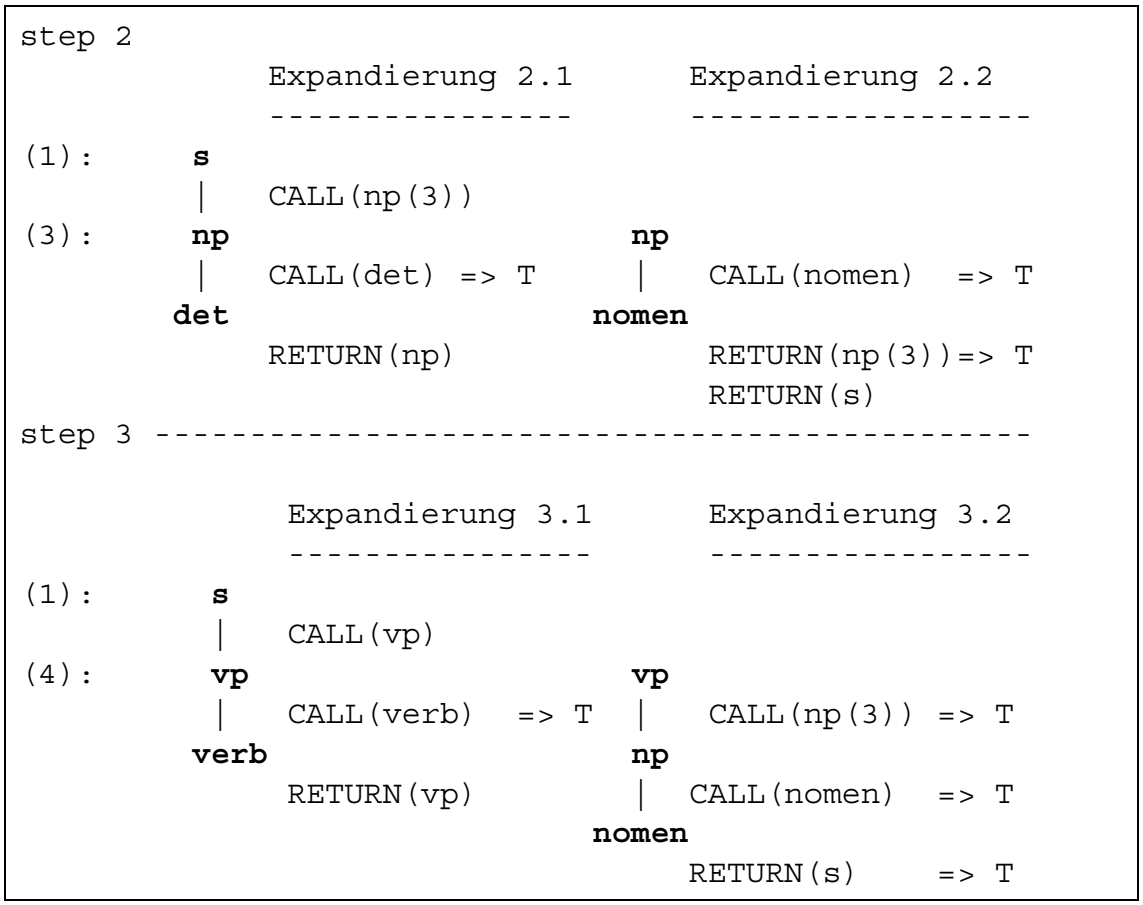


Bild 1.7 Expandierungsketten der UP-Maschine

Mit der aus Expandierungen zusammengesetzten Ableitung hat man zugleich ein Elementarmodell einer **Tiefensuche** im Prädikationssystem vorliegen. Expandierungen sollen so schnell wie möglich in die “Tiefe” der Hierarchie führen.

Zum Vergleich: in ERNEST wird systematisch Gebrauch von Expandierungen gemacht, um sogenannte *Prädiktionen*, sprich Vorhersagen, d.h. auf Modellwissen gestützte, restriktive Auswahlen von Hypothesen, durchzuführen. Der UP–Aufrufmechanismus zur Regelauswahl von PROLOG wird als Teilmechanismus in eine Maschine eingebaut, als deren Leitprinzip die A*-Suche[17] eingesetzt wird.

Im Fall des Akzeptors setzt die Anwendung des UP-Mechanismus das Bekanntsein des vollständigen Satzes oder der vollständigen Wortkandidatenmenge voraus. Der Verzicht auf genau diese Voraussetzung bildet gerade ein Leitmotiv der Abhandlung.

[17] der A*-Suchalgorithmus wurde von Nilsson [Nil82], [Gen89] untersucht

Eine *inkrementelle Verarbeitung* von Satzteilen könnte mit diesem Mechanismus und strenger Top-Down-Expandierung nur dann erfolgen, wenn diskontinuierliche[18] Satz-Konstituenten ausgeschlossen sind. Inkrementelle Verarbeitung diskontinuierlicher Konstituenten erfordert einen prozeduralen Aufsatz zum UP-Mechanismus wie er z.B. mit der Zusatzkontrolle des Shift-Reduce-Parsers [Eik89] gegeben ist, der im speziellen Kapitel über linguistische Datenverarbeitung vorgestellt wird.

Inkrementelle Verarbeitung meint die stückweise Verarbeitung linksbündig signalüberdeckender Wortketten.

1.2.2 Bottom-Up-Wege

ERNEST stellt Basisoperationen bereit, mit denen das systematische Alternieren von Bottom-Up- und Top-Down-Ableitungen organisiert werden kann. In [See92] habe ich einen Versuch gemacht, diese Basisoperationen abzugrenzen. Das Alternieren, insofern es den Wechsel modell- und daten-getriebener Methoden betrifft, wird häufig als epistemologisch notwendig herausgestellt [Kan80].

Es kann auch durch geschickte Wahl aufeinander aufbauender Subziele erreicht werden. Eine typische Lösung in PROLOG wird durch die Einführung von '*Meta-Prädikaten*' erreicht. Zwar gilt grundsätzlich:

Ableitungen sind generell top-down gerichtet.

Jedoch kann mit Hilfe der Zusatzprädikate wortart() und phrase() in Bild 1.8 eine sukzessive Zielbestimmung vorgenommen werden, wie sie in Bild 1.9 angegeben ist. Beide Prädikate stellen Sammel-Cluster aller Prädikate einer ausgewählten Hierarchie-Ebene zusammen:

wortart() für die unterste Prädikat-Ebene, phrase() für die Ebene der höchsten Satz-Konstituenten. Die Clusterung wird als Disjunktion von Prädikaten ausgedrückt.

```
(5) wortart(X,Y,W)
   :-(det(X,Y);nomen(X,Y);verb(X,Y)), X=[W|Y].
(6) phrase(X,Y):-(np(X,Y);vp(X,Y)).
```

Bild 1.8 Meta-Prädikate zur Bottom-up-Suche

Die Verwendung der zusätzlichen Variablen W in Regel (5) dient dazu, die Information 'aktuell ausgewähltes Wort der Eingabekette' an spätere Subziel-Bestimmungen "durchreichen" zu können.
Die Verwendung ist ablesbar anhand der Zielformulierung mittels phrase() in Bild 1.9:

phrase([W|Y],Z).

[18] Ketten aus nicht aufeinanderfolgenden Subketten von Lexemen

```
1 ?- wortart([the,fox,eats,goose],Y,W),
|    phrase([W|Y],Z).
  W = the
  Y = [fox,eats,goose]
  Z = [eats,goose]
```

Bild 1.9 Bottom-Up–Bestimmung von Subzielen

Bottom-Up-Kontrolle der Suche ist subsequente Auswahl und Ableitung hierarchie-höherer Subziele passend zu Elementargegenständen der Domäne.

In Bild 1.10 werden zu den 'Elementargegenständen' [the] und [eats] durch Anwendung der Clusterprädikate jeweils Subzielfolgen 1.9 bestimmt. Die verknappte Darstellung des CALL-RETURN-Ablaufs der UP-Maschine soll zeigen, daß jeweils nach der 'partiellen' Ableitung des höchsten Subziels der Folge zur vollständigen Ableitung automatisch eine Expandierung erfolgt. Im Fall des linguistischen Prädikationssystems ist an dieser Stelle erneut die Problematik der Bearbeitung diskontinuierlicher Konstituenten zu beachten.

Man benötigt den in PROLOG nicht betrachteten Begriff der **partiellen Ableitung** eines Prädikats. Wie Bild 1.10 zeigt, kann das Prädikat np bezüglich Regel (3) entlang der Subzielkette als partiell abgeleitet betrachtet werden, nicht jedoch bezüglich Regel (2) (angezeigt durch TRUE-, FALSE-Marker).

Partielle Ableitungen werden von der UP-Maschine zur Realisierung von Expandierungsketten intern mittels Erzeugung lokaler, ungebundener Variablen durchgeführt.

Während Ableitung allgemein durch domänen-unabhängige Automatismen der Regelauswertung erfolgen kann, kommen bei der Subziel-Bestimmung Freiheitsgrade in Betracht, die beim Entwurf der Wissensbasis in Form von Meta-Prädikaten vorbereitet und durch eine problem-orientierte Kontrolle ausgenutzt werden können.

problem-orientierte Kontrolle umfaßt Programme

- zur Wahl der Subziele
- zum temporären Einfrieren partieller Ableitungen
- zur Auswahl von Expandierungen
- zum Alternieren der Ableitung
- von Subzielen und Expandierungen

Bei den semantische Netzen des KL-ONE-Typs wird das Verhältnis von Prädikaten und Meta-Prädikaten thematisiert. Der Netzformalismus ERNEST präzisiert mit einem eigenen problem-unabhängig formulierten System von Metaregeln den Mechanismus der partiellen Ableitung.

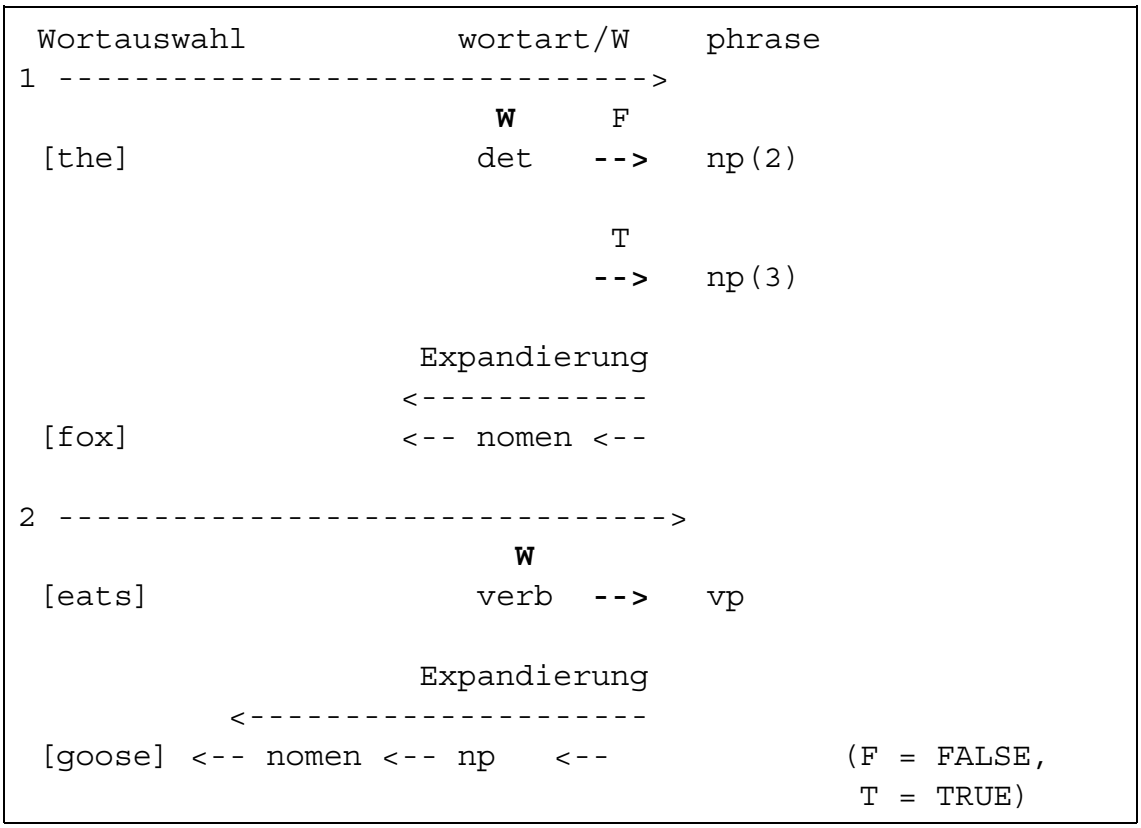


Bild 1.10 Alternieren von Bottom-Up– und Top-Down-Suche

1.2.3 Parser erzeugen Strukturbeschreibungen

Zur Vertiefung soll an dieser Stelle bereits die Weiterentwicklung des Satzakzeptors zum Parser vorgestellt werden, die Eikmeyer in [Eik89] vorgenommen hat.

```
(1)  s(s(S1,S2),X,Y)  :- np(S1,X,Z),vp(S2,Z,Y).
(2) np(np(S1),X,Y)    :- nomen(S1,X,Y).
(3) np(np(S1,S2),X,Y) :- det(S1,X,W),nomen(S2,W,Y).
(4) vp(vp(S1,S2),X,Y) :- verb(S1,X,Z),np(S2,Z,Y).
(5) wortart(S1,X,Y,W)
  :-(det(S1,X,Y);nomen(S1,X,Y);verb(S1,X,Y)),X=[W|Y].
(6) phrase(S1,X,Y)    :-(np(S1,X,Y);vp(S1,X,Y)).
det(det(the),[the|X],X).
det(det(a),[a|X],X).
nomen(nomen(goose),[goose|X],X).
nomen(nomen(geese),[geese|X],X).
nomen(nomen(fox),[fox|X],X).
nomen(nomen(foxes),[foxes|X],X).
verb(verb(eats),[eats|X],X).
verb(verb(eat),[eat|X],X).
```

Bild 1.11 Erweiterung des Satzakzeptors zum Parser

Ein ’**Parser**, d.h. ein Verfahren zur automatischen Sprachanalyse, hat eine zweifache Aufgabenstellung zu lösen:

1. er muß grammatisch wohlgeformte Sätze akzeptieren und nicht-wohlgeformte Sätze zurückweisen,
2. er muß die wohlgeformten Sätze mit einer **Strukturbeschreibung** (Hervorhebung v. Verf.) versehen.

Beschränkt man sich nur auf die erste Aufgabe, hat man es mit einem **Recognizer** zu tun.’ Das folgende Bild 1.11 stellt die Erweiterung des Recognizers zum Parser dar.

Mit Bild 1.12 werden verschiedene Effekte der Strukturbeschreibung ausgewiesen. Die Form der Strukturbeschreibung ist:

1. Namen stehen für Verzweigungsknoten.
2. Klammern hinter Namen stehen für den Cluster der vom zugehörigen Knoten abgehenden Verzweigungen und Unterverzweigungen.

```
1 ?- s(Struktur,[the,fox,eats,goose],[]).
       Struktur =  s(  np( det(the),
                           nomen(fox)
                         ),
                       vp( verb(eats),
                           np(nomen(goose))
                         )
                     )
```

```
2 ?- wortart(SW1,[the,fox,eats,goose],Y1,W1),
|    phrase(SP1,[W1|Y1],Y2),
|    wortart(SW2,Y2,Y3,W2),
|    phrase(SP2,[W2|Y3],Y4).
     W1  = the
     W2  = eats
     Y1  = [fox,eats,goose]
     Y2  = [eats,goose,
     Y3  = [goose]
     SW1 = det(the)
     SW2 = verb(eats)
     SP1 = np( SW1, nomen(fox) )
     SP2 = vp( SW2, np(nomen(goose)) )
```

Bild 1.12 Strukturbeschreibung eines Satzes

Anfrage 1 zeigt, wie die normale Arbeitsweise des Akzeptors zur parallelen Erzeugung des Strukturbaums[19] benutzt werden kann.

[19] man spricht auch vom ’Syntaxbaum’

In Anfrage 2 wird eine vollständige Subziel-Sequenz mittels *Strukturvariablen* SW1/2, SP1/2 in der Absicht notiert, auch die Zwischenstufen beim Aufbau des Strukturbaums sichtbar zu machen. Dadurch wird der korrekte Gebrauch der Metaprädikate wortart, phrase zur Steuerung der Bottom-Up-Suche erst transparent. Die Strukturbäume beider Anfragen müssen identisch sein.

Jedem abhängigen Prädikat wird in Bild 1.11 eine zusätzliche Variable an identischer Position im Argumenttupel eingefügt, um einen zweiten, parallelen Ableitungsfaden aufzubauen, der zur bisher für sich betrachteten Bindung der Wortketten-Variablen korrespondiert.

Die Bindung der zusätzlichen 'Struktur'-Variablen erfolgt jedoch völlig entkoppelt von der erstgenannten, indem gesonderte 'Regeln innerhalb von Regeln' folgender Form aufgebaut werden:

$a(X_1, \ldots, X_n)$:- $b_1(X_1), \ldots, b_n(X_n)$.

Für die Regeln (1), (2) ergibt das:

(in (1)) `s(S1,S2) :- np(S1),vp(S2).`,

(in (2)) `np(S1) :- nomen(S1).`

Eine der wesentlichen Neuerungen von ERNEST besteht darin, die Strukturbeschreibung zu einem wesentlichen Element der Kontrolle auszuarbeiten. Das ist kein fertiger Zustand, sondern ein Entwicklungsprozeß, der von neuen Anwendungen lebt. Aufgaben inkrementeller Verarbeitung machen massiven Gebrauch von der Analyse der Strukturbeschreibung.

1.3 Semantisches Netzwerk als Kernstruktur der Regelbasis

1.3.1 Elementarisierung von Regeln

Eine PROLOG-Wissensbasis ist ein 2–dimensional angeordnetes Regelsystem:
— vertikal als Reihenfolge der Regeln, welche dem Mechanismus der *Regelauswahl* zugrunde liegt,
— horizontal als Reihenfolge der zu betrachtenden unabhängigen Prädikate der Regel. Netz- und Regelformalismus gehen im landläufigen Verständnis nicht notwendig zusammen. So wird nicht selten die deklarative/prozedurale Semantik als exklusiver Vorzug von PROLOG angeführt.

Es soll versucht werden, die Verbindungslinie zu ziehen.

Im ersten Schritt wird der Satzakzeptor als Graph angeordnet. Vereinfachend wurden die Kanten zu den lexikalischen Items (unterster Hierarchie-Level) zusammengefaßt:

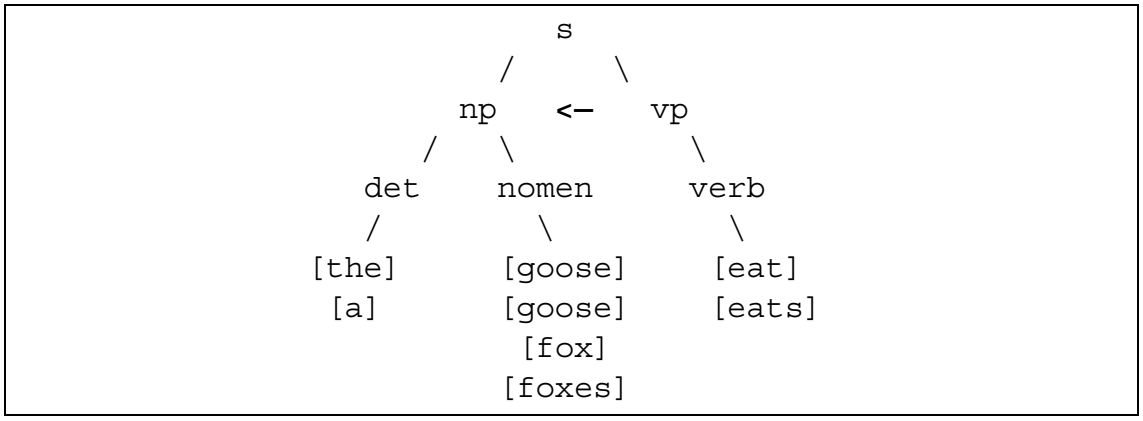


Bild 1.13 Hierarchie des Satzakzeptors

Der Pfeil 'np <— vp' soll zum Ausdruck bringen, daß das Prädikat vp() zwar von np() abhängt, jedoch die umgekehrte Beziehung nicht besteht. Faßt man das Vorliegen der Kante zwischen 2 Knoten inhaltlich auf, so handelt es sich in Fortsetzung der Beziehung np/vp um:

die *gerichtete binäre Relation* **(R)** von einem abhängigen zu genau einem seiner unabhängigen Prädikate, wobei die Richtung die irreversible Abhängigkeit festhält.

Betrachtet man das System der Abhängigkeitsbeziehungen im Akzeptor, so sind alle sonstigen Kanten ebenfalls als (nach-unten) gerichtet aufzufassen. Trägt man diese Beziehungen als Pfeile in Bild 1.13 ein, so kann man leicht die Azyklizität[20] entlang sämtlicher Kantenpfade des Netzes ablesen.
Auch rekursive Bezüge eines Prädikates auf sich selbst in der Position des Elementarprädikates kommen nicht vor, was mit dem Stichwort *Nicht-Rekursivität der Prädikation je Regel* festgehalten werden soll. Elementarisierung, Azyklizität und Nicht-Rekursivität, mit obiger Einschränkung, sind die Kennzeichen der Netzform des Satzakzeptors.

Wir orientieren uns im Folgenden nur noch an solchen Gestaltungsmitteln des PROLOG-Formalismus, die bei Ausbau des ohnehin unserer Anwendungsaufgabe nahestehenden Akzeptors und anderen computerlinguistischen Ansätzen relevant sind. PROLOG als Universalsystem muß die Möglichkeit von Azyklizität und Rekursivität verständlicherweise zulassen.

Sagerer hat in [Sag90] die Entwicklungslinie der semantischen Netze zu den KL-ONE ähnlichen Systemen und von dort bis zum System PSN ('procedural semantic network') [Lev79] nachgezeichnet. ERNEST steht in der Linie der Fortsetzung von PSN, indem Prädikationen (in gängiger Redeweise auch *Konzepte* genannt) zusätzlich an Berechnungsprozeduren gebunden werden, die außerhalb des Netzformalismus in einer rein prozeduralen Sprache codiert werden. Nicht die Universalumgebung PROLOG schlechthin, sondern ihre spezielle computerlinguistische Verwendung bietet dazu interessante Analogien. Zwar ist das Prädikationssystem des Akzeptors streng nicht-rekursiv aufgebaut. Für ergänzende Constraint-Beziehungen, z.B. definiert auf Zwischenvariablen wie Z in

a(X,Y) :- b(X,Z), c(Z,Y).,

[20] der Ausschluß gerichteter Kantenpfade, die zu ihrem Ausgangspunkt zurückführen

kann es aber durchaus rekursive Berechungen geben. Erst recht machen die ohnehin algorithmisch zu fassenden Mechanismen wie z.B. der Shift-Reduce-Parser [Eik89] von Rekursion Gebrauch. Dieser stellt eine typische Problemlösung der Computerlinguistik für die Links-Rechts-Verarbeitung von Wortketten dar.

Wir analysieren nun die intuitive Leitvorstellung, die es uns erlaubt, ein dem Satzakzeptor zugrunde liegendes "semantisches Netz" in der Weise von Bild 1.13 mittels Knoten und Kanten darzustellen.

Die Netz-Darstellung ist eine Abstraktion der Regelbasis. Wovon wird abstrahiert?

1. Regeln sind im Netz nicht mehr explizit gegeben. Explizit gegeben sind nur die Beziehungen (R) zwischen Prädikaten und Elementar-Prädikaten, aus denen sich die Regeln zusammensetzen.
2. Die rechte Seite einer Regel fungiert als Kapsel für die Definition von Vorrangbeziehungen zwischen Elementar-Prädikationen. Der Vorrang von np() vor vp() in s()[21] geht im Netz verloren.
3. Die Ableitbarkeit eines Prädikats mittels verschiedener Regeln, wie z.B. np(), wird unsichtbar. Sichtbar wird eine Kollektion aller unabhängigen Prädikate für jedes abhängige Prädikat.
4. Von Variablen wird kein Gebrauch gemacht.
5. Prädikate zu Variablen und Fakten verbinden sich zu einem gemeinsamen Netz, das in Hierarchie-Ebenen gegliedert ist. Fakten bilden die unterste Ebene.

Eine einheitliche Definition semantischer Netze ist in der Literatur nicht zu finden. Der hier gewählte Zugang stützt sich auf den mit der DCG aufgewiesenen Wissensbasistyp des Prädikationssystems. *Prädikation* verweist auf den Aufbau 'höherer' Prädikate, die damit schärfer gefaßt werden als in PROLOG, das auch Prädikate rein prozeduralen Inhalts zuläßt. Die Besonderheit semantischer Netze besteht darin, die Hierarchie eines Prädikationssystems, seine **Taxonomie**, zu Lasten anderer Aspekte hervorzuheben. Das mag die einseitige Vorstellung eines reinen Begriffsnetzes nahegelegt haben. Ist allgemein von 'Taxonomie' die Rede, so bleibt noch offen, ob es sich um eine strenge Etagenteilung einer Hierarchie wie in Bild 1.13 oder eine beliebige Baumstruktur handeln soll, mit der man Vererbung z.B. im Sinne der objekt-orientierten Programmierung durchführen kann. In ERNEST sind beide Aspekte wirksam, wobei der Vererbungsaspekt spezielle Fassung erhält.

Es gibt in der Literatur eine Reihe von Versuchen, Wissensbasen in Regelform versus Netzform zu vergleichen. Dabei werden insbesondere verschiedene Schlußfolgerungen aus der **Elementarisierung** des Beziehungstyps, sprich der Reduktion auf den **(R)**-Typ, gezogen.

M.Richter sagt semantischen Netzen generelle Schwierigkeiten bei der

"Repräsentation von
– *anderen als binären Prädikaten,*
– *logischen Verknüpfungen,*
– *Quantoren* [Ric89]"

nach, wobei er als "Prädikate" die Beziehungen $R(p(),p_i())$ hernimmt. Damit gerät er

[21] im Englischen durch die *word order* gefordert

auf die falsche Fährte bei der Suche des Äquivalents für logische Verknüpfungen im semantischen Netz:

"Die Konjunktion zweier Prädikate wird einfach dadurch ausgedrückt, daß die entsprechenden Pfeile (Kanten, d.Verf.) im Netz vorhanden sind" [Ric89].

Den Vorteil der expliziten Taxonomie zieht er nicht in Erwägung.

Grisham präsentiert in [Gri86] die gängige Meinung, semantische Netze seien, im Unterschied zum Kalkül der Prädikatenlogik, nicht standardisiert und man könne sich bei ihrer Besprechung lediglich an Applikationen halten. Auch er beläßt es bei der Wiederentdeckung der Prädikate in den Netzkanten[22], die er dann noch als spezielle Kantensorte faßt[23]. Offenbar versteht sich die Auffassung semantischer Netze als Prädikationssysteme, die über einen Ableitungsmechanismus verfügen, nicht von selbst.

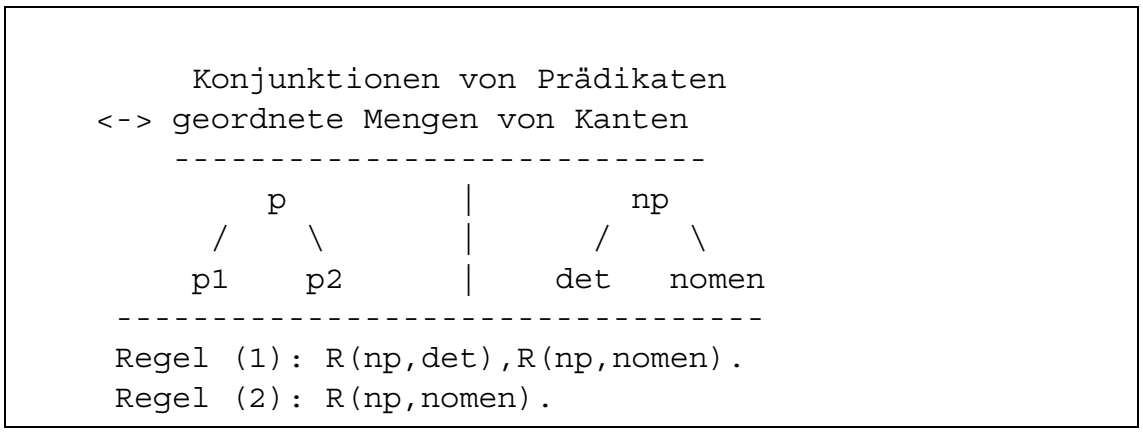


Bild 1.14 Regeln als geordnete Kantenmengen

Im semantischen Netz entspricht allgemein der konjunktiven rechten Seite der Ableitungsformel die geordnete Menge derjenigen Kanten, die vom gleichen Knoten abgehen:

A. Netz-*Knoten* werden als
   Kapseln aller der Regeln, sprich Kanten-Konjunktionen, eingesetzt, die zur Ableitung je eines Prädikats bestimmt sind. In ERNEST wird die Idee in Form der *Modalitäten*, sprich zulässigen Kanten-Mengen eines Konzepts (Prädikats) realisiert.
B. Die Taxonomie der Knoten als der Regelbasis vorgeschaltete Informationsquelle effektiviert die Regelauswahl.
C. Variation über die Reihenfolge und Anzahl der abhängigen Prädikate einer Regel läßt sich in Form von *Adjazenzen*, sprich Matrizendefinitionen der zulässigen Anordnung von je 2 Kanten zusammenfassen. Im linguistischen Anwendungsfall handelt es sich z.B. um die 'word order' der Kategorien, d.h. der Wort-Arten wie Determinans, Nomen.

1.3.2 Taxonomische Abstraktion

In ERNEST werden streng taxonomisch gegliederte Wissensbasen aufgebaut, indem auch die Umkehrung der Zuordnung gilt:

[22] Figure 3.1, [Gri86], S.96
[23] Figure 3.2, [Gri86], S.96

semantische Netzknoten modellieren durchweg Prädikate.

1. ERNEST-Netze sind im doppelten Sinne taxonomisch:
   — als Etagenteilung in syntaktische/semantische/pragmatische Ebenen der Prädikation,
   — innerhalb jeder Ebene durch die absteigende Stufenfolge von Meta-Prädikaten zu Prädikaten (Kante SPEZIALISIERUNG in Bild 1.15).
2. ERNEST-Netze enthalten ausschließlich Knoten, die Prädikaten zugeordnet werden.
3. Alle Knoten oberhalb der Fakten-Ebene HYPOTHESEN (siehe Bild 1.15) sind Regelkapseln, keineswegs bloße Speicher für Konstanten wie Daten, Eigennamen, Wörter.
4. Die in anderen Formalismen gängige Typenvielfalt der Kanten wird auf 3 taxonomische Grundrelationen (siehe untenstehendes Bild)[24] reduziert.

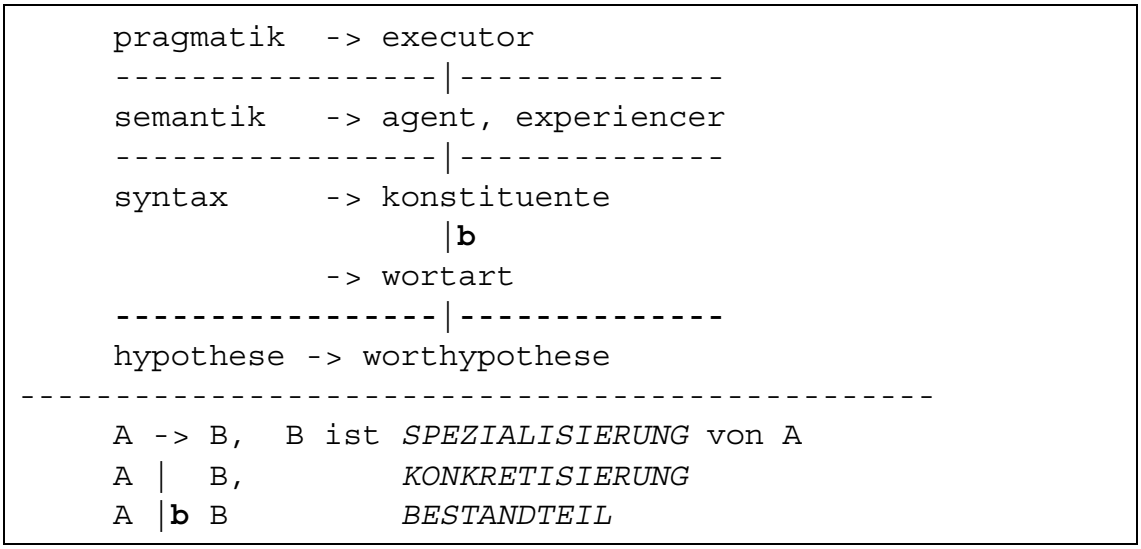


Bild 1.15 2D-Taxonomie der Wissensbasis zur Zugauskunft

Für den Kantentyp SPEZIALISIERUNG wurde anhand der Metaprädikate wortart, phrase (konstituente in Bild 1.15) die Funktion der Unterstützung der Regelauswahl aufgezeigt.

Die horizontale Hauptachse der Prädikation in ERNEST wird gemeinsam durch die Kantentypen KONKRETISIERUNG und BESTANDTEIL gebildet. Für diese Differenzierung gibt es inhaltliche und formelle Gründe. Eine Regel wie

executor(X):-agent(X),pragmatik_klasse(X)= exekutierend.

drückt eine Verschärfung der Rollen-Abstraktion aus. Die Gegenstandsgesamtheiten der Domäne werden aus einem erheblich veränderten Blickwinkel betrachtet.

Prädikationssysteme sind Abstraktionssysteme: sie unterscheiden *Sichten*.

Das trifft auch auf den Satzakzeptor zu. Seine Taxonomie enhält genau einen Abstraktionsschritt, den Übergang von der Phrasen- zur Satzbildung. Das erkennt man an der Gestalt von Regel (1) des Satzakzeptors. Sie fordert die Zerlegung einer Äußerung in 2 höchste Konstituenten NP und VP und bindet dadurch die Akzeptanz

[24] zur “Wiedergeburt” der ’verdrängten’ nicht-taxonomischen Kantentypen siehe Kapitel-Abschnitt über Attribute

von Äußerungen an Grammatikalität im Sinne von Satzgrammatik. In Hinsicht auf diese Einschränkung war ja auch vom *Satz*-Akzeptor gesprochen worden. Als Parser, der auf spontan-sprachliche Äußerungen angewandt werden soll, stellt er eher einen Sonderfall dar.

Die Modellierung im System ERNEST+ICZ verzichtet auf Regeln der Satzgrammatik und steht damit im Bereich sprachverstehender Systeme nicht allein, wie sich aus den Veröffentlichungen von W.Ward [War91a] entnehmen läßt. Allein die Systeme, welche dieser Leitprämisse folgen, sollen daher auch im engeren Sinn zum Systemvergleich herangezogen werden. Mit Blick auf bekanntgewordene Applikationssysteme wird folgende Vorausschau gewagt:

Applikationssysteme mit spontan-sprachlicher Eingabe verzichten auf Satzgrammatikalität.

Für die Gestaltung des *Netzwerk-Compilers*[25] sind Kanten des Typs KONKRETISIERUNG nützlich, um Prüfbedingungen betreffs der Ebenengliederung eines vorgegebenen Netzes aufzustellen. Statt die Netzkonsistenz anhand von Kanten-Namen vorzunehmen, wird z.B. mittels Navigation getestet, ob jeder Durchgangsweg durch einen Verbindungspfad über Netzknoten von der Hypothese-Ebene bis zu den Endknoten der Pragmatik die gleiche Anzahl Kanten dieses Typs enthält. Die formalisierten Testbedingungen hat Sagerer in [Sag90] formuliert.

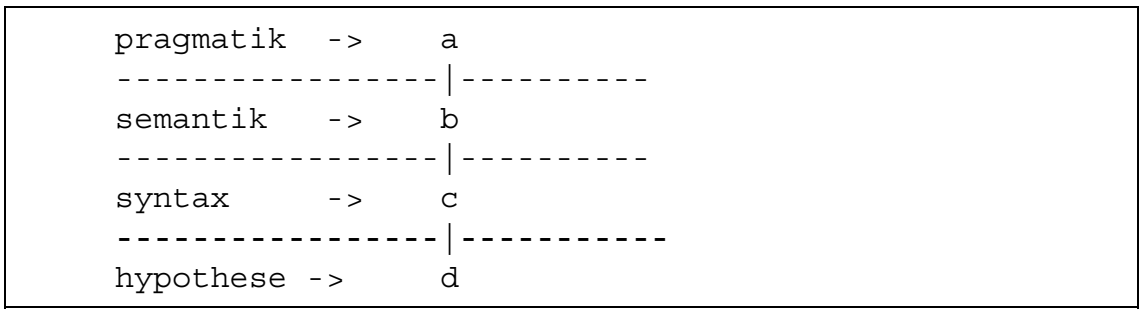


Bild 1.16 Schichtentrennung mit der Kante KONKRETISIERUNG

Aufgrund der in Bild 1.16 veranschaulichten expliziten Schichtentrennung kann dem User ein Meta-Attribut *grad* zur Verfügung gestellt werden:

$1 \leq \text{grad(konzept)} \leq$ Anzahl der Hierarchie-Ebenen.

ERNEST-Netze kann man dazu verwenden, ein Parsing[26] über den Eingabedaten durchzuführen. Zur Analyse der Zwischenstufen der entstehenden Strukturbeschreibung kann der User z.B. durch Abfrage des grad(konzept) für bereits (gegebenfalls partiell) abgeleitete Prädikate eigene Kontrollprogramme entwerfen.

---

[25] Übersetzung von Textdeklarationen in exekutierbaren Code, z.B. von Eigennamen auf den in Kapitel 3 benutzten numerischen Identifikator-Datentyp ID – fasse ich als Compilation auf

[26] Parsing im verallgemeinerten Sinn, siehe den Abschnitt über Tiefenstruktur-Parsing in Kapitel 2

Mit einem Beispiel aus der Bildverarbeitung werde der epistemologische Sinn der folgenden Unterscheidung aufgezeigt:

```
"SPEZIALISIERUNG"  vs.  KONKRETISIERUNG
----------------       ---------------
Grünfläche(X)          gras(X):-
:-gras(X);               bildregion(X),
 (bildregion(X),         bildfarbe(X)=grün.
  bildfarbe(X)=grün).
```

Bild 1.17 Kontroverse Verwendung taxonomischer Kanten

'Gras' und 'Bildregion' sind Termini aus verschiedenen Begriffswelten. Zu Ihrer Verbindung via Prädikation stellt ERNEST die KONKRETISIERUNG bereit. Die "SPEZIALISIERUNG" wird ausdrücklich nicht dafür verwendet. Als Begründung formuliert Sagerer im Anschluß an Myloupoulus (von mir sinngemäß wiedergegeben):

Ein Prädikat kann im Verhältnis zu einem anderen Prädikat nur dann 'spezialisierend' wirken, wenn es der gleichen Terminologie und Abstraktionsebene zuzurechnen ist.

Als deutliche Anzeichen für das Vorliegen eines Abstraktionsschritts sind zu werten:

das Überschreiten der Grenzen einer Terminologie, eines Paradigmas (im Sinne von Kuhn [Kuh79]) oder der Wechsel des Betrachterstandpunkts.

Insofern kann der Begriff nicht allein mit den Mitteln der formalen Logik definiert werden. In den Erkennungsdisziplinen, die ein Sprach- oder Bild-Signal bearbeiten, drückt er den qualitativen Fortschritt der Analyse aus.

Man findet wenige Arbeiten zur Netzwerkmodellierung, welche die Charakteristik des Abstraktionssystems betrachten, so Levesque und Myloupolus in [Lev79]. Es überwiegen Darstellungen, die nur genau eine Hierarchieachse betrachten und diese als Achse der 'Spezialisierung' auffassen, was immer noch zu entsprechenden Rezeptionen in der Computerlinguistik führt, z.B. Sabah in [SAB93].

Weitere allgemein-methodologische Aspekte der Semantik der Kanten folgen im Kapitel-Abschnitt 'Kanten-Indikation'.

In der objekt-orientierten Programmierung wird Taxonomie ausgenutzt, um Vererbung von Bestandteilen der Systemkonstruktion durchzuführen. Als Ziele werden die bessere Wartbarkeit und Transparenz der Konstruktion der Programme angestrebt. Der ERNEST-Netzcompiler führt entlang der Kanten-Achse SPEZIALISIERUNG innerhalb von Abstraktionsebenen die Vererbung von Deklarationsbestandteilen von Konzepten durch. Die Deklaration allgemeinerer Konzepte zentralisiert Elemente derselben für speziellere Konzepte und muß so vom Gestalter der Wissensbasis nur einmal vorgenommen werden.

1.4 Relationalstruktur der Konzepte

1.4.1 Kanten-Rollen

In ERNEST werden Regelbasen in den Überbau eines semantischen Netzes eingebettet. Die Knoten des Netzes modellieren Konzepte und werden als Frames aufgebaut, die selbst wieder Substrukturen enthalten.

In Bild 1.19 werden die wichtigsten Untereinheiten der Frame-Deklaration des Konzepts SY_NG im ICZ-Netz dargestellt.

Die Bezeichnungsweise für Konzepte sind gegliedert in:

1. Prefix —> Hierarchie-Ebene:
   Hypothesen, **SY**ntax, **S**emantik, **P**ragmatik
2. Suffix —> Konzeptname, z.B.
   _NG für **N**ominal**G**ruppe (np)

Betrachtet man ERNEST als einen Formalismus zur Herstellung strukturierter Regelbasen für Hypothesen-Suchprobleme, so wird die in Bild 1.19 zur Unterstützung eingezeichnete TEIL 1/ 2–Gliederung des Gesamt-Frames verständlich. Beide Teile haben die Aufgabe, eine Nach-Innen-Projektion der Relationen eines Knoten zu anderen Knoten des Netzes zu realisieren. Es geht darum, die Information über alle eingehenden und abgehenden Kanten des Knotens in ihm selbst zu zentralisieren. Bench-Capon hat in [BC90] versucht, die Präzisierung der prozeduralen Semantik semantischer Netze im wesentlichen mit 2 Maßnahmen zu erreichen:

1. Frame-Strukturierung der Netze,
2. Clusterung der Kanteninformation in den Knoten.

Eigentlich müßte dann eine gesonderte Gestaltung von Kantenelementen überflüssig sein.

Um Netzkonstruktionen zu unterstützen, die einen zielgerichteten Ableitungsprozeß steuern können, hält sich ERNEST an die:

Rollenform der Regeln im semantischen Netz

Die Abhängigkeit eines Konzepts p von Elementarprädikationen p_j, (j=1,..,n), heiße **rollenverteilt**,
falls in jedem Argumenttupel $Arg_j(p_j)$
nur genau eine Gegenstandsvariable auftritt. Anders ausgedrückt:

die Kantenorganisation eines semantisches Netz ist rollenverteilt, wenn je Regel und Regel-Kante genau eine Gegenstandsvariable zu binden ist.

Datenbanken kennen die ’Rolle’ als ’Feld’–Definition zur Strukturierung von Datensätzen[27].

Versuchte man die Rollenform von Netzregeln äquivalent zu PROLOG als Einschränkung der allgemeinen Regelform zu notieren, so käme am ehesten eine Metaregel in Frage:

---

[27] vergleichbar sind auch die Bestandteile eines ’struct’ in der Programmiersprache C

$p(X_1,...,X_n)$ $_{meta}$:- $rolle_1(X_1),...,rolle_n(X_n)$. (i=1,..., n)

Bild 1.18 Regelschreibweise für Konzepte im semantischen Netz

Die Metaregel — eine Abstraktion aus einer Menge von Einzelregeln — erkennt man hier an dem Tatbestand, daß die abhängigen Prädikate (die Rollenträger) durch ihre Aquivalenzklasse 'rolle$_i$', (i=1,..., n), ausgedrückt werden.

Von den 3 in ERNEST verwandten, miteinander korrespondierenden Rollen-Bestimmungen für Kanten entspricht die 'rolle$_i$' dem Mittelglied in:

```
Allgemeines          Besonderes       Einzelnes
-----------------------------------------------
taxonomische Rolle   Rollentyp     Rollenträger
```

Die taxonomischen Rollen realisieren die Asymmetrie, sprich Nicht-Austauschbarkeit der Positionen in den gerichteten Beziehungen
SPEZIALISIERUNG, KONKRETISIERUNG, BESTANDTEIL.
Es wird deshalb vertikal und horizontal unterschieden in der:

```
                        Hierarchie-Richtung
   ---------------------------------------
   SPEZIALISIERUNG          nach unten
   SPEZIALISIERUNG_VON      nach oben
```

Der Frame TEIL 1 (siehe Bild 1.19) widmet sich ausschließlich der Nach-innen-Projektion von Kanten durch Bestimmung der taxonomischen Rollen des Nach-oben-Typs. Es werden alle *Quellkonzepte* Q, sprich Rollenträger, zusammengefaßt, die dem zu deklararierenden Konzept K in der _VON-Rolle gegenüberstehen:

```
K = SPEZIALISIERUNG_VON (Q);
    KONKRETISIERUNG_VON (Q);
    BESTANDTEIL_VON (Q).
```

TEIL 2 clustert die *Zielkonzepte* Z, zu denen K nun seinerseits in der _VON-Rolle steht, und verteilt diese wiederum rollengerecht auf 3 Listen. Im Fall von SY_NG gibt es nur den Subframe

```
START_BESTANDTEILE ... ENDE_BESTANDTEILE
über alle Z: Z = BESTANDTEIL(K)
```
[28].

Das Tripel von Subframes enthält die Liste der Rollentypen:

1. Typname der Rolle
2. Liste der Rollenträger, sprich Zielkonzepte

[28] zu lesen: K hat BESTANDTEIL Z

```
START_KONZEPT SY_NG

   TEIL 1
 ------------------------------------------------
 |  SPEZIALISIERUNG_VON ( SY_KONSTITUENTE )      |
 |  KONKRETISIERUNG_VON ( S_AGENT, ...  )        |
 |  BESTANDTEIL_VON     ( SY_PNG )               |
 ------------------------------------------------
   TEIL 2
 ------------------------------------------------
 |  START_BESTANDTEILE( 9 )                      |
 |     ROLLE ( nukleus )                         |
 |        ZIELKNOTEN ( SY_PRON, SY_REFLPRON,... )|
 |     ROLLE ( adjektiv )                        |
 |        ZIELKNOTEN ( SY_ADJ )                  |
 |     ....                                      |
 |  ENDE_BESTANDTEILE                            |
 | ---------------------------------------------- |
 | |  START_MODALITAETEN( 3 )                  | |
 | |      MODALITAET                           | |
 | |        OBLIGATORISCH( nukleus )           | |
 | |        OPTIONAL                           | |
 | |        ( adjektiv,artikel,zahl,ordinalzahl| |
 | |          frageartikel, adverb )           | |
 | |        INHERENT ( ... )                   | |
 | |    ---------------------------------------| |
 | |    |                                    | | |
 | |    |   ADJAZENZ(   0 1 0 0 0 0 0 0 1    | | |
 | |    |               ................. )   | | |
 | |    ---------------------------------------| |
 | |        KOHAERENT( JA )                    | |
 | |      ....                                 | |
 | |  ENDE_MODALITAETEN                        | |
 ------------------------------------------------
 ENDE_KONZEPT
```

Bild 1.19 Konzept-Deklaration in ERNEST

In Frame TEIL 2 in Bild 1.19 folgt der Subframe

`START_MODALITAET ... ENDE_MODALITAET.`

Modalitäten sind Cluster der zur Ableitung des Konzepts K insgesamt zu betrachtenden Regeln. Bildungselement von Modalitäten sind Rollentypen.

Der Titel 'Modalität' — Möglichkeitsform — wird durch die Aufgabenstellung motiviert, die Rollen, sprich Kanten, mit Blick auf deren spätere Interpretation als Rollenträger in Pragmatik-Konzepten nach pragmatischer Relevanz klassifiziert. Dabei wird nicht an eine lineare Rangordnung der Relevanz gedacht. Relevanz ist eine

Reflexionseigenschaft[29] ausgewählter Rollen im Verhältnis aufeinander. Ein typisches Beispiel stellt die Rolle eines 'nukleus' (wie in SY_NG) im Verhältnis zu ergänzenden Rollen dar, z.B. nukleus für Nomen und Artikel als Ergänzung.

Modalitäten stellen eine Verallgemeinerung der Regelform dar. Sie verkörpern den syntagmatischen Typ eines Rollenensembles (fest bis auf Variation der Reihenfolge, Anzahl der Rollen) und den paradigmatischen Typ (fest bis auf Variation der Rollenträger). Die Syntagmenbildung kann durch Ja/Nein[30]-Matrix ADJAZENZ weiter eingeschränkt werden. Das ist eine Art von Korrelationsmatrix, die je Matrixelement die domänen-spezifische Nachbarschaft für je 2 Rollen der Modalität formalisiert.

Bild 1.19 zeigt die Relevanzmarker für Rollen innerhalb der Modalitäten: OBLIGATORISCH versus OPTIONAL. INHERENT markiert die nicht notwendig im Signal codierten, z.B. nicht-geäußerten Rollenträger, die aus Defaultannahmen rekonstruiert werden müssen.

In der Sphäre der Tier-Dominanzen sollten beispielsweise die Sätze

```
"the foxes eat goose"
    "foxes eat goose"
```

semantisch und pragmatisch gleichwertig sein, was durch eine Präzisierung der Regel (3) ausgedrückt werden kann:

```
np(X,Y) :- det(X,Z,OPTIONAL),
                  nomen(Z,Y,OBLIGATORISCH).
```

In PROLOG wird Rollenvorrang im allgemeinen lediglich durch die Reihenfolge ausgedrückt. Es müssen sämtliche Rollen gebunden werden. Der von ERNEST begründete Default-Regelinterpreter verwendet eine Metaregel, welche von einem generellen Vorrang der obligatorischen Rollen zur partiellen Ableitung des abhängigen Prädikats ausgeht.

User-Programme können diesen Vorrang durch Zusatzregeln abwandeln, was im Einzelfall, z.B. in der inkrementellen Wort-Verarbeitung, dringend geboten ist. Für die inkrementelle Verarbeitung ist der Eintrag KOHAERENT, der zur Markierung der Kontinuität/Diskontinuität von Konstituenten verwendet wird, dringend zu beachten.

Durch die Konzentration auf die Bindung obligatorischer Rollen wird die Einführung eines Mechanismus der **partiellen** *Ableitung* motiviert.

Im folgenden soll eine Übersicht zur Clusterung von Regelmengen vorgestellt werden:

TEIL 1:
Clusterung der von Y abhängigen Prädikate X1,...,Xm

- nach Hierarchie-Positionen relativ zu Y:
  BESTANDTEIL_VON, KONKRETISIERUNG_VON
- nach Positionen der Hierarchie-Ebene von Y:
  SPEZIALISIERUNG_VON

---

[29] epistemologischer Ausdruck für Eigenschaften eines Gegenstandes, die nur aus dem Verhältnis zu anderen enstehen

[30] es ist keine 0–1–Matrix, da Rollen-Anzahlen ausgedrückt werden können

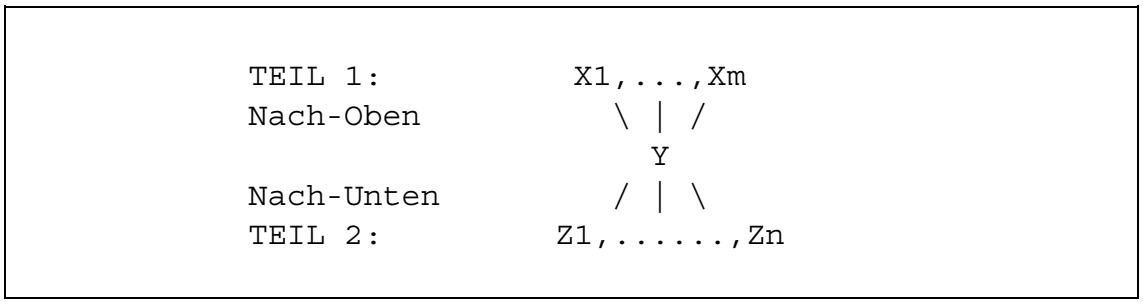


Bild 1.20 Nach-Innen-Projektion der Konzeptkanten in seinen Frame

TEIL 2:
Clusterung von Regeln
aus den Elementar-Prädikaten Z1,...,Zn

- KANTE oder ROLLE
  Cluster xor-disjunktiver Elementar-Prädikate
- MODALITAET:
  Konjunktionen der Rollenträger
- ADJAZENZ:
  Variation von Reihenfolge und Anzahl
  der Prädikate innerhalb der Modalität

1.4.2 Prinzip Faktor

Modalitäten stellen eine Anwendung des Prinzips der **Faktorisierung** von Regeln dar, das von Woods in [Woo89] (ebenfalls anhand einer PROLOG-Notation) formell eingeführt wurde.

```
p(X,Y) :- a(X,U), (m(U,V) or n(U,V)), b(V,Y).
```

Die äußere Klammerung des Mittelgliedes in

'(m() or[31] n())'

steht hier für die Rollen-Abstraktion. Die Prädikate m(), n() können stellvertretend füreinander zur Ableitung herangezogen werden. Im Falle der Nichtableitbarkeit von m() kann sofort zur Prüfung von n() fortgeschritten werden. Bei entsprechender nicht-faktorisierter Regel-Basis

```
p(X,Y) :- a(X,U),m(U,V),b(V,Y).
p(X,Y) :- a(X,U),n(U,V),b(V,Y).
```

hätte der Regelinterpreter das schon abgeleitete Prädikat a() für die zweite Regel nochmals überprüfen müssen, nachdem die erste Regel an m() gescheitert war. Faktorisierung verhindert überflüssige Aktivierungen des Mechanismus der Regel-Auswahl, was für die Effizienz der Arbeit mit massiven Regelbasen erhebliche Auswirkungen haben kann.

[31] gebräuchlich ist auch das ';'-Zeichen

Würde man in der Regelbasis des Akzeptors die Regeln (2),(3) im Sinne von 2 Modalitäten innerhalb einer neuen Regel (2') mittels der or-Klammerung zusammenfassen, so hätte man eine weitere Form von Faktorisierung vor Augen:

```
(2)  np(X,Y) :- nomen(X,Y).
(3)  np(X,Y) :- det(X,Z), nomen(Z,Y).
-------------------------------------------------
(2') np(X,Y) :-(nomen(X,Y) or (det(X,Z),nomen(Z,Y)).
```

Bild 1.21 Faktorisierung von Regeln

Beide Formen werden von ERNEST zur Anwendung gebracht werden[32].

Nimmt man also die disjunktive Verkettung unabhängiger Prädikate in die allgemeine Regelform auf und faßt mittels Rollenabstraktion und Modalitäten-Kapselung die Regeln zur Ableitung ein- und desselben Prädikats zu einer Regel zusammen, so gewinnt man die spezifische Form einer Regelbasis, der ein semantisches Netz übergestülpt wurde.

Was ein semantisches Netz dazutut, ist zunächst nichts weiter als Faktorisierung zum Zwecke der effizienteren Regelauswahl. Unter dem Aspekt der Vermeidung kombinatorischer Explosionen beim Durchsuchen großer Hypothesenräume mag das allein wichtig genug sein. Die inhaltliche Bedeutung der Netzform des Prädikationssystems wird im Kapitel-Abschnitt 'Tiefenstruktur–Parsing' betrachtet.

1.5 Kanten-Indikationen

1.5.1 Binäre Relationen

Bei der Betrachtung der allgemeinen Regelform und Ihrer Spezifikation für semantische Netze lag das Hauptaugenmerk bisher auf der Bindung der domänenspezifischen Gegenstandsvariablen durch Rollen. Rollen sind naturgemäß in binären, sprich 2–stelligen, Relationen verankert, da nur taxonomische Relationen betrachtet wurden. Insbesondere im Fall der Signalverarbeitung von Äußerungen ist es jedoch nicht sinnvoll, auf 3– und mehrstellige Relationen zu verzichten. Da Gegenstandsbildung den Aspekt des Wiedererkennens der 'richtigen Wörter aus einem Überangebot' umfaßt, hat man Anlaß genug, sich beliebiger Arten restriktiver Relationen bedienen zu können.

Den umfassenderen Problemkontext analysierte Woods 1975 in dem Artikel "What's in a link ?" [Woo75].

' If we are to mix the two notations together as in:

```
JOHN
    HEIGHT    6 FEET
    HIT       MARY
```

then we need either to provide somewhere an *indication* that these two links are of different types and therefore must be treated differently by the procedures which make inferences in the net.'

[32] die Erstgenannte findet sich explizit in der in Kapitel 3 behandelten Netzflußinformation von ERNEST

Bis zu diesem Zeitpunkt wurden z.B. folgende Kantentypen ohne Einschränkung miteinander verknüpft:

1. Realisierungen der allgemeinen Form
   attribut (entität, wert).
   attribut_anzahl_räder (fahrrad, 2).
2. Kanten mit Einbindung in ein Rollenensemble,
   z.B. hit (JOHN, MARY) im Zusammenhang mit den Rollen
   agent (JOHN), recipient (MARY).
3. Kanten des Typs 'is-a' (allgemein: Relationen taxonomisch-klassifizierenden Inhalts)
   is_a (Adler, Vogel) — in gerichteter Lesart, da Asymmetrie der Relation gilt:
   'Adler is-a Vogel'.

Indikation der Kanten verstand Woods als Typisierung der 'link names', z.B. der in 1., 2. in obenstehender Auflistung angesprochenen Kantensorten:

1. 'stand for attributes of the node', zeigt auf Knoten, der den Wert des Attributs enthält ('attribut-value'-Paare),
2. 'arbitrary relations between the node and other nodes'.

Das sind offenbar Mindestforderungen, die gestellt werden, um den methodologischen Sinn der Kanten nicht zu verwischen.

In ERNEST werden ausschließlich Indices der Fälle 2.,3. zugelassen:

Generelle taxonomische Indikation zur Unterstützung von

- Compilerfunktionen:
  Konsistenztest
- Analysefunktionen:
  Bottom-Up-Suche

und eine

spezielle Indikation des Rollen-Ensembles, der **Kontext,** zur Unterstützung kontext-abhängiger partieller Ableitungen.

Die Indikation, sprich Einbindung von Kanten in einen Rollen-Kontext mit Auswirkung auf den Mechanismus partieller Ableitung, wird im System ERNEST+ICZ systematisch für alle Kanten durchgeführt, die von Konzepten der Ebenen Semantik und Pragmatik abgehen.

Die Besonderheit der Kantenindikation in ERNEST kann folgendermaßen charakterisiert werden:

Taxonomie und Rollenkontext bestimmen *domänen-übergreifende* Indikationen
=> (problem-unabhängige) Metaregeln der Kontrolle.

Der bereits in 'What's in a link ?' problematisierte Kanten-Mix wurde von Bench-Capon speziell bezüglich der Attribut-Kanten anhand des in Bild 1.22 gezeigten Netzes thematisiert.

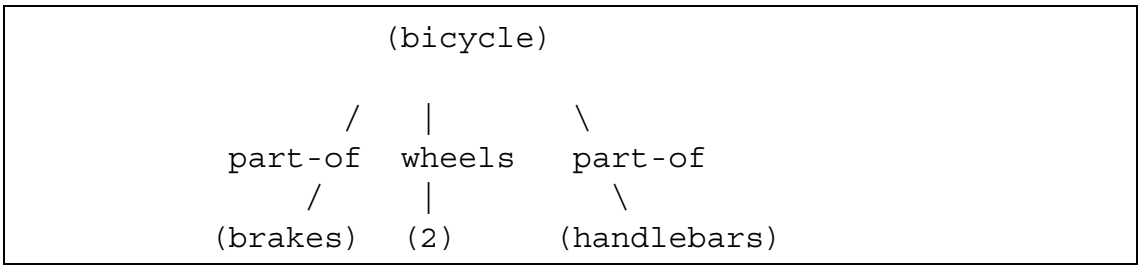


Bild 1.22 Vermischung taxonomischer und attributierter Kanten [BC90]

Die Regelfassung für das in Bild 1.22 gezeigte Subnetz zur Bestimmung des Prädikats bicycle() würde lauten (wobei Kantennamen als Indices zu Prädikaten notiert sind):

bicycle(X):- $p1_{part\text{-}of}(X)$, $anzahl_{wheels}(X,2)$, $p3_{part\text{-}of}(X)$.

Dadurch wird die allgemeine Kanten-Notation verletzt, da X im Prädikat '$anzahl_{wheels}$' als reine Attributvariable fungiert.

Die Lösung wäre in folgender Richtung zu suchen:

bicycle(X,Y,Z,..):- $wheel1_{part\text{-}of}(X)$, $wheel2_{part\text{-}of}(Y)$,... .

Der Satz-Akzeptor wird durch eine Attributvariable NUM so erweitert werden, daß der im 1. Abschnitt skizzierte Semantiktest (siehe Bild 1.2) befriedigt werden kann. Folgende Regeln werden modifiziert:

```
 s(X,Y) :- np(X,Z,NUM), vp(Z,Y,NUM).
np(X,Y,NUM) :- nomen(X,Y,NUM).
np(X,Y,NUM) :- det(X,Z), nomen(Z,Y,NUM).
vp(X,Y,NUM) :- verb(X,Z,NUM), np(Z,Y).

nomen([goose|X],X,singular).
nomen([geese|X],X,plural).
nomen([fox|X],X,singular).
nomen([foxes|X],X,plural).
verb([eats|X],X,singular).
verb([eat|X],X,plural).
```

```
1 ?- s([foxes,eat,a,goose],[]).
No
```

Bild 1.23 Bottom-Up-Propagierung von Attributwerten

Tritt die Attributvariable auf beiden Seiten der Regel auf, so vermittelt sie eine Bottom-Up–Propagierung der den lexikalischen Items zugeordneten Attributwerte {singular, plural}. In s() erzwingt die Gleichnamigkeit der Attributvariablen in np,vp den Test auf Gleichwertigkeit ihrer durch Propagierung erzeugten Belegung.

An den oben skizzierten Pragmatiktest läßt sich der Akzeptor anpassen, indem Eintragungen von Konstanten {dominant, undominant} in s(), vp() sowie in den lexikalischen Items vorgenommen und eine Attributvariable DOM zur Top–Down–Oben-nach-Unten-Propagierung eingesetzt wird.

In beiden PROLOG-Beispielen wurden Attributwerte durch alle Hierarchie-Schichten der Phrasenbildung hindurchpropagiert, d.h. derselbe Variablentyp geht aus rein prozeduralen Gründen in die Definition aller Prädikate ein, die auf den Bindungspfaden liegen.

```
 s(X,Y) :- np(X,Z,dominant), vp(Z,Y).
np(X,Y,DOM) :- nomen(X,Y,DOM).
np(X,Y,DOM) :- det(X,Z), nomen(Z,Y,DOM).
vp(X,Y) :- verb(X,Z), np(Z,Y,undominant).

nomen([goose|X],X,undominant).
nomen([geese|X],X,undominant).
nomen([fox|X],X,dominant).
nomen([foxes|X],X,dominant).
```

```
2 ?- s([geese,eat,foxes],[]).
No
```

Bild 1.24 Top-Down-Propagierung von Attributwerten

1.5.2 Sphäre der Attribute

In ERNEST lassen sich funktional 3 Schichten der Netz-Architektur unterscheiden, die allesamt innerhalb von Konzeptframes realisiert werden:

I. die Taxonomie der Konzepte

II. die Faktorisierung

III. das Constraintsystem (Attributbewertungen)

Die in Bild 1.19 begonnene Darstellung des Konzeptframes ist noch um die Subframes der Attributbeschreibung zu ergänzen. Es wird zunächst der Subframe ATTRIBUTE angelegt, wobei jedes Attribut an einen eindeutig bestimmten Rollentyp gebunden wird.

```
START_KONZEPT SY_NG
 ------------------------------------------------
 | TAXONOMIE: vert./horiz. Hierarchie nach-oben  |
 ------------------------------------------------
 | TAXONOMIE: nach-unten,  MODALITAETEN          |
 ------------------------------------------------
   START_ATTRIBUTE( 6 )
      ATTRIBUT( attributname )
         WERTETYP
         WERTEANZAHL
         RESTRIKTIONSBEREICH
         WERTEBERECHNUNG
         BEWERTUNG
         ...
   ENDE_ATTRIBUTE
 -------------------------------
   START_ANALYSEPARAMETER( 7 )
      ATTRIBUT()
      ...
   ENDE_ANALYSEPARAMETER
 -------------------------------
   START_STRUKTURRELATIONEN( 1 )
      RELATION
         BEWERTUNG
            UMKEHRFUNKTION
      ...
   ENDE_STRUKTURRELATIONEN
 ------------------------------------------------
   BEWERTUNG ( ATTRIBUT,.... )
 ENDE_KONZEPT
```

Bild 1.25 Subframe der Attributbeschreibung im Konzept-Frame

Die in Bild 1.26 gezeigte Beschreibung des Attributs *praep* im Konzept P_ANKUNFTSORT wird an die einzige, vom Konzept abgehende Kante *sem_real* (steht für 'semantische Realisierung') gebunden, was sich in der Argumentdefinition der gleichnamigen C-Funktion niederschlägt: *sem_real* . *praep*[33].

Das Attribut drückt z.B. folgenden Constraint aus:

ankunftsort(X) :- goal(X), praeposition(X) = [nach,in].

[33] in C sind die Bestandteile einer Struktur struct A mit 'A.bestandteil' zugreifbar

```
ATTRIBUT( praep )                          Erklärung
-------------------------------------------------
WERTETYP( INTEGER )                        wortnummer
WERTEANZAHL( minimal:1, maximal:1 )        kantenzahl=1
RESTRIKTIONSBEREICH( EINZELWERTE )
-------------------------------------------------
    WERT( 'nach' )                         zulässige
    WERT( 'in' )                           Präpositionen
-------------------------------------------------
WERTEBERECHNUNG( C_praep )
    ARGUMENTE( sem_real.praep )            C-Funktionen
    INVERS( C_invers_name )
BEWERTUNG ( C_name )
```

Bild 1.26 Constraint für Präpositionen im Konzept P_ANKUNFTSORT

Vererbt der ERNEST-Netzcompiler Rollentypen zwischen Konzepten, so stets auch die anhängenden Attribute.

Mehrstellige (über mehrere Rollen bestimmte) Attribute werden mit der STRUKTURRELATION erfaßt. Da Signalverarbeitung betrieben wird, benötigt man ANALYSEPARAMETER wie z.B. Signal-Anfangs-/Endpunkt eines Wortkandidaten.

Die Rollenform der Netzregeln bestimmt sich näher zu:

$p(X_1,...,X_n)$ $_{meta}$:-
$rolle_1(X_1),...,rolle_n(X_n)$,
..., $attribut_i^1(X_i)$, ..., $attribut_i^l(X_i)$,....,
..., $analyseparameter_j^1(X_i)$, ..., $analyseparameter_j^m(X_j)$,....,
..., $relation_{rs}(Arg_{rs})$,... .,
mit Teilfolge Arg_{rs} von (X_i).

Die 5 Untereinheiten des Subframes ATTRIBUT dienen der direkten Spezifikation von Datenstrukturen, die für die Codierung der Attributbewertungen in der Sprache C benötigt werden.

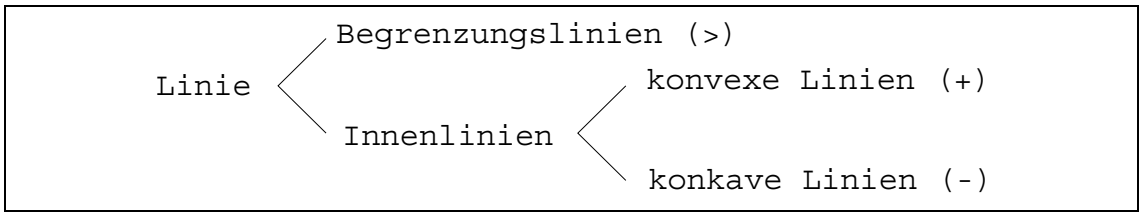


Bild 1.27 SPEZIALISIERUNGS-Hierarchie von Linien im Constraintsystem von Waltz

Es seien nur 2 Überlegungen vorgestellt, die zu einer ganz anderen allgemeinen Sicht der Methode von Waltz, als sie Richter gegeben hat, führen:

er betreibt *Gegenstandsbildung* – 'Wort'-Gruppen je Eckpunkt mit je einem Symbol je ankommender Linie:

1. Den 'physisch zulässigen'[34] Gruppen kann das Prädikat 'Körperecke' zugeschrieben werden, die betrachtete Bild-Ecke wird so zum Knotenpunkt weiterer Betrachtung.
2. Die Art der Köperecke wird durch die Beziehungen zu Nachbarecken eingeschränkt.

Waltz betrachtet dazu 2 STRUKTURRELATIONEN an den so bestimmten Knoten; in Erwiderung auf Richter – Strukturrelationen werden üblicherweise den Knoten zugeordnet, nicht den Kanten. Es sind die Relationen

1. der zulässigen Winkelkonfiguration der Linien für diese Symbolkette,
2. des übereinstimmenden Symbols der Verbindungskante von Eckenpaaren.

Verwendete ANALYSEPARAMETER sind z.B. Eckpunktkoordinaten, um die Relaxation der Markenmengen möglichst von einem Eckpunkt starten zu können, der mit (geometrisch begründeter) Sicherheit auf der äußeren Begrenzungslinie der Polyeder-Aufstellung liegt.

Waltz operiert auf einem semantischen Netz, das man um so leichter übersieht, weil es von sehr flacher Hierarchie ist.

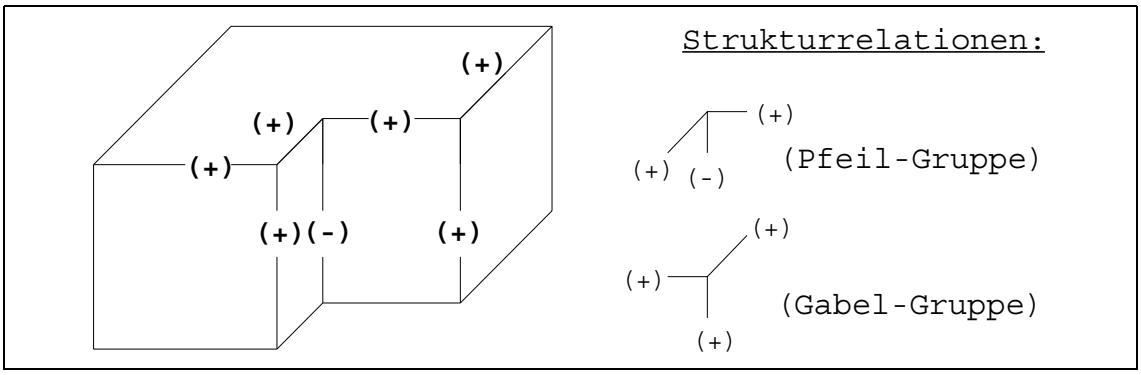


Bild 1.28 Strukturrelationen in Waltz' Constraintsystem

Trotz der fragmentarischen Analyse sollten damit die Stichworte angegeben sein, um von irgendwelchen "Dualismen" im Verhältnis von semantischen Netzen und Constraintsystemen abzulassen und eher von einem Ergänzungsverhältnis auszugehen. Relaxationsverfahren (die "Entspannung" der ursprünglichen, überproportionierten Markenmengen) tendieren zur Breitensuche nach allen für eine 'Aktivationsausbreitung' relevanten Netzknotenpunkten, während Suche im semantischen Prädikationsnetz zur Tiefensuche strebt. Das könnte sich aber als bloßer Indikator für das Wesentliche der Komplementarität erweisen.

Das Relaxationsverfahren von Waltz muß meines Erachtens bezüglich seines Konvergenzverhaltens[35] in anderen Anwendungen überprüft werden.

Constraintsysteme betrachten Attribute mit möglichst lokal begrenztem Zugriff von einem Knoten zu den nächsten Nachbarknoten. Bild 1.29 zeigt die Grenzen des Definitionsbereichs für Attribute und Strukturrelationen in ERNEST.

---

[34] das wird durch einen Symbolkatalog beschrieben

[35] nach Mackworth hängt es von sehr problem-spezifischen Annahmen ab [Mac76]

Durch die Forderungen seiner Konzept–Deklaration erzwingt ERNEST Spezifikationen, die der Erzeuger einer PROLOG-Regelbasis durch Reihenfolgebestimmung der Regeln[36] sowie Verwendung von ODER- und Constraint-Formeln eigenständig optional vornehmen kann. Der Vorteil der Anwendung von PROLOG ist ein geschlossener Regeltext, der von demselben Compiler übersetzt werden kann. Der PROLOG-Compiler ist eine Art Universalcompiler für eine Wissensbasis. Letztere kann im Einzelfall vollständig auf ein Semantisches Netz abbildbar sein. Der Compiler wurde jedoch so allgemein gehalten, daß er auch konventionelle Datenbankprobleme lösen kann. Taxonomische Organisation, die Woods für die Wissensverarbeitung heraushebt, wird nicht explizit unterstützt, was auf zusätzliche interne Abarbeitungsschritte des Regelinterpreters zur hierarchie-treuen Regelauswahl hinausläuft.

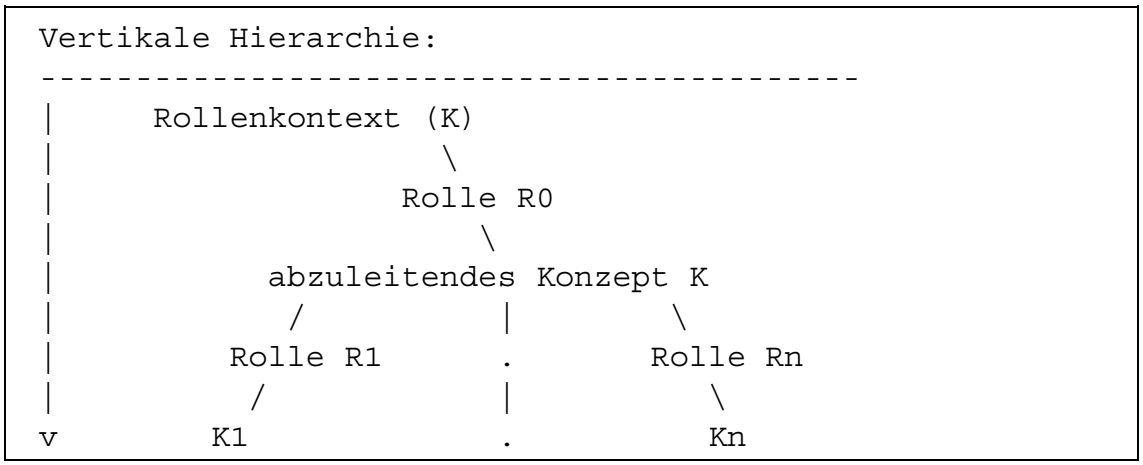


Bild 1.29 Grenzen des Zugriffsbereichs für Attribut-Berechnungsfunktionen in ERNEST

Im Falle ERNEST handelt es sich um einen Spezialcompiler für Regelbasen, denen eine Taxonomie übergestülpt wird und deren Attributwert-Propagierung. Die Entscheidung für Universal- versus Spezial-Compilierung stellt eine Crux der internationalen Software-Entwicklung dar. Ein Idealweg scheint nicht gefunden.

ERNEST verwendet einen Spezialcompiler[37] für die Binär-Codierung der Deklarationen (Schicht I,II) und C-Codierung/Compilierung für separate Prozeduren (Schicht III und Kontrolle). Der unselbständige Binärcode wird bislang durch die (in C) prozedural formulierte Kontrolle eingelesen und verarbeitet, so daß letzlich auch hier ein Universalcompiler die Oberhand bekommt. Damit sind gewichtige Voraussetzungen für die Gestaltung effizient und portabel programmierter Applikationslösungen gegeben. Andererseits geht dabei etwas verloren. Die Abhandlung beschreibt die vollständige Umgestaltung der Kontrolle einer gegebenen ERNEST-Applikation.

Der Anwender des PROLOG-Satzakzeptors hat immerhin die deklarative Notation der Ziel-Formeln zur Verfügung, so kryptisch sie auch sein mag. Damit wird ein breiter Kreis von Nutzern eingeladen. Die derzeitige Implementation von ERNEST priorisiert die durchgängige Anwendung der Systemprogrammiersprache C/C++, auch für rein deklarativ faßbare Inhalte. Das schränkt den Anwenderkreis erheblich ein. In Kapitel 3 wird der Ansatzpunkt eines Auswegs aus diesem Dilemma gesucht.

---

[36] z.B. die hierarchietreue Anordnung der Akzeptorregeln

[37] siehe frühere Fußnote zum Compiler-Begriff

1.6 Tiefenstruktur-Parser

1.6.1 Strukturbeschreibung: ’Konzeptgraph’

In semantischen Netzen lassen sich Regeln bündeln, indem man sie unter Suchbegriffen (Konzepten) zusammenfaßt. Das zu lösende Retrieval-Problem lautet nun nicht mehr:

welche Regel wende ich an?

sondern

welches Konzept will ich zuordnen?.

In derselben Arbeit, in der Woods in [Woo89] das ’basic principle of **factoring**’ formulierte, gibt er dem so vollzogenen Sichtwechsel eine weitere Zuspitzung:

’conceptually factored, taxonomic representation systems such as KL-ONE appear to be well suited to such applications. *Such knowledge structures can be used to perform a kind of abstract “parsing” of a situation, using the patterns and schemata of the knowledge base as a “grammar”’*.

Woods betätigte sich als Neuerer auf dem Gebiet der Frage-Antwort-Systeme [Woo79]. Für seine Netzwerkvorstellung hat man zuallererst die erweiterten, rekursiven Übergangsnetze (RTN)[38] zu betrachten. Sie sollen schlaglichtartig betrachtet werden:

Rekursive Übergangsnetze sind Grammatiken.

Man nehme die Regel connect() in Bild1.30. Die geklammerten unabhängigen Prädikate bilden je ein RTN. Formal sind das Finite-State-Automaten. Aufgrund der Statezahl 2 sind sie dazu bestimmt, Wortketten der Länge 2 zu matchen. Es geht darum, die Variablen der pragmatischen Konzepte *flight(), place()* unmittelbar durch Wortkonstanten zu binden. Den ziemlich direkten Zugriff zur Wortebene mittels Matching bezeichnet man als *Konzept-Spotting*.

Die zu erkennende SITUATION in Satz ’AA-57 fly from Boston to Chicago’ besteht aus der Zerlegung des Satzes in die kontinuierlichen Wortketten G1,2,3 durch das Matching.

```
connect(G11,G22,G32)
:-(flight(G11), G12= fly),
  (G21= from, place(G22)),
  (G31= to,   place(G32)).

SITUATION:
G1=[AA-57,fly],G2=[from,Boston],G3=[to,Chicago]
=>
AKTION = connect(AA-57,Boston,Chicago)
```

Bild 1.30 Situation-Aktion-Regel für Konzept–Spotting, [Pal75]

Um Wood’s Verständnis des Terminus ’Aktion’ zu belegen, sei eine weitere Aktion angedeutet:

```
list(G_departure_time, G11,G22).
```

[38] <engl.> Recursive Transition Network, v. Woods eingeführt[Woo70b]

Die Aktion list() soll die für eine konventionelle Datenbankanfrage benötigten Substrings extrahieren und gegebenenfalls re-interpretieren. Solche Arbeiten bilden auch in der Applikation ICZ die Voraussetzung, um z.B. nach der Zuordnung von Kalendertagen zu verbalen Zeitbestimmungen auf eine Datenbank des Bundesbahn-Fahrplans zugreifen zu können [Hil93a].

Die aus Übergangsnetzen kombinierte Regel hat die Form:

```
Situation => Aktion
```

und dient dazu, die wichtigsten pragmatischen Bestimmungsstücke der Satzhandlung herauszufiltern und zu einem Gegenstand [x,y,z] mit der Bedeutung 'Flug-Abflugort-Ankunftsort' zusammenzufassen.

Die RTN-Modellierung stellt eine spezielle Möglichkeit dar, die pragmatisch-bestimmte Tiefenstruktur des Satzes freizulegen. Man kann sie als Vorläufer eines **Tiefenstruktur-Parser** bezeichnen.

Die Ableitungsmaschine für die erweiterten, rekursiven Übergangsnetze kann ebenfalls nach dem Prinzip der UP-Maschine (vergleiche die oben zitierte Definition nach[Aho86]) realisiert werden.

ERNEST realisiert den Tiefenstruktur-Parser noch konsequenter. Die einzelnen Entwicklungsstadien der Strukturbeschreibung und vor allem die partiellen Ableitungsgrade der Konzepte werden transparent gemacht.

Die Analyse der Strukturbeschreibung selbst wird thematisiert und für den Zugriff des Users offengelegt. Deshalb sollte es erlaubt sein, zu sagen:

```
ERNEST-Netzwerke sind  Meta-Grammatiken.
```

Die ERNEST-Strukturbeschreibung bestimmt durchgängig den Grad der partiellen Ableitung von Konzepten. Die 2 gravierendsten Ableitungszustände für ein Konzept K werden durch folgende Zustandsmarker gespeichert:

```
M(K): unvollständig (partiell) abgeleitet,
I(K):   vollständig abgeleitet
```

Die Terminologie für Zustandsmarker schließt sich Gebräuchen aus dem Gebiet der Frame- und Constraintsysteme an. Mit dem *Instantiieren* meint man dort das Binden von freien Variablen oder Rollen. Parsing mit ERNEST-Netzen hat es wesentlich mit der Betrachtung von Ableitungsgraden, d.h. der Rollenbindung, zu tun. Man beachte den Unterschied zur Redeweise der objekt-orientierten Programmierung, die '*Instanzen*' auf Basis eines vorgegebenen Klassensystems bestimmt und damit Klassen-Individuen meint. Die wohlunterschiedene Fassung beider Aspekte ist um so nötiger, insofern beide in die Strukturbeschreibung eingehen:

Netzwerk-Konzept	Strukturbeschreibung
---------------->	---------------------------
Klasse	Instanz
Schema	Individuum
Typ	Einzelfall der Zuschreibung

Bild 1.31 OOP- und Netzwerk-Begriffe

Instanzen als Strukturmarker in ERNEST drücken beides aus:

```
  Instanzbildung  (Vererbung des Klassenschemas)
+  Instantiierung (vollständige Bindung)
```

Die folgenden Markersymbole werden ebenso zur Kennzeichnung des Ableitungsgrads der Marker als auch der von ihnen ausgedrückten Prädikation verwendet. Ihnen werden die bisher in der ERNEST-Literatur gewählten Termini zugeordnet:

```
M(): partiell  abgeleiteter Strukturmarker
       -> modifiziertes Konzept
I(): vollständig  abgeleiteter Strukturmarker
       -> Instanz
```

Ich schlage die Sprachregelung vor, nach der das semantische Netz die Typen der Konzepte definiert, während die Strukturbeschreibung die zum Typ gehörigen Individuen-Exemplare registriert. Sie heiße demzufolge **Konzeptgraph**.

Verwendet wird ab sofort die *'Sprechweise über Konzepte'*:
Ist von 'Konzepten' schlechthin die Rede, so sind stets die Strukturmarker gemeint, d.h. es geht um die abkürzende Rede über Zuschreibungen von Konzepten.

Die Schema-Funktion der Netzwerk-Konzepte wird in der Implementation physisch realisiert. Neben den Speicherräumen für das Netz, die Konzeptgraphen und den Suchbaum wird ein Look–Up–Tabellenspeicher zur Generierung der Individual-Schemata angelegt. Letztere werden parallel zum Eintrag der M()-Marker angelegt. Der Tabellenindex wird als Nummer des Individualschemas im Marker verankert (in Bild 1.35 weggelassen).

Die Strukturbeschreibung gewinnt in ERNEST den Rang einer Steuerungsanweisung für die Analyse. Zum Vergleich sei an die Einflußlosigkeit der dem Satz-Akzeptor zugehörigen Strukturbeschreibung erinnert. Bisher findet man aber in der ERNEST-Literatur keine geschlossene Thematisierung eines *Konzeptgraph-Analysers*. Durch die Aufgabenstellung der inkrementellen Verarbeitung rückt die Subzielbestimmung und die Bestimmung von SITUATION-AKTION-Regeln in den Vordergrund. Das Hauptaugenmerk generalisierender Darstellungen zur ERNEST-Kontrolle konzentrierte sich bislang auf den A*-Algorithmus zur Suchbaum-Kontrolle und eine Metabasis von 6 (problem-unabhängigen) Regeln zur Expansion von Konzeptgraphen.

1.6.2 Subziele zum Analyseziel

Die Strukturbeschreibung unterstützt die situationsgerechte Bestimmung von Subzielen und die Möglichkeit quasi-parallel an verschiedenen Subzielen zu arbeiten.

```
Subziele werden als Strukturmarker angegeben.
```

M() soll immer auch für die rein hypothetische Zusprechung eines Prädikats stehen, vergleichbar der 'Ziel-Anfrage' in PROLOG. Subziele werden formuliert, um die Ableitung für ein *Wurzel-Prädikat*[39] zu steuern.

Subziel-Definition im Konzeptgraphen:

Es sei ($\{M_i(), I_j() \mid (i=1,..,n;\ j=1,...,m)\}$, $\{k\}$)
der aktuell betrachtete Konzeptgraph von n+m Strukturmarkern
und einer Menge {k} von Verbindungskanten, die ausschließlich der

[39] im Satzakzeptor ist das s()

Organisation der Vorgänger/Nachfolger-Beziehungen der Marker (homolog zur Taxonomie des ERNEST-Netzes) dient.

Ein Marker M(), der mit mindestens einem anderen Marker verbunden ist, heiße ein *offenes Ziel*.
Es wird zugelassen, der Knotenmenge des Konzeptgraphen isolierte, unverbundene Marker M(K) für beliebige Netzkonzepte K hinzuzufügen, um dadurch ein weiteres Subziel — es heiße das **Neuziel** — vorzugeben.

Der letzte Satz artikuliert einen Aspekt der Userkontrolle, der in der ERNEST-Literatur bisher ausgeklammert wird:

Soll es dem User gestattet sein, sozusagen 'von Hand' Neuziele zu erzeugen und damit die Analyse des Konzeptgraphen zu steuern?

Interessant ist auch, unter welchen Umständen und für welche Konzepte dies sinnvoll ist. Um diese Fragen zu klären, muß man zunächst die allgemein akzeptierten Elemente der Kontrolle zusammenfassen.

Die Ableitung der Subziele erzeugt eine Reihenfolge von Konzeptgraphen, die durch die Beziehung EXPANDIERUNG_VON() zu ordnen ist. Die Interdependenzen der Konzeptgraphen werden zum Gegenstand einer übergeordneten Suchbaumkontrolle. Konzeptgraphen erhalten Knotenidentifikatoren in der natürlichen Folge Ihrer Entstehung. Man vergleiche die an den Kanten des Suchbaums in Bild 1.35 angetragenen Nummern. Die Entfaltung des Suchbaumes wird außer durch den Ableitungsgrad der Konzepte durch sogenannte *Aufspaltungsgründe* [Pre89] sowie die vergleichbaren, numerischen Bewertungen der Konzeptgraphen differenziert. Diese seien zunächst betrachtet.

1.6.3 Exkurs: Richtschnur Tiefensuche

Ein Exkurs soll der Besprechung des Grundprinzips der Suche im ERNEST-Basisschema vorausgeschickt werden. Hinter Tiefensuchverfahren verbirgt sich stets der Determinismus der UP-Maschine. ERNEST macht Vorschläge zur Flexiblisierung der Tiefensuche. Einen Teil der größeren Variabilität macht die veränderte Handhabung der in Tiefensuchverfahren im Allgemeinen verwendeten **OFFEN-Liste** aus. Beim Abstieg in die Tiefe hat man sich aller ausgelassenen Varianten zu merken, zu denen er im Fehlerfall zurückkehren muß. Sie kommt in jedem graphentheoretischen Buch vor, das sich mit rein deterministischen[40] Suchproblemen beschäftigt.

Es folgt eine kurze Betrachtung zum Einsatz von Tiefensuche und OFFEN-Liste in anderen Sachgebieten. In den eigenen Arbeiten zur Low-Level-Bilderkennung [Kov86], speziell der Regionen-Segmentierung, wurde ein Verfahren zur pixelweisen Markierung mit je einem einheitlichen Label benötigt. Es leitet den Übergang zur symbolischen Verarbeitung der extrahierten Bildregionen ein. Theoretisch und praktisch schien alles gelöst:

A. Von Seiten der Graphentheorie wird mit Argumenten aus der Theorie der Berechenbarkeit/Komplexität [Hop88] genau ein Algorithmus priorisiert:
   die Markierung in Tiefensuche nach einem adjazenten Regionenpixel[41] in einem

[40] ich verwende 'deterministisch' etwas frei in dem Sinne: alles geht nach einem Basisprinzip.

[41] das Kriterium der Pixel-ADJAZENZ hat besonders Kovalevski in [Kov84] problematisiert

region-inneren Pfad von einem Startpunkt aus. Die Restnachbarn des jeweiligen Pfadpunktes kommen in die OFFEN-Liste. Dem Verfahren wird die Komplexität O(n) mit Pixelzahl n pro Region ausgewiesen – Begründung: jeder Label wird nach dem Tremaux-Verfahren genau einmal geschrieben [Eve79] (S.59 f.).

B. Von Seiten der Computergrafik, die aus Polygonzügen Rasterbilder künstlich erzeugt, wurden für das Polygon-Filling diverse Techniken verfügbar gemacht.

Trotzdem erwiesen sich beide Ansätze für die Belange der Bilderkennung als ergänzungsbedürftig:

1. Die Startpunktmenge ist nicht vorauszusetzen, sondern erst zu finden.
2. Anzuwendende Lookahead-Verfahren der Startpunktsuche können gleichzeitig für die Erkennung des Regionenzusammenhangs aus elementar markierbaren Teilgebieten eingesetzt werden (Problem der Verzahnung von Aufgaben).
3. Wünschenswert ist ein inkrementelles Labeling, das beim Durchlaufen des Bildes von Region zu Region springt statt gefundene Regionen “in einem Stück” zu verarbeiten. Das erscheint mir analog zur partiellen Ableitung von Konstituenten. (Problem des “Malens an vielen Stellen eines Gemäldes”).
4. Die Parallelisierbarkeit der Problemstellung wurde durch A. wie B. nicht behandelt [Hir79].

In einem von Kovalevski angeregten Vortrag am Zentralinstitut für Kybernetik und Informationsprozesse der ostdeutschen Akademie der Wissenschaften habe ich 1986 die reellen Aufwendungen (die zur Gesamtlösung benötigten Kontrollmittel insgesamt) verglichen. Ich konnte zeigen, daß der im Sinne der Komplexitätstheorie beste Algorithmus höhere Aufwendungen[42] verlangt als eine von Kovalevski und mir vorgeschlagene Innovation. Zu verweisen ist auch auf die Diskussion der Konnektivitäts–Eigenschaft von Regionen, topologisch gesprochen: Gebieten, als nicht durch ein Perzeptron detektierbar – nachzulesen im “Anti-Perzeptron”-Buch von Minsky/Papert [Min69]. Das scheinbar triviale Problem sollte Anlaß geben, die theoretisch begründete Kontrolle (via Graphentheorie, via Linguistik) und die Aufgaben–orientierte Kontrolle gleichberechtigt zu behandeln. Offenbar geht es nicht um den Vorrang einer Seite, sondern um ein Ergänzungsverhältnis. Das scheint mir Marr in seiner Skizze einer Epistemologie der Informatik [Mar82] gemeint zu haben, wo er zwischen Basis-Theorien und Implementationen das Vermittlungsglied der ’*computational theory*’ einführt. Interessanterweise exemplifiziert er die Differenz ’formal vs. computational theory’ gerade im Gebiet der Sprachverarbeitung: anhand der Grammatik-Theorie Chomskys [Cho65] und dessen Ergänzung durch Marcus [Mar80]. Ich kenne bisher keine adäquate deutsche Benennung des Mittelglieds, es sei denn, man begnügt sich mit dem Begriff der Berechenbarkeit aus der theoretischen Informatik.

Ähnliche Schwierigkeiten der Einordnung treten zu Begriff und Methode des *Backtracking* auf, je nachdem welche Seite des Ergänzungsverhältnisses man vorher eingenommen hat. Ich habe im Abschnitt “Sphäre der Attribute” (im Zusammenhang des Constraining-Verfahrens von Waltz) einen Artikel von Mackworth [Mac76] besprochen, der Backtracking als Versuch-Irrtum-Verfahren in der Fehlerbehandlung kritisiert. Bildhaft gesprochen, wird es mit “Trashing Behavior”[43] gleichgesetzt. Gerade die

[42] in diesem Sachgebiet zählt man Lese/Schreiboperationen auf Bildern

[43] Planloses Verhalten

inkrementelle Verarbeitung braucht eine “Rückverfolgung” zu sicheren Aufsetzpunkten in der Aktivationsorder der Knoten. PROLOG-Backtracking basiert auf der UP-Maschine. Sie hat den eindeutig bestimmten Rückkehrfaden auf dem Kontrollstack vorliegen (siehe Abschnitt “Expandierungen der UP-Maschine”).

1.6.4 Bewertung

Konzeptgraphen werden im Suchbaum registriert und bekommen aufgrund ihrer Gesamtbewertung einen Platz in der Rangordnung der OFFEN-Liste. Den Konzepttypen der Strukturmarker des Graphen sind in der Netzdeklaration Attibutbeschreibungen zugeordnet. Der Slot **BEWERTUNG** setzt innerhalb der Attributbeschreibung den Endslot des Frameaufbaus (siehe Beispiel in Bild 1.26 in Abschnitt 1.5 von Kapitel 1). Er umfaßt ein Tupel aller für eine Gesamt-Konzeptbewertung als maßgeblich zugewiesenen Attributbewertungen. Die mit jeder Rollenbindung für ein Konzept K berechneten Bewertungen der Attribut-Constraints der Rolle werden im Individualschema von M() gespeichert. Will man die Vorgaben der Wissensbasis absolut setzen, so müßte für den Zusammenhang der Einzelbewertungen die folgende Verfahrensweise gelten:
Übertragungsprinzip *– ergibt nur eine der in der Attributbeschreibung verlangten Einzelbewertungen den Wert* UNZULAESSIG*, so wird er auf die Gesamtbewertung übertragen.*

PROLOG-Kenner, die einen Überblick zu ERNEST zu gewinnen suchen, stellen die Frage, wie sich *Ableitung und Bewertung* miteinander “verschwägern” lassen. Diese Frage stellt sich inbesondere auch betreffs der Fortschreibung des Übergang:

```
 Menge der Konzept-Bewertungen
—> Gesamtbewertung des Konzeptgraphen.
```

Setzt man die UP-Maschine [Aho86] (und ihren strengen Determinismus) voraus, so ist in der Tat fraglich, inwiefern Bewertung, also eine Methodik, die mit Graden und Abstufungen umgeht, stattfinden kann. Es wäre kein Problem vorhanden, würde man die scharfe JA/NEIN-Entscheidung des Übertragungsprinzips der Konzeptbewertung auch für den Konzeptgraphen insgesamt geltend machen. Aber ein solcher Determinismus würde z.B. die in ERNEST gemachte Unterscheidung obligatorischer vs. optionaler Bestandteile außer acht lassen. Im Sinne der Robustheit der Anwendung liefert diese Unterscheidung einen Fingerzeig zur Codierung der problem-abhängigen Bewertungsfunktion für Konzeptgraphen. Dem Mechanismus der Bewertung von Konzeptgraphen ist im Basisschema der Kontrolle nur ein Platzhalter zugeordnet, über dessen Inhalt die Anwendung entscheidet. Sie verfügt auch über die Wahl der

Abweichung vom Übertragungsprinzip (für Konzeptgraphen):

UNZULAESSIGE Strukturmarker, die für die Modalitäten der beiden zentralen Konzepte der Steuerung, Kontext und Auskunftstyp, nicht obligatorisch sind, werden

1. vom Übertragungsprinzip ausgenommen,
2. ihre noch zulässigen Bestandteile werden, sofern sie situationsbedingt als Prior in Frage kommen, auf UNZULAESSIG gesetzt.

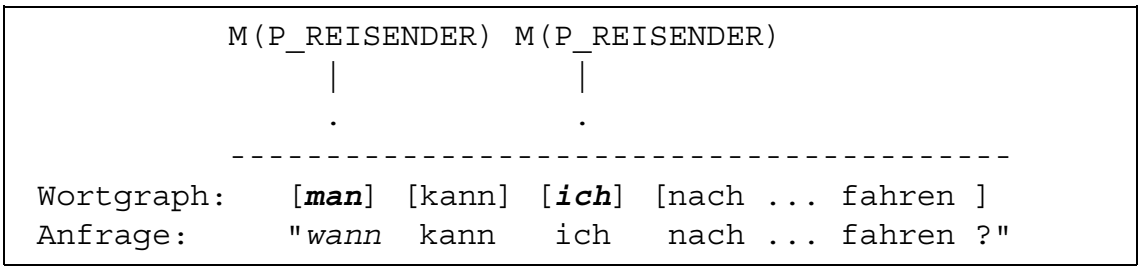


Bild 1.32 Wettbewerb pragmatischer Rollenträger

Zurück zur Frage der Anwendung/Einschränkung des Übertragungsprinzips. Strikte Anwendung würde dazu führen, daß der Konzeptgraph als UNZULAESSIG aus der OFFEN-Liste entfernt wird. Aber in Annahme der Möglichkeit, daß eine Fehldetektion im Wortgraph die gleiche Bewertungssituation in den Nachfolgerknoten der OFFEN-Liste reproduzieren könnte, ist das Verfahren prophylaktisch auch für obligatorische Bestandteile anzuwenden. In Bild 1.32 wird ein weiterer Fall notwendiger Abweichung vom Übertragungsprinzip vorgelegt:

Durch eine Fehlererkennung im Wortgraphen (Auftreten des Wortkandidaten [man] statt des gesprochenen “wann”) und infolge noch zu beschreibender situationsbedingter Kontrollmaßnahmen (mit Bezug zum Verbrahmen für [fahren]) können die beiden Pronomen nur zur Aufwärtsableitung mit identischem Konzepttyp P_REISENDER führen. Handlungsschemata mit 2 Reisenden sind aber nicht vorgesehen. Mindestens einer dieser beiden Marker ist sozusagen “von Hand” (nicht infolge der Attribut-Bewertungen) UNZULAESSIG zu setzen, wobei noch ganz offen ist, welcher in Frage kommt. Ersichtlich haben beide Varianten sogar identische pragmatische Interpretation. Um eine allgemeingültige Lösung unter Einsatz des Ableitungsmechanismus zu erreichen, empfiehlt es sich hier, “von Hand” eine Suchbaumaufspaltung vorzunehmen: Verdopplung des Konzeptgraphen von Bild 1.32 und Setzung je genau eines der beiden konzepttyp-gleichen Marker in je einem Duplikat auf UNZULAESSIG. Weiter unten folgt noch eine Metaregel zur Behandlung solcher Fälle. Im obigen Beispiel würde die Expansion beider Pfade der so initiierten Suchbaumaufspaltungen zum Erfolg kommen. Trotzdem verspricht die Wahl der inkrementell später erzeugten pragmatischen Bestimmung einen Vorteil: die Voraussicht auf Versprecher und Repetitionen des Sprechers als mögliche natürliche Ursache der Verdopplung. Der Effekt einer Metaregel (der problem-abhängigen Steuerung) zur Bevorzugung des gemeinten Suchbaumknoten kann schon dadurch erreicht werden, daß er an die höchste Stelle der ranggeordneten OFFEN-Liste gesetzt und im nächsten Expansionsschritt weiterentwickelt wird. Vor der näheren Betrachtung solcher Expansionen ist zunächst das Grundprinzip der Suche in ERNEST-basierten Anwendungen zu erklären.

Je Anwendung ergeben sich weitere Problemsituationen des Typs der doppelten pragmatischen Bestimmungen, z.B. die Verdoppelung des Verbrahmen-Markers. Sie müssen durch *globale Bewertungsfunktionen* behandelt werden. Diese sind global, weil sie Zugriff auf die Gesamtstruktur des Konzeptgraphen benötigen. Die von ERNEST geforderten, auf begrenztem Zugriffsraum definierten Attribut-Berechnungsfunktionen sind nicht-global (vergleiche Bild 1.29).

Für die Zwecke der Erkennung der falsifizierenden Wirkung dieser Funktionen innerhalb einer experimentellen Usershell habe ich sie mit einer Protokollierung versehen.

Suchbaumknoten als Datenstrukturen enthalten noch mehr Informationen als den zugeordneten Konzeptgraphen. Durch Verzeigerungen im Arbeits-, sprich Instanzen-Speicher der Analyse besteht ein eindeutiger Bezug auf die assoziierte Wortkette. Auf diese richtet sich ein anderer Typ globaler Bewertungsfunktionen, die *akustischen* Bewertungfunktionen. Für derartige Bewertungen stehen in der ERNEST-internen Framestruktur für Suchbaumknoten je 5 ganzzahlige Slots (Typ int) und 5 reellwertige Slots (Typ float) zur Verfügung.

int-Wert 3	linguistische Gesamtbewertung des Konzeptgraphen (Wert ZULAESSIG/ UNZULAESSIG)
int-Wert 5	linguistische Vorrangsteuerung (zum Erzwingen einer TOP-of-Queue Stellung in OFFEN)
float-Wert 1	akustische Pfadbewertung der assoziierten Kette
int-Wert 1	reale Überdeckung (Summe von einzelnen Wortkandidaten überdeckt. Intervalle)
int-Wert 2	SKIP-Überdeckung (Anfangsframe - Endframe d. Kette)
int-Wert 4	Aufgabenkontrolle (Reihenfolgen von Suchbaumaufspaltungen)

Bild 1.33 Bewertungstabelle des besten Knoten der OFFEN-Liste

Die in der ICZ-Anwendung von Kummert verwendeten Parameter
— maximal langes Intervall einer Subkette der assoziierten Wortkette,
— minimale Anzahl von deren Subketten
sind im Zusammenhang der SKIP-Verarbeitung nicht mehr sinnvoll. Bezüglich int-Maß 3 UNZULAESSIGE Knoten kommen nicht in die OFFEN-Liste. Sieht man von diesem Parameter ab, so zeigt die Tabelle von oben beginnend die Reihenfolge beim stellenweisen Vergleich der Bewertungstupel von Suchbaumknoten an.

Zusätzlich zu den int-Maßen (quasi als 6. int-Wert) wird auch die beim Knotenvergleich stets verfügbare Knotennummer ausgewertet, von der vorausgesetzt wird, daß sie im Proporz zur Zeitordnung vergeben wird. Sie drückt also das frühere/spätere Auftreten des Knoten auf der Expansionsleiter aus. Damit wird verhindert, daß durch ein formales Prinzip der Einsortierung in die linear-gelinkte OFFEN-Liste ein in allen anderen Stelligkeiten gleichbewerteter Knoten früherer Expansionsstufe bei der Auswahl des aktiven Knoten vorgezogen werden kann.

An oberster Stelle der Tabelle steht die Bewertung der linguistischen ZULAESSIGKEIT. Für den eigentlichen Knotenvergleich zur Bestimmung des besten Knoten der OFFEN-Liste werden nur die folgenden 5 Maße benötigt. UNZULAESSIGE Knoten entferne ich vor der Bestenbestimung aus OFFEN. Bei der Bestimmung des “Besseren” von je zwei vorgelegten Suchbaumknoten werden die Maße der Tabelle in Bild 1.33 zeilenweise, von oben beginnend, berechnet. Der zeilenweise Vergleich stoppt bei der ersten Ungleichheit. An die erste Stelle des Vergleichs war

zwingend ein Steuerungsparameter zu plazieren. Das in Bild 1.34 gezeigte Beispiel soll die Notwendigkeit verdeutlichen. In der aktuellen Konzeptgraph-Expansion sei mit der Neuziel-Setzung M(H_WORTHYP) der nächste Inkrementierungschritt vorbereitet worden. Als aktuell auszuwählende Wortkandidaten seien [Berlin] und [man] in je eigenen Suchbaumknoten als Instantiierungen des Neuziels gegeben. Die akustische Qualität, sprich Pfadbewertung, der assoziierten Wortkette für [man] sei besser als die für [Berlin].

Auch an die letzte Stelle habe ich ein kontrollbestimmtes Vergleichsmaß gesetzt. Damit kann man zwischen ansonsten gleichbewerteten Knoten differenzieren und die Reihenfolge der Bearbeitung von Suchbaumaufspaltungen beeinflussen.

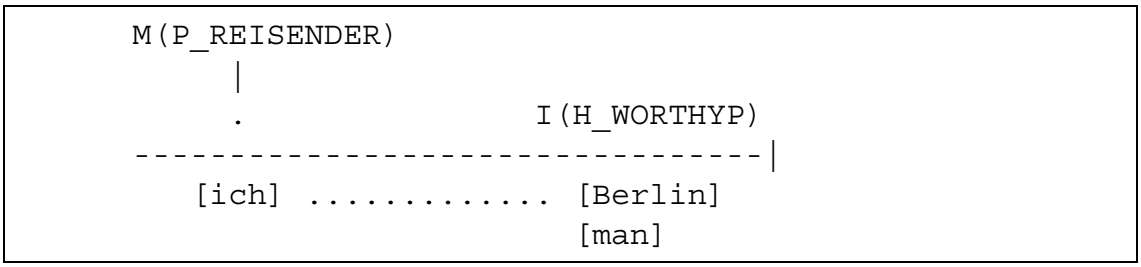


Bild 1.34 Wettbewerb von Wortkandidaten

Zur Unterscheidung der pragmatischen Relevanz der lexikalischen Einträge sei eine domänenspezifisch ausgerichtete Boolsche Testfunktion gegeben, die z.B. dem Lexem [Berlin] angesichts der ICZ-Domäne (Stadt mit Intercity-Bahnhof) den Wert TRUE, dem Lexem [man] aber den Wert FALSE zuordnet. In diesem Fall möchte man durch eine Setzung des Steuerparameters (int-Wert 5) erzwingen, daß der Konzeptgraph mit I(H_WORTHYP, [Berlin]) zuerst expandiert wird, obwohl er die schlechtere akustische Bewertung besitzt:

> User-Regel (Bewertung) 'linguistische/akustische Bewertung':
> linguistische Prioritäten stehen höher als akustische Bewertungen.

Auf die Beziehung von SKIP-Überdeckung und realer Überdeckung werden Testfunktionen zur Bestimmung des globalen Instantiierungsgrads des Konzeptgraphen angewandt. Bei positiver Differenz werden sogenannte RESKIP-Versuche gestartet. Es wird versucht, die in einer Liste gesammelten Wortkandidaten, die wegen linguistischer Mehrdeutigkeit und anderer Gründe übersprungen wurden, nachträglich aufwärts abzuleiten. Die reale Überdeckung besitzt erst dann die größere Aussagekraft, wenn die inkrementelle Verarbeitung den Signalendpunkt erreicht hat, so daß diese Maße deutlich situationsabhängig auszuwerten sind.

1.6.5 A*-Suche

Der explosiven Zunahme von Suchbaumknoten versucht eine übergeordnete Pozedur der ERNEST-Kontrolle entgegenzuwirken, indem sie auf den **optimalen Weg** der Knotenexpansion hinsteuert. Aus der Theorie der Suche ist bekannt, daß Breitensuche als auch Tiefensuche nicht hinreichend sind. Deshalb wird der A*-Algorithmus als allgemeines Prinzip der (Suchbaum-)Kontrolle eingesetzt.

Eine allgemeinverständliche Erläuterung des Prinzips gibt Winston:

’**Die Prozedur A*** ist eine Verzweige-und-Begrenze-Suche mit einer Schätzung der verbleibenden Entfernung, kombiniert mit dem Prinzip der dynamischen Programmierung.
Wenn die Schätzung der Restentfernung ein unterer Grenzwert der tatsächlichen Entfernung ist, dann ergeben sich aus A optimale Lösungen.’*[Win87b], (S. 130).

Winston diskutiert (auf S.130 f.) die Frage, ob auch die in A* kombinierten Einzelverfahren in bestimmten Anwendungen zum Optimum führen können. Sie wäre an anderer Stelle weiterzuverfolgen. Vor allem geht es um die Realisierung von A* unter den Bedingungen inkrementeller Verarbeitung, d.h. die Restschätzung ohne Kenntnis eines Signalrestabschnitts.

Der Suchbaum besteht aus Teilwegen, die jeweils durch ihren zuletzt expandierten *offenen Knoten* vertreten werden können. Sie bilden in ERNEST die OFFEN-Liste. Da die Suchbaumknoten stets bewertet werden, bevor sie in der OFFEN-Liste abgelegt werden, ist es insbesondere für den interaktiven Testbetrieb ratsam nur die zulässigen Knoten dorthin zu bringen. Das Übertragungsprinzip, welches die Gesamtbewertung des Suchbaumknoten, sprich Konzeptgraphen, aus den Bewertungen seiner Strukturmarker nach dem Maßstab ’Alles-oder-Nichts’ abliest, liefert zumindest eine hinreichende Bedingung für den Begriff des *zulässigen Konzeptgraphen*. Sind einzelne Marker als linguistisch UNZULAESSIG bewertet, so fordert der für Anwendungszwecke offene Bewertungsmechanismus von ERNEST nicht notwendig, den Tatbestand auf die Gesamtbewertung zu übertragen. Durch geeignete Wahl der für Steuerungsaufgaben freigehaltenen Einzelmaße des Bewertungsvektors ist dann zu gewährleisten, daß in der OFFEN-Liste gelegene Knoten geringeren Entwicklungsgrades überhaupt noch zur Auswahl für den nächsten Expansionsschritt kommen können.

Ein zulässiger Konzeptgraph heiße eine *Expansion des optimalen Weges*, wenn der zugehörige Suchbaumknoten offen ist und kein anderer Knoten existiert, der bezüglich des Werttupels

(Restmaß, $Maß_2$,...., $Maß_n$)

im komponentenweisen Vergleich besser ist.

ERNEST strebt die Expansion eines optimalen Weges entlang von Konzeptgraphen an. Statt des Backtrackings wie in PROLOG[44] greift der ERNEST-Netzinterpreter bei Wertung UNZULAESSIG für den besten offenen Konzeptgraphen stets auf das nächste Element der nach Bewertung geordneten OFFEN-Liste zurück.
Suchbaumaufspaltung ist eine Konsequenz der ERNEST-Modellierung:

`Modalitäten erzwingen Aufspaltung.`

Soll die aktuelle Strukturbeschreibung um ein offenes Ziel oder ein Neuziel M(K) erweitert werden und hat K mehrere, nämlich n Modalitäten, so ist der Suchbaum aufzuspalten und mit je einem Konzeptgraphen je Modalität anzulegen. Es werden

[44] Kritiken des Prinzips resümiert Mackworth in[Mac76]

n Individual–Schemata zu M(K) angelegt und die jeweilige Modalität darin vermerkt. Bild 1.35 bietet das Gegenstück der Akzeptor-Analyse (siehe Bild 1.7) für das Ziel:

s([the, fox, eats, goose],[]).

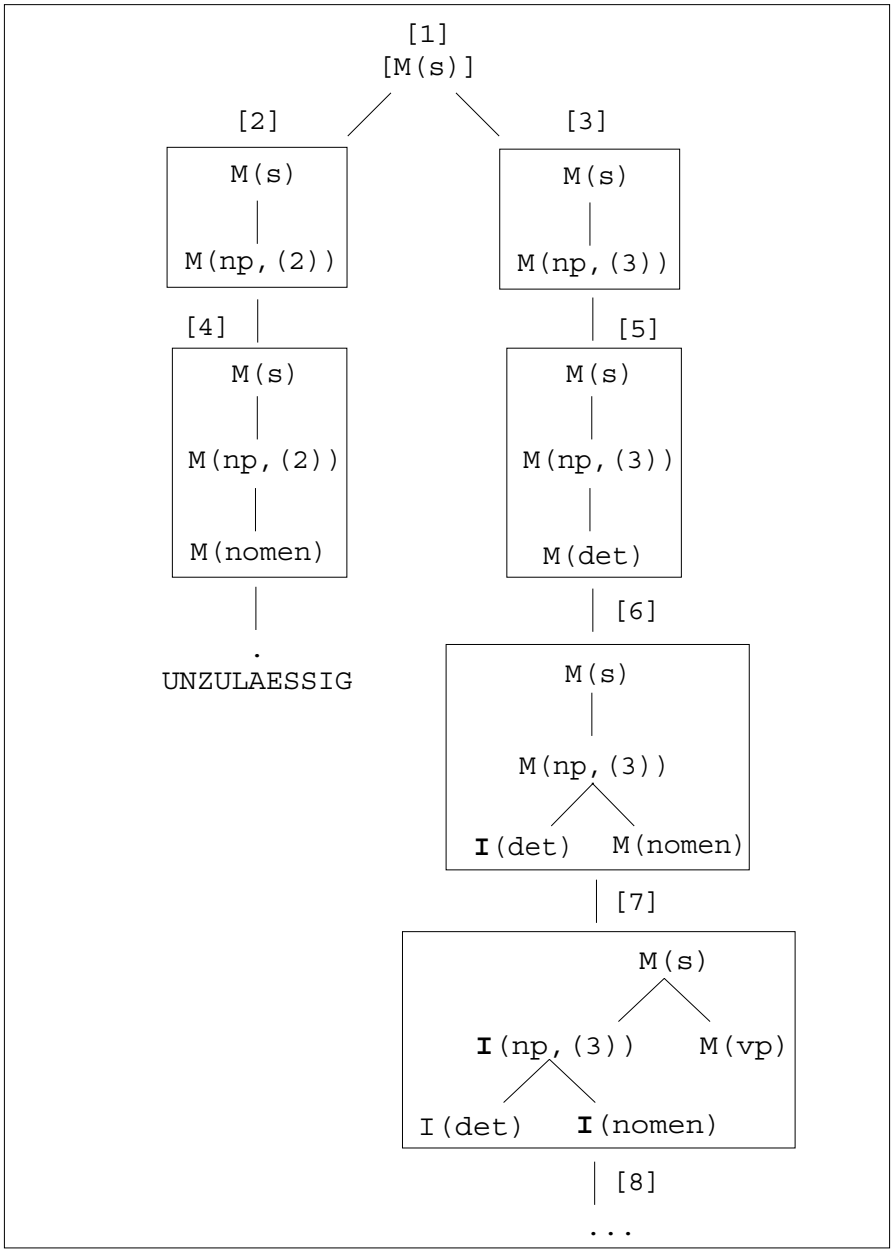


Bild 1.35 Suchbaum–Expansion

Für das Prädikat np werden automatisch Suchbaumknoten 2 und 3 entsprechend der Modalitätenzahl angelegt. Zuerst wird Knoten 2 expandiert, No.3 wird in OFFEN gemerkt. Der Suchweg endet in einem unzulässigen Knoten, weil die Gegenstandsvariable des Prädikats nomen nicht gebunden werden kann. Die A*-Maschine greift automatisch auf Knoten 3 zurück.

Ein anderer typischer Aufspaltungsgrund entsteht auf der Signalebene. M(det) in Knoten 5 wird zu I(det) in Knoten 6, weil ein Element der Wortkette eingebunden wurde. Im allgemeinen Fall wird die Signalebene durch einen Wortgraphen oder eine Menge von Wortketten vertreten sein, so daß für die Bindung konkurrierende Wortkandidaten

bereit stehen. Für jedes legt der ERNEST-Netzinterpreter einen I()-Marker in je einem eigenen Konzeptgraph an:

`Instantiierung auf Signalebene erzwingt Aufspaltung.`

I(K) steht für die vollständige Ableitung oder Instantiierung von K in einem Konzeptgraph. Zur Instantiierung auf Signalebene ist genau eine Rolle zu binden. Für den allgemeinen Fall sieht ERNEST eine spezielle Auffassung von *Vollständigkeit* der Ableitung vor.

ERNEST-Auffassung der vollständigen Ableitung:

Ein Strukturmarker M(K) des Konzepttyps K heiße im Konzeptgraphen *vollständig* (oder instantiiert),
falls alle zu M(K,Modalität) gehörigen obligatorischen Rollen gebunden sind.
Das wird durch den Beschreiber I() anstelle von M() festgehalten.
Weitere Bindung optionaler Rollen führt lediglich zur Erweiterung der Beschreibung I() im Arbeitsspeicher und zu einer Veränderung von Einzelmaßen des Bewertungsvektors von K.

Bild 1.35 stellt die Startphase der Entwicklung des **Suchbaumes** in Parallele zur Ableitung mittels UP-Maschine in Bild 1.7 dar. Im Bild stehen die I()-Marker für eine im buchstäblichen Sinne 'vollständige' Ableitung, wie sie der PROLOG-Interpreter verlangt. In Knoten 7 werden alle Rollen des in 6 noch offenen Ziels M(np, Modalität 3) gebunden, so daß es zur Instantiierung kommt. Das Beispiel soll nur zeigen, daß man die PROLOG-Ableitung auch mittels ERNEST-Ableitung adäquat durchführen kann. In ERNEST bieten sich aufgrund des formal schwächeren Ableitungsprinzips weitere Varianten an.

Generell wird die Suchbaumexpansion so durchgeführt, daß spätere Konzeptgraphen ihre Vorgänger sozusagen als "Klones" enthalten. Beim 'Opentracking' (dem Pendant zum Backtracking) findet man nützliche Vorstufen des verworfenen Graphen gleich mit vor. Die Wiederwendung von Vorstufen, auch von echten Fragmenten jüngerer Konzeptgraphen, sei als *Klonierung* angemerkt.

Die gewachsene Steuerungsfunktion der Strukturbeschreibung in ERNEST zeigt sich an der Aufdeckung verschiedener Aktivationspunkte im Konzeptgraphen, so daß gewisse Abarbeitungsvarianten zu verfolgen sind.

Bei Vorliegen von gleichrangigen Konzepten befragt die Prozedur den sogenannten KANTENVORRANG, der in den Kantenbeschreibungen der Konzeptframes deklariert werden kann.

1.6.6 Regelform: SITUATION –> AKTION

Wurde der Aktivierungspunkt gewählt, so ergibt sich die **AKTION** im ERNEST-Basisschema der Kontrolle bislang als eine den Ableitungsgrad des (partiell) abgeleiteten Ziels erweiternde Expandierung. ERNEST bietet für jeden Konzeptgraphen als Aktivationspunkt das **Prior**–Konzept an, ermittelt durch eine Standardprozedur, die sich von folgender Gewichtung der offenen Ziele leiten läßt:

1. `durch Signaldaten verifizierbare Konzepte`

2. a) Rollenkontexte
   b) Pragmatik-Konzepte
   -> Aufdeckung der Tiefenstruktur

Bild 1.36 problematisiert in verallgemeinernder Form die Bestimmung des Aktionspunkts in Knoten 5 des abgebildeten Suchbaums.

Muß die Expandierung des Konzepts b automatisch der von a vorgezogen werden?

Dafür gibt es auch nach dem Verzicht auf den UP-Automatismus einen Grund. Konzept b ist signalnah und kann unmittelbar durch Datenbindung verifiziert/falsifiziert werden.

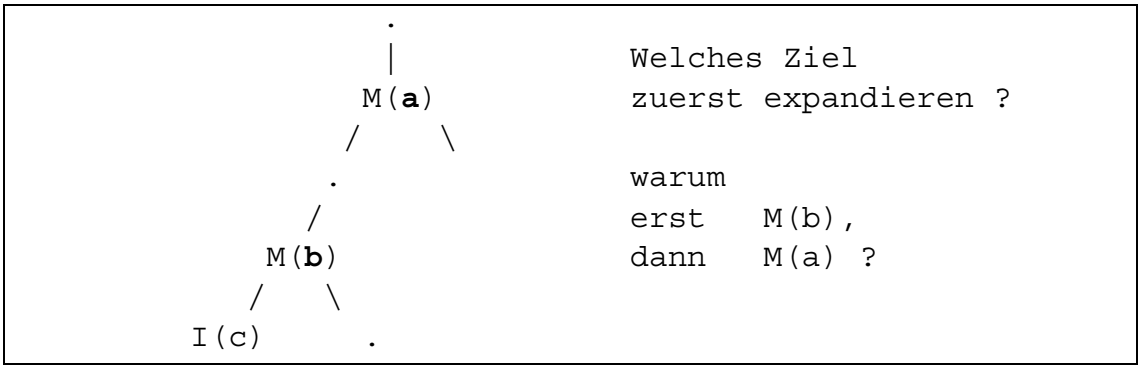


Bild 1.36 Wettbewerb der Aktivationspunkte im Konzeptgraph

Die Prior-Bestimmung entscheidet sich eindeutig für M(b). Aber angenommen, Konzept b hat den Typ NICHT–KOHAERENT[45]. Will man die inkrementelle Verarbeitung durchhalten, so kommt man nicht umhin, den automatischen Vorrang signalnaher Konzepte abzusetzen. Das gilt allgemein für die Konzepte, deren AKTION, sprich Expandierung, die Bindung diskontinuierlicher Wortketten erfordern würde. Gleiche Einschränkungen gelten für den a priori, also situationsentbunden definierten Kantenvorrang[46].

Der vorgelegten Problemlösung wurde deshalb ein verändertes Grundschema der ERNEST-Kontrolle zugrunde gelegt. Die Veränderung ermöglicht es dem User signalnahe Neuziele aufzubauen, die Ableitung streng daten-orientiert, inkrementell und in einem 'wait-and-see'-Modus zu halten.

Im 2. Kapitel wird der erweiterte Spielraum des Users und das Problem der User-Kontrolle generell thematisiert.

1.7 Zusammenfassung: ERNEST-Prädikationssysteme

Im ERNEST-Netz werden ein Prädikatensystem und Knotensystem in beiden Richtungen eindeutig aufeinander abgebildet. Die taxonomische Organisation der Prädikate bildet die oberste oder die Netzwerk-Schicht der Wissensbasis. Die Liste der Regeln je Prädikat wird im Knoten eingekapselt. Die Einzelregeln innerhalb einer Knoten-Definition werden als *Modalitäten* bezeichnet. Modalitäten sind Cluster von Regeln,

[45] b ist 'nicht-notwendig kontinuierlich'

[46] er ist v.a. bei primär modell-getriebenen Applikationen sinnvoll

deren Variation durch Adjazenzen gegeben wird. Durch Definition von Modalitäten wird unmittelbar ein nicht-formaler Mechanismus der partiellen Ableitung vorbereitet, indem Modalitäten auch die Funktion haben, den Stellenwert eines Elementarprädikats für die vollständige Ableitung eines abhängigen Prädikats zu taxieren (obligatorisch/optional).

Mit ERNEST werden Prädikationssysteme gebaut. Prädikation setzt einen mitlaufenden Prozeß der Gegenstandsbildung voraus, der selbst ein formeller Prozeß ist. Linguistische Datenverarbeitung geht von Wörtern und damit von Entitäten aus, die im Verhältnis zu ihrer Signalgestalt bereits höchst-abstrakte Form haben. Derartige Abstrakta sind kombinierbar zu neuen Abstrakta, zu Substrings. Substrings sind spezielle Gegenstände, sie haben z.B. eine Länge, der von Computerlinguisten Einfluß auf den Parsing-Prozeß als auch auf das "Verstehen" von Äußerungen gegeben wird (L.Frazier/J.Fodor [Fra78]).

Linguistische Erkennungs-/Verstehensprozesse sind zur Zeit besser definiert als viele Aufgaben der Bildverarbeitung. Bei Letzteren erweist sich die Form der Gegenstandsbildung und die Arten von Handlungsschemata der Aufgabenstellung als schwer zugänglich. Die Bildverarbeitung setzt bei weitaus signalnäheren Abstrakta an, deren Kombinierung zu maschinell verarbeitungsfähigen formalen Gegenständen problematisch bleibt. Ein Verb kann unmittelbar ein Handlungsschema reflektieren, den Seh-Gegenständen steht ihre Handlungsbedeutung nicht "auf der Stirn" geschrieben. Aufgrund experimenteller Erfahrungen in der Bildverarbeitung [Kov86] und Kenntnis wahrnehmungspsychologischer Untersuchungen [Leo73], [Hol75] scheint mir Vorsicht betreffs der Übertragung von Prinzipien der linguistischen Datenverarbeitung auf den Bildbereich geboten. Eine Unterscheidung von Prädikation/Gegenstandsbildung innerhalb der Bildverarbeitung bereitet Probleme. Nur eine Andeutung sei erlaubt. Wörter sind selbst Substrings, einelementig aufgefaßt. Die computer-linguistische Gegenstandsbildung hat in den Wortketten ihre abstrakte Basis. Bild-Primitiva, die zu den handlungsrelevanten[47] Sehgegenständen führen können, sind nicht auf eine einheitliche Form reduzierbar.

Bei Beachtung der genannten Vorbehalte: die Gegenstandsbildung macht selbst einen Teil des Erkennungsprozesses aus. Deshalb unterscheidet ERNEST Prädikate von bloßen Attributen. Erstere sollen der Identifikation des abstrakten Handlungsschemas der Erkennungsaufgabe dienen. Letztere dienen dazu, die Gegenstandsbildung einem zusätzlichen Constraining-Prozeß zu unterziehen. Er ist um so nötiger im Fall konkurrierender Mengen von Ausgangsabstrakta, z.B. Wortkandidaten.[48] Um den Unterschied von Prädikation und Attribution auch sprachlich festzuhalten, kommt der von sonstigem kognitiven Beigeschmack gereinigte Terminus 'Konzept' gerade recht und wird so zum Vehikel des Prädikationsschemas.

Um die maschinelle Verarbeitung zu organisieren, ist ein Prozeß der gezielten Ableitung von Teilergebnissen anzuleiten. Die Ableitungspotenz semantischer Netze ist bislang weidlich unterschätzt worden und sie spielt, diese Aussage sei gewagt, auch in der Literatur über ERNEST nicht die geziemende Rolle. Eine erste Ahnung dieses Mankos hat zu einigen kleineren Abweichungen von anderen ERNEST-Darstellungen in [See92] geführt, indem versucht wurde, die Analyse von Äußerungen auf ein systematisches Zusammenspiel elementarer Kontrollprozeduren zurückzuführen. Aber

---

47 '*Handlung*' ist hier soziologisch gemeint, i.S. von Rollenhandeln, so auch in pragmatischen Rollen

48 'Wortkandidat' wird anstelle von 'Worthypothese' verwandt, weil alle Prädikationen hypothetisch sind

auch in diesem Text werden die Kontrollelemente weder auf allgemeine Ableitungsmechanismen zurückgeführt, noch wird ihr Zusammenhang zur Form des Netzaufbaus herausgearbeitet.

Retrieval von Information ist das Geschäft vieler Systeme, die das Etikett 'semantisches Netz' tragen. Wie im Hypertext, der womöglich prototypisch für eine Spielart semantischerNetze ist, werden Suchfäden ausgelegt [Müh94]. So unterstützen auch die Netzfäden von ERNEST zunächst die regel-gestützte Ableitung durch Bestimmung von Metaregeln und durch Regelclusterung. Insofern mag es erlaubt sein, die oberste Ebene der ERNEST-Netze als Hypertexte zu verstehen, welche die Regelauswahl der Maschine beschleunigen. Aber das Hauptgeschäft ist Ableitung und Gegenstandsbildung. Mit experimentellen Untersuchungen ist zu prüfen, ob die statische, vorcompilierte Information der Wissensbasis ausreicht, die Signalinterpretation vor kombinatorischer Explosion zu bewahren.

Deren Steueranweisungen sollten idealerweise regelhaft codierbar sein. Die Zulassung der User-Eingriffe hängt systematisch mit dem erstmalig von ERNEST gegangenen Weg der partiellen Regel-Ableitung und der Einbeziehung der Strukturbeschreibung in den Analyseprozeß zusammen. Die Konzeption der Implementation ist offen, auf die Ergänzung durch Userprogramme angelegt. Es wäre nicht richtig, der Userprogrammierung den gesamten problemabhängigen Teil der Kontrolle zuzuordnen. ERNEST entwickelt die Möglichkeit, mit problem-**un**abhängigen Kontrollregeln das deklarierte Wissen des semantischen Netzes prozedural in problem**ab**hängiges Kontrollwissen "übersetzen"zu können. Der näheren Untersuchung beider Anteile der problemabhängigen Kontrolle dient das nächste Kapitel.

Kapitel 2 Basis-Kontrolle und Aufgaben-Kontrolle

Das 1.Kapitel hat allgemein-methodologische Aspekte der für ERNEST typischen Beziehung von Wissensbasis und Suchsteuerung herausgestellt. Im 2.Kapitel rückt nun die Frage der inkrementellen Steuerung[1] des sprachverstehenden Systems ICZ in den Mittelpunkt.

Die Detailbeschreibung der funktionsfähigen Steuerung im 3. Kapitel wird vorbereitet. Die bisher im Basis-Schema der Kontrolle eingebauten und von mir erweiterten Userfunktionen besitzen eine linguistische Motivierung, die in Form von 'User-Regeln' (eine Art Metaregel) ausgesprochen werden. Den ständigen Gebrauch des bedenklichen Ausdrucks 'User' bitte ich zu entschuldigen und als reines Unterscheidungsmittel für zwei Funktionsbereiche zu betrachten. Wie die Schlußbemerkung des letzten Kapitels zeigt, fällt eine allesumfassende inhaltliche Abgrenzung nicht leicht. Als Orientierung mag das Leitmotto dieses Kapitels dienen: die ***Basiskontrolle*** überprüft die formelle linguistische Konsistenz der gefundenen Satz-Interpretation gemäß der Wissensbasis, die ***Aufgabenkontrolle*** überprüft Entwicklungsstand und Fortschritt der Interpretation. Sie führt Fehler- und Endebehandlung durch.

Die Lösung ICZ soll als Beispiellösung für weitere ereignisgesteuerte Anwendungen von ERNEST kenntlich werden.

2.1 Inkrementelle Wortgraphen

2.1.1 Struktur von Wortgraphen

Ansatzpunkt der inkrementellen Verarbeitung ist in der linguistischen Domäne ein Eingabestrom von Wortkandidaten. Den Strom kann man als Linie von Zeitereignissen auffassen. 'Zeit' macht sich hierbei erstens als Grenze kenntlich:

was ist *bis zu diesem Zeitpunkt* aus dem Signal ableitbar ?

Zweitens geht es um den 'Zeitpunkt', das aktuelle Ereignis, auf das spezifisch zu reagieren ist:

was ist *aus genau diesem Zeitpunkt, sprich Ereignis,* ableitbar ?

Zur Behandlung des zweiten Aspekts benötigt die Maschine eine Segmentierung des Zeitstromes in diskrete Einheiten. Im Fall des Sprachsignals sind das Signalintervalle, bei einer Bildfolge das Einzelbild. Bei der Einzelbild-Analyse ist die ereignis-orientierte Betrachtungsweise ebenfalls relevant, doch kann darauf im Sinne der Bemerkung in Abschnitt 1.7 nicht eingegangen werden.

Eine sehr gebräuchliche Form der Intervall-Segmentierung des Sprachsignals ist der **Wortgraph**. Seine Kanten gelten als Entsprechungen der Signalintervalle. Eine Menge ***W*** der von einem Worterkenner gelieferten Wortkandidaten wird an den Kanten notiert.

Von der Menge W muß im allgemeinen angenommen werden, daß sie die faktisch vom Sprecher verwendete Wörtermenge $\mathbf{W_S}$ nicht vollständig umfaßt. Trotzdem kann man sich in vielen Fällen das Ziel stellen, wenigstens die Grundbestandteile der

[1] 'Steuerung', 'Kontrolle', 'Analyse' werden synonym gebraucht

(pragmatischen) Tiefenstruktur der Sprecheräußerung – gegebenenfalls unvollständig – zu entdecken. Da es sich um die Gestaltung eines Dialogsystems, also insbesondere um die Generierung von Frage-Antwort-Zyklen handelt, ergeben sich daraus gerade die gezielten Rückfragen. Außerdem wird von vornherein ein relativ freies Sprecherverhalten vorausgesetzt. Läßt der Sprecher in der Erstäußerung Elemente der Tiefenstruktur aus, so hat das System die Aufgabe, das Defizit zu entdecken und gezielt zurückzufragen.

Wie sieht nun ein Wortgraph im Detail aus ? Ein Sprachsignal wird mit einem bestimmten Zeitraster diskretisiert. Das Einheitsintervall der Diskretisierung heiße ein *Frame* (10 ms). Jeder Wortkandidat ist einem Zeitintervall zugeordnet und durch Frameanfangs– und Frameend-Nummer definiert. Mit dem Terminus Wort-**Kandidat** soll immer schon mitgemeint sein, daß ein Wort als eines von mehreren möglichen Konkurrenten für ein Signalintervall auftritt.

Im Normalfall werden Sprechpausen mitdetektiert. Die Menge *W* sei diesbezüglich eingeschränkt. Durch eine Vorverarbeitung seien bereits alle Pausen-Zeitabschnitte und – Bewertungen mit passenden lexikalischen Hypothesen verschmolzen und damit zum Verschwinden gebracht. Die passenden Hypothesen werden gemäß einer *Nachbarschafts*–Relation bestimmt. 2 Fälle sind zu unterscheiden:

(1) die *Knoten-zentrierte* Darstellung:

Die Menge *W* wird als Kantenmenge eines Wortgraphen gewählt, indem den Endpunkten der segmentierten Zeitintervalle eine Knotennumerierung zugeordnet wird. Dabei können gleichen Framenummern differerierende Knotennummern zugeordnet sein. Daraus resultieren dann einander ausschließende Verbindungspfade entlang von Wortkanten durch den Graphen wie in Bild 2.1 dargestellt.

Das linksbündig stehende Spaltenpaar verkörpert die Knotennumerierung der Kanten. Die Listenstruktur des Wortgraphen wird als (im allgemeinen sortierte) Aufzählung der Kantenmenge gegeben. Die Kantenliste in Bild 2.1 wurde mit der Sortierindexfolge

1. Anfangsknotennummer, 2. Endknotennummer

hergestellt. Ebensogut hätte man über die im rechtsbündigen Spaltenpaar angegebenen Framenummern sortieren können. Besonders wichtig ist die nicht-normierte, nicht-kumulierte sogenannte ’akustische Qualität’ des Wortkandidaten.

Die Listenform entspricht einer im BMFT-Verbundprojekt “ASL Verbmobil” standardisierten Form. Da es innerhalb des Projektes geteilte Zuständigkeiten von Projektpartnern ausschließlich für Worterkennung einerseits und linguistische Interpretation andererseits gab und diese als modulare Softwarekomponenten implementiert wurden, diente das Textformat der Liste als Fileinterface zwischen den Modulen.

Knoten			Qualität	Frames	
BEGINN_WORTGRAPH					
0	1	ich	20.50	2	10
1	2	moechte	35.47	10	40
2	3	morgen	51.02	40	73
2	6	einen	32.65	40	73
3	4	frueh	16.89	73	103
4	5	in	17.40	103	118
5	9	Einsiedel	85.89	118	184
6	7	Zug	37.87	73	103
7	8	nach	19.74	103	118
8	9	Dortmund	90.01	118	184
9	10	ankommen	41.22	184	220
ENDE_WORTGRAPH					

Bild 2.1 Listenstruktur des Wortgraphen

Entsprechend der Knotennumerierung wird in Bild 2.2 eine 2D-Anordnung des Wortgraphen aufgebaut.

Wortgraphen erleichtern den Zugriff auf die Nachbarschaftsrelation. Die Nachbarschaft von Wortkanten kann durch bloßen Vergleich einer End- mit einer Anfangs-Knotennummer bestimmt werden und es ist wird erleichtert, Pfade zu verfolgen. Bild 2.1 zeigt eine vereinfachte Darstellung von Pfaden. Im Normalfall gehören zu jeder Kante mehrere Wortkandidaten.

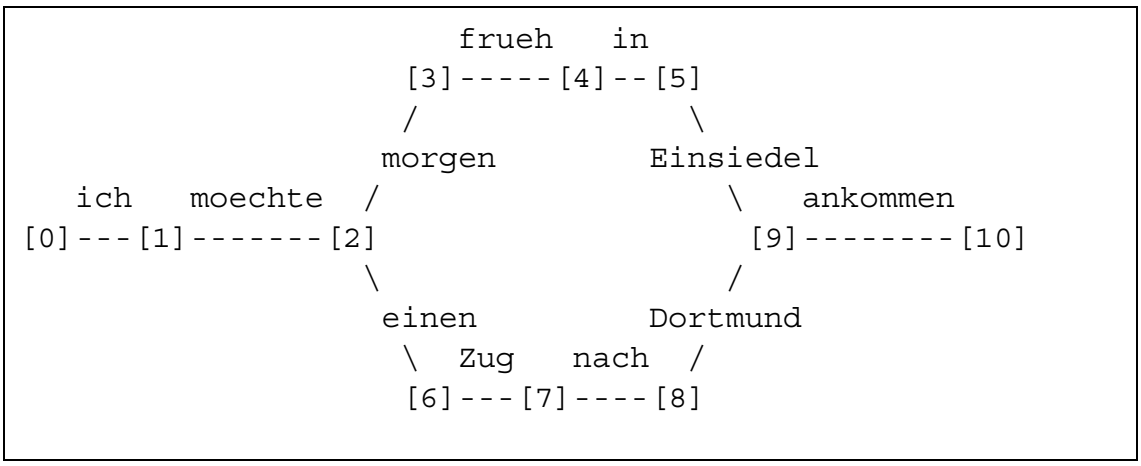


Bild 2.2 Wortgraph in 2D

(2) die *frame-überlappende* Darstellung:

Die einfachere Form des Erkenner-Resultats verzichtet auf die Knotennumerierung und eine damit verbundene Verrechnung der Framebegrenzungen des Zeitintervalls der Hypothesen. Die Nachbarschaftsdefinition wird abgeschwächt und läßt eine gewisse Anzahl gemeinsamer (überlappender) Frames je Paar Wortkandidaten zu. Wie im Fall (1) werden die Bewertungen der Wörter in einem additiven Sinn verstanden, so daß sich Bewertungen von Pfaden und Teilpfaden benachbarter Kandidaten sinnvoll aus der Summe ihrer Einzelglieder berechnen lassen.

In beiden Darstellungen muß jeweils bekannt sein, wie das Verhältnis von numerischer Größe und akustischer Qualität eines Wortkandidaten zu sehen ist. Es ist durchaus üblich, negative Zahlenwerte infolge von Logarithmierung zu verwenden oder ein umgekehrt proportionales Verhältnis von Zahl- und Qualitätsgröße zu verwenden. Allgemein ist die vom Erkenner berechnete Qualität nicht als verläßlicher Absolutmaßstab anzunehmen. Kurze Stringlänge eines Wortes hat im allgemeinen einen Bewertungsbonus (*Problem der kurzen Wörter*). Manche Erkenner flechten aus Wortpaarhäufigkeiten[2] berechnete Multiplikatoren ein, was sich in der linguistischen Suche als nachteilig erweisen kann und besser herauszurechnen wäre.

Mit Blick auf die Graphtypen (1), (2) ist eine weitere Grundannahme der Verarbeitung von Wortgraphen zu nennen. Vom Mengendurchschnitt der Wörtermenge W_S des Sprechers mit der Menge W wird *nicht* notwendig verlangt:

Typ (1): er sei Pfad-zusammenhängend,
Typ (2): er liege geschlossen in einer der n-besten Ketten.

Für die Applikation wurden Wortgraphen der in [Oer93], [Kuh94], [Rau93] beschriebenen Erkenner getestet. Bezogen auf diese ist der Graph in Bild 2.2 als 'worst case' aufzufassen . Pfade werden in diesem Fall nicht nur durch konkurrierende Wortkandidaten aufgespannt. Sie drücken die Beschränkung aus, gewisse, im Sinne von Typ (2) benachbarte, Kandidaten nicht verketten zu dürfen. Die Wortkandidaten innerhalb des obengelegenen Teilpfads über Knoten [4] werden von der Verkettung mit sämtlichen Gliedern des unteren Teilpfads über Knoten [7] ausgeschlossen und vice versa.

Statt einer mathematisch-exakten Definition der von beiden Pfaden eingeschlossenen 'Masche' ist an eine operationale Definition der Abprüfung der Suchbeschränkung gedacht. Der Anhang "Wortgraphen-Interface" erläutert die eingeführte Zusatzfunktion, welche es ermöglicht die 'Maschen'-Information zur Beschränkung der Hypothesensuche auszunutzen.

Der häufigere Fall der Konkurrenz von Pfaden im Wortgraphen entsteht, wie untenstehend dargestellt, dadurch, daß derselbe Wortstring als Kandidat für verschiedene, einander überlappende Intervalle auftritt. Das kann man sich zumeist als Resultat sogenannter Einfügungen/Löschungen (engl. Insertions/Deletions) – je nachdem, welchen Pfad man als Meßlatte hernimmt – erklären, die im Suchverhalten des Erkenners begründet sind.

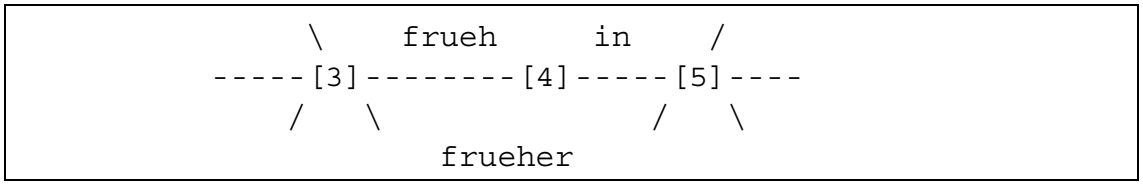


Bild 2.3 Insertions/Deletions im Wortgraph

Für die Unterscheidung der 2 Sorten Wortgraphen gibt es keine einschlägigen Termini. Den Letzteren kann man im wesentlichen als Einzelpfad-Typ charakterisieren,

[2] solche Paare heißen Bigramme, ebenso werden Tri-Gramme betrachtet

bei dem sich sozusagen ein Geflecht von Pfaden (eine ’Ranke’) um eine virtuelle Hauptachse windet.

Für den Graphen mit Frame-Überlappung wird im allgemeinen die Rekombination der analysierten Wortkette aus Gliedern verschiedener Anfangsketten als zulässig betrachtet. Das System ICZ wurde bislang auf diesen Typ ausgerichtet. Durch meine Vorarbeiten können jetzt wahlweise beide Graphentypen behandelt werden (Anhang “Wortgraphen-Interface”).

Der **inkrementelle Wortgraph** kann nach den bisherigen Erfahrungen gewisse strukturelle Unterschiede zum vollständigen Wortgraphen aufweisen. Im Allgemeinen kann nicht vorausgesetzt werden, daß alle Pfade an genau einem Endknoten zusammenlaufen, noch daß alle Endknoten von Pfaden die bis auf Überlappungstoleranz gleiche Framenummer haben müssen:

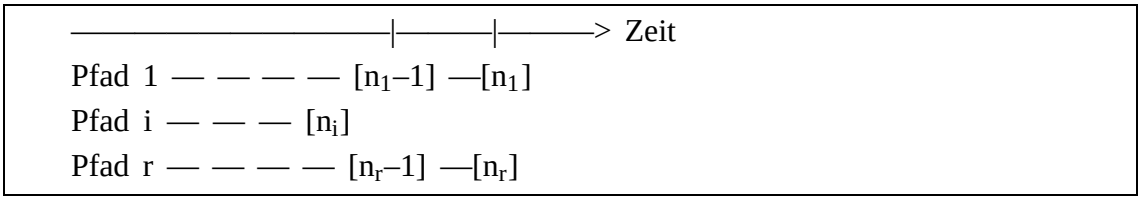


Bild 2.4 Endsegment im Wortgraphen

Bei der zeitsynchronen Kopplung der Einheiten Erkennung/Interpretation ist von Beginn an mit dem Auftreten von Wortkanten ohne Fortsetzungskante zu rechnen. Bedingt wird dies Phänomen durch den zeitlich begrenzten Fokus des Betrachtungsfensters des Erkenners. Im Anschluß wird gezeigt, welche Konsequenzen für die Pfadbewertung sich aus dem Unterschied ergeben. Bei der asynchronen Kopplung ist es aufgrund der vorhandenen Gesamtübersicht über den Wortgraphen möglich, derartige Fragmente durch eine Vorverarbeitung zu eliminieren. Die Anwendung der Pfadbewertung für vollständige Graphen erfordert es. Die inkrementelle Erzeugung von Wortgraphen beschreibt Rautenstrauch anhand ihrer Realisierung im Rahmen des BMFT-Projekts “VERBMOBIL” [Rau94].

2.1.2 Pfad-Bewertung im vollständigen Wortgraphen

Prädikation und Gegenstands-, sprich Kettenbildung, sind die Seiten ein- und desselben Analyseprozesses. Die numerische Bewertung von Knotenpunkten des Suchraums, sprich der einzelnen Strukturbeschreibungen, wird auf die numerische Bewertung der prädizierten Wortketten zurückgeführt.

Für die sogenannte ’akustische Qualität’ der Wortkandidaten wird üblicherweise die *Annahme der Additivität* gemacht,
d.h. die Bewertung einer Kette von Wörtern kann durch Addition der Einzelbewertungen berechnet werden.

Das Quantum ’akustische Qualität eines Lexems’[3] ist im Normalfall ein logarithmierter Wahrscheinlichkeitswert. Die Wahrscheinlichkeit eines Pfades ergibt sich aus dem Produkt der Einzelmaße. Für Logarithmen hat man den Übergang vom Produkt zur Summe zu vollziehen.

[3] was **ein** *Wort* ist, entscheidet das verwendete Lexikon – z.B. ’guten_Tag’

Während die Bewertung des einzelnen Wortes eine durchaus relative, nach der Wortlänge zu wichtende Größe darstellt, heben sich in der Kette solche Relativierungen potentiell auf. Zu jedem Wortkandidaten gibt es im Wortgraph einen bestbewerteten Pfad und damit ein eindeutig bestimmtes Maß. In Bild 2.5 wird für ausgewählte Kanten des Graphen in Bild 2.1 das Pfadmaß, gesplittet in Vorpfad (einschließlich der Einzelbewertung) und Nachmaß, angegeben und die akustische Qualität normiert.

	Frames		norm. Qualität	Vor- summe	Nach- summe	Knoten	
ich	2	10	2.278	20.50	247.89	0	1
...							
morgen	40	73	1.501	106.99	161.40	2	3
einen	40	73	0.960	88.62	188.84	3	4
...							
ankommen	184	220	1.114	268.39	0.00	9	10

Bild 2.5 Pfadbewertung imWortgraph (Vor- und Nachmaß)

Im Anhang “Wortgraphen-Interface” wird die von mir erstellte Prozedur angegeben, die für alle Wortgraphen des ASL-Formats (vergleiche Bild 2.1) den entsprechenden pfadbewerteten Graphen erzeugt. Ihre Pfadberechnungen sind invariant gegen die vom Worterkenner verwendete Methode der Knotennumerierung. Es können folgende Bestimmungen vorgenommen werden:

1. der beste Pfad durch den Graphen,
2. für jeden Ereignis-, sprich Frame–Zeitpunkt mit Knotenzuordnung:
   der bis dahin beste Teilpfad[4], vertreten durch den besten Einzelkandidaten seiner letzten Kante.

Sollen Worterkennung und linguistische Interpretation getrennte Arbeitsphasen und der Wortgraph im ASL-Listenformat gegeben sein, so kann man aus den Pfadmaßen der Wortkandidaten eine optimistische Restschätzung der Bewertung von Konzeptgraphen gewinnen. Dazu betrachte man die jedem Konzeptgraphen eindeutig zugeordnete Wortkette:

Man bestimmt die **assoziierte Wortkette**
eines Konzeptgraphen, indem man

1. alle I(HYPOTHESE)-Marker, d.h. alle Instanz-Marker für die unterste, signalnahe Ebene des semantischen Netzes auswählt,
2. die zugehörigen Wortkandidaten bestimmt,
3. und diese in aufsteigender Reihenfolge der zugehörigen Signalintervalle ordnet.

[4] Gegenstück zum Viterbi-Pfad des Erkenners

Wortketten sind keine Mengen. Sie können ein Wort mehrfach als Wortkandidaten enthalten. Die assoziierte Wortkette ist die Grundlage der Gegenstandsbildung, d.h. der Extraktion linguistisch “wohlgeformter” Subketten aus dem Wortgraphen.

Es sei [w1,...,wn] die assoziierte Wortkette des Konzeptgraphen.

Die **optimistische Restschätzung** einer Wortkette ergibt sich:

1. aus den Pfadmaßen aller Kettenglieder,
2. aus der Wahl des schlechtesten Maßes unter diesen.

Das Maß heiße das *Maß des schlechtesten Kettengliedes*.

Das Maß ist optimistisch oder

’*ein unterer Grenzwert der tatsächlichen Restentfernung*’ (Winston [Win87b][5])

in folgendem Sinne:

alle später zur assoziierten Wortkette hinzukommenden Wortkandidaten können den aktuellen Wert nur verschlechtern.

Bei Verwendung der n-besten Ketten ist dies offensichtlich, da die gegebenfalls linguistisch nicht ZULAESSIG erweiterbare Kette durch Austauch mit Gliedern schlechterer Ketten abgeändert werden muß.

Die *Kettenlänge*, d.h. die Summe der überdeckten Signalframes, ist mit der assoziierten Wortkette ebenfalls eindeutig bestimmt. Sie wird als zusätzliches Vergleichsmaß hinter die Restschätzung im Bewertungstupel für Konzeptgraphen eingeordnet.

Bei der Definition einer Restschätzung ist stets folgendes zu beachten:

Da Ketten- oder Pfadmaße auf ein Produkt von Wahrscheinlichkeiten hinauslaufen, sind kürzere Ketten stets besser, sprich wahrscheinlicher, als längere. Die Frage der *Länge von Pfaden* relativ zueinander wird für Subketten ein- und desselben Pfades gestellt.

Die Steuerung der Suchbaumentwicklung muß eine automatische Bevorzugung der Strukturbeschreibungen kürzerer Ketten beim Opentracking (Rückverfolgung mit Hilfe der OFFEN-Liste) verhindern. Es käme zur Breitensuche.

Für vollständige Wortgraphen findet das Längenproblem mit dem Maß des schlechtesten Kettengliedes seine Lösung:

In der akustisch besten Wortkette haben alle Subketten die gleiche Restschätzung.

2.1.3 Pfad-Bewertung im inkrementellen Wortgraphen

Vollständige Wortgraphen kann man inkrementell verarbeiten, indem man die vorab berechneten Pfadmaße verwendet. Das Ziel der inkrementellen Verarbeitung besteht aber letzten Endes in der **zeitsynchronen Kopplung** mit einem Worterkenner, der in der Lage ist, Wortpfade inkrementell aufzubauen. Nur im zeitsynchronen Verhältnis beider ist es möglich, aus dem Tiefenstrukturparsing Rückkopplungen für die weitere Pfadverfolgung des Erkenners selbst zu beziehen.

Anhand des Wortgraphen von Bild 2.1 sei zunächst die inkrementelle Verarbeitung eines vollständigen Wortgraphen skizziert, der eine ’Masche’ enthält. Für das Quantum

[5] siehe das umfassendere Zitat im 1.Kapitel

der akustischen Qualität sei ein umgekehrt proportionales Verhältnis zur Zahlgröße angenommen. Die Aufgabelung des Graphen in 2 Pfade (zum Vergleich Bild 2.2) wird durch folgende Wortkandidaten eröffnet:

Pfad 1: [2]—morgen—[3],

Pfad 2: [2]—einen—[6].

Angenommen, das Pfadmaß des linguistisch nicht korrekten Pfades 2 sei besser als das von Pfad 1. Der konstruierte Fall wäre ein ’worst case’, insofern die streng Links-Rechts zugreifende inkrementelle Verarbeitung die UNZULAESSIGKEIT erst am letzten Inkrement bemerken würde. Ein rein auf akustische Maße gestützter A*-Algorithmus wäre angesichts derartig konkurrierender ’Maschen’-Pfade nicht sinnvoll. Er würde zwar für die obigen 2 Wortkandidaten konkurrierende Suchbaumknoten anlegen, aber zunächst nur einen Pfad – den falschen – weiterverfolgen. Damit wäre im Verhältnis zu einer nicht-inkrementellen Verarbeitung, wie sie bisher im System ICZ auf vollständigen Wortgraphen des Typs (2) durchgeführt wurde, eher ein Nachteil entstanden.

Der ERNEST-Kontrollalgorithmus sieht eine Kopplung stabiler Basis- und veränderlicher User-Routinen vor. Die Kontrolle des Wortgraphen, z.B. betreffs kritischer Gabelungen, kann mittels Userfunktionen erfolgen, indem kontrollbestimmte Kennzeichnungen im Bewertungstupel eintragen werden. Ihnen ist dann Vorrang vor den akustischen Bewertungen zu geben. Der A*-Algorithmus bleibt zulässig, aber er arbeitet nicht notwendig optimal, denn es wird im Allgemeinen mehr als eine Minimalzahl von Expansionen durchgeführt:

Kontrollbestimmte User-Bewertungen erhalten die Zulässigkeit der A*-Suche bei Verzicht auf Optimalität.

Eine solche Änderung der Fragestellung ist umso naheliegender, wenn es um die direkte Gestaltung der zeitsynchronen Kopplung geht. Durch die Rückkopplung sollen dem Worterkenner zur Beschränkung seiner Suche – dem sogenannte Prunen von Wortpfaden – inhaltliche Fingerzeige gegeben werden. Für die nicht-optimale Expansionszahl in der linguistischen Suche erhält man ein Zeitäquivalent eingesparter Erkennerpfade und einen Zugewinn an Robustheit.

Im Normalfall erzeugen Worterkenner Pfade, die wie ’Ranken’ um eine gemeinsame virtuelle Hauptachse gewunden sind. Man denke sich zu Pfad 1 des betrachteten Graphen eine zusätzliche ’Umrankung’ (siehe Bild 2.6). Rankenbildungen mit verstreuten Insertions/Deletions sind typisches Produkt der Viterbi–Suche von Worterkennern. Die Funktion der Viterbi-Suche im Rahmen der Hidden-Markov-Modellierung für statistische Worterkenner behandelt Rabiner [Rab78], [Rab85], zu einer korrekten populärwissenschaftliche Einführung ergänzen sich die Artikel von Spies/Steinbiß[Spi94] und [Ste94].

Bei rein akustischer Bewertung würde derjenige Wortkandidat der Kante [4]—[5], der die bessere Einzelbewertung besitzt, über die unmittelbare Fortsetzung der Suchbaumexpansion entscheiden. Auch dieses Beispiel legt es nahe, zusätzliche Usersteuerungen zu verwenden, um gegebenenfalls *alle* relevanten, linguistisch zulässigen Expansionen der Strukturbeschreibung zu realisieren. Im obigen Fall würde es sich um die parallele Expansion des Auftretens einer Orts- versus einer Zeitkonstituente handeln.

Bei der Bewertung von Subketten steht linguistische ZULAESSIGKEIT höher als akustische Qualität.

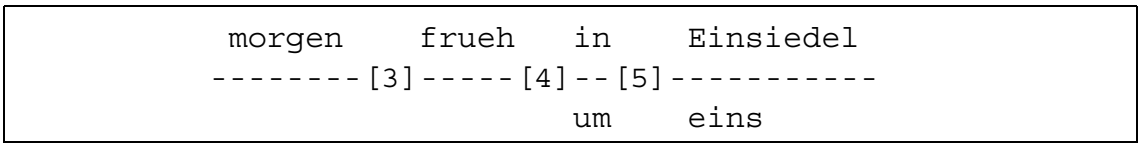


Bild 2.6 “Ranken” im Wortgraph

Linguistische ZULAESSIGKEIT ist von A*-Zulässigkeit zu unterscheiden – das soll die Großschreibung anzeigen. Sie bedeutet Übereinstimmung mit der Wissensbasis.

Andererseits ist es gerade wünschenswert, eine linguistische Verarbeitung zu haben, welche die akustischen Bewertungen systematisch ausnutzen kann. Dazu ist das bereits für vollständige Wortgraphen diskutierte *Problem der Kettenlänge* zu lösen. Es stellt sich bereits in der Worterkennung, wenn A*-Suche über den Viterbi-Pfaden durchgeführt werden soll. Eine Übersicht der Problemlage hat D.Paul 1990 in “Algorithms for an Optimal A* Search” [Pau92] (Titel von mir verkürzt) gegeben:

’A* criterion is the difference between the actual log-likelihood of reaching a point in time on a path (Pfad i, Zeitpunkt t, logarithmierte Wahrscheinlichkeit $\mathbf{L_i(t)}$ – d. Verf.) and an upper bound on the log-likelihood of any path reaching that point in time (bezeichnet mit **ubL(t)**, meine Einfügung)’ (S.201).

A*-Kriterium (Worterkennung) ist die Differenz

$L_i(t) - ubL(t)$,

d.h. die Normierung der Wahrscheinlichkeit des Pfades i auf Länge.

Je Zeitpunkt t sei **lubL(t)** die kleinste untere Grenze aller ubL(t).

Ist ein Worterkenner in der Lage, für jeden Framezeitpunkt des Wortgraphen, dem ein Graphknoten assoziiert ist, das Maß lubL(t) zu berechnen, so kann auch zur (linguistischen) Analysezeit ein für die A*- Suche zulässiges Maß durch einfache Differenzbildung[6] berechnet werden. Für (als kohärent unterstellte) Wortketten bedeutet dies:

Kettenmaß – lubL(Ketten-Endpunkt).

Kennt man die Größe von lubL(Ketten-Endpunkt), so wäre die (rein akustisch gesteuerte) A*-Suche optimal. Sie benötigte eine Minimalzahl von Suchbaumexpansionen.

2.2 Inkrementelle Verarbeitung der Wort-’Ereignisse’

2.2.1 Ziel-Konzepte im ICZ-Sprachnetz

Das in Abschnitt 2.1.2 behandelte Beispiel verkörpert eine typische Problemsituation der inkrementellen linguistischen Analyse und der Links-Rechts-Verarbeitung allgemein. Einerseits geht es darum, den akustisch-besten und linguistisch-zulässigen Pfad durch den Wortgraphen nicht zu verpassen. Andererseits ist auf inhaltliche Problemsituationen zu reagieren, die sich an den Strukturbeschreibungen der OFFEN-Knoten ablesen lassen.

[6] für Logarithmen ist das ein Quotient, sprich ein Normierungsverhältnis

Die erste Verarbeitungsphase ist damit beschäftigt, den Wortgraph von links-nach-rechts durchzuarbeiten, d.h. dem Signal (bis auf Lücken) genau eine (Teil-) Kette eines Pfades des Wortgraphen zuzuweisen. Die linguistische Konsistenz der Wortkette wird durch ihre Zerlegbarkeit in syntaktische Modell-Konstituenten des semantischen Netzes erwiesen.

Die Taxonomie des semantischen Netzes, in diesem Fall der Spezialisierungsachse gibt vor, welche Konstituenten modelliert sind:

```
SY_KONSTITUENTE(X)
  :-(spez)     SY_NG(X),SY_PNG(X),SY_VG(X),
               Z_ZEITANGABE(X),ADVGRU(X).
```

NG steht für Nominalgruppe, PNG für Präpositionalgruppe, VG für Verbalgruppe, ZEITANGABE für Zeitkonstituente, ADVGRU für Adverbialgruppe.

Das Sprach-Netz sagt aus, welche Konstituenten vorgesehen sind.

Die Konstituenten haben im ICZ-Netz eine eindeutig bestimmte Stellung in der vertikalen Taxonomie und können sich nicht rekursiv – wie bei reinen Syntax-Parsern üblich – als Sub-Konstituente enthalten. (z.B. np als Bestandteil von np) . Eine SY_NG kann aber Bestandteil einer SY_PNG sein.

Der Konstituenten-Begriff wird dem Tiefenstrukturparsing angepaßt.

Eine Subkette heiße eine **Konstituente**, wenn sie als Träger einer pragmatische Rolle fungieren kann.

Als recht speziell mag die Form der Zeitkonstituente erscheinen. Das vom Zugauskunftssystem auszulösende Retrieval im Bundesbahnfahrplan erfordert scharfe Suchindices. Die Aufteilung der Modalitäten des Konzepts Z_ZEITANGABE sollen deren Festlegung unterstützen:

1. *an welchem Tag wollen Sie ... ?* (Z_TAG)
2. *in welchem Tagesabschnitt ... ?* (Z_ABSCHNITT)
3. *um welche Zeit ... ?* (Z_UHR)

Das durch den Prefix Z_ herausgehobene System der Zeitkonzepte besitzt domänenübergreifende Verwendung, z.B. für Terminabsprachen. Tendenziell läuft es auf einen Akzeptor für alle Phrasen hinaus, deren Aussage auf ein Intervall der Kalenderzeit abgebildet werden kann.

Wie schon betont wird auf Satzgrammatikalität verzichtet. Das Konzeptsystem ist auf die Bindung der semantischen und pragmatischen Rollen mittels der Konstituenten zugeschnitten.

In Bild 2.7 wird exemplarisch eine wichtige – nicht die einzig mögliche – Konzeptgraph-Entwicklung der Wortkette [ich, moechte, morgen, frueh] vorgestellt.

Die Gegenstands-, sprich Ketten-Segmentierung, ergibt:

[ich] — [moechte] — [morgen, frueh].

Die Strichlinie soll die Richtung der nach oben offenen Weiterentwicklung der Konstituenten anzeigen (HYP=hypothese, PRON=pronomen, MVERB=modalverb und ADV=adverb).

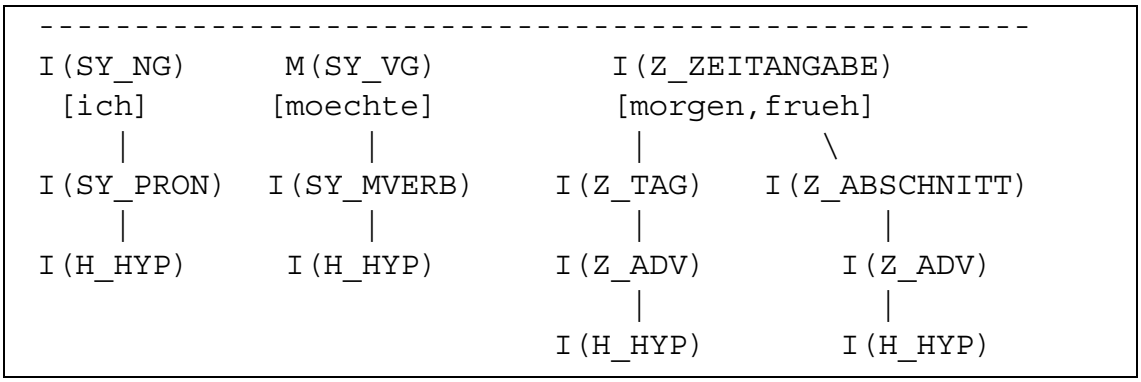


Bild 2.7 Konzeptgraph der Kette [ich, moechte, morgen, frueh]

Es stellt sich die Frage, ob Z_ZEITANGABE eine Konstituente im syntaktischen Sinne ist. Zeiten und Orte weisen eher auf Semantik. Betrachtet man die von SY_KONSTITUENTE resümierten Konzepte, so ergibt sich fernerhin die scheinbare Inkonsistenz:

Neben der Z_ZEITANGABE, grammatisch eine Adverbialgruppe oder eine Präpositionalgruppe, gibt es auch noch die SY_PNG und die SY_ADVGRU. Dagegen gibt es keine gesonderte ORTSANGABE. Sie fällt aktuell – in der Zugauskunftsdomäne – unter die SY_PNG. Die Z_Komponente redupliziert zum Zwecke der Modularisierung einen Teil der Syntax-Konzepte.

Solche Einteilungen haben sehr praktische Gründe oder widerspiegeln generelle Schwierigkeiten der Gestaltung von Wissensbasen für Applikationen:

1. Jede Wissensbasis unterliegt, wenn sie eine Applikation stützen soll, dem trade-off zwischen ihrer Aussagekraft und der Effizienz ihrer Verwendung [Woo75].
2. Die Komponenten der Wissensbasen sind in unterschiedlichen Phasen entstanden, besitzen unterschiedlichen Allgemeinheitsgrad und unterschiedliche Modularisierung (wie Z_) zwecks Wiederverwendung in anderen Applikationen.

Anzustreben ist deshalb die

weitestgehende Invarianz der Kontrolle gegen Änderungen der Wissensbasis.

ERNEST grenzt zu diesem Zweck eine veränderliche Schale wissens- und problemabhängiger Kontrollfunktionen von einem unveränderlichen Kern, der **Basis-Kontrolle** (dem *Kontrollalgorithmus*) ab. Beide Teile liegen aktuell in C–Code vor und werden durch das Unterprogramm-Prinzip gekoppelt.

User-Routinen kapseln die änderungsabhängigen Teile der Kontrolle.

2.2.2 Einführung der Aufwärtsableitung

Konstituenten sind besonders geeignete Subziele der Analyse. Ihre Vor-Bestimmung gehört zur veränderlichen Schale der Kontrollarchitektur. Sie sollen deshalb *User-Ziele* heißen. **ERNEST-User** sind Personen, welche eine in ERNEST formulierte und compilierte Wissensbasis auf einen Datenbestand, der die zu prädizierenden Gegenstände enthält, anwendet und dazu die von ERNEST geforderten User-Kontroll-Routinen[7] codiert.

Zum Vergleich, PROLOG-User ist derjenige, der dem PROLOG-Interpreter die entsprechenden Ziel- und Subziel-Formulierungen zur Ableitung vorgibt. Ich habe eine Prototyplösung geschaffen, um eine vergleichbare Testumgebung für ERNEST-Nutzer verfügbar zu machen.

Subziele sind Stationen auf dem Weg bis zur Ableitung des Analyseziels. Analyseziel ist die Prädikation eines Konzepts, das die pragmatische Tiefenstruktur einer Wortkette zusammenfaßt.

Ableitung in ERNEST kann parallel mehrere Subziele verfolgen

Subziele können als Neuziele gestellt, d.h. als Strukturmarker im zu expandierenden Konzeptgraphen plaziert werden, obwohl dieser bereits offene Subziele enthält. Die Ausschöpfung der Varianten des Ableitungsmechanismus von ERNEST erfolgt in der Aufgabenkontrolle, die durch Userfunktionen implementiert wird.

Die Programmierung der Basiskontrolle ist dadurch ausgezeichnet, ausschließlich in problem-unabhängigen, formellen Termini der Netzwerktechnologie formuliert zu sein. In erster Näherung (da insbesondere für den User bisher nicht spezifiziert) gilt:

Basis-Programme: formelle Netzwerk-Termini

User-Programme: konzeptuelle Linguistik-Termini

Vorauswahl der Subziele soll durch Userprogrammierung erfolgen. Sind sie nicht ebenso aus der bewußt gewählten semantischen Netzhierarchie mit rein formellen, sprich domänen-unabhängigen, Mitteln ablesbar ?

Sagerer hat in der bisher umfassendsten Monographie über ERNEST [Sag90] einen Kontrollalgorithmus für das System ICZ angegeben, der auf den ersten Blick ohne User-Ziele auskommt. In der Initialisierungsphase (Schritt 1–7, S. 267) werden die Subziele automatisch aus dem Netz entnommen:

'Schritt 4: Generiere für jede ausgewählte Worthypothese ..... Instanzen ... und ein ... Konzept, das eine syntaktische Wortart repräsentiert'.

Somit sind die Subziele mit der ausgezeichneten Menge der Wortartkonzepte automatisch aus dem semantischen Netz bestimmbar.

In Schritt 8 entnimmt er die nächsten Subziele wiederum einer ausgezeichneten Konzeptmenge:

---

[7] sie wurden in dem 1991 vorgelegten ERNEST-Manual spezifiziert: Funktionsname, Argumenttypen und Ausgabetyp

'die Konzepte, die der Pragmatikebene des Netzwerks zugeordnet sind und keine Bestandteile haben'. Damit sind die pragmatischen Rollen[8] gemeint und die Rollen-Kontexte ausgeschlossen.

Das generelle Ziel von Subzielbestimmung hat Sagerer wie folgt formuliert:

'Über die Abstraktionsebene Pragmatik ist die inhaltliche Kompetenz (des Auskunftssystems, meine Einfügung) festgelegt. Jedes der Konzepte dieser Ebene, das einen Auskunftstyp realisiert, repräsentiert ein mögliches *Analyseziel*. Für die Effizienz der Analyse ist somit eine möglichst schnelle und frühzeitige Auswahl von potentiellen – im Idealfall eines – Analyseziels von großer Bedeutung (S.239)'.

Anhand des Ziels M(P_REISENDER [ich]) in Bild 2.8 (es enthält VERB= vollverb, ADJU= unflektiertes_adjektiv) sei erklärt, wie in ERNEST Bottom-Up-Ableitung durchgeführt wird. Es markiert die in Sagerers Kontrollalgorithmus beabsichtigten

'"Sprünge" von einzelnen Worthypothesen unmittelbar zu beliebig hohen Abstraktionsebenen' (S.239) für die Wortkette [ich, moechte, morgen, frueh].

Formell werden Subziele als Neuziele in die Strukturbeschreibung eingetragen. Auf diese Weise kann man zwar sprunghaft in der Konzepthierachie hinaufsteigen, aber ein isoliertes Ziel muß wie in PROLOG expandiert werden. Die Absichten der inkrementellen Verarbeitung – ein immer wiederkehrender Respekt – lassen das nicht generell zu. Erinnert sei nochmals an das Problem der diskontinuierlichen Konstituenten.

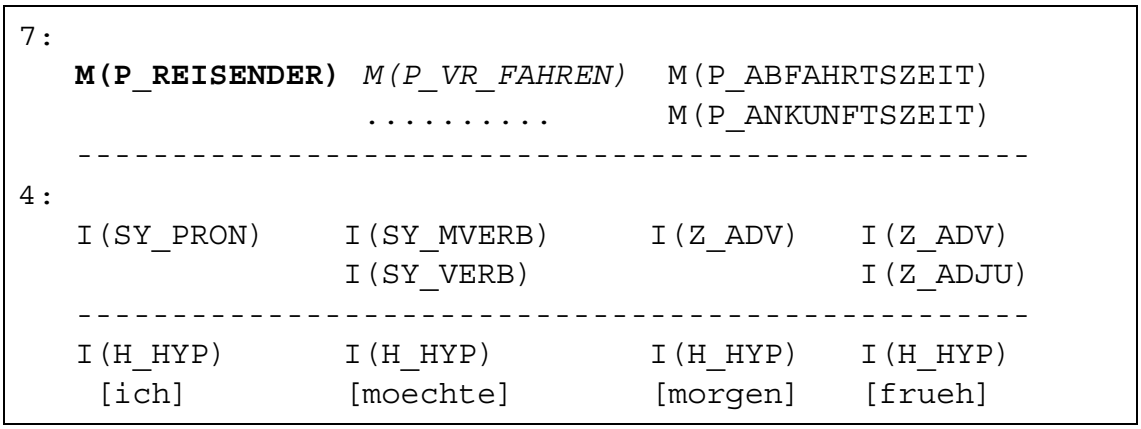


Bild 2.8 Subziel-Finder

Eine andere Möglichkeit, die ERNEST bietet, nachdem die partielle Ableitung ermöglicht ist, stellt die Bottom-Up-Ableitung von Subzielen dar.

Wie in Abschnitt 1.2 gezeigt wurde, kann man in PROLOG bottom-up vorgehen, indem man geeignete Metaprädikate wie wortart, konstituente (im ICZ-Netz: SY_KONSTITUENTE) bis zur Fakten-, sprich Signalebene, expandiert – und zwar vollständig. Wie ihr Gegenstück, die Expandierung, entwickelt diese Ableitungsform einen im semantischen Netz zusammenhängenden Pfad von Rollenbindungen entlang der Vertikalen[9].

---

[8] in Abschnitt 1.4 wurde erklärt, daß Rollen auch als Konzeptmenge, den zulässigen Rollenträgern, aufgefaßt werden kann

[9] man spricht auch von der Dekompositionsachse der Bestandteile und Rollenabstraktionen

Die Ableitungsform kann konsequent entlang der vertikalen Taxonomieachse – ohne Zwischenschritte über Metakonzepte – durchgeführt werden. Sie kann wie die Expandierung partiell bleiben. Sie heiße – ganz im Sinne der von Sagerer ausgesprochenen Motivation – die **Aufwärtsableitung**. Allerdings vollzieht sich diese Ableitung schrittweise. Die Einzelschritte beziehen sich jeweils auf ein nächsthöheres Konzept, ein *Zwischenziel*, im Netz. Sie ist insofern nicht sprunghaft, aber sparsam. Dies wurde durch eine von mir in ihrem Zugriff erweiterte, die möglichen Zwischenziele beschränkende User-Routine verbessert (Kapitel 3, Abschnitt 'Zwischenziele').

Um den Unterschied der Aufwärtsableitung zur Bottom–Up–Ableitung mittels Metaprädikaten zu klären, wird ein Beispiel betrachtet:

```
  Kette von Elementarableitungen
(4)       M(P_REISENDER [ich]) :-
(3)           M(S_AGENT [ich]) :-
(2)               I(SY_NG [ich]) :-
(1)             I(SY_PRON [ich])
```

Bild 2.9 Zwischenziele der Aufwärtsableitung von P_REISENDER

Man vergleiche die Ableitungsweise des UP-Automaten. Für ihn wäre es ein Unding, den Return von Regel (3) zu Regel (4) auszuführen, bevor M(S_AGENT [ich]) instantiiert ist. Das ist noch nicht der Fall, da der Konzepttyp S_AGENT Tiefenkasus (siehe ICZ-Netzübersicht in Bild 2.17), also Teil eines Kontextes, sprich Rollenensembles, ist. Folgerichtig müßte er ein passendes Element der folgenden KONTEXT_VON–Liste in der Konzeptdeklaration von S_AGENT expandieren:

```
START_KONZEPT S_AGENT
      ...
      KONTEXT_VON( S_VR_FAHREN, S_VR_NEHMEN,...)
      KONTEXT ()
      ...
ENDE_KONZEPT
```

Da keinerlei Änderungen an der Wissensbasis vorgenommen werden, wird auf ihre ausführliche Beschreibung verzichtet und auf die Abhandlung von Kummert [Kum92a] verwiesen. Wesensmerkmale, welche das ICZ-Netz für das Tiefenstrukturparsing geeignet machen, werden en passant erläutert. So zeigt die ICZ-Netzübersicht die im wesentlichen auf den sogenannten *Verb- und Nomenrahmen* beruhende Zusammensetzung der Netzschichten Semantik und Pragmatik. Die Auswahl der Verb/Nomenrahmen zusammengefaßten Tiefenkasusrollen in ICZ lehnt sich an deren theoretische Begründung durch Fillmore [Fil71] an. Die Modellierung von Kontexten verweist auf die computerlinguistische Beschreibungsebene der Dependanzgrammatik [Hel90]. Formelles Gestaltungsmittel in ERNEST zur Zusammenfassung der Tiefenkasus in den Rahmen sind *kontextabhängig* definierte BESTANDTEIL-Kanten.

Nach der inhaltlichen Analyse der Situation in Bild 2.9 nun zurück zu den formellen Besonderheiten der Aufwärtsableitung gegenüber der Bottom-Up-Ableitung mit der UP-Maschine. Sie läßt sich aus dem Bild wie folgt ablesen:

Aufwärtsableitung gestattet es,
die Elementarableitung von Zwischenzielen nach oben fortzusetzen,
obwohl dieselbigen erst partiell abgeleitet sind.

Insofern ist es berechtigt, im Sinne von Sagerer von ’sprunghafter’ Entwicklung zu sprechen. Aufwärtsableitung erfordert die Ersetzung der zu starren UP-Maschine [Hop88] durch eine virtuelle Maschine zur systematischen Analyse der netzartigen Sturkturbeschreibung, in der alle durch Aufwärts-’Sprünge’ offengebliebenen Bindungen verzeichnet sind.

Die Ausschöpfung der Möglichkeiten der Aufwärtsableitung kann als differentia specifica der inkrementellen Verarbeitung unter den Vorzeichen der ERNEST-Basiskontrolle angesprochen werden. Die Weiterentwicklung der Bottom-Up-Ableitung in PROLOG-Umgebungen wird durch den sogenannten Shift-Reduce-Parser gegeben. Eine einführende Darstellung desselben gibt [Eik89]. Die betreffs der SKIP-Verarbeitung relevante Darstellung ist in [LA93] enthalten und zeigt die Einbettung des Prinzips in die GLR*-Konzeption [Tom86].

2.2.3 Anwendungsspielräume der Aufwärtskontrolle

Im Folgenden soll eine Übersicht zur Beeinflussung der Aufwärtsableitung durch Basis- und User-Routinen aufgestellt werden. Die Anteile beider Seiten werden mittels der Label ’Basis’ , ’User’ extra angezeigt:

1. Aufwärtsableitung eines Subziels ist eine Hintereinanderschaltung der Aufwärtsableitung von Zwischenzielen. Letztere bilden im semantischen Netz einen Bindungspfad vom Konzepttyp des Startmarkers zum Subziel (Konzept).
   User: Zwischenziele jeweils Subziel -> Aufwärtsableitung genau eines Bindungspfads.
2. Je höher das Subziel, desto mehr Gabelungen des Bindungspfads. Die Basiskontrolle erzwingt jedesmal Suchbaumaufspaltung.
   User: Nebenpfade weglassen
   -> Gefahr den richtigen Pfad zu verpassen
   am sichersten:
   Subziel nicht zu hoch ansetzen,
   alle Pfade ableiten
   User: Subzielhöhe, Pfadzahl
   auf Ambiguität/Ungewißheit abstimmen.
   Beispiel: die Aufwärtsableitung eines pragmatischen Verbrahmen (P_VR_..) von einem Hilfsverb-Marker, I(HYP [moechte] in Bild 2.8 ist wegen Ambiguität der Auswahl unter diversen Verbrahmen zu vermeiden.
3. Dem Erreichen des Endpunkts einer Aufwärtsableitung entlang einer Kette von Elementarableitungen folgt stets eine Top-Down-Propagierung von Restriktionen der Attributwerte im gesamten Ableitungsbaum des erreichten Markers und damit die Einschränkung der Gegenstandsbildung an den tieferstehenden Markern desselben.
   User: Subziel höher ansetzen -> schärfere Restriktion propagieren.
4. Aufwärtsableitung eines netz-nächsten Zwischenzieles, sofern nicht schon als Strukturmarker gesetzt, ist identisch mit Setzen des Neuziel-Markers und Expandierung desselben. Expandierung läuft hier auf genau eine Rollenbindung hinaus, welche

die Modalität einschränken kann. Zwischenziele werden (bisher generell) modalitätsbestimmt aufwärts-abgeleitet, was einen Suchbaumknoten je Modalität nach sich zieht.

Basis:
- modalitätenfreie Ableitung
von M()-Markern aus M()-Markern integrieren
- Top-Down-Propagierung
der Gegenstandsrestriktion sichern.

Diese Form der Steuerung wird zur Zeit nicht verwendet.

5. Sind für Sub- oder Zwischenziele schon M()-Marker abgeleitet, so sind diese zuerst zu expandieren:
im Beispiel (Bild 2.8) ist eine unike Zeitkonstituente linguistisch gefordert, d.h. von I(HYP [morgen]) und I(HYP [frueh]) ausgehende Bindungspfade müssen zusammenlaufen.
Basis: realisiert (v. Verf.) -> neue Expandierungsroutine für Aufwärtsableitung
6. Kontextbindung eines Strukturmarkers wird durch Aufwärtsableitung des Kontextkonzepts realisiert.

Aufwärtsableitung wird durch geschickte Wahl der Sub- und Zwischen-Ziele beeinflußt. User-Routinen bestimmen diese Ziele und benutzen:

Heuristiken der **Userziel**-Bestimmung für Problem-SITUATIONEN in Wort- und Konzeptgraph.

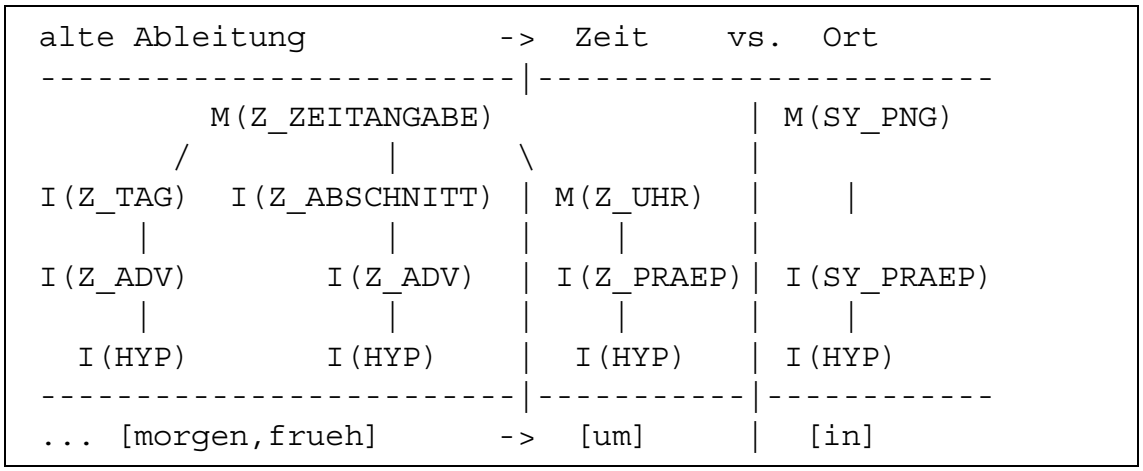


Bild 2.10 Raum-Zeit-Ambiguität

Es geht nicht allein darum, Teile der Kontrolle, die Änderungen unterliegen, in Subroutinen auszulagern, sondern auch darum, Kontrollwissen, z.B. computerlinguistischer Art, einzubringen. Zur Erläuterung sei die Problemsituation des Wortgraphenpfades in Bild 2.6 hergenommen. Inkrementell sind als nächstes die Wortkandidaten ’um’ und ’in’ zu verarbeiten. Die Ableitung befindet sich in der Phase der Konstiuentenbildung. Phasenerkennung fällt dem User zu. Die Basiskontrolle läßt die Ergänzung der bereits vollständig abgeleiteten Zeitkonstituente um ein obligatorisches als auch um ein

optionales Bestandteil zu. Die Wissensbasis enthalte eine Modalität mit obligatorischer Uhrzeitangabe. Zum anderen kann dieselbe offen gelassen und ein weiteres Subziel zur Bildung einer Ortskonstituente (Konzept SY_PNG) angegeben werden.

Die folgende Übersicht soll zeigen, welche unterschiedlichen Blickrichtungen von Heuristiken betrachtet werden können:

1. **akustische A*-Bewertung** —
   die akustisch besseren Kandidaten werden zunächst bevorzugt. Die Fehlerbehandlung sowie die Erkennung linguistischer Ambiguitäten überläßt man dem Opentracking.
2. computerlinguistisches Prinzip der
   Rechts-Assoziation nach Kimball [Kim73]:
   ’*terminal symbols* (’um’, d.Verf.) *optimally associate to the lowest nonterminal node* (M(Z_ZEITANGABE), d.Verf.)’
   (zitiert nach L.Frazier/J.D.Fodor [Fra78], die eine ausführliche Diskussion des Prinzips vornehmen). Die Begründung ist auch kognitiver Art, insofern argumentiert wird, die Anbindung an die letzte linke Konstituente unterstütze ’Verstehen’ und Ambiguitätsreduktion. Es wird angenommen, daß Anbindungen an weiter vorn liegende Konstituenten vom Sprecher/Hörer absichtlich vermieden werden.
3. **situative Bewertung des Konzeptgraphen** —
   Wurde bereits eine pragmatische Bestimmung aus einer Ortskonstituente (aufwärts) abgeleitet[10]? Eine der beiden Suchbaumexpansionen durch eine extra User-Bewertung des Bewertungstupels auf– oder abwerten (Userroutine ’Knotenbewertung’).
4. **situative Bewertung des Wortgraphen** —
   [in], [um] sind besonders kurze Wörter und Kandidaten für Insertionen. Es wird von der strengen Links-Rechts-Auswahl der Wortkandidaten abgewichen und ein längeres Intervall abgewartet (Verzögerungs- oder SKIP-Methode).

Die Liste verweist auf unterschiedliche Aufgaben der User-programmierten Steuerung, welche der im ERNEST-Manual 1991 dokumentierte Satz von User-Routinen nicht abdeckt.

Funktionen wie die Subzielbestimmung sind nicht nur problem-abhängig zu betrachten und auf die Verarbeitungsmethode abzustimmen. Andere Funktionen wie die Bewertung der Suchbaumknoten, die deren Platz in der OFFEN-Liste regelt, können teils auf objektivierte und mathematisch–analytisch begründete Maße (Restschätzung u.a) begründet werden, teils Teilbewertungen mit empirisch-linguistischen Annahmen wie in Fall 2 und 3 enthalten. Einige Userfunktionen, sprich Eingriffsstellen, mußten hinzugefügt werden. Teilweise greifen sie in Basis-Routinen ein, z.B. die Ereignissteuerung. Oder sie verändern das problem-abhängige Arrangement der Ressourcen der Kontrolle (Faktenbasis, Wissensbasis, Konzeptgraphen, Instanzenspeicher u.a), welches durch eine Initialisierungsfunktion eingerichtet werden muß. Im 4. Fall, der auf die ’wait-and-see’-Taktik des Users Bezug nimmt, war erst der Zugriff der Userkontrolle auf die ’Faktenbasis’ (den Wortgraphen) einzurichten.

ERNEST räumt der User-programmierten Kontrolle größeren Spielraum ein als z.B. eine Standardumgebung wie PROLOG. Die Begründung bezieht sich nicht allein auf

[10] eine Rolle spielt dabei die Verwendung einer Defaultannahme: Abfahrtsort = Ort der Zuganfrage

Effizienz und Flexibilisierung der prozeduralen Komponente, sondern vor allem auf die Ausnutzung des erweiterten Ableitungs-Mechanismus.

Im Rahmen meiner Abhandlung will ich eine vollständige Übersicht dieser Funktionen geben.

2.3 Metaregeln zur Entwicklung des Konzeptgraphen

Durch die Einführung der partiellen Ableitung, der Aufwärtsableitung und die Einbeziehung der Strukturbeschreibung in den Ableitungsvorgang ergibt sich eine erhebliche Auffächerung der Steuerungsmöglichkeiten. Als einheitliche Form bieten sich Metaregeln zur Erweiterung des Konzeptgraphen an.

2.3.1 problem-unabhängige Metaregeln

In meiner Sicht besitzt ERNEST 5 allgemeine, problemunabhängige Metaregeln des Typs:

SITUATION der Strukturbeschreibung

—> AKTION der Ableitung

Die Regel R6 entspricht nach meiner Auffassung nicht der Form der übrigen Ableitungsregeln und wird extra betrachtet. In Bild 2.11 steht der fettgedruckte Marker für den Abzuleitenden, * in K* für das abgeleitete Konzept , ()* für den zuletzt erzeugten Marker.

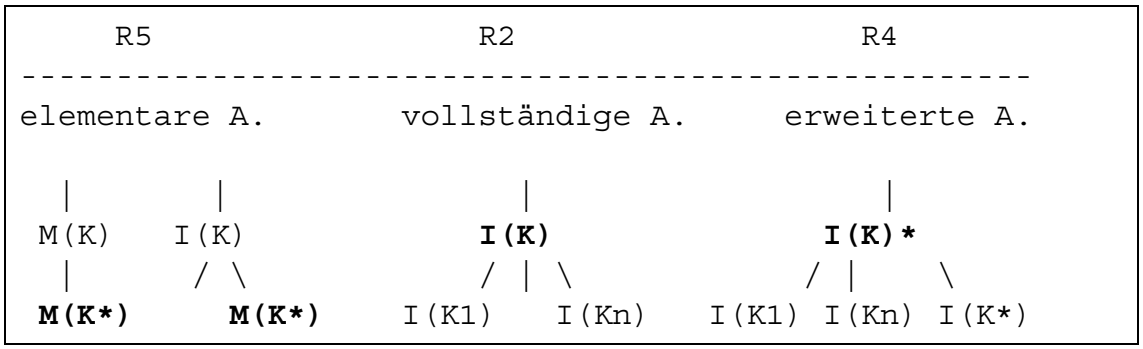


Bild 2.11 Metaregeln der partiellen Ableitung von Strukturmarkern

Stellvertretend für alle 5 Regeln sei die Metaregel R5 angegeben:

SITUATION: *Strukturbeschreibung enthält Marker M(K) oder I(K) des Konzepttyps K,*
die Wissensbasis enthält Konzepttyp K mit:*
K = KONKRETISIERUNG (K) oder = BESTANDTEIL (K)*
und Modalität mod(K) mit:
K(Arg) :-$_{(mod)}$ K(X),.. . (X in {(Arg)})*
*—> **AKTION**: binde die Modalität in M(K,mod) und erzeuge den Rollenbinder M(K*)**
bewerte den neuen Strukturmarker durch Berechnung der an die Kante gebundenen Attribute

Eine feinere Detaillierung von Subaktionen innerhalb der AKTION der Regeln R1 – R5 (auch die Systematik ihrer Numerierung) findet sich in den Basistexten [Sag90], [Kum92a].

Man entnimmt der AKTION eine – auch in der PROLOG-Ableitung erkennbare – Abfolge:

erst Strukturmarker erzeugen, *dann* bewerten.

In PROLOG beschränkt man sich auf den Binarismus der TRUE/FALSE-Bewertungen. In ERNEST können anwendungsspezifisch graduelle Bewertungen eingesetzt werden.

Sowohl der Inhalt als auch der Zusammenhang der Bewertungen, z.B. das in Abschnitt 1.6 beschriebene Übertragungsprinzip, gehen nicht explizit in die Formulierung der Metaregeln R1–R5 ein. Das Übertragungsprinzip stellt jedoch eine Art "Ausführungsbestimmung" zu den Metaregeln dar, von der nur in der Anwendungssituation begründete Abweichungen vorgenommen werden sollten.

Zur Anwendung von R5 ist festzuhalten, daß sie in ERNEST die Ausführung von Expandierungsketten bis zur Signalebene reguliert. Eine erfolgreiche Expandierung ergibt eine elementare Ableitung des Startmarkers. Aufgrund der besonderen Fassung der vollständigen Ableitung in ERNEST mittels R2, d.h. der notwendigen Bindung lediglich der obligatorischen Rollen, kann auch der bereits instantiierte Startmarker expandiert werden – zum Zwecke der Bindung optionaler Rollen. Für die Erweiterung von Instanzmarkern mit optionalen Bindungen gibt es eine zusätzliche Metaregel R3. Zusammen konstituieren die 3 Metaregeln den Mechanismus der *partiellen Ableitung in ERNEST*.

Die Aufwärtsableitung wird mit den Schemata in Bild 2.12 resümiert. R4 zeigt die Varianten zur Durchführung genau eines Aufwärts-Schrittes an. Er kann zur Instantiierung des Neuziels führen, z.B. wenn dessen Modalität nur eine Rolle enthält oder alle übrigen obligatorischen Rollen bereits gebunden sind. R1 verdient besondere Aufmerksamkeit. In der für den aktuellen Stand von ERNEST wesentlichen Darstellung von Kummert [Kum92a] wird auf Seite 56/57 die Anwendung der Metaregel erklärt. R1 zeigt die Verschränkung zwischen der Instantiierung von Markern kontextabhängiger Konzepte und einem Kontext-Marker. Soll die Kontextbindung eines ansonsten vollständig gebundenen Markers erreicht werden, so wird – wie bereits dargestellt – die (partielle) Aufwärtsableitung des Kontextmarkers vom Bestandteilsmarker aus vollzogen. Dafür wird ein extra Kontextmarker Ip() eingeführt.

2.3.2 Einführung aufgaben-orientierter Metaregeln

R1 – R5 sind ausschließlich in problemunabhängigen Termini formuliert. Für die R6–Formel trifft dies nicht zu. Sagerer ([Sag90], Seite 239) und Kummert ([Kum92a], Seite 65) geben zwar die Ausführungsbestimmungen der AKTION an, um Subziele (einschließlich des Analyseziels) als Neuziele in die Strukturbeschreibung einzuführen. Man betrachte die Definition der SITUATION für R6 :

'*wenn für ein Konzept A Werte für Attribute oder Analyseparameter bekannt sind*' [Sag90],
'*oder gemäß einer konkreten Analysesituation ein Konzept A als mögliches Analyseziel gegeben ist*' (Hinzufügung in [Kum92a]).

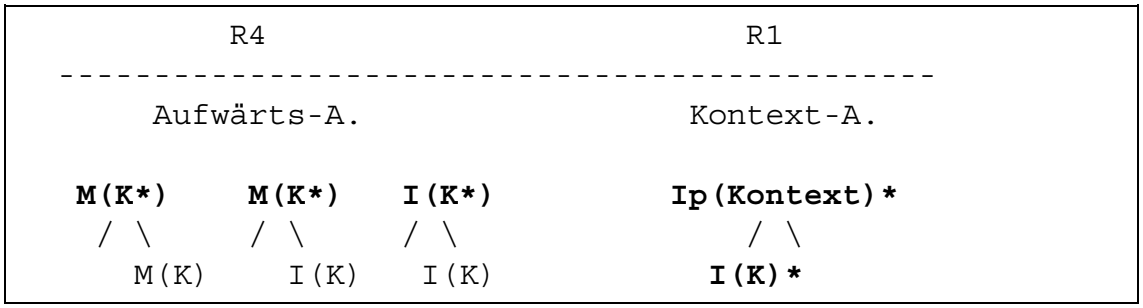


Bild 2.12 Metaregeln der Aufwärtsableitung

Kummert hat in [Kum92a] eine bestimmte Art und Weise des Wechselspiels von Basis- und Aufgabenkontrolle (Userroutinen) vorgestellt. Die Bestimmung der SITUATION für Aufwärtsableitungen erfolgt vollständig innerhalb der Aufgaben-Kontrolle anhand folgender Grundfragen:

1. *Wo liegt der Aktivationspunkt des Konzeptgraphen ?*
2. *Soll von diesem ausgehend ein neues Subziel gestellt werden ?*
3. *Welche Konzepte sollen das Subziel bilden?*

Das Bild 2.13 resümiert die Initialisierungsphase und Hauptschleife der Basiskontrolle von Kummert.

Sie stellt die verdichtete Darstellung des in [Kum92a](Seite 82) entwickelten Schemas dar, in dem der Platz der Userroutinen (mit Bezeichnern des Typs '*anwend*_funktionsname') explizit festgehalten ist.

I. initiale **User**-Neuziele stellen
II. Suchbaum, OFFEN-Liste initialisieren
III. weitere **User**-Initialisierung ('Faktenbasis' u.a.)
IV. Kontroll-Vorphase:
    je User-Neuziel Suchbaumknoten mit AKTION(R6) anlegen
    und in OFFEN ablegen
V. **User**-Funktion: SITUATION(n) (Knotenbewertung)

Bild 2.13 Basiskontrolle nach Kummert [Kum92a],

Zum Verständnis der Hauptkontrollschleife sei ein operativer Terminus der SITUATIONs-Analyse eingeführt:
ein Marker der Strukturbeschreibung heiße **instantiierbar**, wenn genau eine obligatorische Rollenbindung offen (geblieben) ist.

Das von Kummert gegebene **Basisschema** zur Plazierung der Userroutinen wird der inkrementellen Verarbeitung zugrunde gelegt. Es enthält 2 initiale Routinen, die speziell auf Inkrementalität (Kontrolle, Wortgraph) anzupassen sind. Die Userroutine zur Bestimmung des besten OFFEN-Knoten ist objektivierbar. Es kann durch die in Abschnitt 2.1.2 angegebenen Restschätzungen auf Inkrementalität abgestimmt werden. Ansonsten wird auf die schon bei Kummert [Kum92a] beschriebenen Maße des Bewertungstupels zurückgegriffen. In diesem Tupel gibt es Dummyfelder, die für zusätzliche numerische Situationsmarkierungen in Userroutinen verfügbar sind

und in der Userfunktion knotenbewertung() gesetzt werden. Diese Funktion wurde konsequenterweise in Initialisierungs- wie Hauptphase an das Ende gesetzt. Beide Phasen bedeuten je einen Schritt, sprich Step oder Takt, der noch zu beschreibenden virtuellen A*-Maschine, die einer Gestaltung eines interaktiven ERNEST-Regelinterpreters vergleichbar zu PROLOG das “Schrittmaß” vorgibt, worauf noch einzugehen ist. Jeder A*-Takt produziert eine gewisse Zahl neuer OFFEN-Knoten, deren Bewertung den “Schlußakkord” bildet.

Die Plazierung der Userfunktion *knotenbewertung()* bedeutet keine Änderung zu Kummerts Basisschema, der die Knotenbewertung jeweils am Ende der Basisroutinen *zielschätzung(), instantiierung(), expandierung()* eingekapselt hat.

In Basis-Routinen werden weitere für die inkrementelle Kontrolle wichtige Userroutinen des Typs SITUATION—AKTION plaziert, zusätzlich zu den in der Hauptkontrollschleife eingeführten 4 Funktionen. Diese 3 Basisroutinen gehören somit zum Basisschema, das als Rahmen der Plazierung von Userfunktionen zu kennzeichnen ist.

Die Routine ’zielschätzung’ realisiert Schleifen von Elementarschritten der Aufwärtsableitung je Subziel, ausgehend von einem Startmarker. Der Terminus ’Zielschätzung’ hat mindestens seit der Realisierung der Kontrolle von Kummert für das ICZ-System größeren Gebrauch in der Darstellung der ERNEST-spezifischen Kontrolle gefunden. Mit ’Schätzung’ meint man beides:

1. die SITUATIONs-Analyse zur Bestimmung von Subzielen,
2. die Form der Aufwärtsableitung selbst im Gegensatz zu einer Einzel-Expandierung.

Der Vergleich mit PROLOG-Prädikationssystemen, der im 1. Kapitel behandelt wurde, verweist auf die Relevanz beider Aspekte und legt es nahe, sie gesondert zu behandeln und zu bezeichnen:

- *Subziel-Bestimmung* in der Perspektive inhaltlich bestimmter User-Aktionen,
- *Aufwärtsableitung* als formale Ableitungsmethode und Entwicklung der Strukturbeschreibung.

Die Zusammenfassung der im Basisschema interagierenden Elemente der Steuerung gibt der folgende Satz im ERNEST-Manual [Hil91] an:

’Durch die sechs Regeln, die lokale Steuerungsinformation (Priorität und Vorrang) und die Festlegung von Zielkonzepten ist die Instantiierungsreihenfolge aller Konzepte eines Netzes vorgegeben.’

Die Basiskontrolle gründet sich auf *Metaregeln* zur Analyse der Strukturbeschreibung und verwendet vorcompilierte (also situations-unabhängige) *Prioritäten* für Konzepte, Kanten. Sie gibt aber auch der anwendungsspezifischen Situationskontrolle Platz. Das bisherige Basisschema und seine Implementation in den aktuellsten Applikationen war auf das Zusammenspiel dieser Komponenten hin zu untersuchen. Welcher Stellenwert den *Zielen* zukommt, läßt sich an der Hauptkontrollschleife in Bild 2.15 eindeutig ablesen. Sie sind das bestimmende erste Glied der Rangfolge:

```
Ziele   >   Metaregeln   >   Prioritäten
```

Bild 2.14 Trinität der Steuerelemente der ERNEST-Basiskontrolle

Untenstehend wird die Rangfolge der Steuerelemente (1) – (3) im Basisschema zur Sicherung der inkrementellen Verarbeitung dargestellt:

1. Wenn die User-Analyse neue Subziele ergibt, so wird die AKTION Aufwärtsableitung erzwungen.
   SUBAKTION: Da (1), (2) darauf keinen Einfluß nehmen, muß die Auswahl der Netzpfade (alle oder ausgewählte) durch eine Userfunktion inkrementell gesteuert werden.
2. Wenn die Metaregeln instantiierbare Strukturmarker ausweisen, so bekommt dieser die höchste Aktionspriorität und es wird die AKTION Instantiierung eingeleitet.
3. Wenn lediglich Expandierungen möglich sind (im allgemeinen mehrere), so wird entsprechend der vorcompilierten Prioritäten ein zu expandierender Marker ausgewählt:
   SUBAKTION: vergleichbar zu 1. muß die Wahl der zu expandierenden Rollen ebenfalls durch eine (neu im Basisschema zu plazierende) Userfunktion inkrementell steuerbar sein.

Hauptkontrollschleife (solange OFFEN-Knoten existieren):
1. **User**-Funktion: bester OFFEN-Knoten ?
2. besten Knoten n aus OFFEN entfernen
3. **User**-Funktion: SITUATION(n) (Analyseziel erreicht ?)
4. nein: **User**-Funktion: SITUATION(n) (neue Subziele stellen ?)
   ja: goto 5., nein: goto 7.

5. **User**-Funktion: SITUATION(n) (Subziele ?)
6. Basisroutine ’zielschätzung’ auf Subziele anwenden
   goto 8

7. Metaregel-SITUATION(n): (existiert instantiierbares Subziel ?)
   –> Basisroutine ’instantiierung’,
   sonst ’expandierung’
   erst obligatorischer, dann optionaler Bindungen

8. **User**-Funktion: SITUATION(n) (Knotenbewertung)

Bild 2.15 Hauptkontrollschleife von Kummert [Kum92a]

2.3.3 Analysephasen des Konzeptgraphen

Tiefenstrukturparsing ist immer zugleich Gegenstandsbildung im Bereich der Eingabedaten – in ICZ die Suche nach dem *bestbewerteten und linguistisch-zulässigen Subpfad des Wortgraphen*. Über der assoziierten Wortkette spannt sich ein Suchraum möglicher Prädikationen auf. Die Phasen der linguistischen Suche sind durch markante Zwischenergebnisse der Prädikation dieser Kette bestimmt:

Phase I ist abgeschlossen, wenn alle Signalintervalle wie in Bild 2.16 gezeigt – bearbeitet wurden.

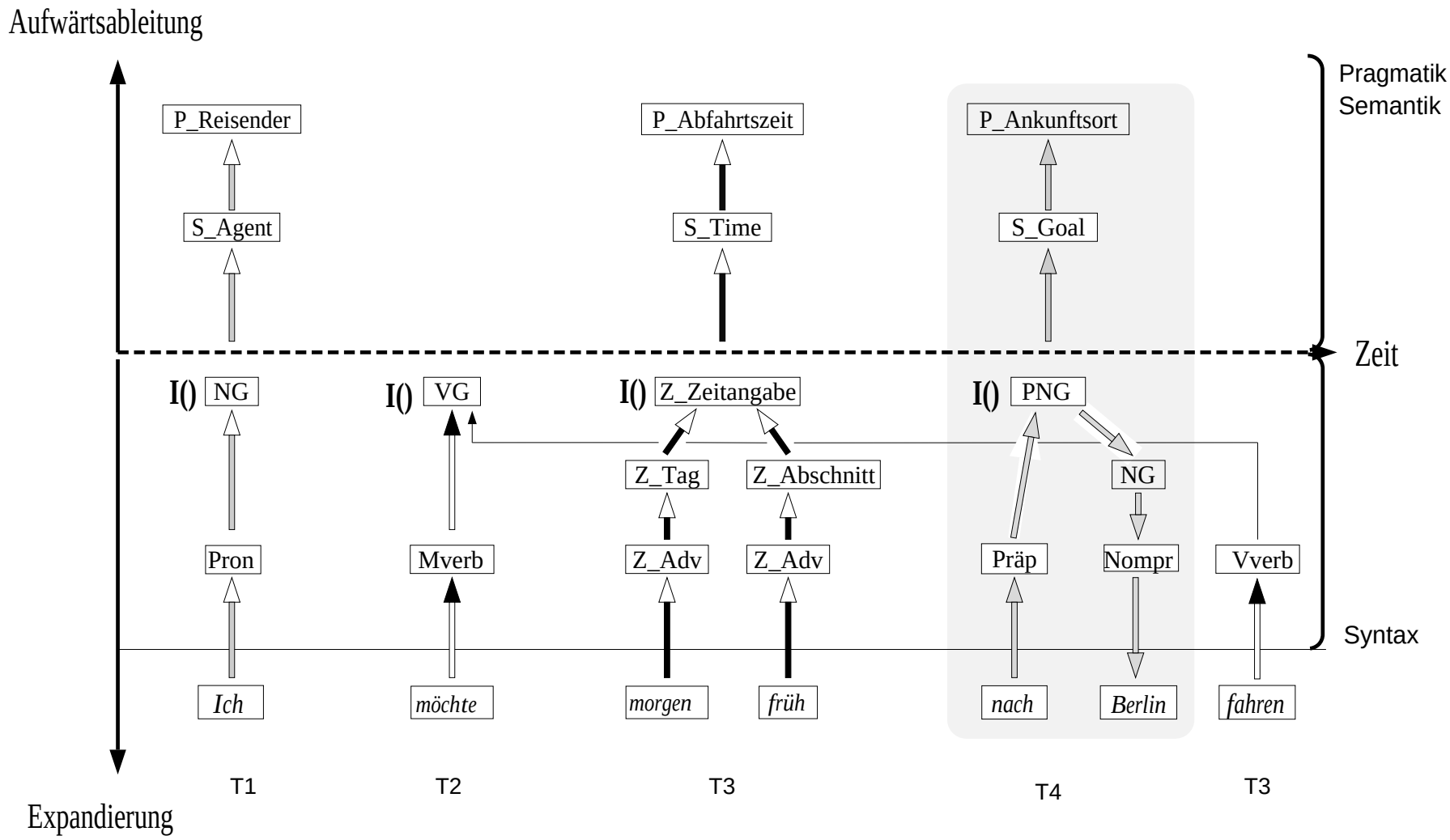


Bild 2.16 Analysephase I

Bild 2.16 zeigt eine Strukturbeschreibung am Ende von Analysephase I. Die verschiedenartige Pfeile dienen nur zur Unterscheidung der Ableitungsbäume. Ihre Richtungen korrespondieren zur Bedeutung der Achsen.

Das in Bild 2.16 gezeigte Analysebeispiel erfüllt die:

***Kriterien der** idealen **Termination** von Phase I:*

1. die assoziierte Wortkette des Konzeptgraphen ist signalüberdeckend,
2. jedes Kettenglied ist in eine Konstituente eingeordnet,
3. alle Konstituentenmarker sind instantiiert.

Alle 3 Kriterien sind als nicht immer erreichbare Norm aufzufassen. Die abgeschwächten Kriterien waren durch die Analyse von Datenmaterial zu finden (siehe Kapitel 3).

Bereits vor Beendigung dieser Phase können die in Bild 2.17 zwischen beiden Strichlinien plazierten semantischen und pragmatischen Rollen der Konzepte (aufwärts) abgeleitet worden sein. Das ist nur möglich, wenn die Usersteuerung schon in Phase I die entsprechenden Subziele stellt.

Im Abschnitt “Anwendungsspielräume der Aufwärtsableitung” wurde das möglichst frühzeitige Stellen von Subzielen der Pragmatik angesprochen, um deren (lexikalische) Restriktionen ausnutzen zu können. Die Usersteuerung kann dieses Problem fallweise lösen, indem jede Konstituente für sich betrachtet wird. Wäre ein formell netztechnisches Kriterium gewünscht, so könnte man stattdessen definieren:

User-Regel ’semantische/pragmatische Subziele’ –
Subziele für Konstituentenmarker sind erst **dann** *zu stellen,*
wenn *diese bereits instantiiert sind.*

Den Endpunkt von ***Phase II*** bildet die Instantiierung eines pragmatischen Verbrahmens, allgemein eines Kontextes. Dazu sind zunächst die restlichen Aufwärtsableitungen ausgehend von Konstituenten zu realisieren. Mit Blick auf Bild 2.16 soll ein Hilfsterminus zur Kennzeichnung der Struktur von Konzeptgraphen definiert werden:
alle Pragmatik-Marker im Konzeptgraphen spannen je eigene Ableitungsbäume auf. Sie heißen **die Ableitungsbäume** innerhalb der Strukturbeschreibung der ICZ-Basiskontrolle. Im Bild sind dies die (bis auf T3 vollständigen) Bäume T1–T4. Die zu den Bäumen gehörigen pragmatischen Konzepte

P_REISENDER, P_ABFAHRTSZEIT und P_ANKUNFTSORT

heißen ihre **Wurzeln**.

So gesehen besteht die für Phase I und II charakteristische inkrementelle Verarbeitung darin, zu den Eingabedaten des Wortgraphen die linguistisch zulässigen Ableitungsbäume zu konstruieren.

Auch Verbrahmen können als Wurzeln einen Ableitungsbaum aufspannen. Kann man ihn auf kürzestem Wege durch Aufwärtsableitung von einem Startpunktmarker (SY_VG [X]) erzeugen (wie T1,T2,T4 im Beispiel)? Dazu hätte man aus der Subkette X den passenden Verbrahmen abzulesen.

Erinnert sei an die in Bild 1.26 (siehe Abschnitt 1.5 von Kapitel 1) exemplarisch dargestellte Attributbeschreibung der Konzeptdeklaration in ERNEST. Der dazugehörige Subframe RESTRIKTIONSBEREICH für das Attribut der Kante

S_VR_FAHREN(X) :- (konkretisierung) SY_VG(X)

könnte die im Worterkenner-Lexikon verzeichneten Wortnummern der zulässigen Vollverben in X von SY_VG(X) festschreiben:

```
ATTRIBUT( )                                    Erklärung
-------------------------------------------------------
....
RESTRIKTIONSBEREICH( EINZELWERTE )      Wortnummer
-------------------------------------------------------
    WERT( 'fahren' )                     erlaubte
    WERT( 'faehrt' )                    Vollverben
-------------------------------------------------------
```

Das obige Bild zeigt den einfachsten Fall der Restriktion: Aufzählung der lexikalischen Wortnummern. Die Wahl des systematischen Orts im deklarativen Bereich, heiße die **WNR-Restriktion** (WNR=Wortnummer). Sie betrifft nicht nur Subziele von Verbalgruppen. Ideale Basis der Usersteuerung wäre überhaupt die explizite Plazierung aller Restriktionen der Gegenstandsbildung im deklarativen Bereich. Die Codierung im Programm könnte dann die Regelform annehmen.

User-Regel 'Verbrahmen':
Bestimme die WNR(vollverb(X)) im Marker(SY_VG, [X])
finde den zutreffenden Verbrahmen durch Vergleich mit den WNR-Restriktionen der Verbrahmenkonzepte der Wissensbasis .

Die Darstellung für den S_VR_FAHREN ist ein Vorschlag. Er hätte bereits beim Entwurf der Wissensbasis realisert werden müssen, als noch nicht im Detail an die inkrementelle Verarbeitung gedacht wurde. Klarerweise ist die Restriktion zur Analysezeit an sich vorhanden – prozedural verschlüsselt in den Funktionen der Attributebene. Würde nur eine geringe Zahl von Verbrahmen vorliegen, so wäre die Universalmethode eines jeden Prädikationssystems anwendbar: expandiere sämtliche Verbrahmen bis zum Marker SY_VG und berechne die Attibute, in denen die WNR-Restriktion verschlüsselt ist. Ein Blick selbst auf kleine Domänen zeigt, wie schnell die Zahl der notwendigen Verbrahmen steigt. Das ist auch ein Hauptproblem moderner maschineller Textübersetzungssysteme [Lea94]. Sie erzeugen eine dem Konzeptgraphen ähnliche Strukturbeschreibung, die sogenannte Interlingua, in der dependanz-grammatische Relationen wie Verbrahmen die gleiche tragende Rolle spielen wie in ICZ. So gesteuerte automatische Übersetzung einer technischen Gebrauchsanweisung erfordert z.B. die Modellierung sämtlicher handlungsrelevanter Verben des Textes. Man nehme zum Vergleich solch Textbeispiel zur Hand.

Sind die WNR-Restriktionen zur Analysezeit nicht decodierbar, so bietet sich nachträgliche 'Deklaration' in der Userfunktion an, wenn man denn mit Aufwärtsableitung arbeiten will. Aus der Analyse der an den Kanten von Verbrahmen

zu Verbalgruppen hängenden Attributfunktionen geht hervor, daß die Bindung der Rolle(verbalgruppe) für jeden Verbrahmen

VR_*nennform*

aus der Nennform des Verbs oder ihren flektierten Formen erhalten werden kann.

Es sei noch ein anderes Problem von Phase II betrachtet. Liegen z.B. durch einen Versprecher oder eine Repetition des Sprechers mehrere Marker des gleichen pragmatischen Konzepttyps (gegebenfalls unterschiedlichen Ableitungsgrads) vor, so hat die Regel eine gewisse psychologische Motivierung:

User-Regel 'partiell identische Ableitungsbäume'

wenn *der Konzeptgraph Ableitungsbäume mit identischem Wurzelmarker enthält,* **dann** *wähle stets die Wurzel zur Bindung des Kontextes, deren assoziierte Wortkette im Signal später auftritt.*

Die Regel kann aber nur einen Vorrang bedeuten, der noch falsifiziert werden kann. Es können Erkennungsfehler vorliegen. Aus Bild 2.17 geht eindeutig hervor, daß allen pragmatischen Rollenkontexten ein entsprechender semantischer Kontext zugeordnet ist. In gewissen Sätzen fehlt der Verbrahmen. An seiner Stelle kann ein Nomenrahmen die Funktion der Einbindung der semantischer Bestimmungen, der Tiefenkasusrollen, übernehmen. Als Beispiel sei der folgende Satz angegeben:
`[ich,moechte,eine,Direktverbindung,nach,Bonn].`

Die Gegenstandsbildung in Phase I, welche der Prädikation einer Präpositionalgruppe zugrunde liegt ergibt die Subkette

X = [eine, Direktverbindung, nach, Bonn].

Aus dem Marker SY_VG([X]) ist in Phase II der richtige Nomenrahmen abzuleiten. Wiederum sind dazu geeignete WNR-Restriktionen erforderlich. Endet Phase II mit der Bindung von Kontext schlechthin —wie die resümierende Darstellung in Bild 2.18 zeigen will, so ist die möglichst frühzeitige partielle Ableitung z.B eines Verbrahmens doch nicht auszuschließen. Für entsprechende frühzeitige Restriktionen der Wurzeln weiterer Ableitungsbäume liegt dann der Ansatzpunkt in Restriktionen der Kontext-Modalität. Ganz formell definieren läßt sich dazu die:

User-Regel 'Frühe pragmatische Bindung':

wenn *der Konzeptgraph bereits einen Kontextmarker mit gesetzter Modalität enthält,* **dann** *wähle von diesem Moment an markierte Wurzeln von Ableitungsbäumen im Konzeptgraph als potentielle Rollenträger des Kontext.*

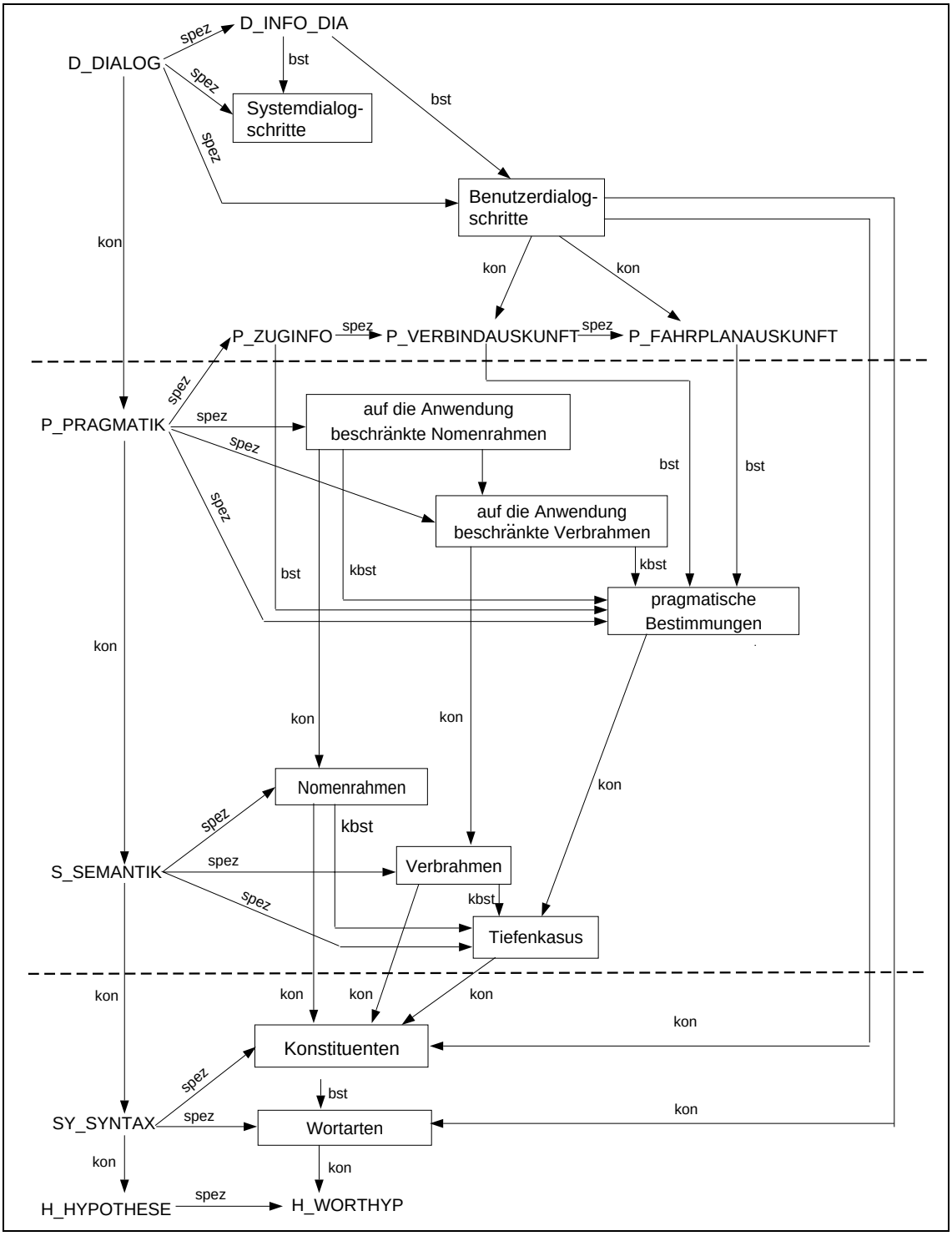


Bild 2.17 Kondensation des semantischen Netzes “Zugauskunft”

Zur Realisierung der entsprechenden AKTION wird eine neue Userfunktion benötigt, welche den Expandierungsmechanismus auf die relevanten Kontext-Rollen der Wurzelmarker hinlenkt.

Man vergleiche die erhaltene Phaseneinteilung mit der von Kummert in [Kum92a] (Seite 183f.) gegebenen Beispielanalyse, die in Bild 2.19 auf die wichtigsten Zwischenetappen der Strukturbeschreibung verdichtet wird. Die Analyse startet mit einem initialen Neuziel (SY_NPR=nomen proprium). Der Wortgraph ist vollständig

bekannt. Das Neuziel soll Wortkandidaten mit besonderer pragmatischer Relevanz binden – Städtenamen, die für Intercitybahnhöfe stehen. Insgesamt finden genau 2 Aufwärtsableitungen (in den Teilbildern a), e) schraffiert dargestellt) statt:

1. in a) : der Wurzel P_ANKUNFTSORT für I(SY_NPR,[Hamburg]),
   (zur Beachtung: in a) – e) wird nur die Ableitung von P_ANKUNFTSORT expandiert).
2. in e): eines Analyseziels zum Wurzelmarker.

Die Phase zwischen beiden Aufwärtsableitungen besteht durchgängig aus Expandierungen. In Teilbild b) wird die Top-Down-Propagierung der in der Wurzel deklarierten WNR-Restriktion über die Präposition durch Expandierung bis zur Signalebene 'verlängert'.
In Teilbild c) folgt nach dem restringierten Zugriff auf den Wortgraphen die Instantiierung hierarchie-höherer Marker (mit schwarzen Pfeilen markiert).
In Teilbild d) steht die Prädikation eines Kontextes im Vordergrund. Sie ist sehr spezifisch auf die nicht-inkrementellen Voraussetzungen der Kontrolle von Kummert zugeschnitten. Die Erläuterung wird im Abschnitt 2.3.4, anhand von Bild 2.20 gegeben. Der Kontextmarker wird (notwendigerweise durch eine Usersteuerung) als Neuziel eingeführt, zunächst durch den Wurzelmarker gebunden. Dann wird er mit mehreren Expandierungen an die Signalebene gebunden.

In Teilbild d) erfolgt die partielle Instantiierung zunächst des semantischen Kontextes. Der hakenförmige Pfeil soll die Verschränkung des Ableitungsgrads von M(SY_GOAL) mit dem des Kontextmarkers anzeigen. Dieser geht notwendig die Instantiierung innerhalb des Ableitungsbaums des Kontextes voraus[11]. Das zeigt exemplarisch die Behandlung solcher Meta-Prädikate (wie Kontexte) in ERNEST:

Metaregel 'Verschränkung Rolle/Kontext'
die Rolle kann nur **dann** *instantiiert werden,* **wenn** *ein sie umfassendes Rollenensemble wenigstens partiell instantiiert ist.*[12]

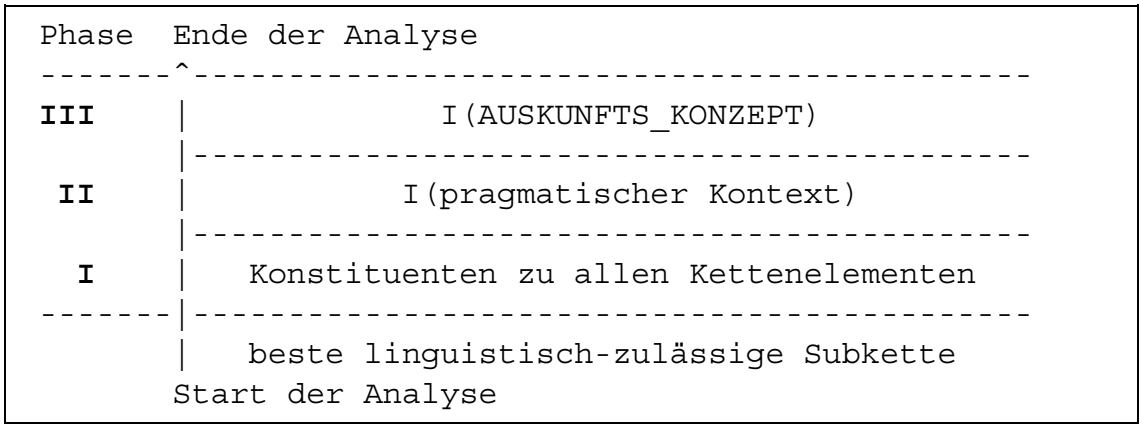


Bild 2.18 Phaseneinteilung der Analyse

Um den Endpunkt der ***Phase III***, die Instantiierung eines Analyseziels zu erreichen, ist – wie in Teilbild 2.19d) (schraffiert) – ein 'aussagekräftiger' Startpunkt zur Aufwärtsableitung zu bestimmen. Hierzu wird eine ergänzende Faktorisierung des

[11] die in [Kum92a] formulierte Regel R1 zeigt die spezifische SITUATION nicht so deutlich an
[12] zur Frage nach der Simulation in PROLOG hat der Verf. keine Lösung

semantischen Netzes benötigt, die ausweist, welche pragmatischen Konzepte überhaupt obligatorische Rollenträger von Auskunftskonzepten und welche als deren differentia specifica einsetzbar sind:

User-Regel 'Analyseziel'

- notwendige Bedingung -

wenn *der Konzeptgraph einen instantiierten Pragmatik-Marker enthält, dessen Konzepttyp obligatorisches Bestandteil einer Modalität eines Analysezieles ist,* **dann** *gibt es einen Startpunkt für Analyseziele*

- hinreichende Bedingung -

wenn *der Konzeptgraph einen Pragmatik-Marker enthält, dessen Konzepttyp notwendiges Bestandteil nur einer Modalität eines Analysezieles ist,* **dann** *wähle genau dieses Paar (Startpunkt,Analyseziel) zur Aufwärtsableitung*

Im ICZ-Netz (siehe Bild 2.17) finden sich die zutreffenden Konzepte als Gruppe der ***pragmatischen Bestimmungen***[13].

Die Bildfolge 2.19 (es stehen abwärts gerichtete Pfeile für expandierung(), aufwärtsgerichtete für instantiierung(), die restlichen für Aufwärtsableitung) soll ein Beispiel der Analyse von Kummert zum Vergleich anbieten.

Die Beispielanalyse von Kummert entwickelt sich sozusagen "aus einem Stück"[14], sprich aus dem Ableitungsbaum von P_ANKUNFTSORT, dessen Instantiierung die *notwendige und hinreichende Bedingung* für ein Analyseziel erfüllt. Die Ableitung des Baumes bis auf Kontextverschränkung stellt hier Phase I dar. Die in Teilbild 2.19d) verkürzte Folge der Kontextexpandierungen[15] – Phase II – sichert die Instantiierung des Wurzelmarkers. Die Expandierungs–Aktionen werden in Kummerts Basisschema konsequent aus dem Ableitungsgrad des je aktuellen Priors des Konzeptgraphen bestimmt. Am Ende der Phase II läßt sich die letzte Userregel anwenden. Phase III entspricht der inkrementellen Verarbeitung.

Die im Basisschema (siehe Bild 2.15) geforderte Userfunktion zur Abbruchbehandlung wurde als Ausstieg aus dem A*-Algorithmus eingebaut und prüft die Beendigung von Phase III ab:

User-Regel 'Analyseabbruch'

1. scharfe Bedingung:
   Auskunftskonzept ist instantiiert
2. schwache Bedingung:
   Analyseziel wurde aus 'notwendigem und hinreichendem' Marker (aufwärts) abgeleitet und alle restlichen Wurzelmarker (soweit möglich) gebunden

---

13 alle pragmatischen Konzepte außer Kontexte, anders gesagt: alle pragmatischen Rollenträger

14 im Unterschied dazu die inkrementelle Verarbeitung: separate Entwicklung aller Ableitungsbäume in Phase I/II

15 eine erweiterte Darstellung der Strukturbeschreibungen enthält [See92]

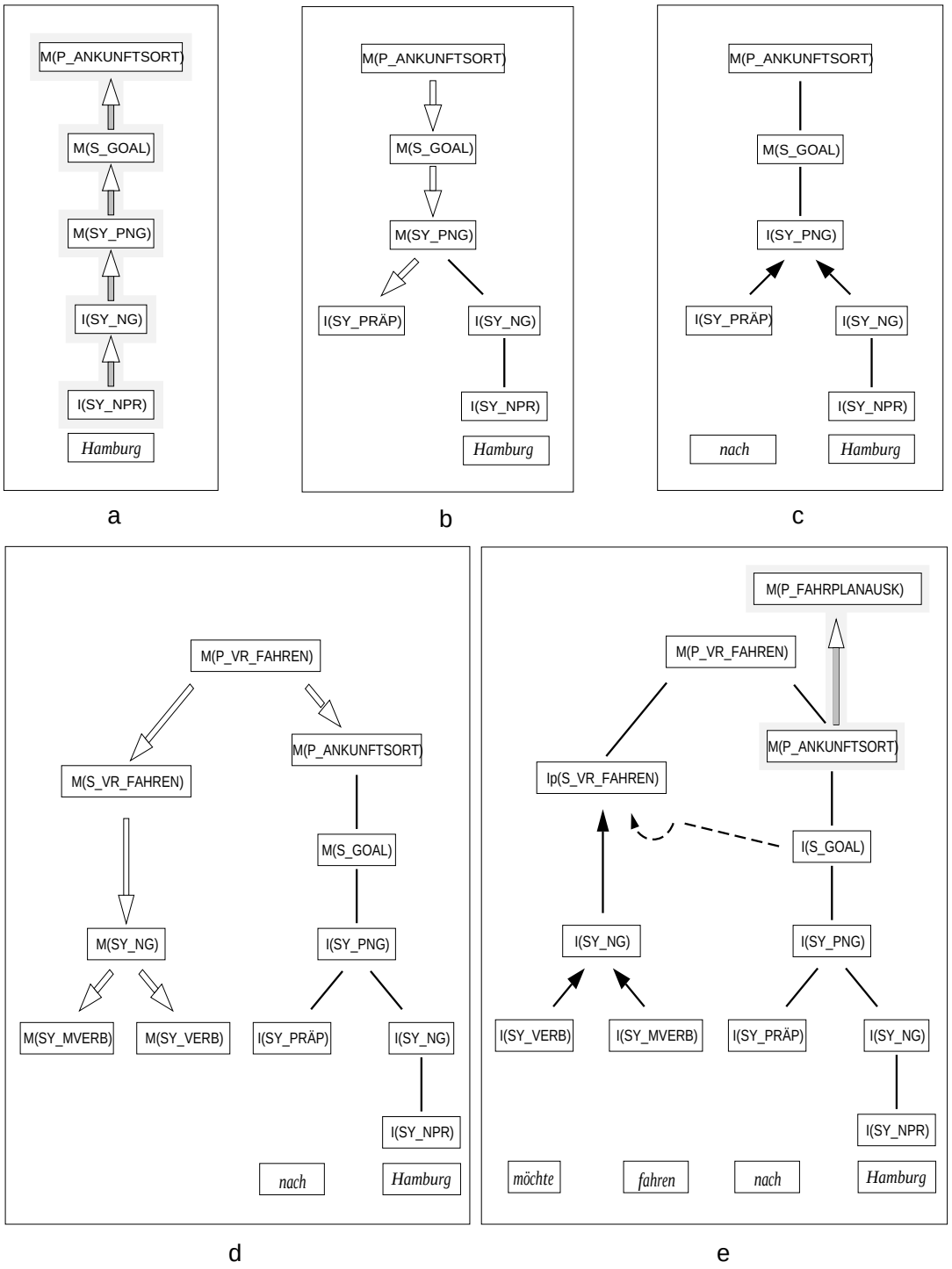


Bild 2.19 Anwendungsbeispiel der Kontrolle von Kummert

2.3.4 Aktionen der inkrementellen Kontrolle

Die Reduktion des Wortinputstroms kann nur die Aufgaben-Kontrolle behandeln. Die Basiskontrolle kann nur den problem-unabhängige Teil der Aufgabe lösen, denn sie ist in rein formellen, netz-technologischen Termini konzipiert und programmiert.

Im Basisschema nach Kummert und in der Rangfolge der Steuerelemente (1) – (3) findet sich noch keine Userfunktion, die explizit für Ereignissteuerung vorgesehen ist. Deshalb wird vorgeschlagen, generell zwischen Schritt 1. und die folgenden

2.+3. der in Abschnitt 2.3, Bild 2.14 dargestellten Trinität der Steuerfolge eine zusätzliche Schnittstelle einzufügen. Es wird eine Plazierung gewählt, welche die Wirksamkeit der bisherigen Steuerelemente aufrechterhält. Alle im Basisschema von Kummert eingesetzten Basisroutinen realisieren Expandierungen im verallgemeinerten Sinne[16]. Diese ergeben sich aus dem Ableitungsgrad eines mittels der Steuerelemente (1) - (3) sowie der Aufgabenkontrolle zu bestimmenden **Aktivationspunktes** oder **aktiven Markers** des Konzeptgraphen und werden durch folgende Subroutinen[17] der Basiskontrolle realisiert:

je nach Ableitungsgrad, sprich offenen Marker-Bindungen,

1. instantiierung(): genau eine obligatorische
2. expandierung(): mehrere obligatorische
3. erweiterung(): nur optionale
4. spezialisierung(): keine

Zunächst sei das inhaltliche Problem betrachtet. Die Wissensbasis kann kontinuierlichen Konzepten das Merkmal KOHAERENZ zuschreiben und damit die inkrementelle Verarbeitung des Signals verschärfen. Dazu betrachte man die Strukturbeschreibung in Bild 2.20. Die bisher assoziierte Wortkette ist [ich, moechte]. Die Strukturbeschreibung ist bis auf die verkürzt dargestellte Aufwärtsableitung von M(P_REISENDER, [ich] wiedergegeben. Welche AKTION hat man an diesem Punkt zu wählen? Entsprechend der in Abschnitt 2.3, Bild 2.14 angegebenen Trinität der Steuerfolge sind folgende Schritte der Situationsanalyse durchzuführen:

1. Sind neue Subziele zu stellen ? Nein – denn M(P_REISENDER) muß an einen Verbrahmen-Kontext gebunden werden. Also müßte erst M(SY_VG) abgeleitet werden. M(SY_VG) ist partiell abgeleitet, aber allein aus der Bindung des Modalverbs [moechte] kann nicht auf den Verbrahmen geschlossen werden, z.B. VR_FAHREN (für das Vollverb [fahren]).
2. Es gibt keinen instantiierbaren Marker im Konzeptgraph.
3. Unter allen offenen Subzielen ist M(P_REISENDER, [ich]) der Prior, da sein Konzept höchste Priorität in der Wissensbasis besitzt – es steht den Analysezielen nahe. Die Metaregeln weisen diesem Marker die AKTION Expandierung zu. Da nur die Kontextbindung offen ist und deren Aufwärtsableitung die gleiche Expandierung bedeutet, gilt wiederum die Argumentation zu 1.

Die Expandierung des zweiten offenen Subziels kommt in der inkrementellen Verarbeitung nicht in Frage, falls es die (im Deutschen) besonders häufige 'nicht-kohärente' Modalität besitzt. Das Signum '(?)' soll anzeigen, daß diese Expandierung in den vom Wortgraphen noch nicht belegten Signalbereich führen könnte.

In der Basiskontrolle nach Kummert tritt das Problem nicht auf, weil der vollständige Wortgraph vorausgesetzt wird. Das entnimmt man der in [Kum92a] gegebenen Beispielanalyse und der Implementierung:

Der Marker M(P_ANKUNFTSORT, [nach, Hamburg])

ist in der Strukturbeschreibung der Beispielanalyse (S.187) der Startpunkt zur Aufwärtsableitung von P_VR_FAHREN.

---

16 im Sinne der in Kapitel 1 als Vergleichsmaßstab gesetzten UP-Maschine

17 *Routinen, Funktionen* sind Computerprogramme, was durch die kleingeschriebene Funktionsnamen angezeigt werden soll

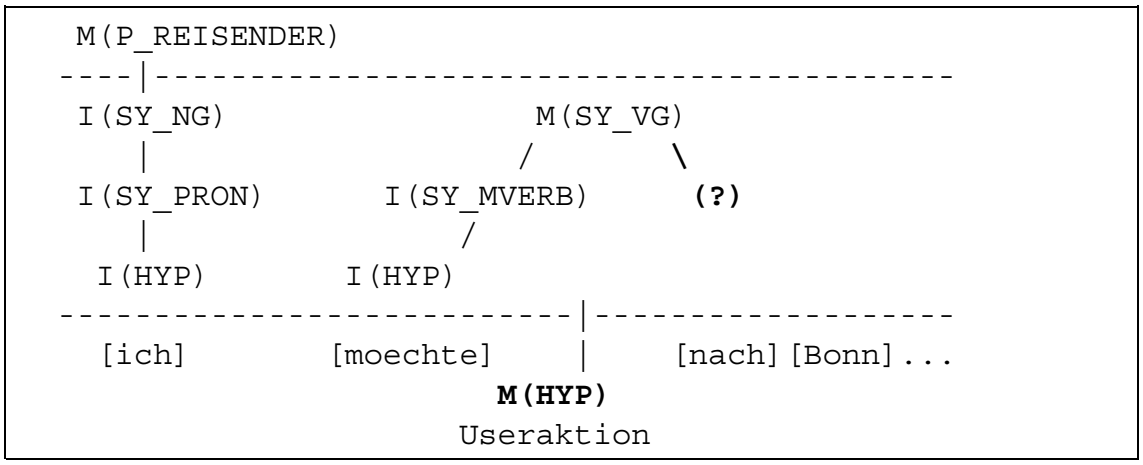


Bild 2.20 Funktion des Neuziels M(H_WORTHYP)

Aus der Codierung der in Kummerts Text [Kum92a] ausgewiesenen Routine etabliere_kontext() lese ich folgende Methode ab. Unter Voraussetzung des vollständigen Wortgraphen werden alle Wortkandidaten der Lexikon-Kategorie 'Wortart Vollverb' zusammengetragen. Die Routine ordnet ihnen je einen Verbrahmen zu. Für jeden wird ein Nachfolgerknoten im Suchbaum expandiert.

Inkrementalität erfordert eine andere Lösung, die in Bild 2.20 bereits angedeutet wird. Der gesamten Expandierungsphase eines A*-Taktes wird eine User-Routine vorgeschaltet. Sie bestimmt situationsabhängig die AKTION:

Setzung eines Neuziels M(HYPOTHESE)
-> Aktivationspunkt der Strukturbeschreibung, d.h
1. Neuziel wird Prior
2. AKTION(Prior) =INSTANTIIERUNG
-> Zugriff zur Datenbasis

Die Formulierungen, so die Wahl eines verallgemeinerten HYPOTHESE-Konzepts (statt H_WORTHYP), sollen die Generalisierbarkeit von Ereignissteuerung anzeigen. Linguistische Datenbasis ist der Wortgraph. Gleiches beabsichtigt auch die Formulierung der untenstehenden

Kriterien der Situationsanalyse:

i. Welches offene Subziel ist Prior?
ii. Welches offene Subziel hat höchste Priorität in der Syntax-Schicht?
iii. Welcher Marker entspricht dem Prinzip der 'Rechtsassoziation' von Kimball? ([Kim73], der '*Lowest Right Nonterminal*', abgekürzt: LRN-Marker)

Zur Prüfung der Expandierbarkeit dieser Marker in absteigender Folge (um die Geltung der bisherigen Steuerelemente zu erhalten) und abhängig von der Analysephase (I-III) dient die:

User-Regel 'Ereignis':

wenn *keines der Subziele i.-iii. (mit Rücksicht auf Inkrementalität) expandiert werden kann,* **dann** *setze das Neuziel M(HYPOTHESE)*

2.4 Einführung eines neuen Basisschemas der ERNEST-Kontrolle

Zur Gewährleistung der inkrementellen Verarbeitung ist es notwendig, ein neues Basisschema der Kontrolle, sprich eine Erweiterung des alten Schemas, einzuführen. Aufgabe des Basisschemas ist es, die Plazierung der Userfunktionen und ihre problem–übergreifende Steuerungsfunktion sichtbar zu machen.

Damit das Basisschema der Kontrolle und ihre wichtigsten Routinen in algorithmischer Form (als Struktogramm) geschrieben werden können, werden formale Bezeichnungen eingeführt. Die Typen von Bezeichnern in untenstehender Liste sind im Sinne von Datentypen von Programmiersprachen zu verstehen. Die Typen selbst als auch ihre einzelnen Datenelemente werden der Kürze halber mit demselben Symbol ausgedrückt. Mit User*funktionsnr*.> werden im folgenden die im Basisschema geforderten Userfunktionen markiert und durchnumeriert. {} notiert Mengen, Ø die leere Menge, ({},{}) die Knoten- und Kantenmenge eines Graphen. Ikone **U:, B:** dienen der Visualisierung der Arbeitsteilung zwischen User- und Basis-Routinen. Die Übersicht beginnt mit den Typen:

```
K                          Konzept
V                          Kantenmenge
KG     ({K}, V)            Konzeptgraph
SB     ({KG},V)            Suchbaum aller KG
OFFEN ({KG})               Liste der offenen KG
Zuordnungen  ------------------------------------
NF()    K    ->  K'        Nachfolger in KG
        KG   ->  KG'       "     in SB
VG()    K    ->  K'        Vorgänger in KG
        KG   ->  KG'       "     in SB
MOD()   K    ->  Ident     Modalität von K
GRAD()  K    ->  Level     Hierarchie-Ebene (H_,..,P_)
K*()    KG   ->  K         zuletzt abgeleitetes Subziel
Z()     KG   -> {K}        Menge der aktuellen Subziele
Kontrollstrukturen    ----------------------------
ALLE ({})                  Elementweise Durchlaufung
FALL (ident)               Fallweise Abarbeitung je nach
                           Situations-Identifikator
```

2.4.1 virtuelle "A*-Maschine"

Die in Bild 2.22 gezeigte A*-Kontrollschleife ist etwas knapper gefaßt als in der Darstellung von Kummert ([Kum92a], S.82), bewahrt aber die Kontinuität zum alten Schema.

Folgende Initialisierungen werden vorausgesetzt: der Suchbaum besteht aus einem leeren Wurzelknoten und einer initialen Aufspaltung seiner Nachfolger entsprechend der Anzahl der durch die Aufgabenkontrolle vorzugebenden initialen Neuziele:

```
ICZ-Steuerung (nach Kummert):   M(SY_NPR), M(SY_NOMEN)
inkrementelle Steuerung:        M(H_WORTHYP)
```

Bild 2.21 Initiale Neuziele der Anwendung “Zugauskunft”

Die Wahl des initialen Neuziels bei Kummert [Kum92a] verwendet Wortart-Konzepte für Lexeme besonderer pragmatischer Relevanz und ist domänen-spezifisch. Dagegen ist für die inkrementelle Verarbeitung die Vorgabe des primitiven Netzwerk-konzepts zwingend. Die anstartende Basiskontrolle wählt die Neuzielmarker zum automatischen Aktivationspunkt einer Expandierung. Damit wird in jedem Fall der erste, direkte Zugriff auf die Datenereignisse ausgelöst, was insbesondere für die inkrementelle Verarbeitung als einzig sinnvoller Startprozeß infrage kommt.

Die Knoten zu den initialen Neuzielen bilden auch die Initialisierung der OFFEN-Liste, wenn die in Bild 2.22 als Struktogramm dargestellte “A*-Maschine” anläuft. Den mit der Zielgebung von Bild 2.21 eindeutig bestimmten Startknoten der OFFEN-Liste für die inkrementelle Steuerung bezeichne ich als ***Initialknoten***. Er wird der Implementierung der inkrementellen Kontrolle als globale Information verfügbar gemacht.

Die A*-Maschine expandiert je Durchlauf den ***aktiven Suchbaumknoten*** — das ist ihre Hauptfunktion. Da die OFFEN-Liste nur die offenen, sprich nicht-expandierten Knoten registrieren soll, beginnt die Exekution dieser Maschine mit dem Löschen des aktiven Knoten, sofern er aus der OFFEN-Liste genommen war.

Die inkrementelle Verarbeitung erzwingt eine Unterscheidung zur Herkunft des aktiven Knoten und eine Änderung der Basiskontrolle. Bisher konnte dieser Knoten ausschließlich aus der OFFEN-Liste geholt werden. Nunmehr muß es möglich sein, auf die ausgezeichneten Knoten der Suchbaumentwicklung zurückzugreifen:

der Initialknoten sowie alle späteren Knoten, deren Nachfolger lediglich um den Eintrag des Neuziel-Markers M(H_WORTHYP)[18] erweitert wurden, heißen die *ausgezeichneten* Knoten oder Returnknoten des Suchbaums. Der zuletzt auf diese Weise erzeugte Knoten heiße der ***aktuelle Returnknoten***.

Die OFFEN-Liste kann bei der inkrementellen Kontrolle bereits vor dem Analyse-Zwischenstop am Signalende leer sein und zwar in dem genauen Sinn, daß sie keinen als ZULAESSIG bewerteten Knoten mehr bereithält.

---

[18] im allgemeinen Anwendungsfall M(H_HYPOTHESE

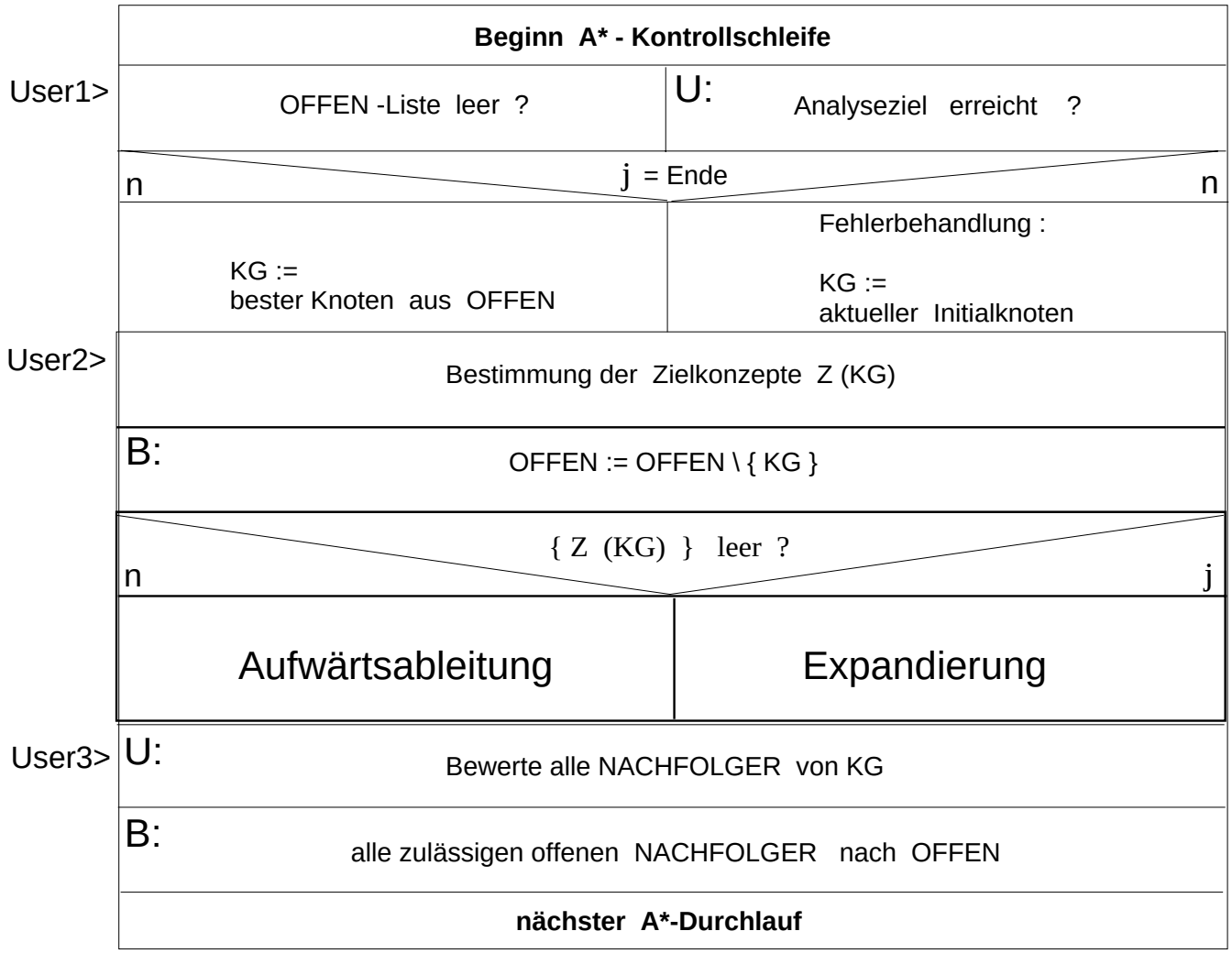


Bild 2.22 Struktogramm der A*-Schleife

Wenn z.B. die Analyse ein verrauschtes Startintervall des Sprachsignals durchläuft und alle Aufwärtsableitungen der Wortkandidaten des Bereichs scheitern, so gehen die Endknoten der Aufwärtsableitungen nicht mehr in die OFFEN-Liste ein. Dies sichert eine bisher nicht vorgenommene Verschärfung in der von mir veränderten Implementation der Basiskontrolle:

Inkrementelle Kontrolle verschärft den Gebrauch der OFFEN-Liste, indem nur offene und ZULAESSIGE Knoten in sie Eingang finden.

Da die OFFEN-Liste definitionsgemäß auch den inzwischen expandierten Initialknoten nicht mehr enthält, wird sie in solchen Fällen vollständig entleert und es besteht die Notwendigkeit einen ausgezeichneten Knoten des Suchbaums zu reaktivieren.

Der zentrale Platz der A*-Schleife in der ERNEST-Basiskontrolle soll mit folgenden Maßgaben gekennzeichnet werden:

- Sie ist die äußere Schleife (die *'main loop'*), die alle Analyse- und Kontrollmaßnahmen einschließt. Eine von ihr umschlossene innere Schleife ist z.B. die Aufwärtsableitung über mehrere Zwischenziele.
- Funktion der A*-Schleife ist im Normalfall (keine Fehlerbehandlung, keine RESKIP-Phase) die Expansion des Suchbaums ausgehend von je genau einem

aus der OFFEN-Liste erwählten Knoten (der aktive Knoten, KG im Struktogramm). Ein Schleifendurchlauf heiße ***A*-Takt***.

- Im Unterschied zu der von Miller, Galanter und Pribram [Mil60][19] bekannt gemachten TOTE-Schleife (***Test-Operation-Test-Exit***) ist das eine TOBE-Schleife (Test-Operation-***Bewertung***-Exit).
- Sie ist der zentrale Ort der Wechselwirkung von Basiskontrolle (Signet B:) und Aufgabenkontrolle (Signet U:, die Funktionen User1 und 2) mit eindeutig bestimmter Platzzuweisung: die TOBE–Schleife wird realisiert durch den Wechsel der Kontrolle in der Folge UBUB.
- Der mit User1/2 markierte Block ***Test*** bestimmt den besten Knoten der OFFEN-Liste. Diese kann bei inkrementeller Verarbeitung leer sein, weil unverarbeitbare Signalbereiche auftreten. Dann wird die Analyse auf den aktuellen Returnknoten zurückgesetzt und der Signalbereich durch SKIP-Steuerung übersprungen (Fehlerbehandlung). Zum Test-Block gehört auch die Abbruchbehandlung, die das Erreichen des Analyseziels oder eines Subziels testet, und den Ausstieg aus der Gesamtkontrolle erzwingt.
- Der Block ***Operation*** (Aufwärtsableitung/Expandierung) wird von der Aufgabenkontrolle beeinflußt, indem sie die Menge der Subziele vorgibt. Ist diese leer, so wird expandiert.
- Im Block ***Bewertung*** der im Operationsblock erzeugten Suchbaumknoten kann die Aufgabenkontrolle die Expansion eines Konzeptgraphen maximal weit vorantreiben. Sie kann aber auch die Steuerparameter des Bewertungstupels einsetzen, um das Um- oder auch Zurückspringen in einen anderen Suchbaumzweig zu erzwingen. Eine begrenzte Breitensuche ist zeitweilig sinnvoll, z.B. um an die Erkennungseinheit zeitsynchron mehr falsifizierte Hypothesen zurückzugeben.
- Der Block ***Exit*** ordnet die Knoten in die OFFEN-Liste ein und bewirkt den Rücksprung zum Block *Test*.

Gerade aus der Sicht der ingenieurmäßigen Anwendung von ERNEST halte ich es für sinnvoll, die hinter ihr stehende Kontrollmaschine sichtbar zu machen, was meines Wissens in der Literatur über ERNEST bisher nicht geschehen ist. Man sprach stets vom “A*-Algorithmus”, was bei entsprechender Fassung des Algorithmen-Begriffs auf einen abstrakten Automatentyp zurückführt. Die Bezeichung ***“A*–Maschine”*** führe ich in Hochstrichen ein, weil die algorithmische Formulierung von A* üblicherweise an eine Stop-Bedingung geknüpft wird, die unter inkrementellem Regime nicht zu halten ist (vergleiche die allgemeine Form in [Kum92a], S. 30 und die für ERNEST angepaßte Form in [Sag90], S.267):

while (OFFEN nicht-leer), wobei while() für die A*-Schleife steht.

Eine interessante Notierung des A*-Algorithmus gab Winston in [Win87b], indem er sie als Erweiterung der “Branch-and-Bound”-Suche darstellt. Die OFFEN-Liste faßt er als Queue, sprich Liste, von Teilwegen. Das macht nur einen formalen Unterschied aus, da jeder Teilweg durch seinen Endknoten eindeutig bestimmt ist.

Wichtig ist der Unterschied der Stop-Bedingung, den ich aus der englischen Version der 3. und letzten Ausgabe des Buches “Künstliche Intelligenz” [Win87a] (S.94) zitiere:

Stopbedingung der A*-Suche:

[19] Ein wichtiges Buch: macht Informatik-Konzepte (TOTE u.a.) für eine allgemeine Theorie der Handlungssteuerung fruchtbar.

’Until the first path in the queue terminates at the goal node or the queue is empty.’

Die Bedingung des erreichten Analyseziels war auch in der Formulierung des ERNEST-Basis-Algorithmus enthalten, nur diente sie als zweiter, innerhalb der A*-Schleife positionierter “Ausstiegshaken” (siehe auch [Kum92a], S.92).

Steht die Abbruch-Behandlung dagegen direkt in der Testbedingung des Schleifenkopfs, so kann auf “verrauschten” Signalbereichen die Fehlerbehandlung eingeleitet werden, denn das Analyseziel oder ein pragmatisch aussagekräftiges Subziel wurden noch nicht erreicht. Das versuche ich im Kopf des Struktogramms, Bild 2.22, auszudrücken.

Damit sollte gerechtfertigt sein, daß mit einer derartigen Konstruktion A*-Suche gefahren werden kann. Ob sie lediglich eine Branch-and-Bound-Suche durchführt, hängt allein von der Art der Bewertung ab.

Unter dem Stichwort “A*-Maschine” werden folgende Eigenschaften resümiert:

A. Die Flexibilisierung der Kontrolle mit UP-Maschine, z.B. temporäre Breitensuche, Alternieren von Bottom-up/Top-Down-Suche.
B. Die spezielle Fähigkeit zur inkrementellen Verarbeitung von Hypothesen durch Reaktivierung ausgezeichneter Knoten des Suchbaums und SKIP-Verarbeitung.
C. Die ingenieurmäßigen Vorteile beim Programmentwurf – beim Debugging ist die A*-Maschine bis zu den ehemals aktiven Knoten ohne Seiteneffekte taktweise zurück- und stets vorausfahrbar, was den Programmtest sehr erleichtert[20].
D. Die generelle Möglichkeit, eine PROLOG-ähnliche Anfrage-Umgebung zu schaffen, die über einen Debugger hinausgeht und geeignete Haltepunkte sichtbar macht.

Die der Implementierung zugrunde gelegte Maßregel der Suchbaum-Speicherung ist zu kennzeichnen als

Prinzip der Erhaltung des Sub-Suchbaums der aktiven Knoten:

Jegliche Erweiterungen oder Bewertungen werden nicht am *aktiven* Knoten, sondern an einer als Nachfolger in den Suchbaum eingehängten Kopie durchgeführt, um den Knoten für Maßnahmen der Rückverfolgung im Originalzustand zu erhalten.

Erinnert sei an das Problem der Strukturbeschreibungen mit mehreren konzeptgleichen Pragmatik-Wurzeln im Beispiel in Bild 1.32 (Unterabschnitt “Bewertung”) sowie an die in Bild 2.23 skizzierte Maßnahme zur Sicherung des Originalzustands von Returnknoten.

Das Erhaltungsprinzip setzt voraus, daß Suchbaumknoten den Zustand der Strukturbeschreibung am Ende eines A*-Taktes widerspiegeln. Es ist nicht sinnvoll, den Returnknoten von seiner Position n in Bild 2.23 auf n+1 zu versetzen, weil dieser Knoten in vielen Fällen (siehe SKIP-Verarbeitung) benötigt wird. Man betrachte die folgende Situation: ein Konzeptgraph gäbe keine Möglichkeit der weiteren Signalüberdeckung, man will nach einer Fehlererkennung zum Returnknoten zurückgreifen und den Übergang in Phase II erzwingen. Aus Knoten n+1 würde man den störenden Neuzieleintrag mitnehmen, der ein ständiger Aspirant für die Priorbestimmung ist.

---

[20] dazu mußte allerdings der Modularisierungsgrad der Basisroutine erweitert werden.

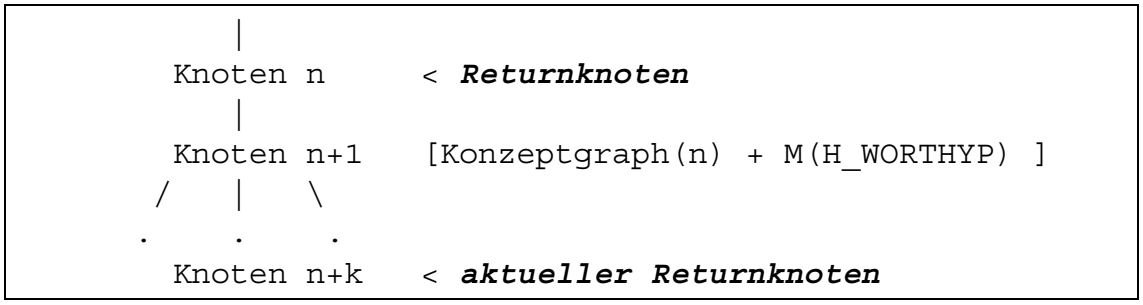


Bild 2.23 Bestimmung des Returnknoten

Bild 2.24 zeigt einen meiner ersten Entwürfe einer Maschine zur inkrementellen Verarbeitung von Worthypothesen. Der von Parsern verwandte Stackautomat ermittelt seine Reaktion auf Eingabesymbole aus einer Lookuptabelle, deren Einsprünge jeweils Paare von Eingabesymbolen und Automatenzuständen sind, aus der “Automatentafel”.

Der entworfene Pipelineautomat gruppiert entlang einer Zentralachse Testfunktionen, die teils auf den Konzepttyp der Hypothese, d.h. auf die Wissensbasis, und teils auf globale Zustandsbeschreibungen zugreifen, z.B. die auf einem extra Stack gesammelten offenen Konstituenten (z_1, z_0, ...). An der Verwendung der ersten Testfunktion (“liegt eine WNR-Restriktion vor?”) erkennt man, daß der Automat auch die Arbeit der Top-Down-Propagierung der linguistischen Attributwerte übernehmen soll. Von Überschaubarkeit/Beherrschbarkeit kann daher keine Rede sein.

Ohne jeden Anspruch auf Verallgemeinerbarkeit möcht ich dazu eine Hypothese wagen:

Je größer die *Multiplizität der Kontroll-Komponenten*, um so wichtiger ist erstens die *Simplizität der Basiskontrollmaschine*. Um so nötiger ist zweitens eine Aufgabenkontrolle, welche *die “Seiteneffekte” der Komponenten auspegeln kann.*

Der Entwurf wird von der Idee beherrscht, eine Art Shift-Reduce-Maschine (vergleiche die Beschreibung des Shift-Reduce-Parsers in 2.5.1) in Pipelineform zu erfinden.

Was hier ein Maschinentakt ist, entscheiden die an die Pipeline angekoppelten Basisprogramme. Sie können die Rückkehr zur Pipeline triggern und somit weitere Programme anstoßen. Sie sind also Subautomaten, die zeitweilig die Kontrolle übernehmen. Wenn kein Test mehr ansteht und kein “Beitrag” zur Strukturbeschreibung mehr zu gewinnen ist, gilt der Takt als beendet und die nächste Hypothese wird geholt. Der Maschine fehlt ein Unterbrechungsmechanismus, um auf Suchbaumaufspaltungen reagieren zu können.

Die A*-Maschine benötigt nach jedem Takt eine Übersicht der eingelaufenen Suchbaumknoten. Es wird so verfahren, als wüßte man nichts über die Entstehungsgeschichte der Knoten. Es interessiert nur ihre Bewertung. Folglich werden auch die um den Neuzielmarker M(H_WORTHYP) expandierten (Nachfolger der) Returnknoten nicht automatisch zu aktiven Knoten des nächsten Taktes, obwohl dies für die inkrementelle Verarbeitung zwingend ist. Sie müssen sich durch ihre Bewertung als solche ausweisen. Die unterschiedlich Sicht von Basiskontrolle (bewertete Knoten ohne “Historie”) und Anwendungskontrolle (Konzeptgraphen mit Reifestufen) kommt in der SR-Pipeline

gar nicht vor. Sie kann daher auch nur begrenzt inkrementell sein, d.h. z.B. nur sehr begrenzt im zeitsynchroner Kopplung auf den Worterkenner reagieren und ihn beeinflussen.

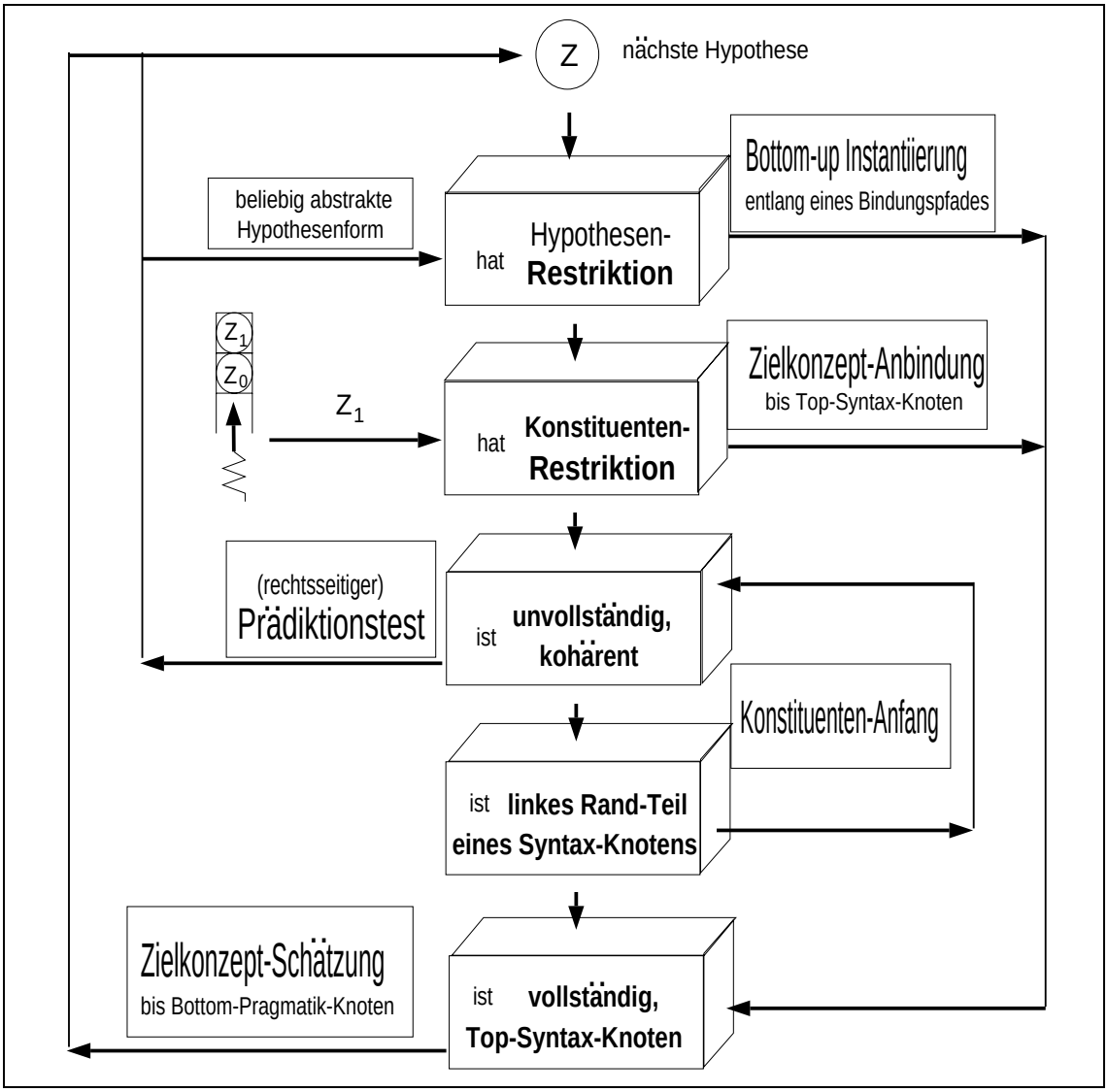


Bild 2.24 Entwurf “Shift-Reduce-Pipeline”

2.4.2 Adjazenz-abhängige Aufwärtsableitung

Aufwärtsableitung wie Expandierung werden in der ERNEST-Basiskontrolle als auch der Aufgabenkontrolle (z.B. Rechts-Assoziativität) auf die ADJAZENZ in den Konzeptdeklarationen[21] abgestimmt. Dieselbige wird als matrizenförmige Anordnung von Boole-Flags notiert, welche die Stellung der Kanten zueinander, sprich die Reihenfolge der abhängigen Prädikate, bestimmt.

Eine gewählte Konzeptmodalität habe die Kanten k_1, ... , k_N:

Die (N+1) x (N+1) -Matrix ($flag_{ij}$) mit Zeilenindex i und Spaltenindex j definiert eine *Vorgänger-Relation* zwischen den Kanten:

[21] siehe Bild1.19 in Abschnitt 1.4

Kante k_j ist Vorgänger von Kante k_i, wenn das $flag_{ij}$ auf TRUE steht. Vorgängerschaft kann auch in nicht-kohärenten Modalitäten angegeben werden. Gesetzte ii-flags (entlang der Diagonale) drücken aus, daß identische Kanten (z.B. mehrere Adjektive) aufeinander folgen dürfen. Eine Limitierung solcher *Mehrfachkanten* kann ausgedrückt werden, indem das Boole-Flag[22] zugleich als Anzahlwert aufgefaßt wird. Die Beginn-Zeile definiert die Rollen, die als erste in einen modalitätseigenen Marker des Konzeptgraphen einzubinden sind.

	Ende	Kante k_1	.....	Kante k_N
Beginn	—	$flag_{01}$	.....	$flag_{0N}$
Kante k_1	$flag_{10}$	$flag_{11}$		
.....	.....	.....	.....	.....
Kante k_N	$flag_{N0}$	$flag_{N1}$	.....	$flag_{NN}$

Bild 2.25 ADJAZENZ-Matrix

Das folgende Beispiel soll den Nutzen des Informationstyps ADJAZENZ, zunächst für die Aufwärtsableitung, belegen. Man betrachte den LRN-Marker in Bild 2.26. Der kritische Punkt M(SY_NG) ergibt sich, weil ein Worterkenner blind für die Groß/Kleinschreibung der von ihm gelieferten Lexeme ist.

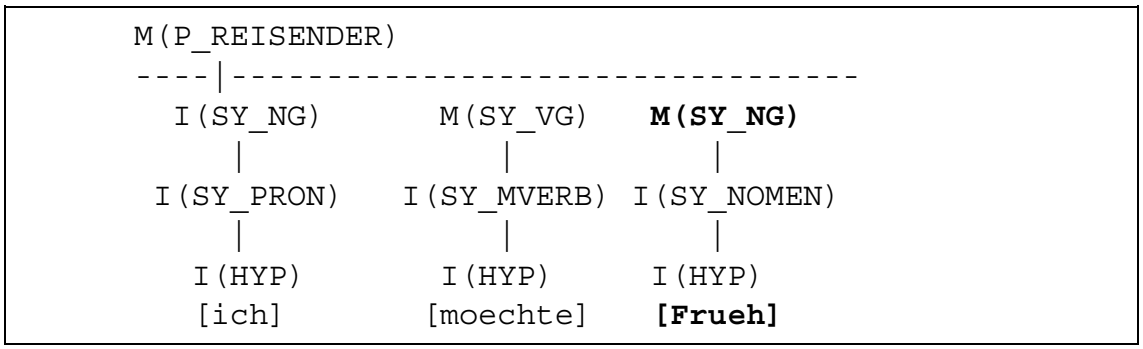


Bild 2.26 Problemsituation: Aufwärtsableitung

Bevor der Wortgraph verarbeitet werden kann, muß erst eine User-, sprich problem–spezifische Datenbehandlung zu vom Lexikon indizierten Fällen Wortgraphkanten 'klonieren', z.B. für [frueh] und [Frueh]. Nun habe man die Verarbeitung des Kandidaten [Frueh] vorliegen und es sind die Subziele anzugeben. Dann braucht man eine relativ spezifische User-Regel:

- Spezialfall –

User-Regel 'Subziel | Wortart Nomen':

wenn ein Wortkandidat die Wortart Nomen hat, **dann** stelle als Subziele genau die Konstituenten SY_NG, Z_ZEITANGABE.

[22] Wert 0 ist das FALSE-flag

- Allgemeinfall –

User-Regel ’Subziel | Wortart x’:

wenn *ein Wortkandidat die Wortart x hat,* **dann** *stelle als Subziele genau die Konstituenten* (SPEZIALISIERUNG(SY_KONSTITUENTE)), *an denen die Wortart als Subkonstituente teilhat.*

Durch die Metaregel zur Aufwärtsableitung des so bestimmten Subziels wird das linguistische Attributwissen der Wissensbasis angefordert und die Falsifizierung des Subziels betrieben. Wäre im untenstehenden Fall für die Regel

SY_NG(X) := SY_NOMEN(X)

nur die standardgemäße (partielle) Elementarableitung zu prüfen, dann käme ein positiver Befund heraus.

Der Forderung nach Inkrementalität wird so jedoch nicht nachgekommen, denn das Datenereignis [Frueh] dürfte an dieser Stelle des Eingabestromes nicht erscheinen. Vom linguistischen Standpunkt darf es keine Konstituente eröffnen, sondern müßte, wie z.B. in der Kette [in, der, Frueh], nachgestellt werden. Deshalb ist eine Zusatzsteuerung nötig, welche bereits die partielle Ableitbarkeit von den Adjazenz-Einschränkungen der Modalität abhängig macht:

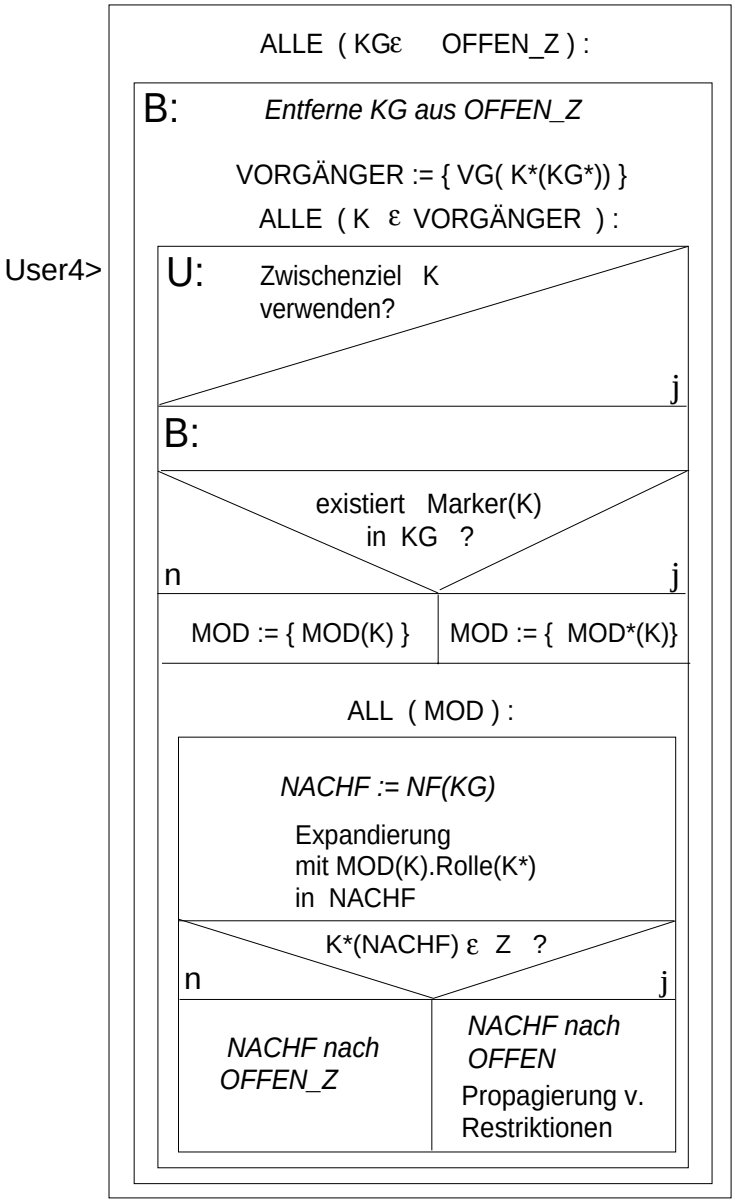
der Marker M(SY_NG) im 2.26 Bild müßte dadurch als UNZULAESSIG bewertet werden.

Gegebenenfalls reicht die definierte Adjazenz eines der Zielkonzepte, sprich Subziele, nicht aus. Für das in der Zugauskunftsdomäne einzig zutreffende Subziel Z_ZEITANGABE ist sie scharf genug, für weitere, rein über die Wortart gefundene Subziele nicht. Also benötigt man eine noch detailliertere Usersteuerung des Falls, eine noch speziellere User-Regel ’ Subziel | [Frueh]’.
Im Effekt ihrer Anwendung dürfte nur ein einziges Subziel, Z_ZEITANGABE, aktiviert werden. Die Anzahl solcher Hilfsregeln müßte allerdings begrenzbar sein.

Für die inkrementelle Verarbeitung benötigt man unbedingt die *adjazenz-abhängige* Aufwärtsableitung für den Kantentyp BESTANDTEIL. Die Quellkonzepte der Kantentypen KONKRETISIERUNG und SPEZIALISIERUNG bestehen nicht aus Syntagmen von Rollen. Es geht um eine formelle, netztechnologische Aufgabe, deren Programmlösung ich realisiert habe. Gegeben seien

M(K,MOD(K)) , M(K*,MOD(K*)) und $\{r^k\}$ (i=1,..,k), d.h.

ein Startmarker des Konzepttyps K*, ein Zwischenziel–Marker des Konzepttyps K für K* (K* = BESTANDTEIL(K)) und die nach Adjazenz geordnete Folge der schon in MOD(K) gebundenen Rollen.



KG Startknoten der Aufwärtsableitung*

OFFEN_Z Typ {KG}
Hilfsmenge von OFFEN: offene Zwischenknoten der Aufwärtsableitung (initial: {KG*})

K*() : KG ---> K
zuletzt abgeleitetes Subziel

VG(K*()) : K ---> K '

MOD*(K)
eindeutig bestimmte Modalität des Markers von K

MOD(K).Rolle(K*)
Regel zur Elementarableitung von K aus K*

Z: KG ---> {K}
Menge der Subziele

Bild 2.27 Schleife der Aufwärtsableitung

Die folgende Metaregel ist eher eine Erweiterung vom Typ der Regeln R1 – R5 denn eine User-Regel:

SITUATION der **adjazenz-abhängigen Aufwärtsableitung:**

1. Bestimme alle Rollen r*: K(Arg) :- $_{(r^*)}$ K*(X_*) (X_* in Arg).
2. Prüfe ob r* in der ADJAZENZ(MOD(K)) rechter Nachbar von r^ksein darf.
3. Falls MOD(K) KOHAERENT ist, bestimme im Wortgraphen den Endknoten der assoziierten Subkette X_k zur Rolle r^k und den Anfangsknoten der Subkette X_* zur Rolle r* :
   teste auf Übereinstimmung der Knotennummern.

4. Falls Bedingung 2.,3. für mindestens ein r* erfüllt sind
   —>
   AKTION: Rollenbindung r*(K*,X_*) in M(K).

Die Regel ist jedoch nicht vom Kontrolltyp unabhängig. Obligatorische und optionale Rollen werden völlig gleich behandelt. Insofern ist die Regel auf die inkrementelle Kontrolle ausgerichtet. Softwaretechnologisch bietet sich zur Einflechtung solcher Metaregeln, z.B. auch für die Expandierung, eine in die betreffenden Basisroutinen fest eingebaute FLAG-Behandlung an, die in der User-Initialisierung der Kontrolle ihre Voreinstellung erfährt.

Das folgende Bild 2.27 gibt die algorithmische Fassung der Aufwärtsableitung mit Akzentverschiebung aus Sicht der Inkrementalität. Bei Kummert tragen die korrespondierenden Struktogramme ([Kum92a], S.89/90) die Bilduntertitel '*Zielkonzeptschätzung*', d.h. Aufwärtsableitung von Subzielen, und '*Propagierung von Erwartungen des Zielkonzepts*'.

Das Struktogramm soll besonders herausheben, daß

- Die *parallele* Ableitung mehrerer Subziele zum selben Startmarker erfordert ein gewisses Management der OFFEN-Liste, z.B. die Hilfskonstruktion einer Subliste OFFEN_Z zur Erfassung der für Zwischenziele angelegten Suchbaumknoten:
  (z.B. im untenstehenden Bild) Subziele P_ANKUNFTSORT, P_ABFAHRTSORT sind von der Präpositionalgruppe [nach, Bonn] (aufwärts) abzuleiten. Expansionsschichten I/II bilden je einen Durchlauf der main-loop des Struktogramms. Knoten [n], [n+1] der Schicht I gehen temporär nach OFFEN_Z und werden Ausgangspunkt für Schicht II.

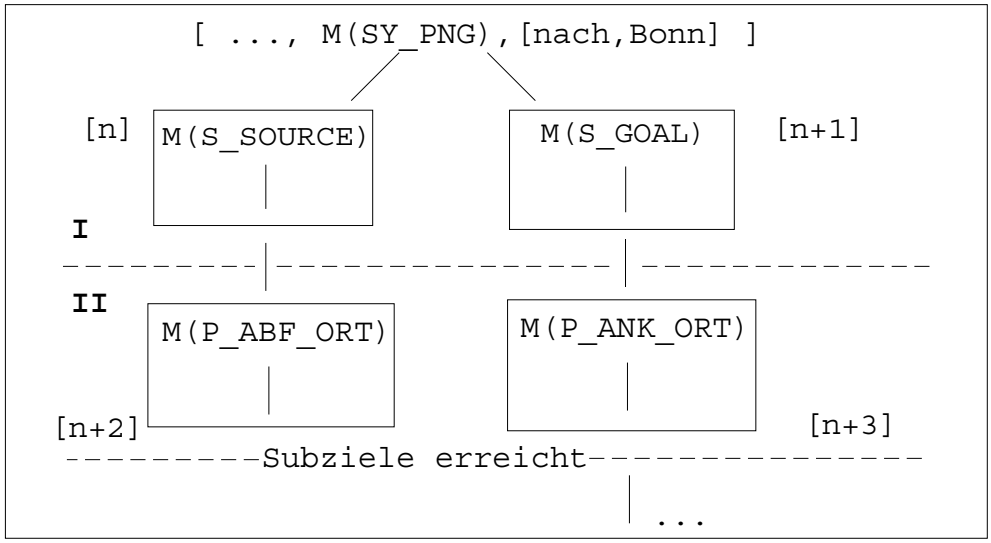


Bild 2.28 Suchbaum nach Expandierung

- Die schichtweise Bearbeitung der Suchbaumknoten, sprich Konzeptgraphen, macht mit dem ausgezeichneten Marker K* den Anfang. Initial ist das der Startpunkt, danach das zuletzt abgeleitete Zwischenziel[23] des Konzeptgraphen.
- Die Funktion *User4* soll die Zahl der Zwischenziele begrenzen. Ihr Grundmuster, d.h. eine Art Codeschema für Userfunktionen, wurde von mir folgendermaßen

[23] bei Kummert ([Kum92a]) heißt er AKTZIEL, bei Sagerer AKTUELL ([Sag90])

gegliedert:
– Zwischenziele fest vorgeben (Minimalisierung)
– Zwischenziele von Basisroutine bestimmen lassen und durch situationsunabhängige Kontrollentscheidungen (darunter die Metaregel 'adjazenz-abhängige Aufwärtsableitung') falsifizieren.

- Der Einführung eines Zwischenziels als Neuziel muß die Durchmusterung der offenen Altziele des Konzeptgraphen vorausgehen.

Metaregel (Aufwärtsableitung) 'Altziel-Vorrang':

wenn *ein Altziel offene Bindung besitzt, die durch M(K*) gebunden werden kann,* **dann** *hat diese Bindungsaktion absoluten Vorrang vor der Neuzielbildung.*

- Die Einführung des Zwischenziels als Neuziel führt zur Suchbaumspaltung der aktuellen Expansionsschicht je Modalität des Neuzielkonzepts: OFFEN_Z enthält Knoten je Anzahl verwendeter Zwischenziele je Subziel (für Neuziele: multipliziert mit deren Modalitätszahl).
- Die letzte Expansionsschicht ist erreicht, wenn gilt: K* = Subziel.

Im Unterschied zu der im folgenden zu beschreibenden adjazenz-abhängigen Form des Expandierens ist noch das folgende Charakteristikum festzustellen:

In der Phase der Aufwärtsableitung wird die Tiefensuche, d.h. der eindeutig bestimmte Suchmodus der UP-Maschine, suspendiert und stattdessen eine auf Subziele begrenzte Breitensuche durchgeführt.

Um so wichtiger ist die korrekte Bestimmung der Subziele im TEST-Block der A*-Schleife.

Das Bild 2.28 der Verzweigung des Suchbaums nach der Aufwärtsableitung verdeutlicht, weshalb das Zurückfahren der A*-Maschine nicht bis zu jedem beliebigen Knoten, sondern nur zu ehemals aktiven Knoten erfolgen kann. Nur von solchen Knoten aus kann man systematisch alle nötigen Weglassungen seiner Nachfolger in der OFFEN-Liste übersehen.

2.4.3 Assoziative Expandierung

Der Übergang zur Problematik der *adjazenz-abhängigen* Expandierung soll anhand des Beispiels in Bild 2.29 vollzogen werden. Die Modalität des (aufwärts) abgeleiteten Markers M(SY_PNG, [nach,X]) ist als KOHAERENT deklariert. In diesem Fall lassen sich erstens aus der Wissensbasis Restriktionen der Subkettenbildung zur Signalebene durchpropagieren.

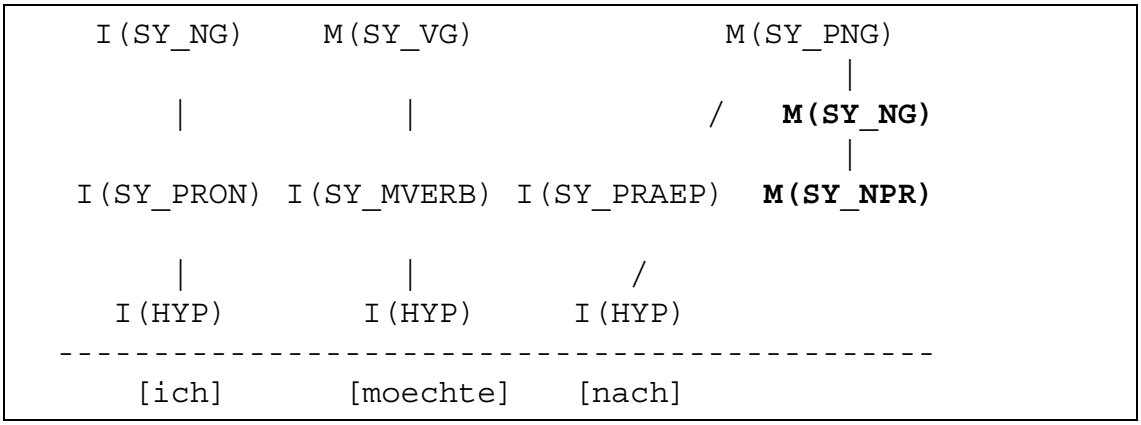


Bild 2.29 User-Steuerung der Expandierung

Zweitens kann man derart durchgängige Expandierungen als Falsifizierung eines von mehreren konkurrenten Suchbaumaufspaltungen vorsehen (vergleiche die Ambiguität in Bild 2.10).

Wiederum läßt sich ganz formell einführen:

User-Regel (inkrementelle Expandierung) 'Kohärenz':
wenn *der Konzepttyp eines potentiellen Startmarkers der nächsten Aktion KOHAERENT ist,* **dann** *gebe seiner adjazenzabhängigen Expandierung Vorrang vor einer Aufwärtsableitung.*

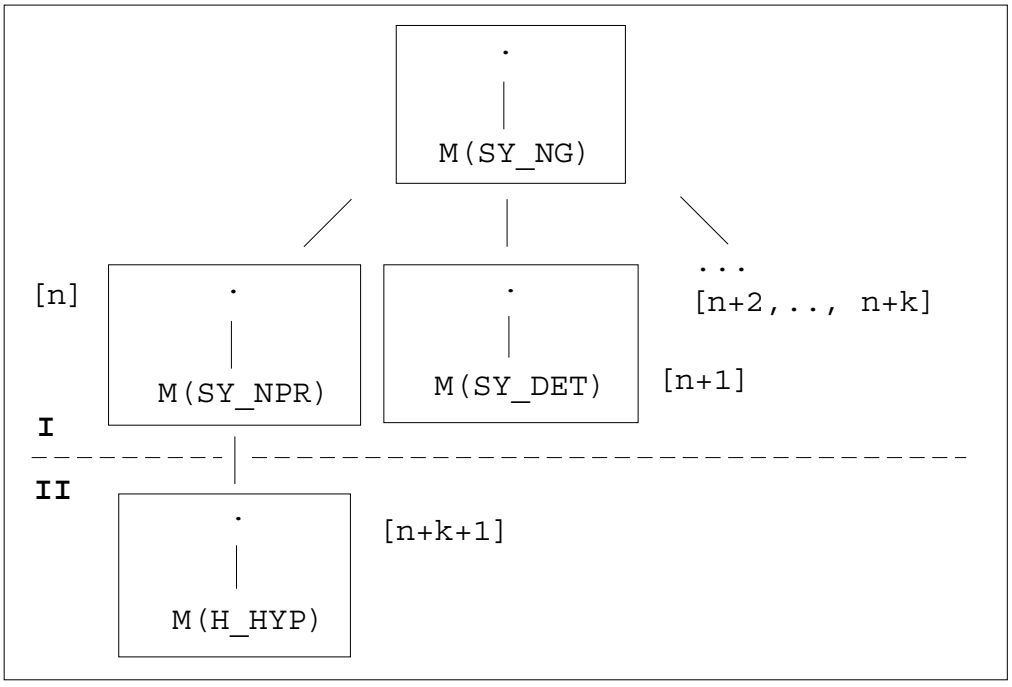


Bild 2.30 Suchbaum-Aufspaltung bei der Expandierung

Das Bild 2.30 zeigt zu der in Bild 2.29 (fettgedruckten) Expandierungskette die Fortsetzung ab M(SY_NG) an. Man erkennt die große Breite der Aufspaltung des Suchbaums aufgrund der Vielzahl von Nominalgruppen-Varianten innerhalb der

Modalität. Im Unterschied zur Aufwärtsableitung ist das Phänomen der '*Breite der Expansionsschicht*' für Expandierungen auf die erste Schicht beschränkt. Danach wird sofort wieder auf Tiefensuche geschaltet, d.h. nur ein Knoten expandiert.

Die Breite in Schicht I legt wiederum vorrangsteuernde User-Regeln nahe. In diesem Fall bietet sich ein ähnliches Vorgehen wie in der Beispielanalyse von Kummert an, der am Analysestart pragmatisch besonders relevante Wortartkonzepte (SY_NPR, Eigennamen) wählte, um bereits in der Anfangsphase der Analyse den Ableitungsbaum eines der Konzepte P_ANKUNFTS-, ABFAHRTS-ORT verfügbar zu haben:

User-Regel (Expandierung) 'pragmatischer Vorrang der Wortart':

wenn *in Expandierungsschicht I eines Syntax-Markers eine große Breite auftritt und einer der Rollenträger zu seiner partiellen Bindung besondere pragmatische Relevanz für einen noch nicht (an der Wurzel) abgeleiteten Ableitungsbaum besitzt,* **dann** *gebe dem mit diesem Konzept expandierten Suchbaumknoten eine User-Bewertung, die seine Auswahl in der Expandierungsschicht II erzwingt.*

Das Struktogramm in Bild 2.31 betont andere Akzente als die Darstellung von Kummert ([Kum92a], S.96, 'Instanzbindung' für Expandierung):

- Die SITUATION—AKTION–Gliederung wird durch getrennte Programmabschnitte deutlich.
- Die Frage der Ereignissteuerung stellt die Vorentscheidung der Situationsanalyse dar. Die rein auf den Prior orientierte Aktionsbestimmung im Konzeptgraphen bleibt als Spezialfall erhalten.
- Die an der entscheidenden Stelle plazierte neue Funktion *User5* steuert die inkrementelle Verarbeitung. Sie betrachtet stets das Paar
  Verarbeitungsphase und LRN.
  Sind keine zwingenden Expandierungsaktionen zu ermitteln, so ist die Reaktion auf neue Datenereignisse zu organisieren. Auch die SKIP–Steuerung und die Rückkopplung an den zeitsynchron arbeitenden Worterkenner fällt in den Coderaum der Userfunktion.
- Das Aktionsschema erhält die alte Gliederung aufrecht: der durch Fallunterscheidung oder Neuzielsetzung erzwungene[24] Aktivationspunkt des Konzeptgraphen KG bestimmt die AKTION.
- Die Aktionsvarianten des so bestimmten Markers AKT(KG) sind durch eine neue Funktion *User6* bei Ereignissteuerung
  – auf die adjazenz-abhängige Expandierung (vor allem in Phase I)
  – sukzessive auf die Kontexteinbindung bereits abgeleiteter Wurzelmarker von Ableitungsbäumen (einschließlich der optionalen in Phase II) hinzulenken. Die alte, auf den Kantenvorrang gestützte Aktionswahl bleibt erhalten.

Der Spezifikation der ausgezeichneten 6 Userfunktionen und den mit ihnen erhaltenen Testergebnissen dient das 3. Kapitel.

---

[24] ein Neuziel H_HYPOTHESE bekommt auf Grund der größtmöglichen Signalnähe automatisch den absoluten Vorrang

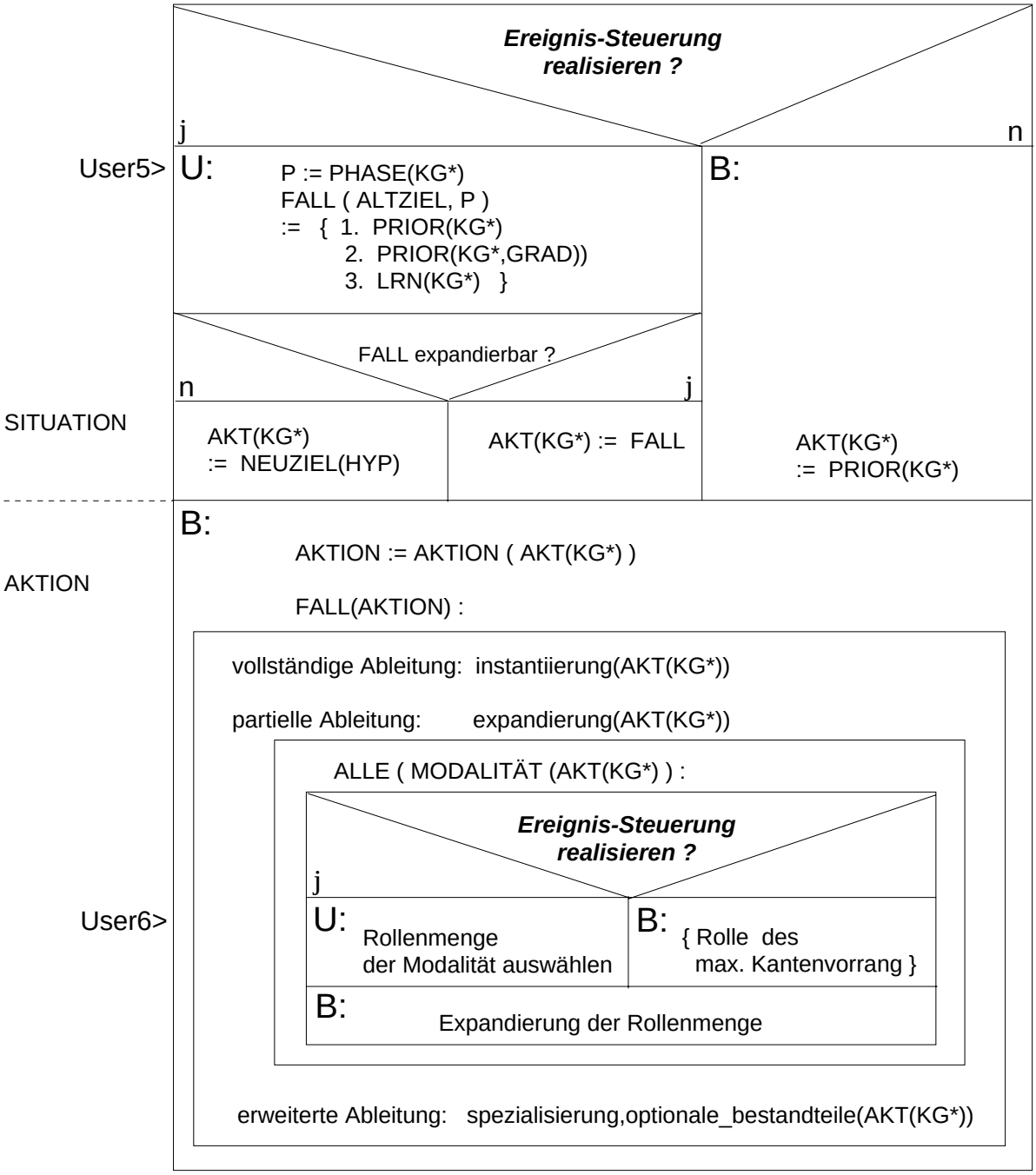


Bild 2.31 Struktogramm der Expandierung

2.5 Analogien zu anderen Kontrollstrategien

In diesem Abschnitt soll die Besonderheit der konzipierten Kontrollstrategie durch Vergleich mit anderen Systemen deutlicher gemacht werden. Besondere Anregungen sind auch den zunächst gewählten, scheinbar ganz sachgebietsfremden Aufgabenstellungen zu entnehmen.

2.5.1 GLR*

In der Einleitung wurde der Shift-Reduce-Parser GLR* [LA93] als Vergleichssystem in das Blickfeld gerückt. Mit seiner Namensgebung sind Ansprüche verknüpft, die ich nur kurz umreißen möchte. Eikmeyer hat in seiner Übersichtsdarstellung [Eik89]

zu Grundzügen der computerlinguistischen Anwendung von PROLOG zunächst die syntaktische/semantische Satzanalyse mit der Definite Clause Grammar (siehe Anfang von Kapitel 1) beschrieben. In der Folge will er der Auffassung vorbeugen, die DCG sei die quasi-natürliche Art der Problemlösung mittels PROLOG. Ich habe weiter oben bereits zu zeigen versucht, daß die Realisierung von Bottom-Up-Ableitungspfaden auf diesem Wege recht umständlich anmutet. In der Tat ist das nicht der Standardweg zur Sicherung einer inkrementellen Arbeitsweise, die vielmehr durch die seit langem ausgefeilte Konzeption des Shift-Reduce-Parsers [Tom86] gelöst wird. Dessen Namensgebung verweist auf eine Stack-Maschine zur Kontrolle der Regelauswahl:

- Ein ***shift()*** ist eine POP-Operation, die das am weitesten linksstehende Terminalsymbol, sprich den Wortkandidaten des Worterkenner-Outputs, auf einen Stack legt. Gleichzeitig wird die Warteschlange dieser Wortereignisse auf das nächste Element adjustiert. Was das für einen Wortgraphen bedeuten würde, ist zunächst nicht ausgemacht.
- Ein ***reduce()*** ist eine TOP-of-Stack-Operation, welche die unmittelbar obenliegenden Symbole (Nonterminal, Terminal) gemäß ihrer Reihenfolge auf dem Stack als passende Sequenz unabhängiger Prädikate auffaßt, die mit einer Regel der Wissensbasis koinzidiert. Dann wird die aufliegende Stack-Sequenz durch ein Symbol des ableitbaren abhängigen Prädikats ersetzt (aus der Stack-Denkweise gesehen: sie wird reduziert).

Die Überschaubarkeit des Maschinenverhaltens ist ausgeprägt: Aktivation geht jeweils vom Wort-Shift aus. Die Stackbelegung in diesem Moment ist deshalb der ***aktive Knoten*** des sich als Aktivationskette darbietenden Suchbaums. Dann wird auf dem Stack sukzessive so oft reduziert als möglich. Eine Shift-Phase mit zugehöriger Reduce-Phase bildet somit den Takt der Maschine.

Wie die PROLOG-Codierung in Bild 2.34 zeigt, muß die in Bild 1.1 eingeführte Beispiel-DCG prozedural verklausuliert werden, damit sie direkt auf den Kontrollmechanismus zugeschnitten werden kann. Aus den linguistischen Prädikationsregeln der DCG werden Fakten zum Prädikat red() (Vorstufe zum Kontrollprädikat reduce()), dessen Argumentliste an die Stack-Reduktion angepaßt:

Arg_1=TOP-of-Stack, ..., Arg_{n-1}= End-of-Stackframe, Arg_n= Reducer.

Außerdem werden Restlisten-Variablen X verwendet, um den Rest der Stackbelegung von der jeweiligen Reduce-Aktion auszunehmen.

Der Stack-Automatismus automatisiert die Möglichkeiten der inkrementellen Abarbeitung von Sätzen oder Zielen und hat in der angewandten Forschung schon lange den Vorzug vor der Anwendung der DCG erhalten. Links-Rechts-Verarbeitung wurde für Computerlinguisten und Compiler-Theoretiker zur “Selbstverständlichkeit”. Die ERNEST-Anwendung, die sich an die Termini und Abstraktionsweisen der Anwendungsgebiete anschließt, ist eher der Tendenz der DCG zuzurechnen. Bei der Entwicklung des Vorgängersystems von ICZ wurde von Kummert bereits gezeigt, wie man Bottom-Up-Interpretation im Rahmen des (pragmatischen) Tiefenstrukturparsing durchführen kann. Die Verzögerung der (mit meiner Abhandlung vorgestellten) inkrementellen Version sollte aus diesem Gesichtswinkel begründet erscheinen. Man wird sich in Kapitel 3 auf entsprechende kontroll-relevante “Spezifikationen” der Codierung einrichten müssen.

```
    Wissensbasis:
  red([vp,np|X],[s|X]).
  red([nomen,det|X],[np|X]).
  red([nomen|X],[np|X]).
  red([np,verb|X],[vp|X]).
  red([the|X],[det|X]).
  red([a|X],[det|X]).
  red([goose|X],[nomen|X]).
  red([geese|X],[nomen|X]).
  red([fox|X],[nomen|X]).
  red([foxes|X],[nomen|X]).
  red([eats|X],[verb|X]).
  red([eat|X],[verb|X]).
1 ?- red([nomen,det],X).
     X = {np]
```

Bild 2.32 “Regeln” des Shift-Reduce-Parsers

Wie beim Rechnen mit den Taschenrechnern, die einer Stackarithmetik folgen[25], erfordert nun die korrekte Codierung von Zielen eine Gewöhnung an die stack-typischen Reihenfolge-Beziehungen (immer vom Top-of-Stack ausgehend). Nichtsdestoweniger bleiben die für den Satzakzeptor in Kapitel 1 aufgezeigten unmittelbaren Abfragemöglichkeiten erhalten. Anfrage 1 in Bild 2.32 zeigt z.B. an, wie man (bei Beachtung der gespiegelten Codierung der Normalform <det,nomen>) eine zulässige Reduktion X erfragen kann.

Erst wenn komplette Regel-Prämissen auf dem Stack liegen, kommt es zu einer Ersetzung. Die *Überschaubarkeit* und auch die *Beherrschbarkeit* (nach Gewöhnung an die Stacksequenz-Codierung) sind zufriedenstellend. Die Maschine stoppt, wenn sie keine Möglichkeit zur Reduzierung der Stackbelegung kennt. Sie kann ohne Zusätze nur Satzakzeptanz prüfen.

```
Reduce-Operationen:
  reduce(STold,STnew) :- red(STold,X),reduce(X,STnew).
  reduce(ST,ST).
2 ?- reduce([nomen,verb,np],X).
3 ?- reduce([np,det],X).
     X=[np,det]
```

Bild 2.33 PROLOG-Regeln für Reduce

Die in Bild 2.33 gezeigten Regeln der Reduce-Phase nutzen PROLOG-spezifische Codierungsmöglichkeiten aus, die sich der in Kapitel 1 dargelegten Regelform entziehen (z.B. Selbst-Referenz des abhängigen Prädikats auf sich als unabhängiges). Ein

[25] Taschenrechner der Firma Hewlett Packard benutzen eine solche, weil die stack–typische klammerfreie Schreibweise den Programmspeicher zur Klammer-Reinterpretation erspart.

elementarer Reduktionsschritt red() aktiviert, während die Selbstreferenz des reduce()-Aufrufs die Vollständigkeit der Reduce-Phase umsetzt. Infolgedessen können in Anfrage 2 nacheinander die letzten 3 Reduktionsschritte von Bild 2.35 in einem Zug durchgeführt werden. Die zweite Regelcodierung[26] soll sichern, daß jeder reduce()-Aufruf abarbeitbar ist, auch wenn keine Reduktionsmöglichkeit (mehr) vorliegt (siehe Anfrage 3).

Je Gesamt-Takt der Stack-Maschine wird ein Element der Wortliste Xel geshiftet und die Restwortliste X bestimmt. Da sich das Prädikat shift_reduce() (siehe Bild 2.34) selbst referiert, kommt es zur wiederholten Aktivierung des shift-reduce-Taktes. Die zweite Regel behandelt die Terminierung der Analyse auf Satzakzeptanz: wie in der letzten Zeile von Bild 2.35 muß der Stackframe des aktiven Knoten zu [s] reduzierbar sein.

```
Shift-Reduce-Parser:
  shift_reduce([Xel | X],STold)
      :- reduce(STold,STnew),
         shift_reduce(X,[Xel | STnew]).
  shift_reduce([],STold) :- reduce(STold,[s]).
3 ?-
```

Bild 2.34 PROLOG-Regeln: Shift-Reduce-Parser

In Bild 2.35 stehen [.] für die identisch bleibende Worteingabeliste und [] für die leere Liste. In der Spalte 'Stack' wurde die Folge der aktiven Knoten eingetragen.

Wortliste	Stack	Operation
[foxes,eat,geese]	[]	
[eat,geese]	[foxes]	shift
[.]	[nomen]	reduce
[.]	[np]	reduce
[geese]	[eat, np]	shift
[.]	[verb,np]	reduce
[]	[geese,verb,np]	shift
[]	[nomen,verb,np]	reduce
[]	[np,verb,np]	reduce
[]	[vp,np]	reduce
[]	[s]	reduce

Bild 2.35 Stackdarstellung eines Shift-Reduce-Parsings

Die von Eikmeyer in Entsprechung zur DCG angegebene Erweiterung um die Strukturbeschreibung sei ausgespart, da sie identischen Einschränkungen unterliegt, d.h. den Ableitungsvorgang nicht beeinflußt.

In der Compiler-Literatur [Aho86] wird der Shift-Reduce-Parser als LR(0)-Parser klassifiziert. Dabei steht das **L** für Links-Rechts-Verarbeitung, **R** für Rechts-Assoziation

[26] Die Abk. ST steht für Stack.

(zur inhaltlichen Diskussion des formalen Prinzips [Kim73], [Fra78]). Der Index k=0 steht für den Tatbestand, daß bei der Reduktion des aktuellen Elements der Wortliste keinerlei Vorausschau auf die verbleibende Restliste gemacht werden soll, was man als formale Fassung des Prinzips inkrementeller Verarbeitung ansehen kann.

Das bereits in der Einleitung zum Vergleich angesprochene System GLR ist ein LR(0)-Parser, der für die Anwendung in der Signalverarbeitung erweitert wurde. Es ist als “Tomita-Parser” bekannt geworden [Tom86]. Die Stack-Maschine wird als Automatentafel compiliert. Zur Kontrolle der Verzweigung von Parsing-Alternativen kommt man nicht ohne einen Suchgraphen der Knoten-Aktivierung aus, was in der oben dargestellten vereinfachten Version überflüssig erschien (Knoten als reine Stackframes). GLR ist ein rein-syntaktischer Parser mit Satzgrammatik, der insofern nicht mit dem in dieser Abhandlung vorgestellten System verglichen werden kann. Interessant sind aber verschiedene Versuche der Erweiterung der Konzeption zur Verbesserung der Robustheit bei spontansprachlicher Verwendung. Die Schärfe der Entscheidung “wohlgeformter Satz – Ja/Nein” wird abgebrochen und eine Fehlerbehandlung für nicht parsbare Wortketten eingeführt.

In dem von mir konstruierten Beispiel in Bild 2.36 versagt die Satzanalyse mit dem regulären Shift-Reduce-Parser ab der Stelle der Strichlinie, weil keine Regel zur Reduktion von [many] vorhanden ist. Das Grundprinzip der Erweiterung von GLR zu GLR* wird jedoch hinreichend erkennbar. GLR* ermöglicht eine Shift-Operation von der inzwischen reduzierten Wortliste unter Voraussetzung eines Rücksprungs zum inaktivierten Knoten (*). Dadurch kommt es zu einer speziellen Version von **SKIP-Verarbeitung**, im Sinne des einfachen, irreversiblen Ausblendens eines Wortkandidaten.

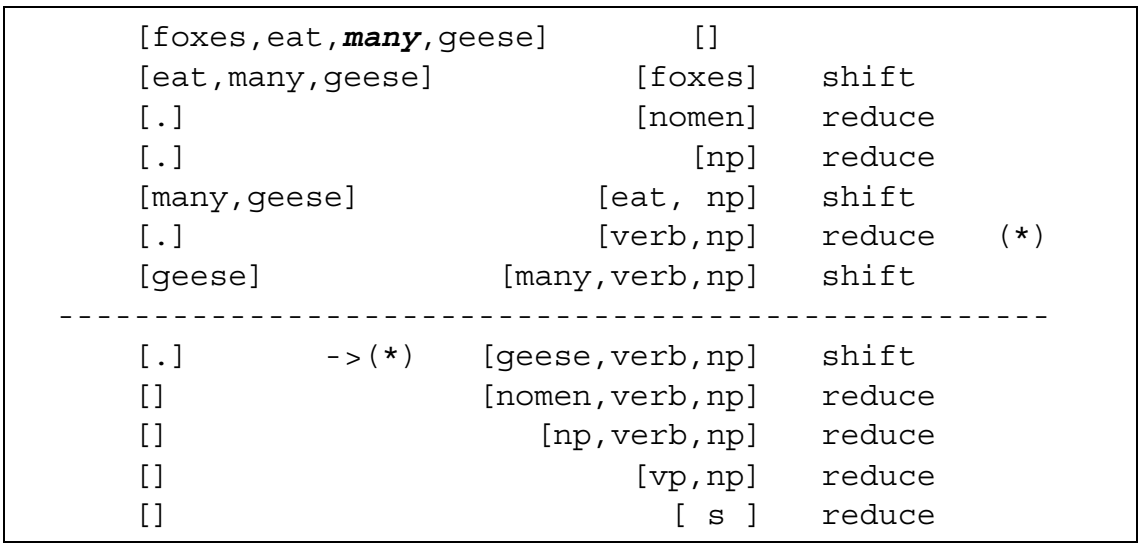


Bild 2.36 GLR*: Rückkehr zum inaktiven Knoten

Beim Parsen von Signalerkennerinput orientiert man sich an der Heuristik, die ***maximale parsbare Subkette*** zu finden, welche durch die größte Signalüberdeckung und kleinste Zahl von Auslasssungen definiert wird. Angesichts der Zielstellung muß man auf eine eindeutig bestimmte Lösung verzichten, muß Zwischenlösungen aufbewahren und später vergleichen. Damit würde man Maßnahmen benötigen, die im Rahmen der ERNEST-Anwendung mit der sogenannten “Kontrolle des dynamischen Abbruchs”

[Kum92b] eingeführt wurden. Details über entsprechende Kontrollmaßnahmen sowie Testergebnisse der Verarbeitung von Wortgraphen wurden bisher nicht veröffentlicht.

Goddeau hat in [God92] ebenfalls einen Vorschlag zur GLR-Erweiterung für spontansprachliche Anwendung vorgelegt. Bei Auftreten nicht-parsbaren Inputs wird eine Rückverfolgung entlang der Kette der deaktivierten Knoten durchgeführt, um eine fallweise Re-Reduktion der Stackdarstellung zu erreichen. Die Art und Weise der Situationsanalyse und die Aufstellung re–reduzierender Funktion (man spricht von 'error functions') geht aus dem Artikel nicht hervor. Offenbar handelt es sich um den Versuch, einen optimalen Pfad der Fehlersuche zu realisieren. Im Gegensatz dazu wäre in GLR* auch die Variante der Subkette [eat, many, geese] (durch SKIP des Startwortes) getestet worden. SKIP-Verarbeitung wird gar nicht erst auf den Fall von "impasses" (d.h. unparsbare Fehler) beschränkt, sondern prophylaktisch "auf Subkette geparsed", um die Zahl der Beampfade aufzufüllen. Da es sich um einen rein syntaktischen Parser handelt, entsteht dabei ein Vielzahl von pragmatisch nicht–adäquaten "parses".

2.5.2 SKIP-Verarbeitung

Die in dieser Abhandlung vorzustellende SKIP-Verarbeitung nutzt die bisher herausgearbeiteten Vorzüge der ERNEST-Kontrolle aus und bereichert sie um ein der Tendenz nach problem-unabhängig ausnutzbares Merkmal. Ich gebe zunächst ein knappes Resümé des bisherigen Untersuchung.

(**T**) Die vorgestellte Anwendung erzeugt mit Hilfe einer pragmatisch-ausgerichteten linguistischen Wissensbasis des ERNEST-Formalismus Tiefenstruktur-Beschreibungen für Sprachsignale.

(**L**) Das Hauptaugenmerk der Untersuchung war bisher darauf gerichtet, Basis- und Aufgabenkontrolle zu modifizieren, damit die inkrementelle Verarbeitung eingeführt werden kann.

(**R**) Expandierung in der inkrementellen Verarbeitung erfolgt rechts-assoziativ. In Kapitel 3 werden weitere Funktionen der Aufgabenkontrolle erläutert, die zur Bestimmung des Lowest Right Nonterminal (LRN) greifen. Damit wird das Prinzip der Rechts-Assoziativität im Sinne von Kimball [Kim73] in die Konzeptgraph-Analyse eingeführt.

(*****) **SKIP**–Verarbeitung bedeutet zunächst nichts anderes als das Überspringen von Wortkandidaten in der Analysephase I. Die Aufgabenkontrolle stützt sich auf Boole-Funktionen zur Bestimmung der Termination ausgezeichneter Entwicklungsstufen der Strukturbeschreibung. Gemäß der gegebenen Charakteristik der A*-Maschine sind solche Phasenmerkmale jeweils am aktiven Knoten des A*-Taktes abzulesen und gelten nur innerhalb des A*-Taktes. Außer den Testfunktionen zu den 3 großen Analysephasen werden kleinere Phasen betrachtet. SKIP-Verarbeitung fasse ich als Traversal durch den Wortgraphen in 2 Pässen auf. Die in Bild 2.37 dargestellte Überdeckungsphase **PASS1** ist die Phase des Überspringens. Unter **SKIP1** will ich den zum SKIP-Parsing in GLR* identischen Prozeß der Rückkehr zu inaktivierten (in ERNEST aus der OFFEN-Liste entfernten) Knoten verstehen. Hier geht es um die Erkennung nicht parsbarer Elemente des Wortgraphen.

Der Ansatz von GLR*, möglichst lange kohärente Subketten zu bilden, wird nicht weiterverfolgt. Das heißt nicht, daß das entsprechende Bewertungskriterium für Überdeckungen abgelehnt wird, obwohl ich es nicht verwende.

Statt dessen betrachte ich ein prophylaktisches Verhalten **SKIP2** zu ambigen und unsicheren Worthypothesen.

Die Ergebnisse beider SKIP-Formen werden in separaten Listen abgelegt:

1. Die (globale) ***Returnliste*** zu SKIP1 umfaßt die aus der weiteren Analyse auszuschließenden und an die Worterkennung als linguistisch UNZULAESSIG rückzumeldenden Worthypothesen.
2. Die ***Skipliste*** zu SKIP2 ist ebenfalls global, weil die abgelegten Kandidaten nicht durch Aufwärtsableitungen in einzelnen Konzeptgraphen falsifiziert wurden. Dann würden sie unter 1. fallen. In die Skip-Liste fallen sie wegen lexikalischer Merkmale (zu viele Wort- und Sub-Kategorien), die bereits auf der Hypothesen-Ebene zu diversen Suchbaumaufspaltungen Anlaß geben würden oder wegen Ambiguität der Konstituenten-Bestimmung.

Die in Bild 2.39 dargestellte Überdeckungsphase PASS1 ist durch 4 Aufwärtsableitungen, 3 SKIP2–Behandlungen besonders ambiger Wortkategorien sowie eine RESKIP1–Maßnahme gekennzeichnet. Letztere meint eine noch in PASS1 erfolgende Einbindung eines Wortkandidaten aus der Skipliste. Dafür gibt es einen zwingenden Grund. Die Modalität des Strukturmarkers M(SY_PNG,[Bonn]) ist KOHAERENT. Also wird zunächst inkrementell im Sinne einer rechts-assoziativen Expandierung nach obligatorischen Bestandteilen des Markers gesucht. Da die Konzeptdefinition (SY_PNG) solches nicht vorsieht, wird automatisch die linksseitige Expandierung veranlaßt. Diese Initiativen gehen von der Aufgabenkontrolle aus, die sich dabei auf die vorcompilierte Netzflußinformation stützt (zum Netzfluß siehe [Kum92a], zur Anwendung die Beschreibung der Funktionen get_user_aktion() und get_user_infolist() in Kapitel 3).

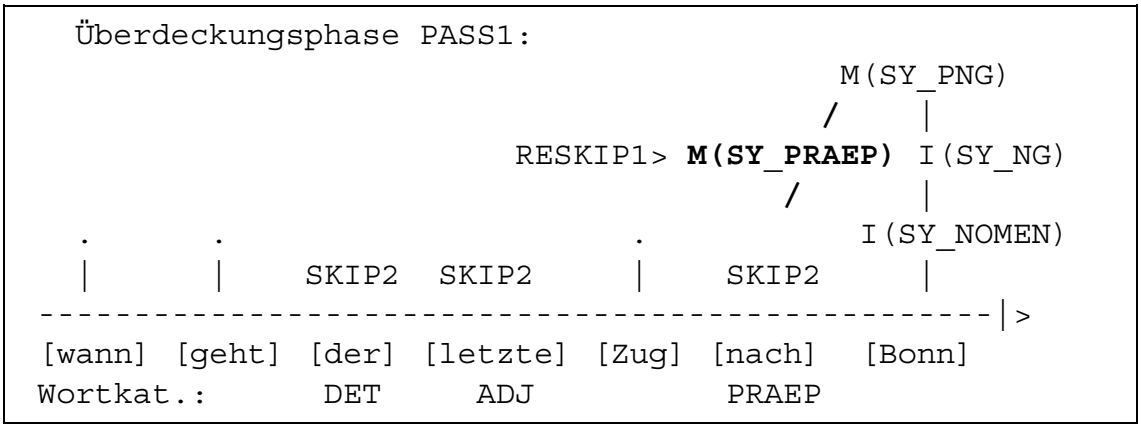


Bild 2.37 Überdeckungsphase I der SKIP-Verarbeitung

Nach erfolgreicher RESKIP-Maßnahme könnte die in der Skipliste abgelegte Hypothese [nach] gelöscht werden, wenn der aktive Knoten bis zum Analyseziel führen würde. Es erscheint zweckmäßig, für jeden Konzeptgraphen eine eigene Skipliste zu führen und nach RESKIP-Maßnahmen lokale Löschungen durchzuführen. Da ein zwingender Grund nicht vorliegt und nur ein geringer Mehraufwand beim Durchmustern der Skipliste entsteht, ziehe ich die einfachere globale Version vor. Konkurrente Wortkandidaten für das Intervall von [nach], die nicht mehr benötigt werden, könnte man nicht nur lokal löschen, sondern in die Rückgabeliste transferieren. Um solche

Fragen beherrschbar zu testen und überschaubar zu lösen, kam es darauf an, ein von anderen Kontrolleinheiten unabhängiges Modul zum Zugriff auf Wortkandidaten und Knotensegmente (Intervalle) zu schaffen (siehe in Kapitel 3 das zentrale Modul der SKIP-Verarbeitung create_skiplist()).

Die in Bild 2.39 dargestellte Überdeckungsphase **PASS2** ist die Phase der Bindung von Lücken der Überdeckung eines Konzeptgraphen durch Elemente der Skipliste. Die Durcharbeitung der Skipliste kann durch verschiedene Ordnungen ihrer Elemente unterstützt werden. Eine erneute Links-Rechts-Verarbeitung bringt Vorteile bei der Zuordnung von SKIP-Elementen und Lücken der assoziierten Wortkette. Dazu muß man die verwendete Methode für RESKIP2 betrachten.

Mit Sicherheit ist es wichtig, den vollständigen Shift des Wortgraphen, sprich der Wortkandidaten, zur Return- oder Skipliste zu realisieren, weil in PASS2 nur mit der Skipliste weitergearbeitet werden soll. Dazu muß man das *Problem des minimalen SKIP-Sprunges* beachten, das in dem Beispiel in Bild 2.38 sichtbar wird. Zur Realisierung von Skip1 und 2 benötigt man einen Zeiger auf den identischen Linksknoten der zuletzt behandelten Gruppe von Wortkandidaten. Im Beispiel wurde für 2 Vorkommen von [noch] SKIP1 veranlaßt. Die Wortkandidaten haben verschiedene Framelänge und besitzen deshalb unterschiedene Rechtsknoten. Bei inkrementellen Erkennern kommt es durch die Fensterbegrenzung häufig zu solchen Bildungen. Würde man bei der Inkrementierung des Zeigers auf einen Rechtsknoten der SKIP-Kandidaten nicht den Kandidaten kleinster Framelänge wählen, so würde einfach durch die Zeigerlogik die pragmatisch relevante Hypothese [nach], die an das kürzere Element anschließt, “übersehen”.

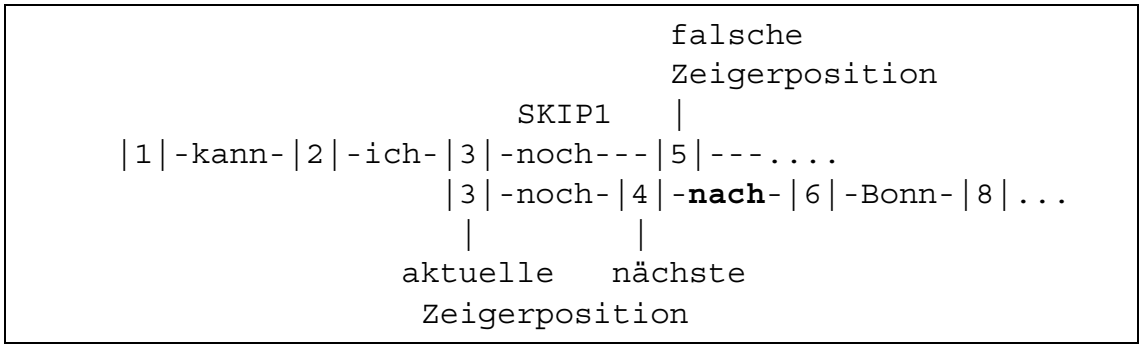


Bild 2.38 Problem der minimalen SKIP-Sprünge

Als Ableitungsmethode für PASS2 bietet sich die Aufwärtsableitung an. Dabei erscheint es angebracht nur solche Subziele zu stellen, zu deren Konzepttypen schon Strukturmarker existieren. In Bild 2.39 wird das um den Wortkandidat [haette] erweiterte Beispiel zu PASS1 wiederaufgenommen.

Eine generelle Bemerkung zum SKIP von Verbformen ist hier einzuschieben. Der zugefügte Kandidat ist wie alle Verben pragmatisch relevant, aber kein Vollverb. Wenn man mit allen Hilfs- und Modalverben SKIP2 durchführt, spart man die für verkürzte Anfragen wie

“Ich möchte nach Hamburg”

notwendige Suchbaumaufspaltung für die Rolleninterpretation

M(SY_MVERB,[moechte]) als Modalverb bzw. M(SY_VERB,[moechte]) als Vollverb. Man wartet ganz einfach das Ende von PASS1 ab.

Wichtiger als die Ersparung ist die Tatsache, daß man auf diese Weise eine einfache Lösung für ein Analyseproblem bekommt:

Nehmen wir an, die Modalverb-Interpretation wurde mittels des linguistischen Steuerparameters im Bewertungsvektor des Konzeptgraphen vorgezogen und nach diversen Expansionen und Aufschichtungen in der OFFEN-Liste stellt man inkrementell fest: der Wortgraph hat kein Vollverb. Dann kann man sich nicht dem Automatismus der Abarbeitung des nächstbesten Knoten der OFFEN-Liste überlassen, denn der relevante Knoten für den Vollverbfall liegt tiefer in der Queue und rutscht durch weitere Expansionen weiter ab. Eine analytische Lösung bestünde darin, die Liste der Returnknoten zurückzuverfolgen und unter den Nachfolgern ihres eindeutig bestimmten Nachfolgers (das ist der Knoten zum Neuziel M(H_WORTHYP)[27]) den Knoten zu I(H_WORTHYP, [moechte]) zu finden. Wie man sieht, hat auch die Simplizität der A*-Maschine ihren Preis.

Zurück zum Beispiel. Die Methodik für RESKIP2 mittels Aufwärtsableitung kann sich auf eine Vorabfalsifizierung der Subziele stützen. Im Beispiel kann man durch Netzflußanalysen (vergleiche dazu Kapitel 3, z.B. die Funktion test_konzept_kontext()) herausfinden, daß die Aufwärtsableitung zu [haette] auf einen zweiten Verbrahmen hinauslaufen würde. Man kann auch durch Ableitung den Konzeptgraphen von Bild 2.39 erzeugen und müßte dann in der Knotenbewertungsfunktion die Kollision der beiden Marker, d.h. den einfachen Tatbestand der Doppelexistenz, als UNZULAESSIG bewerten. Die Rückkehr zum Returnknoten steuert dann die Fortsetung von PASS2 , d.h. die Abarbeitung der Skiplist ein.

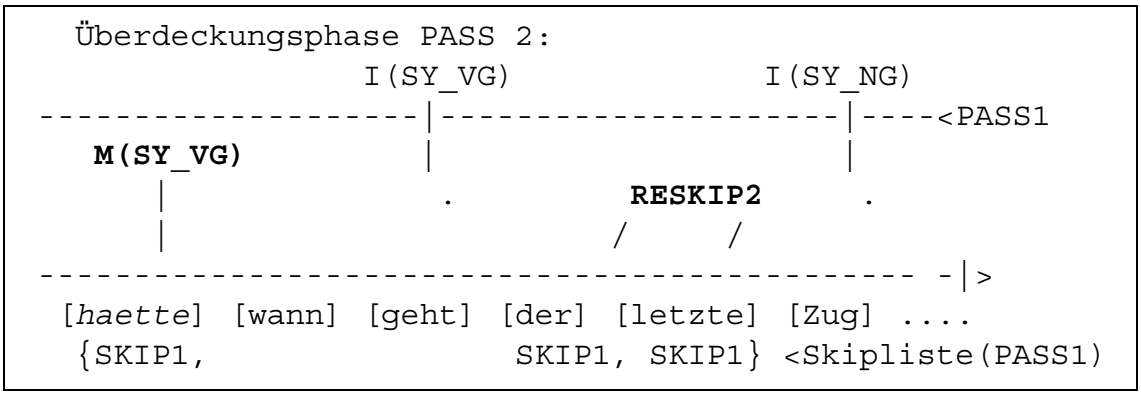


Bild 2.39 Überdeckungsphase II der SKIP-Verarbeitung

Ganz anders liegt der Fall bei der Aufwärtsableitung der Restkette [der, letzte]. Hier sollte es der vollständigen Ableitung auf der Ebene der Netzwerkattribute überlassen werden, herauszufinden, welche Modalität die 3 Lexeme zusammenfassen kann. Es sind diverse Modalitäten zu falsifizieren. Der Knoten mit der richtigen Subkategorie des Nomen kann noch früher entstanden sein als der aktuelle Returnknoten. Dann gibt es keine andere Wahl als die sukzessive Falsifizierung der Knoten, die in der Queue der OFFEN-Liste vorneliegen.

[27] im algemeinen Fall: M(H_HYPOTHESE)

Der Wortgraph in Bild 2.40 entspricht rellen Signaldaten, die unter verrauschten Bedingungen aufgenommen wurden. Ich beziehe absichtlich solche Daten in die Untersuchung ein, um den Beitrag zur Robustheit von Signalinterpretationen belegen zu können. Gemeint sind Daten mit starkem Hintergrundrauschen, das man sich anhand der Nebengeräusche beim Telefonieren in Bahnhofshallen vergegenwärtigen kann.

Infolge der Störeffekte besitzt der Graph ein ganzes Startsegment (Knoten 1–7) von Subketten, unter denen der gesprochene Satz gar nicht vorkommt. Man benötigt einen pragmatisch relevanten Startpunkt für die Tiefenstruktur.

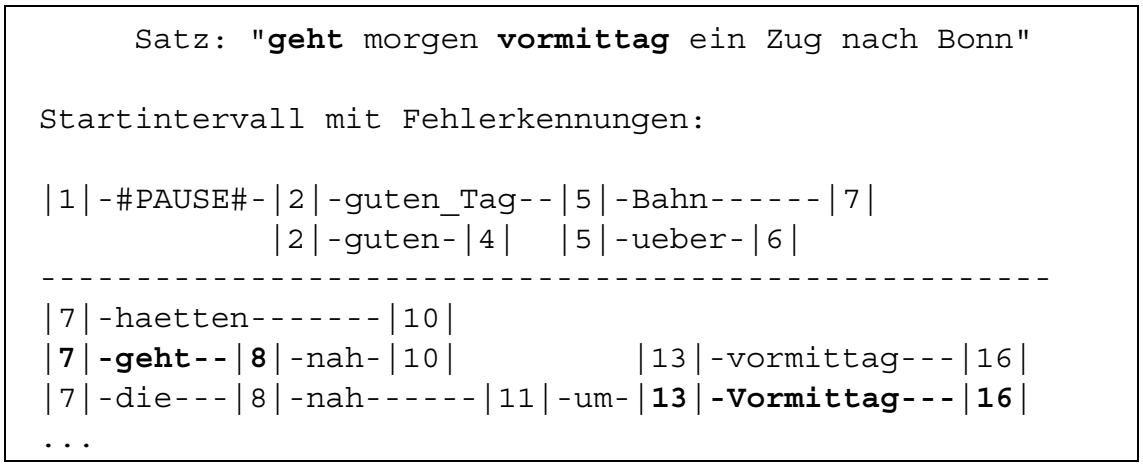


Bild 2.40 Wortgraph mit SKIP-Situationen (Realdaten)

Im oberen Graphsegment 1–7 liegt nur die verwertbare Grußformel, die direkt auf ein Subziel der Dialogschicht hin abgeleitet werden kann und keinerlei restriktiven Einfluß auf die Analyse hat. Man hätte nun die Wahl der beiden Listen. In die Returnliste gehört sie nicht, weil der Worterkenner aus ihr den Feedback zum Pfadpruning beziehen will. In die Skipliste gehört sie eigentlich auch nicht, denn diese Liste stellt für alle folgenden Strukturbeschreibungen das Reservoir für modell-getriebene links-assoziative Expandierungen/Aufwärtsableitungen dar. Der Kandidat [guten_Tag] würde zahlreichen Bindungsversuchen unterzogen werden, die er ohnehin nicht befriedigen kann. Also gibt ihn die Aufgabenkontrolle genau einmal zur Aufwärtsableitung frei. Es wird ein Baum erzeugt, deren Wurzel einen Konzepttyp in der Dialog-Schicht der Wissensbasis hat. Der Baum würde durch alle Suchbaumexpansionen “durchgeschleift”. Will man dies vermeiden, so wäre eine weitere gobale Liste für nachträglich zu substituierende Ableitungen erforderlich.

Alle anderen Kanten des oberen Graphsegments könnte man ebenfalls zur Aufwärtsableitung freigeben, obwohl der pragmatisch-relevante Startpunkt fehlt. Die Folge wären lose in der Strukturbeschreibung hängende Marker ohne Restriktionskraft und eine Aufschichtung von Suchbaumknoten in der OFFEN-Liste. Dabei ist eine Einschränkung zu berücksichtigen, welche die SKIP-Verarbeitung erfordert. Sie wurde in der Titelgebung von Unterabschnitt2.4.2 schon berücksichtigt:

SKIP-Verarbeitung im Sinne des Auslassens linguistisch ambiger und akustisch unsicherer[28] Wortkandidaten erfordert die Relativierung des Prinzips der Rechts-Assoziativität in der Aufwärtsableitung.

[28] erinnert sei an das Problem der kurzen Wörter

Es kann nur Adjazenz im Sinne von *Wortanordnung*[29] (wie in der ADJAZENZ-Matrix, siehe Bild 2.25) gefordert werden. Rechts/Links-Assoziativität verstehe ich als direkte *Benachbarung*. Insbesondere muß die Vorabfalsifikation unterlassen werden, ob eine Aufwärtsableitung zu einem noch nicht im Konzeptgraph (partiell) gebundenen Subziel die Leftmost-Bestimmung der Adjazenzmatrix befriedigt.

Für den Graphen in Bild 2.40 heißt das: würde man z.B. den Wortkandidaten [Bahn] im Startsegment zur Aufwärtsableitung freigeben, so würden sämtliche Modalitäten von M(SY_NG, [..., Bahn,..]) als Suchbaumknoten zu Buche schlagen, obwohl nicht sicher ist, daß man den “Auftakt” der Äußerung in der Hand hat. Stattdessen empfiehlt sich die Einführung einer Userfunktion, die vorgelegte Kandidaten nach Wortart als Aufsetzpunkt falsifiziert.

User-Regel (Startwortarten):

Wenn die assoziierte Wortkette des Konzeptgraphen leer ist, **dann** übergebe die Kandidaten, die keine Startwortart tragen können, der Returnliste.

Die Klassifikation der Startwortarten je Domäne muß genug Spielräume bieten, bei Fehlerkennungen überhaupt eine assoziierte Kette aufbauen zu können (vergleiche die [man]/[wann]-Vertauschung und die Entstehung des nicht-alltäglichen Anfragesatzes “man kann nach .... fahren”, Bild 1.32).

Im unteren Kantensegment von Bild 2.40 wird das Startwort [geht] gefunden. Durch SKIP1/SKIP2–Anwendung gelangt man zu [Vormittag] als Kern einer Zeitkonstituente, die ein RESKIP1 auslöst. so daß die Präposition [um] eingebunden wird.

2.5.3 Gedankenexperiment: Konzept-Spotting

In der Einleitung wurde bereits der Konzept-Spotter PHOENIX [War91a] als Vergleichssystem angesprochen. Ich gebe zunächst die zum Vergleich wichtigen Merkmale des Systems an und leite daraus am Schluß ein für meine Realisierung relevantes Gedankenexperiment ab.

PHOENIX wurde von W. Ward in einem 1994 an der Carnegie Mellon Universität verbreiteten “Overview” so gekennzeichnet:

’a system designed for development of *simple robust Natural Language interfaces* to applications.’

Es verwendet eine Frame–Architektur, um semantische Relationen zu repräsentieren. Jeder Frame verkörpert eine Basis-Operation der Anwendung. Die Frame-Slots werden durch Matching gegen den Eingabesatz gefüllt. Die zum Matchen verwendeten Muster, sprich Wortketten, sind als RTN (’Recursive Transition Networks’, [Woo70a]) gespeichert. Die zulässigen Ketten, die einen Slot füllen können, sind in Form hierarchischer Netze definiert, wie sie in Bild 2.41 auszugsweise für den Slot <interval> dargestellt sind. Ein “Intervall” ist eine pragmatisch interpretierbare Zeitbestimmung von Reisehandlungen.

Im Unterschied zu ERNEST-Netzen besteht keinerlei taxonomische Qualifikation der Netzverbindungen. Es gibt nur die Slot —Subslot-Beziehung, sozusagen eine reine Bestandteils-Hierarchie. Der von W. Ward behaupteten Simplizität entsprechend,

[29] englisch: word order

die auch auf kürzerfristige Erstellungszeiten solcher Netze rekurriert, handelt es sich nicht um homogene Konzeptnetze. Es werden Subnetze (z.B. INTERVAL), mit Sammelkonzepten (z.B. SOMETIME) und einfachen Token (z.B. to) gemixt. Ein * bedeutet die Optionalität des markierten Kettengliedes.

```
<interval>  (*SOMETIME INTERVAL)

INTERVAL
 (between <start_point> CONJ <end_point>),
 (*THAT_LASTS *FROM <start_point> *up TO <end_point>),
 (*THAT_LASTS *for *the +<entire> <unit>),
 (<duration> to <duration>),
 (<duration> to <dur_num>),
 (for *BABBLE <dur_num> to <dur_num> <unit>),
 (FROM <start_point>),
 (*up TILL *AROUND_ABOUT <end_point>)

SOMETIME
 (sometime),(anytime),(somewhere),(anywhere)

FROM
 (from),(starting),(that starts),(that start)

TILL
 (until),(till),(ending),(through),(into),
 (that gets over at),(that goes until),
 (that+ll go until),(that ends)

TO
 (to),(until),(till,(through),(into)

ALL_DAY
 (<all_day>)

THAT_LASTS
 (that goes),(that lasts),(runs),(for)
...
```

Bild 2.41 PHOENIX-Netz <intervall> für ATIS (Flugauskunft)

Der Spotter nimmt den Input-Satz und versucht die verschiedenen Netze (z.B. das Netz <interval>) auf die Wörter im Satz zu matchen. Dabei dürfen Worte auch übersprungen werden. Wird in einer partiellen Interpretation das Ende eines Netzes erreicht, wird in alle möglichen Nachfolgenetze expandiert. Diese sind in separaten Regeln festgehalten. Zeitigt das Matching für ein Slot–Netz Erfolg, so werden alle Super-Slots desselben aktiviert. Sie werden in Beamsuche [Win87b] weiterverfolgt, bei der Pfade

mit vielen übersprungenen Worten “bestraft” und gegebenenfalls herausgeschnitten werden.

Soweit sich die Konzeption beurteilen läßt, kann ihr wie auch immer weitverzweigter Aktivationsbaum im Sinne des strengen Automatismus der UP-Maschine realisiert werden. Eine große Expansionsbreite ist die Folge der Simplizität der Netzkonstruktion.

Problematisch scheint es auch, die Konstruktion auf Sprachen mit noch weniger strengen Regeln der Wortstellung zu übertragen. Das betrifft bereits die in meiner Abhandlung unterstellte relativ restriktive, deutschsprachige Domäne der Zugauskunft. Das hat zur Folge, daß die Konzepte der Netz-Schichten SEMANTIK und PRAGMATIK in ICZ ohne ADJAZENZ, d.h. frei von Stellungsregeln definiert wurden.

```
Konzept Z_TAG

MODALITAET 99  (Typ "fuer morgen")
 cat_heute:Z_ADV          ---> hypothese:H_WORTHYP
[feiertags(1) werktags(1) sonntags(1) sonnabends(1)
 samstags(1) freitags(1)  donnerstags(1) mittwochs(1)
 dienstags(1) montags(1) uebermorgen(2) morgen(2)
 heute(1) ]
 cat_wrt_fuer:Z_PRAEP     ---> hypothese:H_WORTHYP
[fuer(1)]

MODALITAET 100 (Typ "fuer den naechsten Montag")
 cat_montag:Z_NOMEN       ---> hypothese:H_WORTHYP
[Heiligabend(1) Pfingstmontag(1) Pfingstsonntag(1)
 Ostermontag(1) Ostersonntag(1)  Karfreitag(1) Sonntag(1)
 Sonnabend(1) Samstag(1)  Freitag(1) Donnerstag(1)
 Mittwoch(1) Dienstag(1) Montag(1) ]
 cat_fuer:Z_PRAEP:        ---> hypothese:H_WORTHYP
[um(1) ab(2) bis(1) vor(1) zu(1) nach(1) ueber(2)
 gegen(1) an(3) fuer(1) ]
 cat_artikel:Z_DET:       ---> hypothese:H_WORTHYP
[diesen(2) diesem(2) .... eines(1) einen(1) ...
 den(2) dem(1) das(2) der(3) die(2) ]
 cat_naechsten:Z_ADJ:     ---> hypothese:H_WORTHYP
[kommendem(1) kommendes(1) kommender(4) ....
 uebernaechster(1) uebernaechste(4) ... naechste(4)]

MODALITAET 101  (Typ "am kommenden Wochendende")
 cat_werktag:Z_NOMEN:     ---> hypothese:H_WORTHYP ;
[Osterwochenende(1) Pfingstwochenende(1) Vatertag(1)
 Himmelfahrtstag(1) Neujahrstag(1)  Feiertage(1)
 Feiertages(1) Feiertagen(1) Feiertag(1)  Werktagen(1)
 Werktage(1) Werktags(1) Werktag(1) Wochenenden(1)
 Wochenendes(1) Wochenende(1) ]
 cat_am:Z_PRAEP:          ---> hypothese:H_WORTHYP
[zum(1) am(1)]
 cat_fuer:Z_PRAEP, cat_artikel:Z_DET,
 cat_naechsten:Z_ADJ: wie in MODALITAET 100
 .....
```

Bild 2.42 Lexikon*–Expandierung eines ERNEST-Konzepts

Die Basiskontrolle von ERNEST bietet die Möglichkeit, ein beliebiges Netzkonzept entlang der Achse KONKRETISIERUNG/BESTANDTEIL und entsprechend sämtlicher Modalitäten nach unten zu expandieren. Statt des Wortgraphen werde eine Suche über Lexikon* durchgeführt. Damit kann man die Wortnummer-Restriktionen nach unten durchpropagieren und schließlich an allen Blatt-Konzepten die Extension der assoziierten Lexem-Mengen betrachten. In Einzelfällen unzureichende Restriktionen werden bei dieser Gelegenheit überprüfbar. Die stark verkürzte Darstellung des Resultats der Expandierung des Konzepts Z_TAG in Bild 2.42 zeigt die den Subkonzepten[30] (über die Vermittlungsstufe H_WORTHYP) assoziierten Lexem-Mengen. Die Einordnung in diese Mengen ist bis auf die in Klammern stehende ID-Nummer der Subkategorie (vergleiche dazu Kaptel 3) aufdifferenziert. Ihr Umfang sollte ein deutliches Indiz für die zu erwartende kombinatorische Explosion sein, die mit einer Recodierung der Wissensbasis in Form des Konzeptnetzes von PHOENIX verbunden wäre.

Als Gedankenexperiment erscheint mir die Umkehrung der Fragestellung angebracht:

würde man einmalig alle pragmatischen Wurzeln über Lexikon* expandieren, so wäre es möglich für jedes Lexem im Lexikon einen Baum der folgenden Art als direktes Abbild der Ableitungspotenz der Wissensbasis aufzubauen: Teil der Implementierung ist

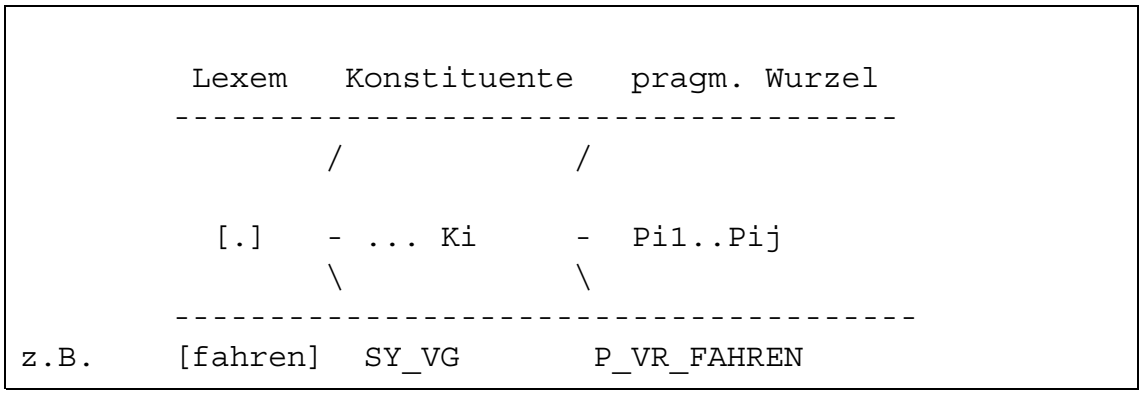


Bild 2.43 “Kurzschlußbaum” für Subziele

zur Zeit eine nur aus den äußeren Polen der unteren Beispielziele bestehende Zuordnung der Kontext-Konzepte, Verb->Verbrahmen, Nomen->Nomenrahmen. In Kapitel 3 werde ich Zweckbestimmung dieser Zuordnung erklären.

Eine mögliche Anwendung der SKIP-Fähigkeit von ICZ besteht darin, Lexeme mit diversen Subkategorien oder mindestens solche mit mehrdeutiger Kategoriezuordnung zu überspringen. Ein Lexem muß aber nicht notwendig mit allen kategorialen Rollen in die Extensionsmengen eingehen. Genau diese Information wäre aus den assoziierten Bäumen zu entnehmen.

Insbesondere fehlt in der aktuellen Implementierung eine Möglichkeit, die eindeutige Indikation (wie Woods sich ausdrückte) [Woo89] von Lexemen als Rollenträger der besonders flexiblen Modellierung von Zeitkonstituenten anders als durch weitgespannte Versuche zur Aufwärtsableitung herauszufinden.

---

[30] vor den Subkonzepten steht die Bezeichnung der Rolle, an der entlang expandiert wurde

2.5.4 “Spiele erfordern andere Suchprozeduren”

Die gewählte Überschrift spielt auf einen Abschnitt in dem Buch von Winston [Win87b] (S. 131f.) an. Er ist nur in dieser Ausgabe enthalten, fehlt aber in der dritten Auflage[Win87a]. Die Überlegungen von Winston fielen mir auf, nachdem ich in einer Vorstufe meiner Abhandlung gewisse Anleihen aus der Spieltheorie erwogen hatte. Das Interesse an diesen Fragen hält an, durch sinnvolle Analogien den Blick für die Besonderheit der Aufgabenkontrolle zu schärfen. Ihre Elemente, die Situationsanalyse und die Betrachtung des Reifegrades des Konzeptgraphen lassen diese als strategisches Verhalten erscheinen, das an das Verhalten des Strategiespielers erinnert. Es geht mir nicht um Anleihen aus der mathematischen Spieltheorie, soweit sich ihr Gegenstand ohnehin mit der Theorie der Suche überlappt [Kum88]. Winston richtete seine Aufmerksamkeit auf besonders anspruchsvolle Strategiespiele wie Schach. Die Diskussion seiner Überlegungen scheint aber gegenstandslos, denn auf S. 106 erklärt er im Untertitel zur Abb. 1.4.:

’Viele Prozeduren widmen sich dem Problem, zufriedenstellende Pfade zu finden (z.B. Tiefensuche). Andere konzentrieren sich auf das schwierigere Problem, optimale Pfade zu finden (z.B. A*, Einfügungen von mir gemäß der Abbildung). *Prozeduren für Spiele unterscheiden sich von den gewöhnlichen Pfad-suchenden Prozeduren, denn sie beschäftigen sich mit Gegnern.’*

Man betrachte aber die Aufgabe des Strategiespiels “Mahjong”. Gegeben ist eine Pyramide aus Steinen, deren Etiketttyp in der Aufschichtung 4 mal vorkommt. Das Ziel besteht darin, durch sukzessives Zeigen auf gleichetikettierte Paare die Aufschichtung vollständig abzubauen. Die Wegnahme der je gezeigten Paare ist nur möglich, wenn die Steine in Randstellungen der einzelnen Pyramidenschichten liegen. Es ist dem Problemlöser “Mensch” im allgemeinen erst nach vielfachem Backtracking möglich, die Aufgabe überhaupt zu lösen. Gutes Spiel besteht aus Zugfolgen, die möglichst viele weitere Paarbildungen freilegen. Man erkennt eine inkrementelle Problemstellung, die sogar Analogie zum Gegenstand der Abhandlung bietet, d.h. zur Reduktion des erst schrittweise sichtbar werdenden Wortgraphen.

Ersichtlich gibt es hier keinen Gegner. Einfache strategische Maßregeln bieten jedoch heuristische Anregung:

1. Wähle nur Züge, welche mindestens die Übersicht verbessern oder den Handlungsspielraum, möglichst aber beides, verbessern.
2. Verbrauche nicht vorzeitig Bindungsmöglichkeiten (Paarbildung vs. Phrasenbildung).

Winston betrachtete Zweipersonenspiele, weil er Bewertungsverteilungen auf einem “geordneten” ***Spielbaum***[31] betrachten möchte. Das Anordnungsprinzip für die Reihenfolge der Suchbaumzweige wird nicht einsichtig. Spricht er z.B. von dem “am weitesten links stehenden Zweig”, so scheint eine beliebige Auflistung von Zweigen gemeint, die je einer vorauszuberechnenden Zugfolge entsprechen. Der Spielbaum besteht nämlich aus Spielsituationen (“Stellungen”) als Knoten und den sie verbindenden Zügen als Kanten und ist nur ein Baum, insofern in jeder Spielsituation mehrere Zugentwicklungen (gedanklich) vorauszuberechnen sind. Vergleichbares behandle ich in Kapitel 3 unter dem Titel *“Vorabfalsifizierung”*.

[31] Pendant zum Suchbaum

Der “Mahjong”-Spielbaum scheint dem A*-Suchbaum direkt vergleichbar, wenn man den A*-Takt als Zug auffaßt und in Kauf nimmt, daß die Definition nachsichzieht, jeweils eine Menge konkurrenter Stellungen in einem Knoten zusammenzufassen. Das hätte den Vorteil, ihre gemeinsame Herkunft aus der Zugfolge rekonstruieren zu können. Die Vorgänger/Nachfolger-Beziehung reicht nach der Einführung des Returnkoten der SKIP-Verarbeitung dazu nicht aus.

Der Zweipersonen-Spielbaum ist ein geschichteter Baum, bei dem mit jeder Schicht die Spielerzuweisung der Knoten wechselt. Eine Parallele zu Zweipersonenspielen scheint sich bestenfalls anzubieten, wenn man die Entscheidungen der Aufgabenkontrolle (Aufwärtsableitung vs. Expandierung, Prior verwenden, Subziele/Zwischenziele) und die Effekte der Basiskontrolle in ein Gegensatzverhältnis zu bringen versucht. Eine andere Parallele paßt noch besser zur inkrementellen Verarbeitungssituation. Der “Anziehende” ist der Sprecher, seine Züge sind die (wie auch immer durch den Worterkenner gebrochenen oder vervielfachten) Wortkandidaten. Gegenzüge sind die Maßnahmen zu deren Reduktion (im Sinne des Shift-Reduce-Parsers). Aber auch in dieser Sicht scheint Stellungsbewertung unter Einbeziehung “feindlicher Absichten” keine brauchbare Analogie.

Winston unterschied insgesamt 5 Suchstrategien für Spiele, die ich stichwortartig besprechen möchte:

1. ***Minimax-Verfahren:***
   Es wird eine Bewertung errechnet, die Stellungen Platzziffern zuordnet (Position in der OFFEN-Liste). Weiter erfolgt Berechung von Vorteilszahlen für voraussehbare Stellungen (“Lookahead-Stellungen”). Man setzt voraus, daß die Lookahead-Prozedur weit genug “expandieren” kann. Man hat einen Zuggenerator (eventuell wie Aufwärtsableitung) zur Erzeugung von Wegen. Beides ist aufwendig und beruht auf der Stellungsbewertungszahl, die unscharf ist.
2. ***Alpha-Beta-Verfahren:***
   Prüfen, ob sich Zugfolgen als ungeeignet herausstellen, obwohl sie nicht bis zum Ende entwickelt wurden. Wenn ein bestimmter Knoten sich als nicht weiterentwickelbar erweist, prüfe ob eine Wiederaufnahme der Vorgänger-Knoten sinnvoll ist. Korrigiere die Bewertung von Vorgängerknoten, wenn sich für ihren Nachfolger ein genaueres Bild ergeben hat.
3. ***Fortschreitende Vertiefung:***
   Berechnen von Zeitfonds je Tiefenstufe des Lookahead je Zugvariante bei Spiel unter Zeitlimit. Bei zeitsynchroner Kopplung an den Worterkenner kann es sinnvoll sein ein Kriterium für solche Zeitscheiben einzuführen, um den Erkenner nicht zu lange auf Feedback warten zu lassen. Man könnte z.B. die Zahl erfolgloser Aufwärtsableitungen eines Wortkandidaten begrenzen.
4. ***Heuristisches Abschneiden:***
   Konzentration auf “plausiblere” Zugfolgen oder die vom Zuggenerator gelieferte “bessere“ Variante ist gefährlich. Es besteht die Gefahr, zugunsten kurzfristiger Stellungsvorteile Auswege aus den Augen zu verlieren (man spricht von “nahenden Katastrophen”: Aufgeben der Stellung). Keinen Lookahead für offensichtlich schlechte Züge durchführen (vergleiche: Filter für Wortarten mit Startfunktion für die Äußerung).

5. ***Heuristische Fortsetzung:***
   Verzögerungszüge (gewisse SKIP-Züge oder UNZULAESSIG-Setzung einzelner Strukturmarker) können negative Indizien aus dem Gesichtsfeld verdrängen (sog. “Horizonteffekt” von H.J.Berliner, Übersicht in[P.W85]). Deswegen kann es ratsam sein, statt Baumbegrenzung zeitweilig Baumerweiterung zu wählen.

Bei der Analogiebildung kann es sich nur um Anregungen handeln, deren Umsetzung nicht eindeutig bestimmt ist. So könnte man die “Heuristische Fortsetzung” im Sinne zeitweiliger Baumerweiterung mit der Subziel-angepaßten Breitensuche in Aufwärtsableitungen in Verbindung bringen. Bei der ereignis-orientierten Aufwärtsableitung für ausgewählte Wortkandidaten kommt es zu Suchbaumaufspaltungen (auch durch Unterscheidung der Subwortkategorien, siehe Kapitel 3). In diesem Fall wird standardmäßig Bestensuche betrieben – gleich ob mit Akzent auf akustische oder linguistische Bewertung. Das kann man unter “Heuristisches Abschneiden” einreihen. Und dabei kann man tatsächlich von einem ***Horizonteffekt*** sprechen. So werden z.B. Suchbaumaufspaltungen durch verschiedene lexikalische Subkategorien erzwungen. Bei der Tiefensuche kann man nicht alle Varianten parallel abarbeiten. So kann es durchaus geschehen, daß die richtige, aber nicht-expandierte Variante im Stapel der OFFEN-Queue relativ weit hinten stecken geblieben ist, d.h. nicht allein über die verwendeten Bewertungen wiederentdeckt werden kann. Es bleibt dann kein anderer Weg, als die obenliegenden fehlerbehafteten Expansionen sukzessive zu falsifizieren.

Als Anregung entnehme ich die Maßregel, alles aus dem Wortgraphen herauszulesen, was pragmatische Bedeutung hat. Es sollte aber auch geprüft werden, ob sich gewisse Verfeinerungen noch lohnen. Erinnert sei an den in Bild 2.39 angedeuteten RESKIP2–Schritt des in PASS1 ausgelassenen Artikels der Kette [der, letzte, Zug]. Die Nominalgruppe wurde durch Aufwärtsableitung allein des Nomen bereits instantiiert und die pragmatisch relevante adjektivische Bestimmung in sie eingebunden. Der im Kommentar zum Bild erläuterte Aufwand der Bindung des bestimmten Artikels [der] erscheint nicht gerechtfertigt, da der Artikel zur weiteren Dialogsteuerung nichts beiträgt. Demzufolge entschließe ich mich, Abstriche von der strikten Anwendung des Kriteriums des maximalen Subketten-Parsings zu machen. Mein Hauptakzent bei der Beurteilung finaler Eigenschaften der Strukturbeschreibung richtet sich auf die phasenbezogenen Kriterien.

Neu und bedenkenswert erscheinen auch die Verteilung von Vorteilswerten in bestimmten Zweigen des Suchbaums und rückwirkende Umwertungen von Spielbaumknoten.

2.5.5 Tendenz zum Intelligenten Tutoriellen System

Die linguistische Wissensbasis in ERNEST/ICZ ist ein komplexes Gemeinschaftswerk von Linguisten und Informatikern. Das System von allein 150 Konzepten kann eine Einzelperson ohne Benutzung von Erklärungshilfen kaum noch übersehen.

Suche unter ERNEST wird je Applikation durch 3 Kontroll-Module mit wie folgt zu unterscheidenden Allgemeinheitsgraden konstituiert:

```
Allgemeines    Besonderes      Einzelnes
-----------------------------------------
A*-            Tiefenstruktur  User-
Suche          Parser          Programme
```

ERNEST stellt aktuell eine Reihe von Defaultstrategien bereit, welche eine bestimmte Interaktion der 3 Einheiten organisiert. Der Bestand an Modellstrategien wird durch die neue Applikation erweitert. Es gibt also Spielräume. Eine Interaktion aller 3 Module zum Zwecke der inkrementellen, streng daten-getriebenen Kontrolle war bislang nicht möglich. Sie kann nicht allein durch bestimmte Userprogramme eingesteuert werden. Es waren Änderungen an der Konzeption und Implementation des Mittelgliedes notwendig. Sie werden in diesem Kapitel beschrieben.

Die dem User nicht direkt zugreifbaren Module seien als '**Experte**' aufgefaßt. Ein Wechsel in das Problemfeld der Expertensysteme soll damit nicht verbunden sein. Intelligente Tutorielle Systeme simulieren einen 'Experten', sprich Lehrer, der den Lernprozeß eines Lerners, Spielers oder Users — das sind an dieser Stelle Synonyme füreinander — anzuleiten hat. Eine Übersicht solcher Systeme gibt [Cla90].

In Hinsicht auf eine Bestimmung von Komponenten eines Intelligenten Lernsystems nach Lesgold in [Les88] wären folgende Module hervorzuheben:

A. Das Laufzeitsystem selbst, die interaktive Ausgabe und Darstellung des Suchbaums und der Konzeptgraphen,
   eine von mir realisierte Interpretershell zur Abarbeitung von User-Aktionen.
B. Die grafisch unterstützte Erklärungskomponente [Pre89], die z.B. den Grund von Suchbaumverzweigungen sowie die Nicht-Erfüllung von Attribut-Constraints angeben kann.
C. Die Überprüfung von User-Eingaben innerhalb der Interaktionsshell.
D. Die in [Fin93b] begründete dynamische Abbruch-Kontrolle, welche den Analyse- oder Lernfortschritt bewertet.

Die Betrachtungsweise als Tutorielles System würde den Rahmen dieser Arbeit sprengen und kann hier nicht weiterverfolgt werden. Die Sichtweise der Lernsysteme scheint aber für die Erweiterung der von mir implementierten interaktiven Usershell für ERNEST durchaus erstrebenswert. So wäre es z.B. sehr nützlich, wenn der Lerner innerhalb der Shell die in der ICZ-Wissensbasis implizit angelegte Funktion des Satzakzeptors testen könnte. Für Lernzwecke um so wichtiger wäre es, bei erkannter linguistischer UNZULAESSIGKEIT einerseits die kritischen Punkte der Strukturbeschreibung angeben zu können. Die Untersuchung von ERNEST als Formalismus für Prädikationssysteme sollte auch in diesem Sinne eine Verbesserung erbringen. Weit schwerer dürfte es sein, die falsifizierende Attributberechnung auf eine linguistisch auslegbare und für verschiedene Lernertypen plausible Klausel abzubilden. Das gleiche trifft auf die bislang rein prozedural codierten User-Regeln zu, die in einer interaktiven Shell als auswählbare Defaultregeln verfügbar zu machen wären.

Kapitel 3 Inhaltliche Aspekte der Implementation

Um den Bezug der nun zu beschreibenden Algorithmik zur reellen C-Code-Implementation der inkrementellen Kontrolle deutlicher herauszuarbeiten, wird eine Art Pidgin-C-Codierung zur Darstellung benutzt, die ohne tiefgehende C-Kenntnisse verständlich sein soll. Dies knüpft an eine Praxis der Darstellung von Algorithmen mit Pidgin-ALGOL an, die seit den 70iger Jahren angewandt wurde (z.B. in [Mac76]). Ich habe versucht, eine formale Pidgin-C-Spezifikation zu entwickeln, die mit möglichst wenig Definitionselementen auskommt.

3.1 Einführung von Pidgin-C

Codiert werden jeweils abgeschlossene Funktionen, beginnend mit einem Funktionskopf:

```
Ausgabetyp Funktionsname([Eingabetyp Typelement, ...])
```

Der mit {..} geklammerte Funktionskörper beginnt mit der Aufzählung der Datentypen in der Form:

```
{ Typ Typelement; ....
```

Die wichtigsten Typen sind:

```
ID  = Identifikator(Einträge,Konzepte u.a.),
KG  = Konzeptgraph,
WH  = Worthypothese,
BF  = Boole-Flag
IDL, WHL sind Liste der Typen ID, WH
```

Die in Erläuterungen stattfindenden Referenzen auf Funktionsnamen werden durch *name()* kenntlich gemacht.

Der im ERNEST-Manual angewandten Konvention zur Vergabe von Funktionsnamen folge ich, um die Ergänzung des Dokuments vorzubereiten. Die Konvention ist einerseits durch vorwiegend englische Funktionsnamen, andererseits durch englisch-deutsche Mischnamen gekennzeichnet. In den Mischnamen stehen englische Prefixe wie *get_*, *insert_* für den Operationstyps der Funktion, während die deutschsprachigen Anteile das primär behandelte Datenelement inhaltlich charakterisieren, z.B. *schaetz_eintrag* für den durch Situationsanalyse zu findenden Startpunkt einer Aufwärtsableitung.

Funktionen, die ein sogenanntes Boolsches oder zweiwertiges Wahrheitsflag nach Durchführung eines Testes zurückliefern, werden generell durch den Prefix *test_* gekennzeichnet. Die in Kapitel 1 anhand des PROLOG-Parsers als adäquates Implementationsmittel ausgewiesene Listenspeicherung, z.B. von ID-Datentypen, werde allein durch die folgend aufgezählten, blackbox–artig gedachten Verwaltungsfunktionen realisiert:

Zur Verwendung des Pidgin-C regten Verwendungen der Interpretersprache AWK für Zwecke der Linguistik an (z.B. als Textscanner). [Tis92]. Dabei handelt es sich um eine C–artige Makrosprache, an deren Vereinfachungen des C-Standards ich mich weitestgehend anschließen möchte.

3.2 Nomenklatur von Prozedurklassen

Die Darstellung der Implementation wirft durchaus inhaltliche Probleme auf. Es zeichnen sich Prozedurklassen ab, denen unterschiedlich starke Aufmerksamkeit zuteil werden muß:

1. **init()** für die Eröffnung,
2. **append()** für die elementweise Erweiterung,
3. **first_element()** für die elementweise Abfrage,
4. **remove()** für das elementweise Löschen,
5. **delete()** für die Aufhebung der gesamten Liste,
6. **test_empty()** für das Vorliegen der leeren Menge.

Ganz ausgeblendet werden somit die statische oder dynamische Speicherverwaltung und alle sonstigen Besonderheiten der Gestaltung der Listenfunktionen. Weiter gelten folgende Notationen:

```
=           Zuweisungoperation
==,!=       Komparatoren auf Gleich-,Ungleichheit
&&,||       Komparatoren auf logisches UND,ODER
;           Anweisungsende
NIL         Listenende
{} in {}    unbenannter Funktionsblock
break       Ausstiegshaken aus {}
return(p)   Funktionsausstieg mit Rückgabeparameter p
continue    Sprung zum Ende von {}
() in ()    Operationsvorrang in ()
```

Vergleiche sind Operationen, die ein Wahrheitsflag erzeugen. Mit **_k_** (wie Suchbaum**k**noten) oder **_ke_** (wie **k**noten**e**intrag) indizierte Funktionen analysieren statische Zustände des Konzeptgraphen insgesamt oder einzelner Einträge aus problemunabhängiger Sicht. Es werden nur solche Funktionen (mit Namen laut ERNEST-Manual) notiert, welche für den Zugriff der situierten Kontrolle hinreichend hochabstrahiert sind. Das sind sie nur dann, wenn man mit ihrer Hilfe unmittelbar inhaltliche Aspekte der Analysesituation erfassen kann.

C-spezifische Eigenschaft von Pidgin-C ist einzig die Beschränkung des return() auf maximal einen Rückgabewert p.

1. ***Prozeßkapseln***, welche die Struktogramme des Basisschemas direkt repräsentieren
   network_init() — die Initialisierung der A*-Maschine, des Suchbaums, der OFFEN-Liste, des semantischen Netzwerks und des Lexikonzugriffs,
   controlstep() — ein Step der A*-Maschine, Bestimmung des besten Knoten der OFFEN-Liste, Abbruch-Kontrolle,
   zielschaetzung(KG n) —Aufwärtsableitung,
   expandierung(KG n) — Situationsanalyse, Ereignisreaktion (durch Neuziel M(HYPOTHESE)) und Expandierung.
   Diese Gliederung kann auf jede ereignisorientierte ERNEST-Anwendung übertragen werden.
2. ***Basis-Routinen*** sind die tragenden Säulen innerhalb der Kapseln[1]. Innerhalb der zielschaetzung(), d.h der Organisation einer Kette elementarer Aufwärtsableitungen

[1] die leider nicht im ERNEST-Manual dokumentiert werden

bis zum Subziel, operieren folgende Funktion je Elementarschritt der Kette:
schaetz_k_knoten() — auf der Netzwerkschicht I (Konzeptauswahl). Sie wurde neu konzipiert. Ihre im wesentlichen erhaltene Subroutine
schaetz_k_konzept() — auf der Netzwerkschicht II (modalitätsbedingte Suchbaum-Aufspaltung und Top-Down-Propagierung nach Erreichen des Subziels). Neu darin ist nur der Aufruf der als nächtes zu nennenden Funktion.
connect_k_eintrag() — auf der Netzwerkschicht III (Attributbehandlung).

3. ***Basis-Kontrollfunktionen*** sind die (mit streng-indizierter Namensgebung[2] versehenen) Zugriffs- und Test-Funktionen innerhalb der Routinen. Für die Anwendung entstanden eine Reihe neuer, durch ihre streng problem-unabhängige Codierung wiederverwendbarer Funktionen. Die Funktionen ließen sich je nach Zugriff zu den Netzwerkschichten I–III ordnen. Zum Vergleich lassen sich die bereits im Manual beschriebenen Funktionen get_gradkon (ID konzept), get_anz_bestandteil (ID konzept) zur Schicht I rechnen, denn sie erlauben netz-positionelle Formbestimmungen inhaltlicher Art.
4. ***Basis-Hilfsfunktionen*** sind ebenfalls problem-unabhängig codiert, doch scheint ihre inhaltliche Relevanz (in der gewählten Kapselung) für andere Anwendung nicht sicher.
5. ***Statische/Dynamische*** **Kontrollfunktionen** realisieren den in Kapitel 2 angegebenen Satz sogenannter Userfunktionen. Es empfiehlt sich, den zunächst zur Anknüpfung an das Basis-Schema von Kummert beibehaltenen Ausdruck zu präzisieren, da er zu falschen Assoziationen Anlaß gibt: 'User' ist einerseits der Auskunftsuchende, dann der Kontrollprogrammierer, der die linguistische Wissensbasis als Fertige voraussetzt, schließlich derjenige, der den Bestand der ERNEST-Basisprozeduren (2.-4.) voraussetzt und damit auszukommen sucht. Würde der von mir vorgeschlagene Ausbau einer 'User'-Shell zum interaktiven Durchspielen der linguistischen Kompetenz des Tiefenstrukturparsers weiterverfolgt[3], so wäre auch eine Nutzung in Form einer einfachen Kommandosprache möglich.
6. **Lookup[4]-Funktionen** sind jene statischen Kontrollfunktionen, die durch ihre tabellarische, und damit bezüglich Kontrollfluß elementar vereinfachte Form eine mögliche Reformulierung mittels einer Deklarationssprache für Kontroll-Regeln unmittelbar vorbereiten.

Die statischen und dynamischen Kontrollfunktionen machen zusammen den problem-abhängigen Teil der Kontrolle aus. Formell läßt sich das an ihrem Programmcode ablesen, indem dort inhaltliche Termini der Wissensbasis, z.B. Namen von Konzepttypen, verwandt werden.

Dynamisch sind Kontrollfunktionen in dem häufig von Informatikern verwendeten Sinn von laufzeitabhängiger Eigendynamik. Es handelt sich um alle die Prozeduren, die aus der Analyse des Konzeptgraphen, genauer von Instantiierungszuständen in der Strukturbeschreibung, Kontrollflußentscheidungen ableiten. Der Fokus liegt dabei immer auf der Filiationslinie eines Konzeptgraphen, sprich eines Suchbaumpfades vom Startpunkt bis zu einem Knoten der OFFEN-Liste. Zu diesem Prozedurtyp gehören mit

---

[2] entsprechend dem ERNEST-Manual

[3] dem dient die unter 1. genannte Kapselung

[4] steht für Nachschautabelle und wird in der Weise benutzt: Einsprungindex —> Element(index)

Sicherheit jene Funktionen, die eine Phasenbestimmung des Konzeptgraphen im Sinne der in Kapitel 2 angegebenen Phasen I-III vornehmen. Teil der phasialen Entwicklung des Konzeptgraphen ist die Bestimmung des Grades der Überdeckung des Sprachsignals durch die assoziierte Wortkette desselben.

Statisch (nicht-dynamisch) sind dagegen jene Prozeduren, deren Kontrollfluß allein durch Regeln der linguistischen Wissensbasis oder Metaregeln der Kontrolle gesteuert wird. Sie haben von der Analysesituation unabhängige Geltung.

3.3 Aufwärtsableitung

Die Umgestaltung der Aufwärtsableitung unter den Gesichtspunkten Inkrementalität, Adjazenz-Abhängigkeit und Rechts-Assoziativität stellt das Kernstück der Implementation von ICZ dar. Bei der Darstellung gehe ich davon aus, daß diese Funktionsmenge als Modell–Beschreibung für weitere inkrementelle Anwendungen gelesen wird. Wichtig zum Verständnis sind die erforderlichen Änderungen an ERNEST-Basisroutinen. Besonders wichtig scheint es mir aber, das Arrangement und die Wechselwirkung jener Kernfunktionen zu erkennen, welche an der Strukturbeschreibung die Voraussetzungen der Aufwärtsableitung bloßlegen. Die Schrittfolge dieser Situationsanalyse beginnt immer mit der Entdeckung des Aktivationspunkts der Aufwärtsableitung durch die Funktion get_user_schaetz_eintrag().

3.3.1 Dynamische Kontrollfunktion get_user_schaetz_eintrag()

Die Funktion wählt in Abhängigkeit von der Analysephase aus der Menge der offenen Subziele des Konzeptgraphen den eindeutig bestimmten Startpunkt der nächsten Aufwärtsableitung. Die Namensgebung ist in folgender Weise motiviert:

Die Marker des Konzeptgraphen heißen im Quellcode und im ERNEST-Manual ***Einträge***. Konzepttyp und Ableitungsgrad des Markers bestimmen wesentlich die Wahl, sprich ***Schätzung***, des neuen Subziels für die Aufwärtsableitung.

Es soll gezeigt werden, daß der betreffende Marker mit Blick auf Inkrementalität durch Navigation im Konzeptgraphen aufgefunden werden muß. In der ERNEST-Implementation werden Konzeptgraphen als verzeigerte Listenstruktur realisiert, auf deren Form ich nicht detaillierter eingehen will[5]. Die Redeweisen von Einträgen respektive Markern dupliziert nicht schlechthin die Sachverhalte, sondern bildet sie auf Beziehungen von Listen-Datenstrukturen ab, die formell[6] interessant sind. ERNEST speichert für jeden Konzeptgraphen 2 situativ-eindeutige Anfangspunkte zur Navigation:

1. ***ziel_eintrag***[7]:
   Jeder Schritt einer Aufwärtsableitung erzeugt einen neuen Eintrag oder modifiziert den Ableitungsgrad eines Altziels (vergleiche die Erklärung der neuen ERNEST-Funktion connect_k_eintrag()).
2. ***end_eintrag***:
   Jeder letzte Ableitungsbaum hat einen zuletzt eingebundenen Zweig (er steht

[5] in der KI schien bis vor kurzem die Beschreibung von LISP-Listentypen wie der ’property list’ diskussionswürdig (Kritik in [Mar82])

[6] ich erinnere an meine Unterscheidung ’formell vs. formal’ am Anfang von Kapitel 1

[7] kleingeschriebene Termini verweisen auf arteigene Abstraktionen der Implementation

*right*most, abgekürzt: er steht in ***RM***-Position). Der Baum ist genau der Bezugspunkt der *Rechts*-Assoziation. Er hat ein eindeutig bestimmtes RM-Blatt. Es heiße der end_eintrag().

Beide Einträge bieten einen spezifischen Zugang zum Durchsuchen des Eintragssystems eines Konzeptgraphen. Es ist nicht mehr möglich, die beiden Suchverfahren als top-down und bottom-up zu charakterisieren. Im Fall der Kontrolle von Kummert traf dies weitestgehend zu, indem jeder Zyklus Aufwärtsableitung nur Subziele stellte, deren Konzepttyp hierarchisch-höher als bei der letzten Zielerreichung lag. Das ist inkrementell nicht durchführbar. Bei jeder Ereignis-Reaktion, sprich Einführung eines Neuziels M(H_HYP), entsteht wieder ein tieferliegender Startpunkt zur Aufwärtsableitung. In Bild 3.1 ist M(Z_ZEITANGABE) das aktuell (aufwärts) abgeleitete Subziel zum Startpunkt M(HYP, [morgen]). Läßt man die sonstigen Ableitungen erstmal beiseite, so ergibt sich:

```
nach Schritt 1:
  ziel_eintrag   =   M(Z_ZEITANGABE),
  end_eintrag    =   I(HYP,[morgen]).
```

Die im Bild 3.1 festgehaltenen Schritte fassen jeweils eine komplette Aufwärtsableitung je Wortkandidat zusammen. In Schritt 2 ist auf das Ereignis [um] zu reagieren, so daß erst das Neuziel und nach dessen Instantiierung der Marker I(HYP, [um]) eingetragen werden. Die von Kummert realisierte Kontrolle bedurfte keines Unterschiedes von *ziel_eintrag* versus *schaetz_eintrag* – der zuletzt abgeleitete Marker ziel_eintrag wurde nach alternierend eingeflochtenen Top-down-Phasen (ohne Situationsanalyse) als schaetz_eintrag der Aufwärtsableitung gesetzt. Dabei stellte sich jedoch eine vergleichbare Situation am Ende von Phase II, wenn man einmal versucht, die Phaseneinteilung der inkrementellen Verarbeitung in Analysebeispielen der Kontrolle von Kummert wiederzufinden. Das versuchte ich anhand der Bildfolge 2.19 in Kapitel 2. Die ERNEST-Basiskontrolle enthält Sonderbehandlungen für Kontext-Marker. Die Aufgaben-orientierte Kontrolle muß der exponierten Stellung derselbigen ebenfalls Rechnung tragen. Nach einem Aufwärtsschritt zum Subziel Kontext benötigt man einen Abwärtsschritt zu den kontext-gebundenen pragmatischen Wurzeln, um den Startpunkt zum Auskunftskonzept zu finden. Zumindest an diesem Punkt hatte man immer schon Navigation und Klassifikation (hier der Wurzeln) zu betreiben.

Im inkrementellen Fall ist dies durchgängig der Fall, wie die weitere Entwicklung des 2. Schritt in Bild 3.1 belegt. Vor Schritt 2 gilt die unveränderte Einstellung des ziel_eintrag auf M(Z_ZEITANGABE). Die Aufgabe der Funktion *get_user_schaetz_eintrag()* muß in dieser Situation darin bestehen, den schaetz_eintrag auf den aktuellen end_eintrag I(HYP,[um]) zu adjustieren, um in die Aufwärtsableitung, sprich ereignis-orientierte Verarbeitung, einzusteuern. Als notwendige Ereignisreaktion ist also gefordert

```
vor Schritt 2:
  schaetz_eintrag  = end_eintrag  = I(HYP,[um]),
  ziel_eintrag     = M(Z_ZEITANGABE)
  Zielkonzept      = Z_ZEITANGABE.
```

Mit dem 'Zielkonzept' ist in verkürzender Redeweise der Konzepttyp eines der möglichen Subziele gemeint, das mit dem von *get_user_schaetz_eintrag()* gelieferten Startpunkt der Aufwärtsableitung zugewiesen werden soll (vergleiche auch die schematische Darstellung zur Aufwärtsableitung im Basisschema, 2. Kapitel).

Der notwendigen Verlagerung des schaetz_eintrag zur Signalebene mit jeder Ereignis-Reaktion schließt sich der ziel_eintrag zu dem Zeitpunkt an, in dem ein durch Aufwärtsableitung gebundenes Zwischenziel (automatisch durch die ERNEST-Kontrollfunktion connect_k_eintrag()) auf ziel_eintrag adjustiert wird, z.B.

```
in  Schritt 2:
  end_eintrag    = I(HYP,[um]),
  Zwischenziel   = I(Z_PRAEP).
```

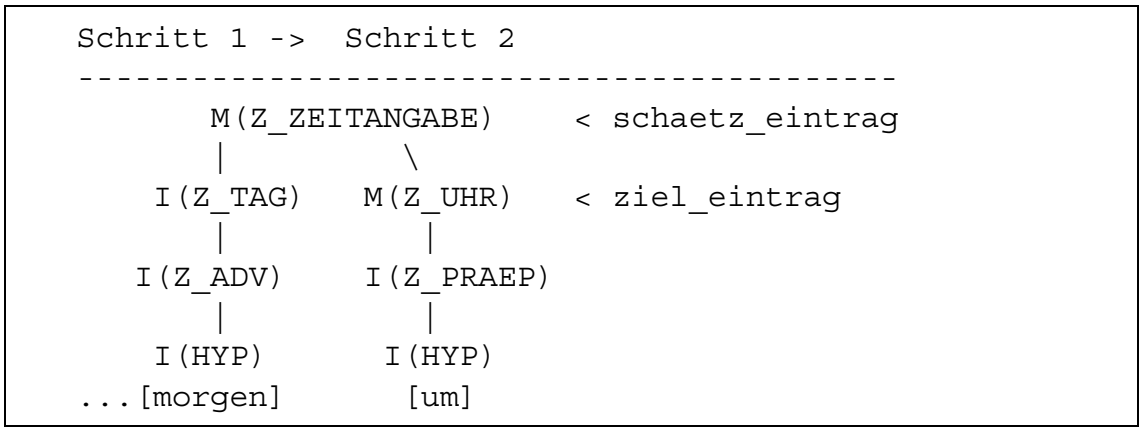


Bild 3.1 Unterschied von schaetz_eintrag und ziel_eintrag

M(Z_UHR) ist am Ende von Schritt 2 der letzte innerhalb einer Aufwärtsableitung dem Konzeptgraphen zugefügte Strukturmarker, d.h. *ziel_eintrag*. Als *schaetz_eintrag* ist für die gegebene Situation aber der Marker M(Z_ZEITANGABE) zu bestimmen, denn er drückt den aktuellen Stand der Ableitung der zur Subkette [morgen,um] gehörigen Konstituente aus. Deren bisheriger Ableitungsgrad steht zur Prüfung an, um zu entscheiden, ob die Konstituente bereits Anlaß zum Tiefenstrukturparsing in höheren Ableitungsschichten bietet.

Ist die gemachte Unterscheidung künstlich und für die situierte Kontrolle irrelevant? Zumindest beachtenswert scheint die alternative Definition:

'ziel_eintrag' ist der zuletzt durch Aufwärtsableitung im Ableitungsgrad modifizierte Strukturmarker des Konzeptgraphen.

Die damit erreichbare Vereinfachung im Begriffsapparat der Situationsanalyse von Konzeptgraphen kollidiert jedoch mit der zweiten Funktion, die der Begriff *ziel_eintrag* für den maschinellen Aufbau der Konzeptgraphen in Form der Listen–Datenstruktur hat. Dazu sei mir eine kurze Bemerkung erlaubt. Die Suche in solchen Listen ist nicht nur für die technische Basis der Situationsanalyse wichtig, sondern schon allein für den technischen Vorgang der Erweiterung des als Liste gespeicherten Graphen durch weitere Knoten und Kanten. Das Beispiel in Bild 3.1 spiegelt gerade eine typische Situation der inkrementellen Verarbeitung wider: ein aus früherem Ereignis partiell abgeleitetes, offenes Subziel wird durch eine weitere ereignis-getriebene

Aufwärtsableitung im Expandierungsgrad erweitert. Der entsprechende Marker war also bereits inmitten des Zeiger-Fadens der alle Einträge verbindenden Liste eingehängt. Die Listen-Funktion des ziel_eintrag ist es, als Anfangspunkt eines Fadens zu fungieren. Um Konzeptgraphen zu handhaben, wird man stets mehrere Zeigerfäden aufspannen müssen. Die vom Eintragungsmoment des einzelnen Markers abstrahierende reine Graphenstruktur reicht nicht aus. Sie besteht situativ aus isolierten Ableitungsbäumen und den Vorgänger/Nachfolger-Beziehungen[8] innerhalb derselben entsprechend der doppelten Netzhierarchie der Konzepttypen. Da man die rein zeitliche Reihenfolge-Beziehung ohnehin benötigt, bietet es sich an, die Liste der Einträge physisch gemäß ihrer Entstehungsfolge zu speichern. Über der rein zeitgeordneten Eintragskette bilden dann weitere Zeiger und Zeigerfäden die strukturellen Beziehungen des Konzeptgraphen ab. Die fundierende Liste heiße die *Basisliste* des Konzeptgraphen. Bild 3.2 bietet in verknappender Darstellung den Ausschnitt der Basisliste dar, wie er sich am Ende obengenannter Schritte 1 und 2 ergibt. Mit der Erweiterung der Basisliste werden entlang der Zeitachse die Postionen von end_eintrag und ziel_eintrag adjustiert (mit end(i), ziel(i) indiziert). Der ziel_eintrag vermittelt einen Zugang zum systematischen Durchsuchen des gesamten Konzeptgraphen. Der end_eintrag eignet sich eher zur Betrachtung des die letzten Ereignisreaktionen abbildenden (lokalen) Ableitungsbaumes im Konzeptgraphen. In dieser Weise werden beide Einträge an diversen Stellen der ERNEST-Basiskontrollfunktion benutzt und diese Implementation galt es zu erhalten.

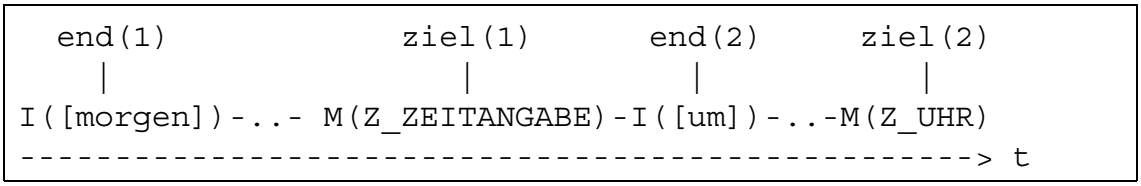


Bild 3.2 Basisliste des Konzeptgraphen

Um die begriffliche Identität von schaetz_eintrag und ziel_eintrag auch in der Implementation durchzusetzen, hätte man jeweils eine Reihe von Verzeigerungen der Basisliste vorzunehmen. Wenn der adjustierte schaetz_eintrag – wie anhand des Übergangs von Schritt 1 zu 2 erläutert – durch eine Ereignisreaktion und Aufwärtsableitung eines neuen Subziels außer Kraft gesetzt wird, so hätte man die Umverzeigerung der Basisliste völlig umsonst durchgeführt.

Die gegebene Implementation von ERNEST-Kontrollfunktionen sollte damit als eine sinnvolle, wenn auch keineswegs einzig mögliche gerechtfertigt sein. Betrachtet man den mit Beginn von Kapitel 3 eingeführten Unterschied von statischer und dynamischer Kontrolle, so erweist sich die vorgeführte Differenzierung nur für die technische Basis der Programmierung der statischen Kontrolle als relevant. Für die dynamische Kontrolle ist sie irrelevant, da sie höhere inhaltliche Abstraktionen als den Begriff ziel_eintrag benötigt. Die folgenden problemunabhängigen Funktionen sind Voraussetzungen der Funktion get_user_schaetz_eintrag().

---

[8] mit Definition gemäß den allgemeinen graphentheoretischen Relationen in Bäumen

3.3.1.1 Basis-Kontrollfunktion get_k_schaetz_eintrag()

Der von der Basiskontrolle geführte *end_eintrag* motiviert die Bestimmung der Wurzel des aktuell bearbeiteten Ableitungsbaumes durch Navigation in der Liste der Einträge: die Aufwärtssuche bis zum aktuellen Endpunkt im Pfad der Struktur-Vorgänger des end_eintrag. Besteht die mit der Funktion get_user_goallist() zu gestaltende Vorgabe der Subziele der Aufwärtsableitung in der Vorgabe zuerst der Konstituenten und danach der pragmatischen Wurzeln, so liefert die situierte Funktion get_k_schaetz_eintrag() dieselbigen für den aktuellen Ableitungsbaum zurück.

```
ID get_k_schaetz_eintrag(KG n)
{ ID e,v;
  e=end_eintrag(n);
  while((v=get_ke_vorgaenger(e)) != NIL)  { e=v; }
  return(e);
}
```

Bild 3.3 Pidgin-C: get_k_schaetz_eintrag()

Wendet man diesen Code auf die anhand von Bild 3.1 erläuterten Aufgaben der Adjustierung des schaetz_eintrag an, so ergeben sich folgende Fälle:

1. vor Schritt 1,2:
   k_schaetz_eintrag = end_eintrag = I(HYP,[..])
2. nach Ende von Schritt 1,2:
   k_schaetz_eintrag = M(Z_ZEITANGABE)

Im Falle einer Expandierung zeigt der end_eintrag per definitionem auf den zuletzt expandierten Marker, so daß get_k_schaetz_eintrag() auch in diesem Fall inhaltlich betrachtet zur Konstituente respektive Pragmatikwurzel führt.

3.3.1.2 Basis-Kontrollfunktion get_k_LN()

Obwohl in den Zusammenhang der situierten Kontrolle von Expandierungen gehörig, soll an dieser Stelle die Funktion zur Bestimmung des LRN[9] eingeführt werden. Im SKIP-Fall muß der aktuelle Ableitungszweig nicht notwendig der RM-Zweig sein.

Bild 3.4 zeigt eine SKIP-Situation, in der nachträglich ein Bindungspfad in die Konstituente M(Z_ZEITANGABE) eingehängt werden muß. Der Wortkandidat [am] sei aufgrund der linguistischen Ambiguität sowie Erkennungsunsicherheit von Präpositionen zum Zeitpunkt der Verarbeitung des zugehörigen Signalintervalls übersprungen worden. Der Ableitungzweig sei der zuletzt Bearbeitete. Wollte man die übersprungene Hypothese durch eine Expandierung einbinden, so würde die Bestimmung des LRN-Markers M(Z_ABSCHNITT) gerade den zutreffenden Ausgangspunkt derselben liefern.

---

[9] engl. Lowest Right Nonterminal, siehe Prinzip der Rechts-Assoziation in Kapitel 2

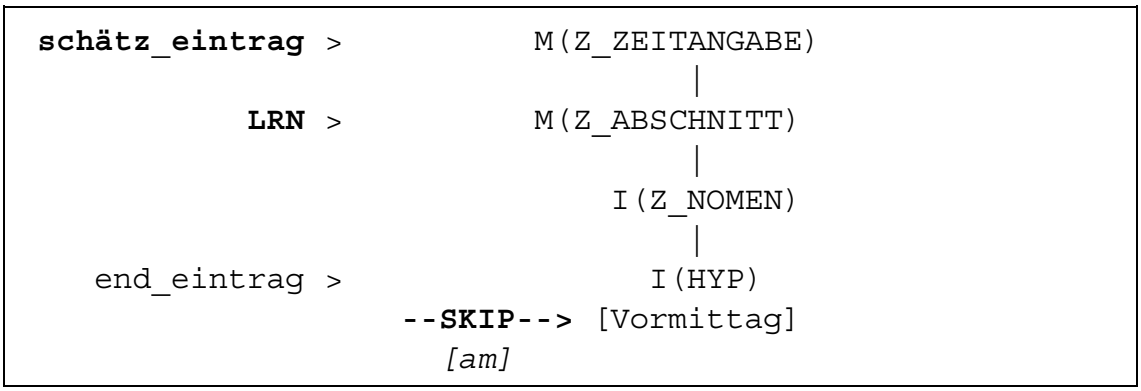


Bild 3.4 Unterschied schätz_eintrag vs. LRN-Eintrag

Der LRN-Zeiger wird stets auf den ersten, offenen Marker im RM-Ableitungszweig adjustiert. Wendet man seine formelle Bestimmung auf RESKIP-Situationen an, so hat man es stattdessen mit dem *Lowest–Nonterminal*–Marker des RM-Ableitungsbaums zu tun. Dafür wähle ich die Bezeichnung **LN**-Marker. Der Code der Funktion get_k_LN() wird gegeben durch:

```
ID get_k_LN(KG n)
{ ID e,v;
  e=end_eintrag(n);
  do { if (ableitungsgrad(e) == partiell) break;
       else
         if((v=get_ke_vorgaenger(e)) != NIL)
           v=e;
     }
  return(e);
}
```

Bild 3.5 Pidgin–C: get_k_LN()

3.3.1.3 Basis-Kontrollfunktion get_k_grad_list()

In Analysephase I ist es nicht ratsam, die vollständige Entwicklung der Ableitungsbäume bis zu ihren pragmatischen Wurzeln voranzutreiben, solange keine Kontext-Information gebunden wurde. Deren Entwicklungsstand ist dann zunächst durch die jeweilige Konstituenten–Wurzel charakterisiert. Eine Zeit-Konstituente z.B. gibt für sich betrachtet keinen Aufschluß über ihre Funktion als Rollenträger einer der pragmatischen Bestimmungen P_ANKUNFTSZEIT, P_ABFAHRTSZEIT in der gegebenen Äußerung.

Im Falle situationsbedingter Unbestimmtheit der pragmatischen Wurzel ist sogar eine Verschärfung der in Kapitel 2 definierten User-Regel ’semantische/pragmatische Subziele’ sinnvoll, indem selbst für instantiierte Konstituenten die weitere Aufwärtsableitung verzögert wird. Um so nötiger ist es dann, spätestens am Ende der Analysephase I diesen Konstituenten-Marker wiederaufzugreifen. Das Ende der Phase wird durch die Funktion test_end_phase1() festgestellt. Zur phasenabhängigen

Klassifizierung von Strukturmarkern und Listenbildung über Klassen von Einträgen dient die in Bild 3.6 codierte Funktion get_k_grad_list(). Sie verwendet den ID *grad* zur Auswahl der relevanten Hierarchieebene. Folgende an den grad angeküpfte (statische) im semantischen Netz formell–definierte Klassifikatoren werden verwendet:

1. grad = SYNTAX –> KONSTITUENTE:
   der Konzepttyp des Eintrags ist eine KONKRETISIERUNG_VON(),
2. grad = PRAGMATIK –> WURZEL:
   der Konzepttyp hat keine Bestandteile,
3. grad = PRAGMATIK –> INFO:
   wie _wurzel-Klasse,
4. grad = PRAGMATIK –> KONTEXT:
   der Konzepttyp ist nicht KONKRETISIERUNG_VON(), hat aber Bestandteile.

Im Unterschied zur Klasse2 der pragmatischen Wurzeln und Klasse3 der Kontexte sind Auskunftskonzepte dadurch gekennzeichnet, daß sie KONKRETISIERUNG_VON() (für die Dialogschicht des semantischen Netzes) sind und Bestandteile haben, d.h. die Kriterien dieser 3 Klassen sind exklusiv zueinander. Die folgenden dynamischen, weil an den Ableitungsgrad gebundenen Kriterien können ebenfalls problem-unabhängig bestimmt werden:

1. KONSTITUENTE: der Eintrag ist noch an keinen höheren Eintrag gebunden,
2. WURZEL: der Eintrag ist noch nicht instantiiert,
3. INFO: der Eintrag ist noch nicht an den Prior gebunden,
4. KONTEXT: ohne dynamische Bestimmung

Der in Bild 3.6 gezeigte Code der Basiskontrollfunktion get_k_grad_list() verbindet beide Arten von Kriterien rein problem-unabhängig mittels Funktionen des ERNEST-Manuals. Das Kontrollelement ’switch’ ist die seit dem Pidgin–ALGOL als ’computed goto’ bezeichnete Kontrollstruktur des Programmflusses, während ’case’ für die einzelnen Fälle desselben steht. Die Manual-Funktion get_gradkon(ID konzept) liefert einen ID der Hierarchie-Ebene von konzept.

Der bisher nicht erwähnte Terminus INFO steht für ein Informationselement der ERNEST–Modellierung, das sowohl durch die Basiskontrolle (zur Regelauswahl) als auch durch die Aufgabenkontrolle (z.B. zur Phasenbestimmung) ausgenutzt werden kann.

Zum Netzfluß, der in [Kum92a] (S.74) behandelt wird, soll nur eine knappe Darstellung gegeben werden. Es werden Zuordnungen der folgenden Art betrachtet:

je eine Modalität — je ein Kontext.

Beide Informationstypen unterstützen die Kontrolle der Regelauswahl auf der Basis semantischer Netze. Jedes Paar definiert genau eine Weise der Instantiierung eines Konzepts und heiße eine PRAEMISSE desselben. So kann z.B. das Konzept P_ANKUNFTSORT in mehr als 10 verschiedene Kontexte eingebunden werden, wobei die Modalität wechselt. Entsprechend viele Prämissen sind zu unterscheiden. Zu den analysevorbereitenden Maßnahmen des Netzwerk-Compilers von ERNEST gehört die Auflistung von ID-Nummern je Kontext und je Kanten-Rolle der Modalität der Prämisse. Genau diese ID, der Elementtyp des Informationsbündels, heiße ***Info***. Infos dienen einer Vorabprüfung der Ableitbarkeit von Prädikaten. Sie ist ohne die in der Attributschicht der dreischichtigen Netzwerk-Architektur modellierten linguistischen

Analysen realisierbar. Im Problemkreis der Strategiespiele hätte sich das Gemeinte als Lookahead-Prozedur bei der Vorausberechnung von Zugfolgen mit dem Ziel einer ***Vorabfalsifikation*** dargestellt. Netztechnisch stellt die Lösung eine Verfeinerung des Prinzips der Nach-Innen-Projektion der Konzeptrelationen in das Konzeptschema dar. Als dessen Realisierung wurde in Kapitel 1 die bloße Auflistung der zulässigen Kontexte und Modalitäten eines Konzepts angegeben.

```
IDL get_k_grad_list(KG n,ID grad,ID klassifikator)
  { ID e,n;
    IDL liste;
    init(liste);
    e=ziel_eintrag(n);
    do { if (get_gradkon(e) != grad) continue;
         switch(klassifikator)
           {
            case KONSTITUENTE:
              if
               (get_anz_konkret_von(konzept(e))) > 0
                    && get_ke_anz_vorgaenger(e) == 0)
                append(liste,e);
            case WURZEL:
              if
               (get_anz_bestandteil(konzept(e)))== 0
                    && ableitungsgrad(e)== partiell)
                append(liste,e);
            case INFO:
              if
               (get_anz_bestandteil(konzept(e)))==0
                && get_ke_vorgaenger(e) != prior(n))
                 append(liste,e);
            case KONTEXT:
              if
               (get_anz_konkret_von(konzept(e))== 0
               && get_anz_bestandteil(konzept(e)))>0)
                append(liste,e);
           }} return(liste);
  }
```

Bild 3.6 Pidgin-C: get_k_grad_list()

Zur Wirkung der Klassifikatorfunktion get_k_grad_list() im Fall INFO sei vorerst festgehalten, daß sie nicht direkt eine Liste vom Typ ID Info liefert[10], sondern zunächst Einträge, deren INFO anschließend zur Vorabfalsifikation heranzuziehen sind. Die ID Info liefert nicht zuletzt ein Steuerungselement zur Auswahl zulässiger Expandierungen unter inkrementellen Bedingungen und wird demgemäß in der zum ERNEST-Standard

[10] es sollte eine vereinheitlichte Funktionalität erhalten bleiben

zugeschlagenen Platzhalterfunktion *get_user_infolist()* angewendet. Platzhalterfunktionen können anwendungsbezogen ausgetauscht werden, müssen aber unbedingt codiert sein, um eine Gesamtkontrolle aufzubauen. Ihr Platz ist im Basisschema der Kontrolle festgeschrieben.

3.3.1.4 Basis-Kontrollfunktion test_konz_kontext()

Im Zusammenhang der Anwendung der ID Info innerhalb der Aufwärtsableitung soll diese Funktion Subziele ausschließen, die eine Vorabfalsifikation bezüglich eines gegebenenfalls etablierten Kontextes nicht überstehen.

```
BF test_konz_kontext(KG n, IDL kontextlist, ID goal,
                                              ID vol)
 { ID pr,e,m,info;
  if(test_empty(kontextlist)) return(TRUE);
  foreach (e in kontextlist)
   {
     m = modalitaet(e);
     foreach (pr in praem_list(goal))
      {
       if (m != pr_modalitaet(p)) continue;
       foreach ( info in info_list(pr, vol))
         {
          if (goal == info_konzept(info))
            return(TRUE);
         }}
  return(FALSE);
 }
```

Bild 3.7 Pidgin-C: test_konz_kontext()

Solange kein Kontext bekannt ist, liefert die Funktion ein TRUE-Flag. Ansonsten wird die Modalität des Kontexteintrags bestimmt und die Prämissen-Liste des zu testenden Subziels goal nach den zu dieser Modalität gehörigen Prämissen p durchsucht. Stimmt sie überein, so werden alle ID Info von p auf Übereinstimmung des Konzepttyps mit goal getestet. Bei der ersten Übereinstimmung gibt die Funktion TRUE als Zeichen zulässiger 'Subziel-zu-Kontext-Bindung' zurück. FALSE ergibt sich, wenn solche Info für keinen der Kontexteinträge gefunden werden kann.

Die Codierung dieser und folgender Funktionen enthält allgemeine Subfunktionen wie konzept(), modalitaet(), die Grundbegriffe von ERNEST prozedural realisieren, und spezielle Subfunktionen praem_list(), pr_modalitaet(), info_list(), die sich auf begriffliche Entitäten der Netzwerk–Implementation beziehen. Ich betrachte sie als hier nicht weiter zu detaillierende Makros im Sinne von Codezusammenfassungen untergeordneter ERNEST-Kontrollfunktionen.

Die praem_list (ID konzept) liefert die Netzfluß–PRAEMISSE [Kum92a] von konzept, der Makro pr_modalitaet() die den Sub-Prämisse dieser Liste zugeordneten

Modalitäten. Die info_list() kann durch den Parameter vol={ALLE, OBL} auf den Test aller oder nur der obligatorischen Bestandteile eingestellt werden.

Die Makros info_list(), info_konzept () greifen auf die gemeinsam mit der Wissensbasis vorcompilierte Netzflußliste zu, welche die Prämissen und die Info codiert.

3.3.1.5 Codierung von get_user_schaetz_eintrag()

Sie wird in Bild 3.8 angegeben. Ihre Gliederung folgt konsequent der in Kapitel 2 vorgestellten Einteilung einer Gesamtanalyse in 3 Phasen, deren Abschluß durch die Testfunktionen **test_end_phase1()** und **test_end_phase2()** überprüft wird.

Die Bestimmung des schaetz_eintrag wird an den jeweiligen Hauptzweck der Analysephase angepaßt. Das macht ihren dynamischen, von der Analysesituation abhängigen Charakter deutlich. Je Phase ist eine spezielle Interdependenz der Hauptfunktionen der dynamischen Steuerung der Aufwärtsableitung zu beachten. In der **ersten Phase** der Konstituentenbildung liefert get_k_schaetz_eintrag() jeweils den Wurzelpunkt des zuletzt eröffneten Zweiges der zuletzt bearbeiteten Konstituente. Es treten 2 Fälle auf, die bereits mit Blick auf die in Bild 3.9 codierte Funktion test_goalestimate() zu betrachten sind, welche über die Einleitung/Aufschiebung der nächsten Aufwärtsableitung entscheidet.

1. Die aktuelle Wurzel liegt in der Schicht HYPOTHESE:
   — Sie ist ein instantiiertes Neuziel und folglich zwingend als Startpunkt der Aufwärtsableitung von Konstituenten zu wählen
   — Sie ist ein via Expandierung abgeleiteter Marker, dessen notwendig partielle Bindung als Kriterium für test_goalestimate() dient, d.h. Aufwärtsableitung wird zugunsten der Instantiierung von Wortkandidaten verzögert.
2. Die aktuelle Wurzel liegt in der Schicht SYNTAX:
   dann kann es in Interdependenz mit der Funktion get_user_goallist(), genauer der Lookup-Funktion get_user_konstitlist() nur eine Konstituente sein, weil unterhalb der netz-modellierten Konstituenten keine Subziele der SYNTAX-Schicht gestellt werden. Deren Ableitungsgrad wird von test_goalestimate() als *notwendige Bedingung der Aufwärtsableitung in Phase 1* ausgewertet.

Bis zum Ende der Phase 1 setzt test_goalestimate() die *hinreichende Bedingung der Aufwärtsableitung in Phase 1*, sprich der Vorbereitung der Vorhersagefunktion des Verbrahmens, als “Bremse” der Aufwärtsableitung aller Konstituenten ein, die nicht vom SY_VG-Typ sind.

Spätestens in der **zweiten Phase** der Kontextbildung ist die Ableitung der pragmatischen Wurzeln der aufgeschobenen Konstituenten abzuschließen. Das ist der einzige Sinn von Aufwärtsableitung in Phase 2. Sie ist genau solange anzusteuern, als die Liste der ungebundenen Konstituenten, die mittels get_k_grad_list() bestimmt wird, nichtleer ist. Die Anwendung der Funktion first_element() dient lediglich der sukzessiven Leerung dieser Liste ohne Bedacht auf eine Vorrangbeziehung zwischen Konstituenten.

```
ID get_user_schaetz_eintrag(KG n)
{ ID  k;
  IDL konstituent_list;
  if (test_end_phase1(n) == FALSE)
     return(get_k_schaetz_eintrag(n));
  if (test_end_phase2(n) == FALSE)
   { konstituent_list = get_k_grad_list(n, SYNTAX);
     if (test_empty(konstituent_list))
       return (NIL);
     else
       return (first_element(konstituent_list));
   }
  if(test_end_phase2(n) == TRUE)
   { k = konzept(get_k_ziel_eintrag(n));
        /*Kriterium: unbekanntes Auskunftskonzept*/
    if (get_k_anz_konkret_von(k) ==0)
      return(get_ke_konz_eintrag(n,P_ANKUNFTSORT));
    else
      return(NIL);
 }}
```

Bild 3.8 Pidgin–C: get_user_schaetz_eintrag()

In der **dritten Phase** der Erkennung des Auskunftsziels sind Vor- und Nach-Phase der Ableitung desselben zu unterscheiden. In der Nach-Phase, sprich bei partiell gebundenem Analyseziel ist dessen Marker notwendig auch ziel_eintrag, da es sich in Phase 3 überhaupt nur um einen einzigen Schritt der Aufwärtsableitung handeln kann: nämlich ausgehend von einer etablierten pragmatischen Wurzel zum Analyseziel. In Umkehrung dessen genügt es nunmehr, den adjustierten ziel_eintrag negativ mit dem bereits genannten rein netz-technischen Kriterium zur Klassifikation von Auskunftskonzepten zu testen. Damit wird die Frage Vor- oder Nach-Phase entschieden. In der Vorphase ist ein passender Startpunkt der singulären Aufwärtsableitung zu wählen.

Die zusätzlich eingeführte Basis-Hilfsfunktion **get_ke_konz_eintrag()** findet zu einem vorgegebenen Konzept den (im allgemeinen nicht eindeutig bestimmten[11]) zugehörigen Eintrag im aktuellen Konzeptgraphen. Mit ihrer Hilfe kann man den 'größten gemeinsamen Nenner' aller Auskunftskonzepte, im aktuellen Netz das Konzept P_ANKUNFTSORT, aus der erreichten Strukturbeschreibung rückfragen. Als Seiteneffekt der gewählten Funktion User-Bewertung ist er notwendigerweise instantiiert, da der Suchbaumknoten des Konzeptgraphen ansonsten keine weitere Beachtung gefunden hätte.

3.3.1.6 Dynamische Kontrollfunktion test_goalestimate()

Die Funktion wird in einer Weise codiert, die implizit ***Seiteneffekte*** der Funktionen get_user_schaetz_eintrag() und get_user_goallist() voraussetzt als auch solche selbst

[11] sie wählt den ersten typ-gemäßen Eintrag, den sie findet

produziert. “Seiteneffekte” werden üblicherweise als unvermeidbarer Nachteil der rein prozeduralen Programmierung angesehen, die man durch gewisse Begrenzungen des Programmierstils, z.B. sparsamen Gebrauch globaler Variablen, strukturierte Programmierung u.a., zu reduzieren sucht. Bei der Codierung der genannten 3 Hauptfunktionen zur dynamischen Kontrolle der Aufwärtsableitung geht es dagegen darum, die sich real ergebenden Problemsituationen und die ihnen angemessene Reaktion zu definieren. So läßt sich anhand der Codierung in Bild 3.9 leicht ablesen, daß der Fall eines in der Hierarchie-Ebene SEMANTIK gelegenen schaetz_eintrag überhaupt nicht vorgesehen ist. Das beruht auf einer Wirkungskette von bewußt intendierten Seiteneffekten:

1. *get_user_goallist()*: konzentriert sich auf das Tiefenstrukturparsing, indem generell Subziele der Schichten SYNTAX (die Konstituenten) und PRAGMATIK (die pragmatischen Wurzeln) vorgegeben werden.
2. *get_user_schaetz_eintrag()*: ermittelt generell nur unter den in 1. genannten Subzielen den neuen Startpunkt (-> Tiefenstrukturparsing), falls der end_eintrag nicht vom Typ H_WORTHYP ist und zwingend zu wählen ist (-> daten-getriebene Inkrementalität).

   Würde man den folgenden Fall hinzufügen, so wäre nichts damit geändert:

   case SEMANTIK: return(FALSE);

Das Problem besteht vielmehr darin, daß die Abstimmung der Seiteneffekte der 3 Funktionen sozusagen ’von Hand’ in Programmierung und Test der dynamischen Kontrollfunktionen erreicht werden muß und durch einen Universal-Compiler Defizite der Abstimmung nicht bemerkt werden können. Anders gesagt, Netzwerk-Compiler und Kontroll-Compiler sind aktuell noch getrennt. Ein für Prädikationssysteme prädestinierter Compiler hätte eine Warn-Nachricht auszugeben, wenn er feststellt, daß für in A ausgewiesene Startpunkte überhaupt kein Test auf Aufwärtsableitung in B vorgesehen ist. Hier können nur gewisse Grundsteinlegungen für einen solchen Compiler gemacht werden, z.B. der Aufweis des prinzipiell deklarativ codierbaren Aufbaus der 3 Hauptfunktionen.

Verwiesen sei nochmals auf den weiteren Seiteneffekt betreffs des Startpunkt-Typs KONSTITUENTE. In der codierten Form werden vor Erreichen des Endes von Phase 1 nur Strukturmarker des Typs SY_VG bis zu einer pragmatischen Wurzel entfaltet. Die somit aufgeschobene, aber gleichermaßen zwingende Entwicklung für die anderen Konstituenten muß in Phase 2 subsequent durch die Startpunktwahl in get_user_schaetz_eintrag() aufgefangen werden. Das entpricht dem verbleibenden TRUE-Fall, falls die 2 voranstehenden Kriterien im ’case SYNTAX’ (unvollständig: Ableitung oder Phase1) dynamisch ausgemerzt wurden. Offensichtlich ergeben sich diesbetreffend weitaus schwierigere Probleme eines Wissensbasis und Kontrolle (trotz Modularität) integrierenden Compilers.

Dazu müßte man offensichtlich Konzepttypen oder Klassen von Typen (z.B. KONSTITUENTE) als solche deklarieren, die durch Aufwärtsableitung zwingend zu entwickeln sind. Zunächst müßte man sich überhaupt entschließen, in die Wissensbasis weitere Kontroll-Charaktere von Konzepten aufzunehmen. Statischen Bestimmungen, die bereits vor Analysebeginn feststehen, wie dem Kantenvorrang, müßte man in der Konsequenz **Dispositionen**, die aus der Analysesituation heraus zu bestätigen sind, zugesellen. Dies wäre im Fall der Aufwärtsableitung vom Startpunkt einer

pragmatischen Wurzel nötig. Die in der aktuellen Domäne gegebene Faktorisierbarkeit eines eindeutig bestimmten obligatorischen Konzepts (P_ANKUNFTSORT) in allen Prämissen aller Auskunftskonzepte ist eher als Ausnahmeerscheinung zu bewerten. Stattdessen müßten Wurzeln mindestens boolewertige Dispositionen für bestimmte Auskunftskonzepte zugeschrieben werden. Diese Problemstellung leitet unmittelbar zur Frage der Subziel-Bestimmung über. Welche Seiteneffekte negativer Art können durch falsche oder unvollständige Subzielvorgabe entstehen? Sollte die Subzielwahl durch Rückgriff auf den Informationstyp Prämisse/Info generell auf den Stand der Wissensbasis ausgerichtet werden? Das würde auf Laufzeit-Untersuchungen innerhalb der Wissensbasis hinauslaufen, die eigentlich zur Compilezeit machbar wären. Wie werden diese Fragen aktuell gelöst?

```
BF test_goalestimate(KG n)
{ ID e,k,a;
  e=get_user_schaetz_eintrag(n);
  k=konzept(e);
  switch(get_gradkon(k))
   {
   case HYPOTHESE:
     if (ableitungsgrad(e) == INSTANZ)
                              return(TRUE);
     else                     return(FALSE);
   case SYNTAX:    /*notwendige Bedingung*/
     if (ableitungsgrad(e) == partiell)
                              return(FALSE);
                   /*hinreichende Bedingung*/
     if (test_end_phase1(n) == FALSE)
       if (k != SY_VG )       return(FALSE);
     return(TRUE);
   case PRAGMATIK:
    if (test_end_phase2(n) == TRUE)
       if (get_anz_bestandteil(k) == 0)
                               return(TRUE);
     return(FALSE);
    }}
```

Bild 3.9 Pidgin–C: test_goalestimate()

3.3.2 Dynamische Kontrollfunktion get_user_goallist()

Zunächst ist die Beschreibung eingeführter Hilfsfunktionen zu geben. Die korrekte Zuordnung der Konstituenten bildet das Hauptproblem des Tiefenstrukturparsings in Phase 1. Ist der schaetz_eintrag vom Typ H_WORTHYP und soll Aufwärtsableitung stattfinden, so muß der Eintrag durch den Test auf Instantiiertheit in test_goalestimate() affirmiert worden seien. Als Instantiiertem muß ihm genau ein Wortkandidat mit folgenden Bestimmungen zugeschrieben worden sein:

1. Lexem: z.B. ’ein’,

2. Kategorie (Wortart) von 'ein': Determinans,
3. lexikalische Subkategorie zu Determinans: definit vs. indefinit.

3.3.2.1 Statische Lookup-Funktion get_wnr_kontextlist()

Die in Bild 3.10 codierte Funktion get_user_konstitlist() vollzieht
eine Zuweisung: Lexem —> Kontext-Konzept,
d.h. sie führt ein Indexikalisierung im Sinne von Woods [] aus.

```
IDL get_wnr_kontextlist (ID lexem, ID wortart)
{ ID NR_FAHREN,...,NR_GEBEN, NR_ZUG,...,NR_BAHN;
  IDL kontextlist;
  init(kontextlist);
  if ( wortart == VERB)
    {                              /* case P_VR_FAHREN*/
     if ( wnr == flexion(NR_FAHREN))
          || wnr == infinitiv(NR_FAHREN)))
                  append(kontextlist, P_VR_FAHREN);

    }
  if ( wortart == NOMEN)
    {                         /* case P_ZUG, P_NR_ZUG*/
     if ( wnr == NR_ZUG || wnr == NR_BAHN  || ...)
                  append(kontextlist, P_ZUG);
    } return(kontextlist);
}
```

Bild 3.10 Pidgin-C: get_wnr_kontextlist()

Die Funktion nutzt vorcompilierte statische Übereinstimmungen zwischen Wissensbasis und Lexikon und sichert damit die Invarianz der statischen problem-abhängigen Kontrolle gegen Änderungen der Wissensbasis. Die Funktion greift auf die vorcompilierte Lookuptabelle zurück, um zu einem Verb-Lexem zu prüfen, ob es mit dem Infinitiv oder der Flexion eines Verbrahmen-bindenden Verbs oder eines Nomenrahmenbindenden Nomens übereinstimmt.

3.3.2.2 Dynamische Lookup-Funktion get_user_konstitlist()

Die in Bild 3.11 codierte Funktion get_user_konstitlist() vollzieht die im allgemeinen

mehrdeutige Zuweisung: Kategorie —> Konstituenten-Konzept

mit Hilfe einer als switch{} implementierten Lookup-Tabelle. Die Funktion setzt auf einem gegebenen Konzeptgraphen, des schaetz_eintrag e und der vorinitialisierten goallist der Subziele auf. Der Einsprung-Index in die Tabelle wird mittels des Makros *wortart* (ID eintrag) aus der instantiierten Wortart des schaetz_eintrag bestimmt. Subkategorien finden keine Beachtung. Der switch erlaubt die weitere

Fallunterscheidung der Tabelleneinsprünge. So wird im 'case NOMEN' zunächst die Wortnummer des Lexems des schaetz_eintrag bestimmt. Dabei wird als implizite Anwendungsvoraussetzung der Funktion unterstellt:

konzept(schaetz_eintrag) = H_WORTHYP.

Anschließend wird betreffs Wortnummer und Klassifikator NOMEN die vorcompilierte Kontexttabelle befragt, ob ein passendes Kontextkonzept existiert. Ist dies nicht der Fall, so wird angenommen, daß es sich um eine Temporal-Bestimmung handelt. Das ist also ein indirekter Schluß: weil vorab bekannt ist, daß aus dem Nomen 'Zug' gewisse Nomenrahmen partiell abgeleitet werden können, kann erschlossen werden, daß der Konstituententyp auf SY_NG, SY_PNG zu beschränken ist. Es wird die Kenntnis der statischen Beziehung von Wissensbasis und Lexikon ausgenutzt. Zur Zeit gibt es keinen tabellarischen, vorcompilierten Ausweis über die Temporalbestimmungen von Wortarten. Diese Information ist implizit in den Codierungen der linguistischen Attribute versteckt und wird erst über den problemunabhängigen Constraints-Mechanismus von ERNEST wirksam.

Die Analyse dynamischer Abhängigkeiten wird in Bild 3.11 noch nicht sichtbar. Dazu denke man sich die Anwendung der allgemeinen append-Funktion innerhalb von get_user_konstit_list() wie folgt phasen-bestimmt: es wird mittels der Funktion **test_end_ueberdeckung()** geprüft, ob die Ereigniskontrolle die Abarbeitung des Signals bis zu dessen Ende vorangetrieben hat, wobei mittels SKIP übersprungene Signalintervalle zugelassen sind. Dadurch werden Vor/Nach-Phase der Konstituentenbildung unterschieden. In der Nach-Phase wird append() nur dann ausgeführt, sprich ein Subziel vergeben, wenn dieses bereits in der Vor-Phase gestellt wurde.

```
IDL get_user_konstit_list(KG n, ID e, IDL goallist)
{ ID wnr;
  IDL kont_list;
  switch(wortart(e))
    {
     case DETERMINANS: append(goallist,SY_NG);
     ...
     case NOMEN:
       wnr=lexem(e,H_WORTHYP);
       kont_list= get_wnr_kontextlist(wnr,NOMEN);
       if (test_empty(kont_list))
         {            append(goallist,SY_NG);
                      append(goallist,SY_PNG);
         }
       else           append(goallist,Z_ZEITANGABE);
      ...
     } return(goallist);
}
```

Bild 3.11 pidgin-C: get_user_konstit_list()

Mit dieser auf Inkrementalität ausgerichteten Gestaltung der Funktion get_user_konstit_list() wird implizit unterstellt, daß nach Erreichen der SKIP-Überdeckung nur noch erweiternde Ableitungen stattzufinden haben.

Die eingeführten Makros *wortart*, *lexem* (ID eintrag, ID konzept) unterstützen die Analyse der in Kapitel 1 angesprochenen Gegenstandsbildung. Jeder Strukturmarker besitzt seinen 'Gegenstand', d.h. eine je nach Ableitungsgrad gebundene Wortkette. Als partiell Abgeleitete hat sie modalitätsbedingte freie Stellen. Beides zusammen macht den 'Gegenstand' einer Prädikation aus. Die Speicherung all dieser gebundenen/freien Zustände ist nicht ratsam. Wirklich gespeichert wird eine Subketten-Fragmentierung der assoziierten Wortkette einer Strukturbeschreibung, wobei die Fragmente zusammenhängende Signalintervalle widerspiegeln. Dieser Zugang zur Gegenstandskonstitution wurde nicht weiterverfolgt. Stattdessen wurde der in diesem Abschnitt dargestellte Weg verfeinerter Navigation im Konzeptgraphen beschritten.

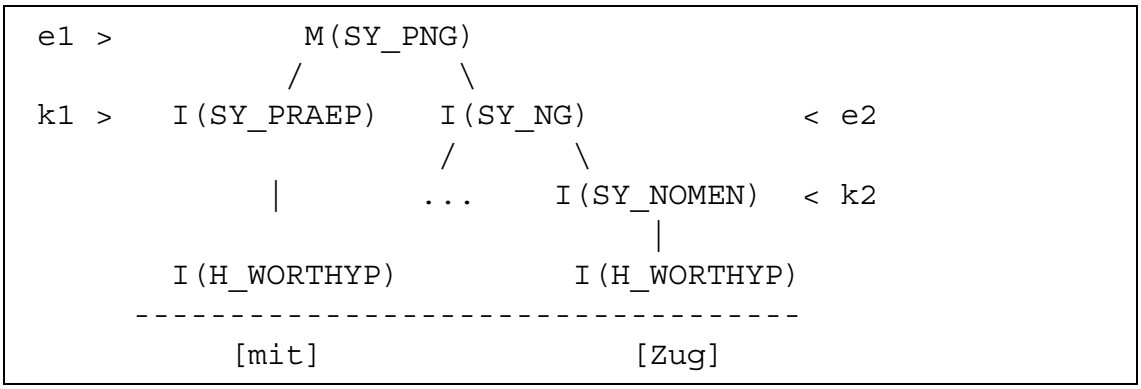


Bild 3.12 Anwendung von lexem() und wortart()

Es ergeben sich diverse Kontrollsituationen, in denen die Glieder der Wortkette eines Strukturmarkers wissenswert sind. Er ist dann wie M(SY_PNG) in Bild 3.12 als Wurzel eines Ableitungsbaumes zu betrachten, anhand dessen die Funktionsparameter (ID eintrag, ID konzept) soweit unten anzusetzen sind, daß nur noch ein einziger Weg zur Signalebene, sprich zu einer die Lexem-, Wortart-Informationen tragenden Instanz I(H_WORTHYP), verbleibt. Will man z.B. die Präposition von M(SY_PNG) erfragen, so wäre das im Bild verzeichnete Paar e1, k1 passend. Soll ein potentielles Nomen erfragt werden, so hat man zunächst mittels der Funktion *get_ke_nachf_konzept*(ID eintrag, ID konzept), den zu konzept (SY_NG) passenden Nachfolgereintrag zu finden[12]. Danach kann man mit dem Paar e2,k2 an die Stelle der gewünschten Instanz gelangen. Konzept k wird bereits am Punkt Eintrag e abgefragt[13].

Die charakteristische Form der durch die letztgenannten Funktionen vorbereiteten Funktion *get_user_konstit_list()* ist eine Lookup- oder Zuordnungstabelle. Das macht sie relativ transparent bezüglich Änderungen und Erweiterungen. Wäre die Zuordnungsweise zu den Wortarten homogen, so wäre die im Laufzeitsystem compilierte Tabellenform sogar als eine zum Zweck des Experimentierens deklarativ Änderbare gestaltfähig. Homogen ist die Konstituenten-Zuordnung deshalb nicht, weil sie für Nomen und Verben mit get_wnr_kontextlist() sogar vorausschauend auf potentiell mögliche

[12] es wird NIL ausgegeben, falls ein solcher nicht existiert

[13] siehe Anwendung von lexem() in Bild 3.11

Kontext-Rahmenkonzepte erfragt werden. Durch die Abfrage der vorcompilierten Korrespondenz zwischen Wissensbasis und Lexikon ist in diesen Fällen die statische Kontrollfunktion mit der Wissensbasis kompatibel.

Eine Subziel-Zuweisung – im allgemeinen als Konzeptmenge, d.h. in Form von Alternativen gefaßt – heiße *mit der Wissensbasis kompatibel oder* **konsistent**, wenn sie

1. gegen deren Detail-Änderungen[14] *invariant* ist,
2. im Sinne der Domäne *vollständig* ist.

Vollständigkeit bezüglich der Domäne und damit zur Befähigung der Gesamtanalyse zum Tiefenstrukturparsing ist jedoch für die übrigen Wortarten ein nicht-trivialer Ansatz. Man nehme das Beispiel der Temporalbestimmungen in Anfragen zur Zugauskunft. Tiefenstrukturparsing in ICZ stützt sich auf einen speziellen Konstituenten-Begriff:

A. Er ist über die konstituierenden Wortarten vermittelt (wie SY_PNG, SY_NG).
B. Er zielt auf bereits pragmatisch vordisponierte Konzepte (wie Z_ZEITANGABE, denkbar künftig auch eine O_ORTSANGABE).

Statisch, d.h. rein vom Standpunkt linguistischer Zulässigkeit betrachtet, sind Überschneidungen beider Fälle abzudecken:

[um, neun, Uhr]

kann linguistisch zulässig als PNG wie als ZEITANGABE abgeleitet werden. Dies ist ein Fall der im Kapitel 1 angesprochenen Mehrdeutigkeit von Wissensbasen. Die letztere Ableitungsweise ist als die pragmatisch bestimmte und deshalb dynamisch, im Zuge der inkrementellen Gegenstandsbildung, einzusteuern. Andererseits sind die nicht-domänenrelevanten Nomen (z.B. [Vormittag]) und die Adverbien sogar insgesamt ([morgen], [frueh] u.a.) geeignete Kandidaten zur Konstitution des Gegenstands von Z_ZEITANGABE in der gegebenen Domäne. Die Wahl dieses Subziels für diese Wortarten fixiert im Codebereich der Funktion get_user_konstit_list() ganz domänenspezifische Inferenzen mit impliziter Schlußweise:

Subziel-Regel 'NOMEN':
wenn *für ein gegebenes Nomen bekannt ist, daß es nicht zur Instantiierung eines Nomenrahmen der Wissensbasis beitragen kann,*
dann *wird es implizit als Z_NOMEN angenommen und die Konstituente Z_ZEITANGABE als Subziel aufgestellt.*

Subziel-Regel 'ADVERB':
wenn *die instantiierte wortart(eintrag) vom Typ ADVERB ist*
dann *wird implizit erschlossen, daß Z_ZEITANGABE als Subziel zu stellen ist, weil konzept(eintrag) nur für Z_Konzepte als notwendiges Bestandteil modelliert ist.*

Zur Aufstellung der Metaregel wurde zunächst nachgesehen, für welche Konzepte die Adverbien obligatorisches Bestandteil sein können. In der aktuellen Domäne kommen lediglich die Subkonzepte von Z_ZEITANGABE infrage. Es handelt sich um die Konzepte Z_TAG (z.B. für [heute], Z_ABSCHNITT (z.B. für [früh]) und Z_UHR (z.B. [etwa] in 'etwa um 3 Uhr'), die wesentlich die pragmatische Vordisposition von Z_ZEITANGABE ausmachen.

---

[14] gemeint sind Änderungen im Sinne von Verbesserungen, ohne Domänenwechsel

Die Regel für Adverbien ist quasi problem-unabhängig (im Sinne von ERNEST[15]) formuliert und insofern als Metaregel anzusprechen. In die Codierung der Lookup-Funktion geht davon nur die einfache Zuweisung ein:

case ADVERB: append(goallist,Z_ZEITANGABE);

Sie drückt sozusagen den Tatbestand der über die Modalitätsinformation restringierbaren Aufwärtsableitung auf genau ein Subziel für Adverbien aus.

Alle weiteren einfachen Zuweisungen sind ebenfalls auf diesen Restriktionstyp und die entsprechende Analyse der Modalitätsstruktur der Wissensbasis zu stützen. Wieder ist dabei zu beachten, daß die in diesem Kapitel behandelten Funktionen nicht von einem Spezialcompiler für die Kontrolle behandelt werden, der ihre Kompatibilität zur Wissensbasis zu prüfen imstande wäre. Da es sich um rein statische Sachverhalte handelt, möchte man allerdings vermeiden, die Lookup-Zuweisungen auf Laufzeitanalysen zur Modalitätsstruktur zurückführen zu müssen. Ideal wäre ein Fachsprachen-Compiler, der eine Markierung von Code-Blöcken als 'statisch' vorsieht und infolgedessen die im Block eingetragenen Anweisungen, sprich hier Analysen zur Wissensbasis, bereits zur Compilezeit ausführen kann. In [Bel73], [Dew75] wurden die Funktionsprinzipien sogenannter TIL – Sprachen [Loe81] herausgearbeitet, die solches gestatten. Solche Sprachen verkörpern einen "Dritten Weg" zwischen traditionellen Compilern und Interpretern. Die von mir in einer ersten Ausbaustufe implementierte Anwendung dieses Modells ist in C codierbar und als Subroutine in die C-Applikation einbindbar, so daß die bekannten Systemvorteile der C-Implementation erhalten bleiben.

Angenommen, das Kompatibilitätsproblem sei auf solche Weise oder wie bisher durch die korrekte Codierung der statischen Kontrollfunktion gelöst, so bleibt der dynamisch aufzulösende Aspekt der notwendigen Mehrdeutigkeit der Subzielbestimmung zu beachten. Das Nomen 'Zug' ist als Bindungsmittel für Nomenrahmen anerkannt. Als Konstituente kommen sowohl SY_NG ('der letzte Zug') als auch SY_PNG ('mit dem Zug') infrage. Solchen Ambiguitäten trägt get_user_konstit_list() durch Mehrfachbestimmung des Subziels Rechnung. Stattdessen hätte man die Aufwärtsableitung zunächst nur bis zum eindeutig bestimmten Zwischenziel SY_NG vorgeben können. Damit wären potentiell Subkonzepte von Konstituenten als Subziel zugelassen, deren zweite Aufwärtsableitung innerhalb der Syntaxschicht[16] nunmehr der Überwachung bedürfte. Die in Bild 3.9 codierte Funktion test_goalestimate() müßte dann im 'case SYNTAX' weiter verzweigt und mit einer weiteren Lookup-Funktion unterstützt werden, die Zwischenzielen ihre Konstituenten zuordnet. Hier war eine Wahlentscheidung zu treffen. Es wurde die Aussonderung der falschen Konstituente vermittels des Ableitungsprozesses gewählt.

3.3.2.3 Codierung von get_user_goallist():

Mit diesen Voraussetzungen kann die Codierung der dritten Haupkontrollfunktion zur Aufwärtsableitung vorgenommen werden.

Mittels switch wird in Bild 3.13 die Bearbeitung der 3 definitiven Hierarchieebenen tabelliert, in welchen Startpunkte durch get_user_schaetz_eintrag() angesetzt werden. Vorab wird generell die Kontextliste aufgebaut, um beim 'case HYPOTHESE

[15] gemeint ist die ausschließliche Berufung auf formell-netztechnische Kriterien

[16] gemeint ist die Aufwärtsableitung bis zur Konstituente

'die Vorab-Falsifizierungen durch get_wnr_kontextlist() durchführen zu können. Die Analysen dieses Einsprungs sind die von get_user_goallist().

```
IDL get_user_goallist(KG n)
{ ID e,ei,k,wnr,nf;
  IDL goallist,;
  if (test_goalestimate(n) == FALSE)  return(NIL);

  e=get_user_schaetz_eintrag(n);
  k=konzept(e);
  init(goallist);
  kontextlist=get_k_grad_list(n,PRAGMATIK,KONTEXT);

  switch(get_gradkon(k))
   {
   case HYPOTHESE:
     return(get_user_konstitlist(n,e,goallist));
   case SYNTAX:
     goallist = subswitch SYNTAX(k);  /*Bild 3.14*/
   case PRAGMATIK:
     if (k == P_ANKUNFTSORT)
      {
       foreach (ei in kontextlist)
        {
        if-one         /*hinreichende Bedingung*/
         (get_ke_nachf_konzept(ei,P_ABFAHRTSZEIT)
         || get_ke_nachf_konzept(ei,P_ANKUNFTSZEIT))
                  append(goallist,P_FAHRPLAUSK);
        else
                  append(goallist,P_VERBINDAUSK);
    }}} return(goallist);
}
```

Bild 3.13 Pidgin-C: get_user_goallist()

Der 'case SYNTAX' läuft in einem subswitch[17] über das Konzept des schaetz_eintrag. Die den Konstituenten als Subziele zuzuordnenden pragmatischen Wurzeln werden für SY_NG, _PNG, _VG (das sind Nomen- oder Verbrahmen) mittels get_wnr_kontextlist() ausgewählt. Die beiden temporal-bestimmten pragmatischen Wurzeln werden mit test_konz_kontext() vorab-falsifiziert[18].

Der 'case PRAGMATIK' hat den eindeutig bestimmten Fall
(*) konzept(schaetz_eintrag) = P_ANKUNFTSORT
zu behandeln. Der Term 'if-one' bewirkt die Abfrage, ob für mindestens einen der eingetragenen Kontextmarker eine temporale pragmatische Wurzel gebunden ist. Das

[17] ein switch innerhalb eines switch: in ihm gelten die Blockvariablen des case weiter

[18] der Parameter 'ALLE' im Funktionsaufruf negiert den Test allein der obligatorischen Bestandteile ('OBL')

entpricht der Realisierung der in Kapitel 2 definierten hinreichenden Bedingung zur Aufwärtsableitung des Analyseziels, während die mit (*) markierte Gleichheit die notwendige Bedingung ausdrückt.

```
subswitch SYNTAX(k)
  {    case SY_NG:            /*subcase SY_PRON*/
         if(get_ke_nachf_konzept(e,SY_PRON) != NIL)
                    append(goallist,P_REISENDER);
         else
           {                   /*subcase SY_NOMEN*/
             wnr = lexem(e,SY_NOMEN);
             if (wnr!= NIL)
               goallist=
               get_wnr_kontextlist(wnr,SY_NOMEN));
           }
       case Z_ZEITANGABE:
         if (test_konz_kontext(kontextlist,
                     P_ABFAHRTSZEIT,ALLE) == TRUE)
                   append(goallist,P_ABFAHRTSZEIT);
         if (test_konz_kontext(kontextlist,
                     P_ANKUNFTSZEIT,ALLE) == TRUE)
                   append(goallist,P_ANKUNFTSZEIT);
       case SY_VG:
         wnr = lexem(e,SY_VERB);
         goallist=
         get_wnr_kontextlist(wnr, SY_VERB));
       case SY_PNG:
          ne = get_ke_nachf_konzept(e,SY_NG);
          wnr= lexem(nf,SY_NOMEN);    /* case NOMEN*/
          if (wnr != NIL)
             goallist=
             get_wnr_kontextlist(wnr, SY_VERB));
          else                        /* case NPR*/
             wnr= lexem(ne,SY_NPR);
             if (wnr != NIL)
               { append(goallist,P_ABFAHRTSORT);
                 append(goallist,P_ANKUNFTSORT); }
    return(goallist);
    }
```

Bild 3.14 Subziele für die SYNTAX-Schicht

3.3.2.4 Dynamische Kontrollfunktion test_user_schaetz()

Die bisher beschriebenen 3 Hauptfunktionen interagieren im Basisschema der Kontrolle, um die Aufwärtsableitung auf Inkrementalität abzustimmen. Die Funktion test_user_schaetz() war bereits im Basisschema von Kummert vorgesehen, um

zusätzliche Vorab-Falsifizierungen der Zwischenziele der Aufwärtsableitung vorzunehmen.

Die Ableitung im Sinne von ERNEST bezieht Prüfungen auf allen 3 Netzwerkschichten, einschließlich der Attributschicht, ein. Bevor sich aufgrund eines Einzelkriteriums die linguistische Unzulässigkeit ergäbe, wären sämtliche Zwischenergebnisse der Instantiierung im Instanzenspeicher einzutragen. Dies reizt dazu an, durch Laufzeitanalyse der ADJAZENZ-Definition von Konzepten eine Vorabfalsifizierung durchzuführen.

Als Eingabeparameter der Funktion sind 3 Marker zu unterscheiden:

1. goal ist der Konzepttyp des aktuellen Subziels.
2. k1 ist der Konzepttyp des aktuellen Zwischenziels.
3. k2 ist der Konzepttyp des schaetz_eintrag.

Im linksstehenden Fall ist inkrementell auf das letzte Datenereignis [Nachmittag] reagiert worden. Die Aufwärtsableitung vom Startpunkt I(HYP) richtet sich auf das Subziel Z_ZEITANGABE (mit <goal[19]markiert) und geht vom aktuellen schaetz_eintrag I(Z_NOMEN) (markiert mit <k2) aus. Zur Prüfung liege das Zwischenziel Z_ABSCHNITT vor.

Als Eingangsparameter der Funktion test_user_schaetz() sind die Konzepttypen goal und k1 in vorgegeben. Die aktuelle Strukturbeschreibung geht in Form des schaetz_eintrag e (anstelle des obigen k2–Konzepts) ein. Die Tabellierung der Funktion mittels switch über das Konzept goal hätte die gemäß der Gestaltung von get_user_goallist() notwendigen Fälle zu betrachten:

- SYNTAX, d.h. goal ist Konstituente,
- PRAGMATIK, d.h. goal ist pragmatische Wurzel.

Wie die in Bild 3.15 angegebene Codierung zeigt, wird der SYNTAX –Fall in dieser Funktion nicht behandelt. Es sei zunächst der PRAGMATIK-Fall betrachtet. Aus der vorangehend beschriebenen Gestaltung der 3 Hauptfunktionen der dynamischen Kontrolle zur Aufwärtsableitung ergibt sich eindeutig: pragmatische Wurzeln werden erst entwickelt, nachdem bereits ein Kontextmarker abgeleitet wurde. Mit der zusätzlichen Basis-Kontrollfunktion **get_k_subkontext**(KG n) wird der im Zuge der Aufwärtsableitung des pragmatischen Kontextes als Zwischenziel entwickelte Subkontext aus der Strukturbeschreibung ermittelt.

Subkontext ist genau der Strukturmarker, der als Nachfolger eine elementare Bindung des pragmatischen Kontext darstellt, und dessen Konzepttyp die netz-technischen Kriterien erfüllt :

1. hat BESTANDTEIL,
2. ist KONKRETISIERUNG_VON().

In der vorliegenden Wissensbasis handelt es sich jeweils um das gleichnamige SEMANTIK-Konzept zum pragmatischen Verbrahmen (z.B. S_VR_FAHREN zu P_VR_FAHREN). In diesem Fall genügt Kriterium 1, um den Marker von den übrigen Kontext-Nachfolgern zu unterscheiden.

---

[19] auch die im ERNEST-Manual auftretenden Anglismen sind keine 1-1-Übersetzungen der deutschen Termini: goal steht für Subziel

```
BF test_user_schaetz(KG n, ID e, ID goal, ID k1)
{BF flag=TRUE;
 ID  subkontext;
 IDL kontext_list;
 if(goal == k1) return(TRUE);
 switch(get_gradkon(goal))
   {
   case PRAGMATIK:
     subkontext=get_k_subkontext(n);
     init(kontext_list);
     append(kontext_list,subkontext);
     if (test_konz_kontext(kontext_list,k1,ALLE) != TRUE)
        flag=FALSE;
   }
 delete(kontext_list);
 return(flag);
}
```

Bild 3.15 Pidgin–C: test_user_schaetz()

Zwischenziele können mit Blick auf den partiell abgeleiteten, zumindest modalitätsbestimmten Subkontext dadurch vorab-falsifiziert werden, indem sie den Prüfungen standhalten, die bereits anhand der Basis-Funktion *test_konz_kontext()* erklärt wurden. Letztere befragt die zur Modalität gehörige PRAEMISSE des Subkontextes, ob ein vorgelegtes Zwischenziel zur (vollständigen oder erweiterten) Ableitung desselben überhaupt verwendet werden kann.

Bild 3.16 zeigt zwei Beispielsituationen der Zwischenziel-Auswahl:

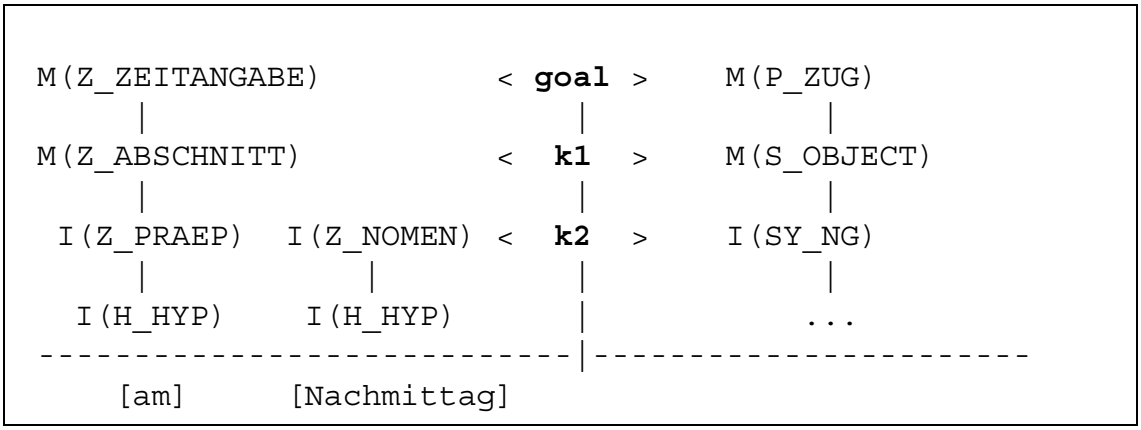


Bild 3.16 Anwendung von test_user_schaetz()

Man denke sich zum rechtsstehenden Beispiel in Bild 3.16 einen vorliegenden Kontextmarker zu P_VR_GEHEN. Von den formellen Netz-Vorgängern S_OBJECT, S_INSTRUMENT zu SY_NG[20] könnte nur S_OBJECT diesen Test bestehen, während S_INSTRUMENT 'negativ' vorab-falsifiziert wäre.

[20] als Konstituente bestimmt

Eine vergleichbare Vorgehensweise für Kontexte wäre zwar wünschenswert, doch steht dagegen, daß der Typ konzept(goal) im allgemeinen von mehreren offenen Konstituenten beansprucht wird. Das linke Beispiel in Bild 3.16 zeigt eine typische Anwendungssituation von test_user_schaetz() im SYNTAX-Fall:

Zwischenziele, sprich Konzepttypen, müssen jetzt zu bereits gebundenen Sub-Markern von Konstituenten zugeordnet werden, im Bild ist das I(Z_ABSCHNITT). Bei der jetzigen Gestaltung der Funktion wurde davon ausgegangen, daß man diesen Marker nicht unmittelbar identifizieren kann, nachdem verschiedene Bestimmungen der Netzwerkschichten I, II sowie über Navigation vermittelte Kriterien wie LN in Betracht gezogen wurden.

Mit der ADJAZENZ zum Konzepttyp des Eintrags hätte man die Information über zulässige Kantenrollen zur Verfügung, welche den schon gebundenen folgen dürfen. Überhaupt ist zu fragen, inwiefern von dieser Information in der Aufwärtsableitung systematisch Gebrauch gemacht wird. Vorgreifend auf die genauere Beschreibung der Kontrolle der SKIP-Verarbeitung sind 3 Phasen getrennt zu betrachten:

1. *Vor* Erreichen der Signalüberdeckung
   hat man bereits die SKIP-Situation zu unterstellen, d.h. man kann z.B. nicht testen, ob die Elementarableitung (entlang einer Kantenrolle) den via Adjazenz zulässigen Anfang einer Konstituente bilden darf[21].
2. *Nach* (gegf. mit SKIP) erreichter Signalüberdeckung
   werden unbearbeitete SKIP-Wortkandidaten durch Aufwärtsableitungen verifiziert/falsifiziert[22]. Dazu wird mit der neuen im Basisschema verankerten Funktion **connect_k_eintrag()** (siehe Struktogramm []) Aufwärtsableitung überwacht, ob mit diesen Kandidaten die offenen Bindungen der Konstituenten gesättigt werden können.
3. Bis zum Ende von *Phase 2*
   sind Rollen des Typs KONKRETISIERUNG bis zu den pragmatischen Wurzeln zu entwickeln[23]. Rahmenkonzepte sind in der aktuellen Wissensbasis adjazenz-frei definiert.

Die Argumentation lautet demnach, in 1. und 3. kann die ADJAZENZ nicht ausgewertet werden. In 2. wird sie ohnehin von der alle vorliegenden Strukturmarker untersuchenden Funktion connect_k_eintrag() eingesetzt, da diese auf allen 3 Netzwerkschichten prüft.

Ein nochmaliger Blick auf die im linksstehenden Beispiel von Bild 3.16 verzeichnete Auswahl temporaler Bestimmungen als Zwischenziele gibt jedoch Anlaß, im SYNTAX-Fall anhand lexikalischer Indizien Vorab-Falsifizierung vorzunehmen. In der in Bild 3.17 angegeben Code-Ergänzung zum switch() in Bild 3.15 wird zuerst mit dem Makro lexem() zum schaetz_eintrag (Konzepttyp Z_NOMEN) das Lexem des instantiierten Wortkandidaten aufgesucht. Es sei [Uhr]. Dann wird überprüft, ob die Konstellation für das Tripel (goal, k1, k2=konzept(schaetz_eintrag)) übereinstimmt mit:

Z_ZEITANGABE — Z_UHR — Z_NOMEN.

Andernfalls soll die Suchbaumaufspaltung aufgrund der Zwischenziele Z_TAG, Z_ABSCHNITT vorab-falsifiziert, sprich vermieden, werden. Die Form des darin

[21] selbst wenn man eine Kontrolle über die Lücken der Signalüberdeckung hätte, muß man Deletions des Worterkenners erwarten

[22] bisher wurde der Terminus 'Falsifikation' als beides übergreifender Vorgang verstanden

[23] ADJAZENZ schränkt nur die Beziehung von Rollen des allgemeinen Typs BESTANDTEIL ein

versteckten Metaregeltyps entspricht der bereits in get_wnr_kontextlist() verwandten Lookup-Tabelle mit Wortnummer wnr() als Einsprungindex. Sie hätte als solche noch kein vorcompiliertes Pendant im Netz.

```
case SYNTAX:
  if (lexem(e,H_WORTHYP) == wnr('Uhr'))
     if (goal == Z_ZEITANGABE && k1 != Z_UHR
                              && konzept(e) == Z_NOMEN)
         flag=FALSE;
```

Bild 3.17 Vorabfalsifizierung eines Zwischenziels

3.3.3 Änderungen der Basisroutine zur Aufwärtsableitung

Zur Realisierung der inkrementellen Verarbeitung mußten auch Änderungen in den Basis-Routinen zur Aufwärtsableitung vorgenommen werden. Den Hauptpunkt stellt die mit der Funktion **connect_k_eintrag()** erreichte Verallgemeinerung zur Abarbeitung des Einzelschritts der Aufwärtsableitung dar. Obwohl folgende Aufrufabhängigkeit zwischen den in diesem Abschnitt zu besprechenden Funktionen herrscht:

schaetz_k_knoten(): call schaetz_k_konzept(),

schaetz_k_konzept(): call *connect_k_eintrag()*,

soll die Funktion connect_k_eintrag() an den Anfang gestellt werden. Sie hat für weitere ereignis-orientierte Anwendungen unter ERNEST Bedeutung. Basisroutinen, obwohl sehr weit oben in der Aufrufhierarchie der Gesamtanalyse gelegen, findet man nicht im ERNEST-Manual beschrieben, da sie normalerweise nicht in den Zugriff der problemabhängigen Kontrolle gehören.

Die Notwendigkeit solcher Änderungen in der ERNEST-Basiskontrolle kann so zusammengefaßt werden:

Bei den bisherigen Anwendungen wurde von der Partialität der Aufwärtsableitung noch in eingeschränktem Maße Gebrauch gemacht, d.h. die elementar abgeleiteten Subziele wurden mit einer Ausnahme stets wieder als schaetz_eintrag gewählt. Die Ausnahme findet im letzten Schritt zum Analyseziel statt. In jedem Fall wurden offene Bindungen der so erreichten Subziele nur durch Expandierungen gesättigt. Jetzt geht es darum, auch durch Aufwärtsableitungen solche Bindungen im Verfolg der datengetriebenen “Oszillation” der hierarchischen Position des schaetz_eintrag erreichen zu können. Diese Möglichkeit war bisher nicht gegeben, gleichwohl aber das Instrumentarium dazu vorhanden.

In Kapitel 2 wurde dargelegt, daß ein Elementarschritt der Aufwärtsableitung auf die Expandierung eines vorher zu erzeugenden Strukturmarkers zum Zwischenziel zurückgeführt werden kann. So gesehen bestand bei der Sättigung kontextueller Abhängigkeiten, realisiert durch die Basisroutine expand_k_kontext(), immer schon eine vergleichbare Problemsituation. Einnert sei an die in Bild 3.16 (linkes Beispiel) verzeichnete Aufwärtsableitung von I(H_WORTHYP, [Nachmittag]) zum Subziel Z_ZEITANGABE. Es besteht die Gefahr bei der partiellen Aufwärtsableitung, den offenen Subziel–Marker der früheren Aufwärtsableitung zu “übersehen”. In der

Konsequenz würde ein zweiter Strukturmarker des gleichen Konzepttyps angelegt, was als Fehler der Strukturbeschreibung bewertet werden müßte.

Genau dies verhindert die in Bild 3.18 dargestellte Funktion connect_k_eintrag(). Sie überprüft jeden Eintrag des Konzeptgraphen:

A. Hat er den Konzepttyp des Zwischenziels?
B. Besitzt er eine offene Bindung, die durch den schaetz_eintrag gesättigt werden kann?

```
 connect_k_eintrag (KG n,ID e,ID subgoal,
                                IDL n_list,ID kopie(e))
{ KG n;
  ID  ei, exp;
  IDL eintrag_list;

  nf=first_element(n_list);
 foreach (ei in n)
  {
   eintrag_list=
  test_ke_expand_kontext(n,ei,e,subgoal,FALSE,TRUE);
    /* Kopien von exp und e in KG nf verbinden*/
      ...
   }
 if (test_empty(eintrag_list)== TRUE)
   { /* Bild3.19: neuen Eintrag hinzufügen, ...*/ ... }
}
```

Bild 3.18 Pidgin–C: connect_k_eintrag(), Aufwärtsableitung

Inhaltlich relevante Eingabeparameter sind der aktuelle Konzeptgraph n, der schaetz_eintrag e und das Zwischenziel subgoal. Unter den beiden übrigen Parametern ist nur die Sammelliste n_list für Suchbaumknoten zu besprechen. Sie wurde bereits in der aufrufenden Funktion schaetz_k_konzept() initialisiert. Nach dem Return wird die n_list dort weiterbehandelt. Sie enthält initial eine Kopie der aktuellen Strukturbeschreibung, von der dann in connect_k_eintrag weitere Klones erstellt werden. Bild 3.18 zeigt zunächst in (mit Kommentarzeilen /*...*/ angedeuteter) verknappter Form die Behandlung der neuen Problemsituation. Alle Einträge von n werden mit der bereits von expand_k_kontext() verwendeten Testfunktion *test_ke_expand_kontext()* zu den letztgenannten Kriterien A,B überprüft. Die Boolschen Parameter sind im Sinne der inkrementellen Kontrolle so zu wählen:

1. FALSE steht für den Programmzweig von test_ke_expand_kontext(), der Nicht-Kontext-Konzepte bearbeitet.
2. TRUE führt dazu, auch optionale Bindungen zur (erweiternden) Sättigung des offenen Subziels miteinzusetzen.

Wird ein passender Eintrag exp gefunden, so werden in einem Duplikat von n die Pendants zu diesem und zum schaetz_eintrag als Vorgänger/Nachfolger verbunden. Die Verwendung solcher Duplikate ist ein durchgängiger Zug der Implementation, der zwar

etwas speicheraufwendig ist, aber dafür den Aufwand zur Analyse der Konzeptgraphen verringert.

Existiert noch kein entsprechendes offenes Subziel, so wird der in Bild 3.19 dargestellte Codeblock wirksam. Hier ist zunächst ein leerer Eintragsframe e0 für das subgoal zu erzeugen. Die von test_ke_expand_kontext() für e0 zurückgelieferte eintrag_list enhält soviele Kopien von e0 wie es zulässige Modalitäten des Konzepts subgoal gibt, die durch konzept(schaetz_eintrag) gebunden werden können. Der diesbezüglichen Suchbaumaufspaltung ist durch Duplizierung des letzten Konzeptgraphen Rechnung zu tragen. In jedem dieser Knoten ist die (nicht näher beschriebene) Verbindung von e0/e und die elementare Ableitung auf der Netzwerkschicht III durchzuführen. Der markante Unterschied der beiden Codeblöcke besteht darin, daß in diesem Fall der ziel_eintrag adjustiert wird. Er zeigt nach Anwendung der Basis-Kontrollfunktion *insert_k_ziel_eintrag()* auf das neue offene Subziel e0. Im obigen Codeblock blieb der ziel_eintrag mangels Adjustierung auf dem zuletzt erreichten Zwischenziel stehen. Das führt genau zu der bereits erläuterten Differenzierung *ziel_eintrag vs. schaetz_eintrag.*

```
if (test_empty(eintrag_list)== TRUE)
 {
  e0 = neu_eintrag(nf);
  eintrag_list=
 test_ke_expand_kontext(nf,e0,e,subgoal,FALSE,TRUE);
  foreach (exp in eintrag_list)
   { /*
     Kopie von KG nf anlegen,
     neuen Eintrag exp mit e in Kopie(nf) verbinden,
     elementare Ableitung e0 :-(elem) e
     */
      ...
      insert_k_ziel_eintrag(kopie(nf),e0);
      append(n_list,kopie(nf));
 }}}
```

Bild 3.19 Codeblock connect_k_eintrag(), Neuziele

Die Funktion schaetz_k_konzept() befaßt sich mit der Behandlung der Beziehung Subziel/Zwischenziel. Sie ist bis auf den Aufruf von connect_k_eintrag() unverändert. Die in der Aufrufhierarchie höchste Funktion der Aufwärtsableitung ***schaetz_k_knoten()*** wurde um eine Steuerungsalternative erweitert, die in Hinsicht auf weitere Anwendungen noch als Vorschlag anzusehen ist, sich aber bisher bewährt hat. Sie betrifft nicht die Vorab-Falsifizierung vorgegebener Zwischenziele, sondern deren Vorgabe selbst. Das anhand von Bild 3.16 eingeführte Tripel

goal — k1 — k2=konzept(schaetz_eintrag)

bildet den Ausgangspunkt der Betrachtung, wobei der ID *goal* für das Subziel der Aufwärtsableitung, der ID *subgoal* für das zu bestimmende Zwischenziel steht. Die Funktion wertet die von get_user_goallist() vorgegebene Liste der Subziele aus. Je Subziel sind nunmehr 2 Varianten zur Auswahl der zugehörigen Zwischenziele wählbar,

zu denen je eine foreach-Schleife über die goallist stattfindet. Die im Codeblock des Bildes 3.21 dargestellte “Netz-Variante” entspricht im wesentlichen der bisherigen Form. Sie wird nur dann wirksam, wenn die “Lookup-Variante”, d.h. der Bezug von Information aus der zusätzlich eingeführten Tabellenfunktion ***get_user_subgoallist()***[24] zustandekommt. Besitzt diese Tabelle mindestens einen goal-relevanten Einsprung, so wird das Übergabe-Flag lookup gesetzt. Die Netz-Variante wird nur wirksam, wenn das Flag nach probeweisem Durchlaufen der Lookup-Variante auf seiner initialen Einstellung FALSE verharrt.

```
schaetz_k_knoten(KG n, ID e, IDL goal_list)
{
 BF  lookup = FALSE;
 ID  k2,konzept,goal,subgoal;
 IDL subgoal_list,k_list;
 k2=konzept(e);
 foreach(goal in goal_list)     /*Lookup-Variante*/
  {
   subgoallist = get_user_subgoal_list(n,goal);
   if (test_empty(subgoal_list)==FALSE)
     {
      lookup=TRUE;
      foreach(subgoal in subgoal_list)
        {
         if(test_netzmat(goal,subgoal) ==TRUE)
         schaetz_k_konzept(n,e,goal,subgoal);
        }
      delete(subgoal_list);
   }}
 if( lookup == FALSE)
     { ... /*Netz-Variante*/ }
}
```

Bild 3.20 Pidgin–C: schaetz_k_knoten()

Die Motivation der neueingeführten Variante hängt gerade mit der inkrementellen Verarbeitung zusammen, die diverse Anwendungen der Basisroutine schaetz_k_knoten() für Aufwärtsableitungen des Typs:

Startpunkt = I(H_WORTHYP, [..]) — Subziel = Konstituente

vorzunehmen hat. Man betrachte in Bild 3.21 die Anwendung der älteren Netzvariante auf diesen Fall. Um die zulässigen Zwischenziel-Kandidaten zu finden, werden zunächst alle Konzepte ermittelt, deren Bestandteil oder Konkretisierung das Konzept H_WORTHYP laut Netzdeklaration sein kann. Die Basis-Subroutine insert_klist() fügt noch alle spezialisierenden Konzepte derselben hinzu. Infolgedessen hat man in besagter Analysesituation jedesmal die Gesamtliste der Wortart-Konzepte durchzuarbeiten. Zur Reduktion genau dieses Tatbestands dient die weiter unten (in Bild 3.23) gezeigte Funktion get_user_subgoallist().

[24] das Pendant zu get_user_goallist() für Zwischenziele

```
/*Netz-Variante*/
    {
     init(k_list);
     foreach(konzept in bestandteil_von(k2)
          || konzept in konkretisierung_von(k2))
       {
        append(k_list,konzept);
        insert_klist_spez(k2, k_list, konzept);
       }
     foreach(goal in goal_list)
       foreach(subgoal in k_list)
         {
          if(test_k_subgoal(n,e,goal,subgoal)==TRUE)
            schaetz_k_konzept(n,e,goal,subgoal);
         }
      delete(k_list);
     }
```

Bild 3.21 Codeblock schaetz_k_knoten(), Auswahl der Zwischenziele

Eine weitere Alternative zur Netz–Variante, deren Besonderheit das Auffinden einer Konzeptmenge zur Laufzeit, die eigentlich statisch, zur Compilezeit, bekannt ist, wäre deren Vorcompilation. Als Kern der Funktion bliebe dann die doppelte foreach-Schleife zuerst über alle Subziele und zu diesen über alle Zwischenzielkandidaten übrig. Innerhalb dieses Kontrollflußblocks kapselt dann die neueingeführte Basis-Subroutine ***test_k_subgoal()*** alle Prozesse der Vorab-Falsifizierung von Zwischenzielen ein, während schaetz_k_konzept() die elementare Netzableitung durchzuführen versucht (siehe Bild 3.22).

```
 BF test_k_subgoal (KG n, ID e, ID goal, ID subgoal)
{
  if (test_netzmat(goal,subgoal))      return(FALSE);
  if (test_user_schaetz(n,e,goal,subgoal))
       return(TRUE);
  else return(FALSE);
}
```

Bild 3.22 Pidgin–C: test_k_subgoal()

Die ERNEST-Kontrollfunktion *test_netzmat(ID k1, ID k2)* sieht im Netz nach, ob überhaupt eine Verbindung der Konzepte k1, k2 vorgesehen ist. Ihre Anwendung soll Zwischenziele vorab-falsifizieren, von denen überhaupt kein Netzpfad zum Subziel führt. Die dynamischen Aspekte prüft die bereits im vorigen Abschnitt “Dynamische Kontrolle der Aufwärtsableitung” beschriebene Funktion *test_user_schaetz()*.

Die in Bild 3.23 codierte Funktion get_user_subgoallist() realisiert die Lookup-Variante zur tabellarischen Auswahl der Zwischenziele.

```
IDL get_user_subgoallist(ID n, ID goal)
{ ID  e,k;
  IDL subgoallist;
  init(subgoallist);
  e= get_k_schaetz_eintrag(n);
  switch(get_gradkon(konzept(e)))
   { case HYPOTHESE:
     if ((k = wortart_konstituente(e,goal)) != NIL)
       append(subgoallist,k);
   } return(subgoallist);
}
```

Bild 3.23 Pidgin-C: get_user_subgoallist()

Mit get_k_schaetz_eintrag() wird zuerst der aktuelle schaetz_eintrag des Konzeptgraphen bestimmt. Der switch behandelt dann die verschiedenen Hierarchie-Zugehörigkeiten des schaetz_eintrag. Zur Zeit ist nur der case HYPOTHESE vorgesehen. Die eigentliche Lookup-Funktion ist hier der Makro *wortart_konstituente(ID e, ID goal)*, der mit Hilfe der ID der in I(H_WORTHYP,[...]) (dem schaetz_eintrag) instantiierten Wortart in die Subtabelle zum Konstituententyp goal einspringt. Vergleicht man die Zuordnungen

A. in get_user_konstitlist():
   wortart(I(H_WORTHYP) —> Konstituenten-Konzept,
B. in wortart_konstituente():
   wortart(I(..), Konstituente —> Wortart-Konzept (als Zwischenziel),

so sollte zwischen beiden Funktionen ein Übereinstimmungsverhältnis gewahrt bleiben, indem jedes Zuordnungspaar (Urbild,Abbild) von A je genau einen Tabelleneinsprung in B definiert. Der Inhalt der Zuordnung B ist als 1–1–Übersetzung der ID wortart auf die ID konzept zu verstehen. Bei noch engerer Bindung von semantischem Netz und Lexikon ist eine vorcompilierte tabellarische Zuordnung entwickelbar.

3.3.3.1 Änderung von Attributberechnungsfunktionen

In einem Unterabschnitt soll festgehalten werden, welche Änderungen von Attributberechnungen vorgenommen wurden, um die inkrementelle Verarbeitung zu realisieren. Diese Funktionen sind dem Bestand der linguistischen Wissensbasis zuzurechnen und Konzepten zugeordnet.

1. ueberd_prim():

Die Funktion berechnet das Attribut 'ueberdeckung' als Voraussetzung von Instantiierungen zum Konzept H_WORTHYP. Um den Zugriff zur assoziierten Wortkette eines Konzeptgraphen einfacher zu gestalten, wurde eine Zwischenspeicherung einer internen ID zur Identifizierung der Elemente der Kette im Instanzenspeicher eingebaut.

2. ueberd_b():

Die Funktion berechnet das Attribut 'ueberdeckung' für Subketten eines Konzeptgraphen. Sie enthielt bisher eine auf die nicht-inkrementelle Kontrollstrategie von Kummert zugeschnittene Beschränkung der Verarbeitung optionaler Bestandteile, die durch eine Flagsteuerung ausgeblendet wurde.

3. hypoth_b():

Die Veränderungen an dieser Funktion waren in mehrfacher Hinsicht gravierend. Sie stellt den Ort dar, an dem das Attribut 'hypothese' bei der Instantiierung eines Eintrags zu H_WORTHYP berechnet wird, d.h in dieser Funktion werden innerhalb von Expandierungen nach unten propagierte Restriktionen ausgewertet:

1. die Restriktion über die Wortnummer (siehe WNR-Restriktionen in Kapitel 2)
2. die Restriktion über die ADJAZENZ zu bereits instantiierten Wortkandidaten bei der Auswahl der Lücken der Signalüberdeckung.

Unter der nicht-inkrementellen Kontrolle kam die Funktion stets nur am Ende von Expandierungsketten bis zur Wortgraphen-Ebene zur Geltung. Von den 3 folgenden Arbeitsschritten eines einzelnen A*-Taktes waren dort nur 1.,3. vorhanden:

1. Aufwärtsableitung,
2. Ereignisbehandlung,
3. Expandierung.

Der Zugriff auf die Kandidaten im Wortgraphen wird aus der Attributfunktion hypoth_b() herausgenommen und in der Funktion **create_skiplist()** (siehe folgenden Abschnitt zur Ereigniskontrolle) zentral zugreifbar. Sie bekommt die Aufgabe zugewiesen, in Schritt 2. des A*-Taktes, der Ereignisbehandlung, die betreffs der Restriktion 2 zulässigen Wortkandidaten zu ermitteln. Gleichzeitig lokalisiert sie die Eingriffsstelle der SKIP-Verarbeitung, welcher ein unmittelbarer Einfluß auf die Restriktion der Signalposition zugestanden werden muß: SKIP oder (im Einzelfall) Vorab-Falsifizieren der Kandidaten eines Intervalls muß zur Verschiebung der Restriktion B führen. Abschließend sei die Weise der Übergabe der von create_skiplist() vorselektierten Wortkandidatenmenge an hypoth_b() erwähnt.

Für jedes Paar (lexem,wortart) wird ein eigener Instantiierungsprozeß, sprich eine Suchbaumaufspaltung, durchgeführt. Die ID wortart ist aber im verwendeten Lexikon in Sub-Wortkategorien unterteilt, z.B. das Lexem [einen] innerhalb der Kategorie Pronomen in 5 Subkategorien zum Gebrauch in verschiedenen Kasusformen.

Die Funktion hypoth_b() gehört allerdings nicht zu jenen Attributberechnungsfunktionen, die auf diese Bestimmung zugreifen. Weil es sinnvoll sein dürfte, die Aufspaltung der datengetriebenen Instantiierung je Wortkategorie/Subkategorie durch die Aufgabenkontrolle einzuschränken, gebe ich dazu Lösungshinweise an. Die Zentralisierung der Kontrollfunktion create_skiplist() bietet die Voraussetzung dafür. In der aktuellen Implementation schreibt die Funktion noch die bis auf Sub-Wortkategorie aufgespaltenen Einzelhypothesen (zum gleichen Lexem, zur gleichen Kante im Wortgraphen) als Records in einen global – also auch für hypoth_b() erreichbaren – Lookup-Bereich im Instanzenspeicher. Bei der inkrementellen Verarbeitung[25] dominiert die Zahl der Aufwärtsableitungen diejenige der Prädiktionen. So wird z.B. bei der SKIP-Verarbeitung der Anfrage

[25] die genannten Funktionen betreffen nur Phase I

[ich, möchte, morgen , früh, nach, Hamburg, fahren]

genau eine Prädiktion benötigt, indem vom Eintrag M(SY_PNG, [Hamburg]) zur kohärent modellierten Modalität von SY_PNG eine Expandierung auf das mit SKIP übersprungene Intervall der Wortgraph-Kante zu [nach] ausgelöst wird.

Nur in solchen Fällen besitzen die im Instanzenpuffer eingetragenen Records ein Zuviel an Information, das durch die Anwendung der WNR-Restriktion in hypoth_b() im Fall ii gegebenfalls verworfen wird.

3.4 Ereigniskontrolle

In Kapitel 2 wurde im Struktogramm der Expandierung auf die im Basis-Schema eingeführte Schnittstelle zur Ereigniskontrolle aufmerksam gemacht. Die Schnittstelle wird wirksam, wenn der aktive Knoten im Test-Block der A*-Schleife an den Operationsblock der Expandierung übergeben wurde.

Um die Zahl der Userfunktionen im Basisschema begrenzt zu halten, habe ich die Schnittstelle zunächst mit nur einer Funktion besetzt, die gleich 2 Aufgaben zu lösen hatte:

1. Analyse, ob eine Reaktion auf den Ereignisstrom erfolgen muß
   —> Merken des aktuellen Returnknoten und Erzeugen eines Nachfolgers mit Neuziel M(H_WORTHYP).
2. Bestimmung des Aktivationspunktes im Konzeptgraph (der *aktive Marker*).

Die Aufgabe 2 ist mit der ersten verzahnt, denn wenn das Neuziel erzeugt wurde, soll es automatisch aktiver Marker werden. Dafür gab es eine sehr einfache Problemlösung. Ich habe die A*-Maschine als eine solche charakterisiert, welche den mit Ende des Maschinentakts erreichten Originalzustand von Sturkturbeschreibungen für gezielte Rückverfolgungen der Suche invariant hält. Die knoten-invariante Lösung, welche die aktuelle Implementierung prägt, sieht man in Bild 3.24 als Variante neu dargestellt.

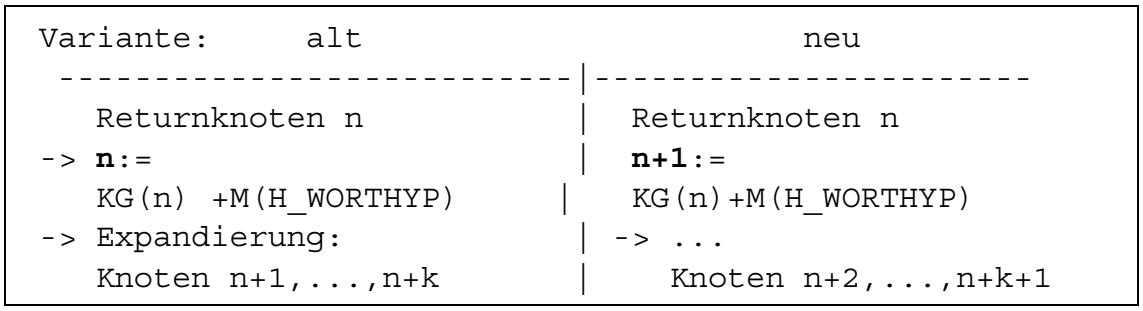


Bild 3.24 Problem des invarianten Returnknoten

Die alte Variante verletzt den Originalzustand des Returnknoten, dessen Bestimmung erst mit Variante 'neu' exakt gegeben ist: Vorgänger eines Knoten mit Neuzieleintrag M(H_...). Der invariante Returnknoten wird benötigt, wenn nach fehlerhaften RESKIP2–Versuchen und Ausstieg aus der Phase PASS2 (siehe Bild 2.39 in Kapitel 2) ein "originaler" Rückkehrpunkt zur Fortsetzung von Analysephase I benötigt wird. Da erweist sich der kurzfristige Vorteil des im gleichen A*-Takt eingehängten und expandierten Neuzieleintrags als grober Fehler. Die Kontrollsituation trat erst beim

Test mit gestörtem Datenmaterial (vergleiche den Wortgraph eines verrauschten Signals im Abschnitt”SKIP-Verarbeitung” in Kapitel 2) auf.

Um die obengenannten Aufgaben 1 und 2 zu entflechten, wurden separate Funktionen eingeführt, die gemeinsam den Test-Block im Struktogramm-Kopf der Expandierung realisieren:

i. *test_user_ereignis(KG n)* gibt TRUE zurück, wenn eine Ereignisreaktion erfolgt ist. Der Operationsblock Expandierung und der A*-Takt werden verlassen. Bei Rückgabe FALSE wird die folgende Funktion aufgerufen.
ii. *get_user_aktion(KG n)* ist eine typische Funktion der Aufgabenkontrolle. Sie bestimmt den Entwicklungsgrad (die Phasen) des aktiven Knoten und in Abhängigkeit dessen den aktiven Marker.

Im alten Basisschema hatte die problem-unabhängige Funktion *get_k_maxprio_eintrag()* die Aufgabe, den aktiven Marker zu bestimmen. Sie liefert den Prior des Konzeptgraphen. Um den Schnittstellencharakter des Einschubs der Funktionskaskade i. -> ii. im Basisschema zu sichern, wird diese auch im inkrementellen Fall wichtige Funktion nicht in ii. aufgerufen, sondern nur im Fall der Rückgabe einer ID NIL. Bild 3.25 zeigt, wie man die Aufruf-Kaskade mit Dummy-Belegungen der Funktionen i./ii. überspringen kann, um in den nicht-inkrementellen Kontroll-Modus einzuspringen.

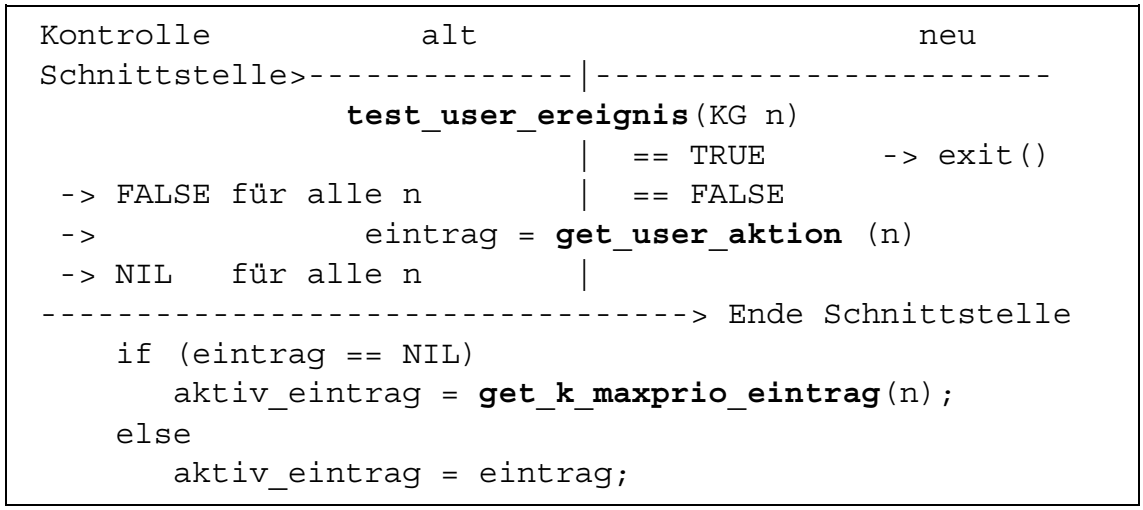


Bild 3.25 Schnittstelle “Ereigniskontrolle” im Basisschema

Das vorgeschlagene Kontrollschema der Expandierung kann in beliebigen ERNEST-Anwendungen angewandt werden.

Ich möchte darauf hinweisen, daß die Aufspaltung in die Funktionen i. und ii. nicht zwingend ist. Es erscheint ebenso möglich, bei Erhaltung des Returnknoten durch Knoten-Klonierung den Neuziel-Knoten im aktuellen Takt zu expandieren. Ausgangs der Schnittstelle hätte man den neuen aktiven Knoten bekannt zu machen und mit dem in get_user_aktion() bestimmten Marker eintrag zu verkapseln. Dazu müßte man in den Code des Basisschemas einen Struktur-Datentyp {KG n; ID eintrag} einführen. Dieser war bisher aus prinzipiellen Gründen nicht erforderlich, da Knoten und Marker generell unabhängig voneinander dynamisch alloziert und verwaltet werden.

Sie verhalten sich übrigens nicht wie Objekte der objekt-orientierten Programmierung. Deshalb erlaube ich mir eine kurze Replik zur Frage der Reimplementation von ERNEST mit OOP-Sprachen wie C++. Die Erhaltung bestimmter Objekt-Charaktere wie die oben angesprochene Knoten-Invarianz erfordert nicht, das Programmierparadigma auf die Kontrolle zu übertragen. Die direkten Konsequenzen für die Aufgabenorientierte Programmierung möchte ich im Rahmen der Arbeit nicht diskutieren. Meine Tendenz habe ich im Anfangskapitel in den PROLOG-Vergleichen ausgesprochen. Zu den Vorteilen der OOP bei der Gestaltung von semantischen Netzen und Framesystemen hat sich Bench-Capon einschränkend geäußert [BC90].

3.4.1 Dynamische Kontrollfunktion test_user_ereignis()

Die Ereigniskontrolle fällt eigentlich ganz in Analysephase I. Durch den Aufruf der Funktion test_end_phase1() wird zunächst in Phase I und Restanalyse verzweigt. Als Restanalysefälle sind weitere Probleme der Erhaltung spezieller Returnknoten zu betrachten. Für den weiteren Ablauf ist noch wichtig, daß in der Testfunktion ein globales RESKIP_FLAG adjustiert wird. Da das Flag den Signalüberdeckungsgrad der assoziierten Kette des aktiven Knoten n abhängt, ist es nur für einen A*-Takt global. Das Flag wird an dieser Stelle nur für PASS2 einegstellt und zwar auf TRUE, solange noch RESKIP2–Aktionen möglich sind. Für RESKIP1 kann das Flag erst in der Funktion get_user_infolist() eingestellt werden, welche zuvörderst die Notwendigkeit zur rechtsassoziativen Expandierung überprüft. Beide Flagsetzungen werden im Interface der SKIP-Kontrolle (Funktion create_skiplist()) ausgewertet.

Zu Phase I: es wird die Hierarchieebene des LN-Markers ermittelt. Liegt sein Konzepttyp in der HYPOTHESE-Ebene, so handelt es sich entweder um den Neuziel-Knoten oder um den letzten Eintrag einer prädiktiven Expandierungskette. In beiden Fällen leitet die Ausgabe des FALSE-Flags zum Aufruf von get_user_aktion() über. Im case SYNTAX der Falltabelle der LN-Konzepttypen wird der Ableitungsgrad des LN betrachtet. get_k_LN() liefert nämlich einen instantiierten Eintrag zurück, wenn im RM-Zweig kein Nonterminal vorkommt. Das ist der Fall für instantiierte Konstituenten, deren weitere Aufwärtsableitung verzögert wurde. In diesem Fall wird die Funktion insert_act_init_knoten() angesprungen, die den Neuzieleintrag produziert.

Ist der LN nicht instantiiert, so ist zu prüfen, ob sein Konzepttyp KOHAERENT ist. In solchen Fällen möchte man die modell-gestützte Vorhersage ausnutzen und über eine Expandierungskette die WNR-Restriktion nach-unten propagieren. Deshalb gibt es hier keine Ereignisreaktion.

Wurde mit Test_end_phase1() das RESKIP_FLAG für den PASS2 auf TRUE gesetzt, so wird die Liste der Konstituentenmarker aufgebaut. Sind deren Elemente durchweg instantiiert, so wird der Neuziel-Knoten erzeugt. Der RESKIP2 soll dann durch Aufwärtsableitung klären, ob gewisse Kandidaten der Skipliste als optionale Bestandteil zur Erweiterung solcher Instanzen verwendet werden können. Andernfalls wird der Absprung zu get_user_aktion() mit dem Ziel vorbereitet, die gefundene unvollständige Konstituente in einem RESKIP2 modell-gestützt expandieren zu lassen.

In allen anderen Fällen wird der Neuziel-Knoten erzeugt und das TRUE-Flag gesetzt, um den A*-Takt zu verlassen. Dazu zählen in Phase I abgeleitete pragmatischen Wurzeln, die sich im RM-Baum als LN-Marker präsentieren..

```
  ID test_user_ereignis(KG n)
{
 ID  eln, ei, eintrag;
 IDL topsynlist;

if (test_end_phase1(n)==E)        /*-> RESKIP_FLAG */
 {
    eln = get_k_LN(n);
  switch(get_gradkon(konzept(eln))
   {case HYPOTHESE:                    return(FALSE);
    case SYNTAX:
     if (ableitungsgrad(eln)!=INSTANZ)
      {
       if (get_k_coherence_eintrag(eln)==TRUE)
                                        return(FALSE);
       if (RESKIP_FLAG==TRUE)
        {
         topsynlist = get_k_grad_list(n,SYNTAX);
         foreach(ei in topsynlist)
          if (ableitungsgrad(ei)!=INSTANZ)
                                        return(FALSE);
    }}}
   insert_act_init_knoten(H_WORTHYP,n);
   return(TRUE);

   }
else

  {
    if(test_k_copy_wurzel(n))           return(TRUE);
    else                                return(FALSE);
 }}
```

Bild 3.26 Pidgin-C: get_user_aktion()

Die Funktion test_k_copy_wurzel() testet in Phase II befindliche Konzeptgraphen auf die Existenz von Wurzelmarkern identischen Konzepttyps. Obwohl rein netztechnisch codiert, steht sie der Aufgabenkontrolle sehr nah. Denn sie geht davon aus, daß solche Fälle in der aktuellen Domäne nicht vorkommen dürfen. Sie können aber durch fehlerhafte Elemente im Wortgraph enstehen. Durch folgende Maßnahmen wird versucht die Interpretation zu retten: es wird ein Klone erzeugt und darin einer der beiden Wurzelmarker mit der Funktion reset_ke_eintrag() auf UNZULAESSIG gesetzt. Das gleiche findet mit dem anderen Marker im Originalknoten statt. Beide werden mit einer Testfunktion der Aufgabenkontrolle bewertet, die vom strengen Übertragungsprinzip (von Markerbewertungen auf die Gesamtbewertung des Knoten) abweicht. Bevor beide Knoten in die OFFEN-Liste kommen, wird einer der Knoten in

die erweiterte Liste der Returnknoten eingetragen. Dieser Knoten wird re-aktiviert, falls sich der zuerst expandierte Knoten durch weitere Ableitung als UNZULAESSIG erweist.

```
  BF  test_k_copy_wurzel(KG n)
{ ID   eintrag, nc, wurzel1, wurzel2, anzahl;
  IDL  toppraglist;
  ...
  toppraglist = get_k_grad_list(node,PRAGMATIK,NOINST);
  if(test_empty(toppraglist)==TRUE)   return(FALSE);
  else
    {
     foreach ( wurzel1 in  toppraglist)
     anzahl=0;
     if (int_mass3(wurzel1)) != UNZULAESSIG)
      {
       foreach(wurzel2 in  toppraglist)
        if (konzept(wurzel1))==konzept(wurzel2))
          {
           anzahl++;
           if (anzahl>1)
            {
             int_mass4(n)= copy_id;
             nc=insert_act_copy_knoten(n);
             act_init_knoten=n;
             append(act_init_knotenlist,n);
             eintrag = search_k_eintrag(n,wurzel1,nc);
             reset_ke_eintrag(eintrag);
             reset_ke_eintrag(nachfolger(eintrag));
             return(TRUE);
    }}}}
  return(FALSE);
}
```

Bild 3.27 Pidgin-C: test_k_copy_wurzel()

Die Funktion insert_act_init_knoten() erzeugt den Neuziel-Eintrag in einem Klone des aktiven Knoten mittels der Funktion insert_k_prim_eintrag. Nur für das initiale Neuziel wird er auch ziel_eintrag. insert_k_knoten() hängt den neuen Knoten im Suchbaum als Nachfolger des aktiven Knoten ein. Dieser wird in die Liste der Returnknoten eingetragen.

```
  ID  insert_act_init_knoten(ID konzept, KG n)
{ ID e, info, nc;
  (global) IDL act_init_knotenlist;
  append(act_init_knotenlist,n);
  act_init_knoten=n;
  nc = kopie(n);
  insert_k_knoten(nc,n);
  e= neu_eintrag();

  if (get_k_ziel_eintrag(n)) == NIL)
    insert_k_ziel_eintrag(n,e);
  else
    insert_k_prim_eintrag(n,e);
  /* Füllen von Eintrag-Slots von e */
  ...
  create_modconcept(n,e);
  /* e auf Arbeitsspeicher adjustieren*/
  ...
  ableitungsgrad(e) = MODKONZ;
  return(e);
}
```

Bild 3.28 Pidgin-C: insert_act_init_knoten()

3.4.2 Dynamische Kontrollfunktion get_user_aktion()

Die Funktion get_user_aktion() bestimmt den aktiven Marker, d.h. den Startpunkt der Expandierung. Die Fallunterscheidung ist ein Spiegelbild derjenigen in der Funktion test_user_ereignis(). Die Entflechtung beider Funktion erfordert es, für den RESKIP2–Fall die gleiche Untersuchung der Konstituentenmarker vorzunehmen. Das hat den Zweck, im laufenden PASS2 die offenen Marker zu expandieren, um Elemente der Skipliste zu integrieren.

Der Neuziel-Marker aus der Aktion von test_user_ereignis() wird als LN gefunden und zum aktiven Marker bestimmt. Gleiches gilt für KOHAERENT modellierte Konzepttypen von LN-Markern der SYNTAX-Schicht.

Befindet sich der aktive Knoten noch in Phase II und besitzt er durchgängig instantiierte Wurzeln, so macht es sich erforderlich, den Kontextmarker selbst zum aktiven Marker zu machen, um nachträglich Bindungen einzutragen.

Die in get_user_aktion() angewandte Funktion test_end_wurzel() stellt für Knoten, die Phase II verlassen haben fest, ob die Liste der noch nicht instantiierten pragmatischen Wurzeln leer ist. UNZULAESSIG gesetzte Marker bleiben außer Betracht. Die Ausgabe TRUE durch die Funktion ist ein Signal dafür, daß die Strukturbeschreibung nicht weiterentwickelt werden kann, falls noch obligatorische Wurzeln für einen Kontextmarker fehlen. Das wird als Abbruch-Kriterium in der Funktion test_user_analyseziel() ausgewertet. In solchen Fällen muß man zur Dialogsteuerung überleiten. Wichtig ist es aber, solche Knoten als potentiell besten Analysezustand aufzuheben.

```
  ID get_user_aktion(KG n)
{
 ID  eln;
 IDL kontextlist;

 if (test_end_phase1(n)==FALSE)
   {
     eln = get_k_LN(n);
   switch(get_gradkon(konzept(eln))
    {
     case HYPOTHESE:
->     create_skiplist(n,eln);
       return(eln);
     case SYNTAX:
      if (ableitungsgrad(eln) !=INSTANZ)
        {
          if ( get_k_coherence_eintrag(eln)==TRUE)
            return(eln);
          if (RESKIP_FLAG==TRUE)
           {
            topsynlist = get_k_grad_list(n,SYNTAX);
            foreach(ei in topsynlist)
            if (ableitungsgrad(ei)!=INSTANZ)
             return(FALSE);

    }}}
 else (test_end_phase2(n)== FALSE
      && test_end_wurzel_phase2(n)== TRUE)
    {
      kontextlist=get_k_grad_list(n,PRAGMATIK,KONTEXT);
      return(first_element(kontextlist));
  }}
 return(NIL);
```

Bild 3.29 Pigin-C: get_user_aktion()

Die Funktion test_end_wurzel() zeigt einen einfachen Weg an, das Problem des *besten Parsings* mit relativ einfachen Kriterien des Tiefenstrukturparsings zu lösen. Ich erinnere daran, daß in GLR* versucht wird, das Problem über die maximale Länge zulässiger Subparses zu lösen.

```
 ID test_end_wurzel_phase2(KG n)
{
 ID eintrag;
 IDL toppraglist;

 if(test_end_phase1(n)== FALSE)       return(FALSE);
 toppraglist=get_k_grad_list(n,PRAGMATIK,NOINST);
 if (test_empty(toppraglist)==FALSE)  return(FALSE);
 foreach( eintrag in  toppraglist }
   if (ableitungsgrad(ei) != INSTANZ) return(FALSE);

 return(TRUE);
}
```

Bild 3.30 Pidgin–C: test_end_wurzel()

3.5 Expandierung

Zur Anpassung der ERNEST-Basiskontrolle an die inkrementelle Verarbeitung wurde der Funktion get_user_infolist() bereits ein fester Platz in der Implementierung eingeräumt. Die Anwendung der Funktion muß über das globale Flag SPACE_TIME_CONT in der Analyse-Initalisierung angefordert werden. Ihre Parameter sind durch den Aufruf in einer Basisroutine der Expandierung festgeschrieben. Die Zweckbestimmung ist eine Vorabfalsifizierung von Rollen als Wege der Expandierung anhand der Netzflußinformation. Die Codierung greift auf ERNEST-Interna zurück, d.h es handelt sich eigentlich um ERNEST-Erweiterungen für nicht in jeder Anwendung erforderliche Kontrollmodi wie Inkrementalität oder rechts-assoziative Expandierung.

3.5.1 Dynamische Kontrollfunktion get_user_infolist()

Die Funktion get_user_infolist() stellt eine Art Manager zur Auswertung der Netzflußinformation dar. Letztere wird durch die Funktion get_k_pr_infolist() entsprechend der Vorgabe von Suchrichtungen und Beschränkungen über die Rollenmenge der Modalität (ALLE oder OBL=nur obligatorische Rollen) geliefert.

Das Vorkommen des Parameters pr (Typ Sub-PRAEMISSE) im Funktionskopf verweist auf den Charakter als systemnahe Steuerungsfunktion. Er ist an die Verwendung der Funktion in der ERNEST-Basisroutine expand_k_knoten() angepaßt. Den Codeblock der Basisroutine, der den Aufruf von get_user_infolist() vorbereitet, habe ich in Funktionen wie test_konz_kontext() importiert (siehe Bild 3.7). Damit soll ein analoger Zugriff der Aufgabenkontrolle auf die Netzfluß ermöglicht werden.

Die Netzflußinformation bündelt Paare (Modalität, Kontext) als Sub-Prämissen zum Konzepttyp. Sollen Strukturmarker expandiert werden, so kennt man die bestimmte Modalität des Markers. Damit kann man in die Sub-Praemisse einspringen und die klassifizierten Rollen (Typ INFO) und Rollenträger der Modalität bestimmen. Will man nachsehen, ob z.B. für einen Expandierungsschritt ein relevanter Ziel-Marker vorliegt, so kann man aus der Info die relevanten Konzepttypen ablesen. Infos sind Konzeptbündel. Infolisten haben den Typ:

$\{K_1, ..., \{K_i, ... , K_{i+l}\} ,...,K_n\}$ mit Konzepten K_j, j=1,...,n.

Die Aufgabe des Auslesens der Konzepte für eine Sub-Prämisse pr ist Sache der Funktion get_k_pr_infolist(). Ihre Anwendung selektiert man mittels der Parameter vol für die Kanten-Klassifikation und dir für die richtungsabhängige Auswahl der (z.B. rechts-assoziativen) Kante, die inkrementell für die Expandierung zu wählen ist. Letzteres löst die Funktion test_dir_info(). Insgesamt geht es um nichts anderes als einen verfeinerten Mechanismus der Regelauswahl.

```
 IDL get_user_infolist(KG n, ID e, ID pr)
{
 IDL infolist;
 (global) WHL skiplist;

 if(test_end_phase1(n)==FALSE)
   {
    infolist = get_k_pr_infolist(n,e,pr,OBL,NACH);
    if (test_empty(infolist)==TRUE)
      {
       if (test_skipped(n)== TRUE
           && test_empty(skiplist)==FALSE)
        {
         infolist=get_k_pr_infolist(n,e,pr,OBL,VOR);
         RESKIP_FLAG = TRUE;
    }}}
 else    infolist=get_k_pr_infolist(n,e,pr,ALLE);
 return(infolist);
}
```

Bild 3.31 Pidgin-C: get_user_infolist()

Die zentrale Funktion von get_user_infolist() besteht nun darin, zu einer n Sub-Prämisse die Liste der aktuell zu expandierenden Infos zu selektieren. In einem A*-Takt können mehrere Expandierungen in einem Konzeptgraphen realisiert werden.

Der einfachste Fall liegt vor, wenn die Teilphase der Wurzelbindung an den Kontext in Phase II erreicht wurde. Dazu genügt der Test des Endes von Phase I, da ansonsten nur Aufwärtsableitungen stattfinden. In dem Moment bekommt man mit get_k_pr_infolist alle auf die vorhandenen Strukturmarker abbildbaren Konzepttypen. Das gleiche gilt für Phase III.

In Phase I wird zunächst stets die Möglichkeit der rechts-assoziativen Expandierung obligatorischer Kanten geprüft. Ist die Info-Menge leer, wird angenommen, daß es sich um einen SKIP-Fall handelt. Die Situation wurde in Bild 2.37 skizziert. Wenn in der Domäne ICZ ein offener PNG-Marker zu einem Eigennamen vorliegt, so läßt die während der Aufwärtsableitung attributiv eingeschränkte Modalität keine rechts-assoziative Expandierung zu. Also wird die Infomenge der Links-Expandierung besorgt. Ist sie nicht leer, wird mit dem RESKIP_FLAG der Zugriff auf die Skipliste angemeldet.

```
  IDL  get_k_pr_infolist
               (KG n, ID e, ID pr, ID vol, ID dir)
{ ID  konzept,grad,mod,netzfluss,modalitaet,info, rk;
  IDL alt_rklist;
  init(infolist);init(grad_infolist);
  grad = get_gradkon(e);
  if (vol==KONTEXT)
       k_grad_list = get_k_grad_list(n, grad, NOKONTEXT);
  else k_grad_list = get_k_grad_list(n, grad, INFO);

  if (test_empty(k_grad_list)==FALSE)
     foreach( ei in k_grad_list)
       append(grad_infolist,get_ke_info(ei));
  konzept=konzept(e);
  alt_rklist=alt_rklist(e);
  if (test_adjazenz(modalitaet(e))==FALSE)
    { if (vol!=KONTEXT)
        {foreach( info in obl_part_infolist(pr))
           { rk=get_finfo_kennung(info);
             if (test_ke_altkenn_rk(e,altkenn_list,rk)
                ==FALSE)
             append(infolist,info); }}
      foreach( info in obl_rest_infolist(pr))
        { if (vol==KONTEXT)
            if (info not-in k_grad_list)
              append(infolist,info);
          if (test_ke_altkenn_rk(e,altkenn_list,rk)
                ==FALSE)
            append(infolist,info); }
      if (vol==ALLE)
        { foreach( info in opt_part_infolist(pr))
           {
            rk=get_finfo_kennung(info);
            if (test_ke_altkenn_rk(e,altkenn_list,rk)
                ==FALSE)
            append(infolist,info); }}}
  else  { /* (s.Bild 3.33)*/ }
  delete(k_grad_list);delete(grad_infolist);
  return(infolist);
}
```

Bild 3.32 Pidgin-C: get_k_pr_infolist()

```
  if (test_adjazenz(modalitaet(e))==TRUE)
    {
     foreach ( info in obl_part_infolist(pr))
       {
        rk=get_finfo_kennung(info);
        if (test_dir_info(e,alt_rklist,pr,info,dir)
                ==TRUE)
            append(infolist,info);
        }}
     if (vol == ALLE)
      {
     foreach ( info in opt_part_infolist(pr))
       {
        rk=get_finfo_kennung(info);
        if (test_dir_info(e,alt_rklist,pr,info,dir)
                ==TRUE)
            append(infolist,info);
    }}}
```

Bild 3.33 Codeblock get_k_pr_infolist(): Inkremente

```
 BF test_dir_info
           (ID e, IDL alt_rklist, ID pr, ID info, ID dir)
{ ID adja, rk;

  rk = get_finfo_kennung(info);
  adja = get_m_adja(pr_modalitaet(pr));
  if (adja > NIL)
    {
     rk=get_finfo_kennung(info);

     if ( test_ke_adja_kante(adja,alt_rklist,rk,dir)
               == FALSE
          || rk_anzahl(e,rk)
              >= get_finfo_numb_min(info) )
       return(FALSE);
    }  return(TRUE);
}
```

Bild 3.34 Pidgin-C: test_dir_info()

Die eingeführte Testfunktion test_ke_adja_kante() testet richtungsabhängig die Rolle rk bezüglich der ADJAZENZ des Eintrags e sowie der in der Liste der in e bereits gebundenen Rollen, alt_rklist. Die Richtung 'Vor' faßt die Rollen in alt_rklist als

Vorausgehende auf. Die Richtung 'Nach' tut das Gegenteil zur Behandlung der RESKIP-Fälle.

3.6 SKIP-Verarbeitung

3.6.1 Wortgrapheninterface create_skiplist()

3.6.1.1 Codierung der Funktion create_skiplist() Der in Bild 3.35 dargestellte Block der Funktion create_skiplist() zeigt den TEST-Block der SKIP-Schleife.

Der **GRAPH-Zeiger** *kohaerent_links* wird auf einen Wortgraphknoten als Linksknoten für die Suche im Wortgraphen adjustiert. Er wird durch test_hypgen(), die eigentliche Zugriffsfunktion für den Wortgraphen, ausgewertet. Zunächst wird die assoziierte Wortkette kette gebildet. Ist sie leer, wird der Zeiger auf den global bekannten ersten Knoten im Wortgraph adjustiert.

Ist die Kette belegt und ist keine RESKIP-Aktion durchzuführen, so ist als nächstes das RETURN_FLAG auszuwerten. Es wird im Test-Block des A*-Taktes gesetzt, wenn die Analyse zum aktuellen Returnknoten zurückgefahren werden muß. Befindet sich das Flag im gesetzten Zustand, sprich TRUE, so wird durch die Funktion test_skiplist() der GRAPH-Zeiger auf den Rechtsknoten des Endelements der assoziierten Wortkette des Returnknoten adjustiert. Ist das nicht der Fall, so wird er auf das Endelement der Kette, das mit get_hyp_endkette() bestimmt wird, adjustiert.

Ist das RESKIP_FLAG gesetzt, so bestimmt die Funktion get_sel_zeitvorhersage() eine Liste der Lückenintervalle der Signalüberdeckung des aktiven Knoten. Diese Liste wird im nächsten Codeblock durchlaufen. Wird innerhalb der Schleife der **SKIP-Zeiger** skipknoten auf einen Knoten im Graphen adjustiert, so läuft die SKIP-Schleife erneut an.

In der SKIP-Schleife werden bei nicht-gesetztem RESKIP_FLAG die Kandidaten des Wortgraphen hyplist, andernfalls die Kandidaten in der Skiplist durchlaufen (siehe Markierung –>). test_hypgen() vergleicht den Links/Rechts-Knoten des Kandidaten mit den Knoten des Lückenintervalls und den GRAPH-Zeiger. Die intervall-gerechten und mit dem Zeiger konizidierenden Kandidaten werden durch test_user_hypothese() bezüglich der Startwortart, Ambiguität und Anzahl der Subkategorien untersucht und klassifiziert. Gemäß der Klassifikation wird der Kandidat einer der 3 (jeweils lokalen) Ergänzungslisten zur Skiplist, Returnlist und zur Liste der Kandidaten der Instantiierung zugewiesen.

Im Fall der Durchlaufung der Skipliste wird zunächst das RETURN_FLAG abgefragt. Ist es gesetzt, so wird mit dem von test_skiplist() gelieferten Zeiger überprüft, ob das vorgegebene Lückenintervall schon bearbeitet wurde. Dabei wird auf den aktuellen Returnknoten zurückgegriffen. Nachdem mit get_ass_whyplist() die Links/Rechtsknoten der assoziieten Kette ermittelt wurden, überprüft test_ass_skiphyp(), ob sich ein Element der Skipliste mit Gliedern der assoziierten Kette intervallmäßig überlappt.

```
  create_skiplist(KG n, ID e)
{ WH wohyp, wh;
  WHL lskiplist, lreturnlist, lhyplist, ass_whyplist;
  RWHL rechyplist;
  IDL kette, b_list, synentrylist;
  ID links, rechts, ketend,minknoten,skipknoten=NIL;
  (global) ID firstknoten, kohaerenz_links;
  (global) BF RET_FLAG, RESKIP_FLAG;
   kette = get_k_kette(n);
   init(hyplist); init(berinstlist);
 do
  {init(lreturnlist); init(lskiplist);
   init(lhyplist);
   if (test_empty(kette)==TRUE)
     { if (skipknoten == NIL)
         kohaerenz_links = firstknoten;
       else
         kohaerenz_links = skipknoten;
     }
   else if (RESKIP_FLAG==FALSE)
         {
           if (RETURN_FLAG == TRUE)
            { kohaerenz_links = test_skiplist(n);
              RETURN_FLAG = FALSE; }
           else
             { wohyp=get_hyp_endkette(kette);
               kohaerenz_links=rechtsknoten(wohyp); }
           kohaerenz_rechts = NIL;
         }
        else
          {b_list = get_berinstlist(Kette);
           get_sel_zeitvorhersage(n,e,b_list,skipknoten);
         }}
   foreach (i in intervall_list) {..} /*Bild3.36*/
   }
 while(skipknot != NIL);
 delete(lhyplist);
 delete(b_list);
 RESKIP_FLAG = FALSE;
}
```

Bild 3.35 Pidgin-C: create_skiplist(), SKIP-Schleife

```
 foreach (i=(links,rechts) in intervall_list)
 {if (RESKIP_FLAG==FALSE)
   {
 -> foreach(wh in whyplist)
     if (test_hypgen(wh,links,rechts))
      {init(synentrylist);
       switch(test_user_hypothese(n,wh,synentrylist))
        {case FALSE:
           append(lreturnlist,wh);
         case SKIP:
           append(lskiplist,wh);
         case TRUE:
           append(lhyplist,wh);
        }
       delete(synentrylist);
      }}
   else
    {
      if(RETURN_FLAG == TRUE)
        {
          if(rechts <= test_skiplist(n)) continue;
          RETURN_FLAG = FALSE;
        }
      init(ass_whyplist);
      get_ass_whyplist(get_k_kette(n),ass_whyplist);
 ->    foreach(wh in skiplist)
       {if (test_ass_skiphyp(wh,ass_whyplist)==FALSE)
         if (test_hypgen(wh, links, rechts))
           append(lhyplist,wh);
        delete(ass_whyplist);
       }
      /* Endebehandlung in Bild 3.37*/ ...
   }
}
```

Bild 3.36 Pidgin-C: create_skiplist(), Hypothesentest

Der Codeblock in Bild 3.37 führt die Ergänzung der globale Skipliste, Returnliste um die in der SKIP-Schleife ermittelten Elemente durch. Falls am Ende der SKIP-Schleife die Liste der zu instantiierenden Wortkandidaten leer geblieben ist, wird der SKIP-Zeiger skipknot mit get_min_endknot() nach dem “Prinzip des minimalen Sprunges” (vergleiche Bild 2.38) adjustiert und ein neuer Durchlauf der Schleife ausgelöst..

```
  if (test_empty(lhyplist)==FALSE)
   { foreach(wh in lhyplist)
      { init(synentrylist);
        foreach(ent in synentrylist)
            append(synentry,ent);
        add_rechyplist(wh,synentrylist);
        delete(synentrylist);
      }}
  else  skipknot=get_min_endknot(lskiplist,lreturnlist);
  if (test_empty(lreturnlist)==FALSE)
    { foreach( wh in lreturnlist)
        append(returnlist,wh);
      delete(lreturnlist);
    }
  if (test_empty(lskiplist)==FALSE)
    { foreach( wh in lskiplist)
        append(skiplist,wh);
      delete(lskiplist); }
```

Bild 3.37 Pidgin-C: create_skiplist(), “minimaler Sprung”

3.6.1.2 Direktzugriff zum Wortgraph: test_hypgen() Die Funktion test_hypgen() bekommt ein Signalintervall vorgegeben, dessen Links/Rechtsknoten durch das Paar (links, rechts) gegeben wird. Es werden die Wortkandidaten ausgefiltert, die nicht in diesem Intervall plaziert sind und nicht die Anschlußbedingung an die GRAPH-Zeiger kohaerenz_links/_rechts erfüllen. Sind diese Zeiger gesetzt, so muß der Wortkandidat, um nicht falsifiziert zu werden, links- oder rechtsbündig in das Intervall hineinpassen. Die aktuelle Codierung der Funktion unterstellt die Voraussetzung, daß die Knoten zeit-linear numeriert sind (nicht notwendig in fortlaufender Folge).

```
 BF test_hypgen(wohyp, links, rechst)
{
 if (lexem(wohyp) == NIL)            return(FALSE);
 if (kohaerenz_links != NIL)
   if(linksknoten(wohyp)!=kohaerenz_links)
                                     return(FALSE);
 else
    if (linksknoten(wohyp) < links) return(FALSE);
 if(RESKIP_FLAG ==FALSE)             return(TRUE);
 /*Rechts-Kohaerenz* s. Bild3.39 */ ...
}
```

Bild 3.38 Pidgin-C: test_hypgen(), Linksknoten-Zugriff im Wortgraph

Der Codeblock zum Rechtsknotenzugriff stellt eine Abschwächung der strengen Forderung der KOHAERENZ für RESKIP–Fälle dar. Indem nur gefordert wird,

daß der Rechtsknoten, d.h. das Ende des Signalintervalls eines Wortkandidaten, vor dem Ende des Lückenintervalls (links, rechts) liegt, soll Robustheit gegen Insertionen des Worterkenners erreicht werden. Das ist besonders wichtig im Fall des RESKIP von Präpositionen, die aufgrund ihrer Mehrdeutigkeit in der nachfolgend beschriebenen Userfunktion test_user_hypothese() zunächst an die Skipliste übergeben werden. Infolgedessen geraten sie genau in den Zugriff der folgenden Regel:

```
{ /*User-Regel abgeschwächte Rechts-Kohaerenz*/
 if(rechtsknoten(wohyp) > rechts)    return(FALSE);

 /* nicht-ausgefilterte Fälle: */
 return(TRUE);
}
```

Bild 3.39 Pidgin-C: test_hypgen(), abgeschwächte Rechts-Kohaerenz

Die scharfe Forderung der KOHAERENZ hätte die Übereinstimmung des Rechsknoten des Wortkandidaten mit dem gesetzten kohaerent_rechts-Zeiger (Ende des Lückenintervalls) bedeutet.

Da für kohärente Konzepte in der Wissensbasis Attributfunktionen existieren, die diesbezügliche Parameter berechnen, muß die Regel durch eine Anpassung von Rechtsknoten und Rechtsframe des Wortkandidaten abgestützt werden. Die Eingriffsstelle dafür bietet sich im Block der Durchmusterung der Skipliste bei der Suche nach Lücken-Kandidaten in der Funktion create_skiplist(). Die Parameter des Wortkandidaten im Wortgraphen bleiben unberührt.

3.6.2 Lookup–Kontrollfunktion test_user_hypothese()

```
BF test_user_hypothese(KG n, WH wohyp)
{ if (test_hyp_endpath(wohyp) == TRUE) return(FALSE);
  if (get_k_kette(n) == NIL)
        return(test_hyp_wortart(n, wohyp,START));
  else  return(test_hyp_wortart(n, wohyp,AMBIG));
}
```

Bild 3.40 Pidgin-C: test_user_hypothese()

Die Funktion test_user_hypothese() schließt zunächst Wortkandidaten aus, die keine Nachfolger haben. Ansonsten klassifiziert sie mittels der Tabellenfunktion test_hyp_wortart() gemäß der Ambiguität der Kategorie. In test_hyp_wortart() werden momentan die zulässigen Startwortarten sowie als ambig gewertete SKIP-Wortarten zum Teil “von Hand” codiert. Generell als AMBIG werden Wortkandidaten betrachtet, die im Lexikon verschiedene Kategorien oder mehrere Subkategorien besitzen (siehe Block nach der foreach-Schleife in Bild 3.42). Die kategoriell eindeutigen Kandidaten können mit Blick auf die Domäne ebenfalls AMBIG sein. Man denke an die Raum-Zeit-Ambiguität von Präpositionen wie [nach]. Die mit (->) gekennzeichneten Wortarten

sind eigentlich gerade in der aktuellen Domäne nicht ambig, sondern verweisen auf Zeitkonstituenten. Sind zum Beispiel im Wortgraphen zugleich die Subketten

[drei, .., Uhr] als auch [dreizehn, Uhr] bildbar, so würde nach einer Aufwärtsableitung anhand von [drei] das Zwischenintervall bis [Uhr] übersprungen und die eigentlich aufgrund der KOHAERENZ des Konzepts Z_UHR geforderte zweite Alternative "übersehen". Die SKIP–Behandlung für Zahlworte drückt somit das Abwarten bis zum Auftreten des Kernlexems für Z_UHR aus.

```
 BF test_hyp_wortart(KG n, WH wohyp, ID flag)
{ ID wnr, wanr, ent;
  IDL synentrylist;
  wnr = lexem(wohyp);
  init(synentrylist);
  foreach( ent in synentrylist)
    { wanr=ent_wortart(ent);
      switch(wanr)  /* SKIP-Wortarten*/
       { case (   DETERMINANS
                || ADJEKTIV    || ADJEKTIV_UNFLEKTIERT
 (->)           || ZAHLWORT    || ORDNUNGSZAHL
                || PRAEPOSITION):
            return(SKIP);
       }
      if (flag == START)
       { switch(wanr)  /* Startwortarten*/
          { case (VERB || PRONOMEN || FRAGE_ADVERB):
              return(TRUE);
            default:
              return(FALSE);
         }}
      if(flag == AMBIG) ... /* siehe Bild 3.42
   } /* end foreach*/


}
```

Bild 3.41 Pidgin-C: test_hyp_wortart(), SKIP-Wortarten

Mit (l/u) in Bild 3.42 wird eine Prüfung markiert, die bereits beim Einlesen des Wortgraphen in read_whyp() vorbereitet wurde. Lexeme, die in Paaren wie [frueh], [Frueh] auftreten, werden durch Setzen des bisher ungenutzten wortart-Parameter innerhalb der Laufzeit-Liste des Wortgraphen vorab als AMBIG (auf NIL) markiert. Weitere, im einzelnen nicht ausgeführte Prüfungen für SKIP- und RETURN(listen)-Fälle sind im Block (flag==AMBIG) weiter zu erproben. Dazu zählen situationsabhängige User-Regeln des Typs:

"Ist bereist ein NOMEN gebunden, so setze ein weiteres Vorkommen zunächst in die Skipliste. Beachte wichtige Ausnahmen wie [Uhr] (siehe oben)". Bei der Verarbeitung stark verrauschter Testdaten erwiesen sich Fehlerkennungen von Nomen als

unangenehme Störfaktoren, da sie relativ leicht zur Instantiierung von Nominalgruppen führen, die schwer zu falsifizieren sind und andere relevante Wortkandidaten, die durch test_hyp_wortart() zunächst als AMBIG ausgewiesen werden, verdecken.

```
    { /* Beginn foreach */
     if(flag == AMBIG)
        {
 (l/u)    if(wortart(wohyp) == NIL) return(SKIP);
          ...
        }

      if (flag == START || flag == AMBIG)

        append(synentrylist,ent);
    }/* ende foreach*/

  if (flag==AMBIG || flag==START)
    if (anzahl(synentrylist) >1) /*wnr ist ambig !*/
      { delete(synentrylist);
        return(SKIP);  }
  else  return(TRUE);
}
```

Bild 3.42 Pidgin-C: test_hyp_wortart(), Ambiguitätsregeln

3.7 Zusammenfassung

Insbesondere die Anmerkungen zu den letztgenannten Funktionen sollten zeigen, daß bereits Erfahrungen in der Anwendung der vorliegenden Implementation auf Daten gesammelt wurden, die besondere Anforderungen an die Robustheit stellen. Die Implementation muß nichtsdestoweniger als Modell-Lösung oder Prototyp für konzeptbasierte Links-Rechts- und SKIP-Verarbeitung betrachtet werden, für deren Tauglichkeit ich bisher 3 Kriterien in Anspruch nehmen will:

1. die Funktionsfähigkeit für Wortgraphen, die domänen-relevante “wohlgeformte” Sätze mit wenigen Alternativkandidaten zur Äußerung darstellen, kann als erwiesen gelten.
2. Anhand diverser Einzelfälle von Wortgraphen zu stark verrauschten Daten konnte nachgewiesen werden, daß sich die SKIP-Methodik so bewährt, daß der Graph über die Analysephase I hinaus entwickelt werden kann.
3. Im Ansatz wurde eine Funktion test_user_analyseziel() aufgebaut, um wenigstens die elementarsten Fälle zu erfassen, die sich nicht über ein gewisses Zwischenstadium der Phase II hinausentwickeln lassen. Potentiell soll die im Test-Block des A*-Taktes verortete Funktion ein Gegenstück zum sogenannten dynamischen Abbruch [Fin93b] darstellen. Die inhaltliche Gegenüberstellung muß weiterentwickelt werden. Der Akzent liegt auf einer noch stärkeren Ausnutzung der Kriterien des Tiefenstrukturparsings.

4. Obwohl noch keine umfangreiche datenmäßige Evaluierung der Implementation vorliegt, geht sie jedoch weit über ein bloßes Testbett der inkrementellen Verarbeitung hinaus und kann bereits als direkte Voraussetzung einer synchronen Kopplung an einen inkrementellen Erkenner betrachtet werden.

Es steht außer Frage, daß der Prototyp in vielen Hinsichten verbessert und erweitert werden kann. Gewisse Fehler der Worterkennung wie Fehlpositionierungen und Verlust einer brauchbaren Basis für den Verbrahmen lassen die Analyse in Phase I stagnieren. Wenn das Verb falsch oder fehlpositioniert ist, sind Hilfskonstruktionen in Erwägung zu ziehen. Manche Interpretationen sind z.B. mit dem “Hilfsverbs” [moechte] rekonstruierbar und somit zu retten.

Der eingeschlagene Weg einer auf die A*-Maschine abgestimmten Rückverfolgung von Fehlerquellen hat sich bereits bewährt, ist aber noch ausbaufähig. Bis jetzt wird nur der jeweils letzte Returnknoten der datengetriebenen Instantiierung betrachtet. Beim Verfolgen dieses Ansatzes wird mein Bestreben sichtbar, dem “Verschwinden” der korrekten kategoriell/subkategoriellen Variante auf der Interpretationsebene der Wortarten vorzubeugen. Es ist für mich weiterhin eine offene Frage, ob die Analyse ohne Einsatz des dynamischen Abbruches immer konvergent gehalten werden kann. Diese Form der Abbruchkontrolle ist auf die Erkennung von Stagnation der Gesamtbewertung der Knoten (z.B. Verbesserung des Überdeckungsgrades) gerichtet.

Die Returnkontrolle schließt zur Zeit noch keine Wiederverwendung korrekter Teilbäume der Ableitung ein. Beim bloßen Wegarbeiten eines die linguistisch bessere Lösung verdeckenden Stapels in der OFFEN-Liste müssen einzelne Bäume, die von vorausgehenden Fehlableitungen nicht betroffen sein müssen, komplett neu erarbeitet werden. Durch SKIP kann man dem zuvorkommen. Der Weg des Ersetzens fehlerhafter Teilbäume sollte ebenfalls verfolgt werden.

Die SKIP-Verarbeitung erlaubt zwar in manchen Fällen die Analyse von Wortgraphen mit erheblichen Defekten und Mehrdeutigkeiten. Ob sich das Verarbeitungsmuster der singulären Einzelregeln der SKIP–Verarbeitung bei der synchronen Kopplung bewährt, ist abzuwarten. Der gekoppelte Erkenner möchte aber schnelle, lokale Falsifikationen als Feedback erhalten. Auch muß der Gefahr begegnet werden, über problematische Bereiche hinweg bis an das Ende des Signals durchzulaufen.

Der Weg der Vervollkommnung des Systems ist jedoch klar vorgezeichnet und in der Implementierung vorbereitet. Einige weitere Hinweise, unter anderem zur synchronen Kopplung, ergeben sich aus den folgenden Anhängen.

Kapitel 4 Anhänge

4.1 Anhang: Wortgraphen-Interface

File *asl_latt.c*:
enthält den Preprocessor **asl_latt()**, der für Wortgraphen des ASL-Standards je Wortkante die Pfadbewertung berechnet
(Beispiel in 2.1.1 "Struktur von Wortgraphen").
Dieser File-Standard enthält BEGIN_, END_WORTGRAPH-Klauseln und erlaubt deshalb die Sammlung diverser Graphen in einem File sowie das Einfügen von Kommentaren (darunter der gesprochene Satz). An den Fileaufruf sind darum stets den einzelnen Graphen indizierende Nummern anzuhängen (i.A. Zählung ab 0)
asl_latt() berechnet das
"Mass des schlechtesten Kettengliedes" je Wortkante
Aufruf: asl_latt wortgraph-file wg ind
Ausgabe: -->pipe: wg_[ind].00
dient als Input der linguisischen Analyse.

File *latt.c*:
enthält das Programm **latt()** zur Bestimmung des optimalen Pfads des Wortgraphen. Eine weitere Option ist:
Liste der je Endknoten ankommenden Kante mit dem besten Vor- bzw. Nach-Pfad, incl. Wortbewertung für Links- bzw. Rechts-Durchlauf
(Optionen-Ausgabe bei parameterfreiem Aufruf)

Aufruf: cat wortgraph-file | **latt**
Ausgabe: -->pipe: Liste

File *NodeDisplay.c*:
Das Programm **NodeDisplay()** liefert die ASCII-Darstellung des Graphen wg(ind), allerdings nur vertikal scrollbar, d.h. es sind Darstellungen von Wortpfaden mit Zeilenlängen möglich, welche die max. Window-Grenze überschreiten (vorher gegf. max. breites X-Fenster einstellen).
Hyperkanten sind die aus der Äquivalenzrelation der Knoten-Nummer-Identität (Anfang+Ende) hergeleiteten Aquivalenzklassen von Kanten.
Aufruf: NodeDisplay [-a|-w|-v] wg ind
ohne option : Hyperkanten zeigen
-a : alle Kanten zeigen
-w : Wörter mitanzeigen
(zur Beachtung: die Index-Zählung beginnt hier bei 1)

File *merger.c*: in Anwendung auf wg_[ind].00
merger() soll die während der Laufzeit der Analyse
ausgeführten Pre-Prozesse extern verfügbar machen,
um Graphen vorab zu testen
a) Pausenmerging durch die Analysefunktion **hypomerge()**
(-> Sprachnetz, hypothese.c)
b) Liste "Paare unverbindbarer Graphknoten"
durch **crea_nonpath()**
(-> Sprachnetz, crea_nopath.c, hypgen.c
Ort der Anwendung: test_hypgen())

Aufruf: -->pipe: worgraph-file | merger option
Ausgabe: -->pipe: Listen entspr. Optionenwahl

1. chg	{OVERLAP}	Liste Hinzufuegungen,Löschungen
2. wg	{OVERLAP}	Ergebnisgraph nach Merging
3. np+	{OVERLAP	Liste aller Knotenpaare, die kein Pfad verbindet
4. np	{OVERLAP}	Untermenge ob.Paare, an denen nicht je eine Kante mit gemeinsamem Anfangs/End-Knoten eingeht

Erläuterung der Optionen:

-> 1.) Merging aller Pausen-Kanten
oder solche mit wortnr=0 (d.h. nicht im Lexikon)
mit den potentiellen Folgekanten
(je nach Frame-OVERLAP, d.h. einer zulässigen Zahl
überlappender Frames - Defaultwert siehe proz.h)

sowie Merging von Kanten an Pfadenden mit
entsprechenden Vor-Kanten
--> Hinzufügungen
es wird geprüft, ob die unverschmolzene Kante weiter
geführt werden muss, weil alternative Kanten mit
positiver Wortnummer an ihrem Anfangspunkt eingehen
--> Löschungen

-> 2.) gibt den nach unter 1. genannten Veränderungen
erhaltenen Graphen aus

-> 3.) gibt die in der Analyse erzeugte und verwendete
Liste aller (ungeordneten) Paare von Knoten aus, die
innerhalb des Graphen nicht durch einen Pfad
verbunden werden können,
z.B. für den Beispielgraphen:

```
                    [2] ueber [3] Frankfurt[5]

 [0] ich [1] moechte [2]                    [5] fahren [6]

                    [2] nach  [4] Dortmund [5]
```

ergeben sich die Paare: (3,4), (4,3)

Solche Paare werden nur im Falle "echter Maschen" gefunden, nicht aber im Fall "simpler Maschen", d.h. einzelner Kanten, die als Konkurrenten zusammenhängenden Kantensequenzen auftreten (bei gleicher Signalüberdeckung).

-> 4.) eine echte Masche wird von 2 Gabelungspunkten (s.o. [2],[5]) aufgespannt, zwischen denen 2 Teilpfade (mindestens je 2 Worte) liegen. Dadurch sollen einander ausschliessende linguistische Alternativen abgebildet werden.
Knotenpunkte, die direkte Nachbarn der Gabelungspunkte sind, werden ausgenommen.

Zweck von 3.,4. innerhalb der Analyse:
bei der Instantiierung nicht-kohärenter Konzepte wird zur Prüfung von Wortkandidaten, die eine Insel in der Signalüberdeckung aufmachen würden, Information über die Verbindbarkeit mit instantiierten Wortkanten benötigt.
Die mit der Option np+
erstellbare Komplettliste wird deshalb in der Analyse vor Beginn jedweder Instantiierungen einmalig erstellt.

Anwendung:
Die Merging-Funktionen sind nur im Fall der asynchronen Kopplung Erkennung/Interpretation relevant.

Zu den ***Voraussetzungen der synchronen Kopplung***:

File *hypothese.c* (-> Sprachnetz):
Die modifizierte Funktion **read_whyp()** wird durch einen aus der Optionenzeile des Analyseaufrufs eingesteuerten

Parameter -SYNC (-> zu beachten in cont_init.c)

auf den synchronen Fall eingestellt.
Während dieser Kopplungsform können die inkrementell-wachsenden Wortgraphen mit der Funktion

read_whyp_aktion()

sukzessive eingelesen werden. Sie verwendet den fertigen Filepointer zu einer vorab zu öffnenden Datei.
Die Form der Codierung des über den Environmentparameter $HYPPATH zugewiesenen Namenskerns sowie Indexerweiterung ist in read_whyp() sichtbar.
Die als Verbindungsglied der asynchronen Kopplung geplante Dateiübergabe ist vom Erkenner mit einer Indizierung auszustatten, aus der die Inkrementierungs-stufe ablesbar sein soll
(-> erlaubt den Test, ob der nächste Graph anliegt).
Die Endebehandlung kann über die einmalig übertragene END_WORTGRAPH-Klausel erkannt werden.
read_whyp() ist innerhalb der zentralisierten Kontroll-funktion zur Hypothesen-Auswahl, **create_skip()**,
(-> Sprachnetz, File create_skiplist.c)
aufzurufen.
Das ist immer dann zu tun, wenn ein Neuziel M(H_WORTHYP) als letzter Struktureintrag erzeugt wurde. Die voraus-gehende Abfrage des Vorliegens des nächsten Wortgraphen ist in der Funktion test_user_ereignis() durchzuführen, welche für die Neuziel-Erweiterung zuständig ist.
Sie sollte gegebenenfalls in Haltezustand gehen, falls der nächste (bzw. ein höher-indizierter) Graph noch nicht vorliegt.

4.2 Anhang: Konzeptgraph-Liste

Als Gegenstück zur im ERNEST-Manual ausgewiesenen Funktion sll_tree (KG n), welche den Konzeptgraphen des gewählten Suchbaumknoten n in einer Kurz- oder Langform auflistet, wurde die Funktion sl (KG n) geschaffen (-> File *sl.c*). Das untenstehende Beispiel zeigt den letzten Knoten der Analyse eines Wortgraphen zur Äußerung

[ich, moechte, morgen, nachmittag, nach, Dortmund, fahren].

Die Liste bietet zunächst die Einordnung in den Suchbaum (Vorgaenger .., Nachfolger: ..). END_KNOTEN bedeutet, daß kein Nachfolger vorliegt. Zeigt die als Listenabschluß angefügte Bewertungszeile nicht den Wert UNZULAESSIG an, so liegt ein END_KNOTEN notwendigerweise in OFFEN. Speziell für die Analyse von Wortgraphen wird die assozierte Wortkette des Konzeptgraphen im Kopf angegeben. Sie wird im Beispiel mit je einer Zeile für PASS1/2 angeordnet. Mit [-] werden die Links/Rechts-Knoten der assoziierten Wortkandidaten notiert, um das mehrfache Vorkommen eines lexikalischen Items unterscheiden zu können. Für die Einträge des Konzeptgraphen wird je eine Zeile mit der folgenden Gliederung geführt:

```
<1517>      I()              (P_VR_FAHREN)        57
eintrag     ableitungs-       konzept      instanznr.
            grad
```

Für den Ableitungsgrad M() handelt es sich in der letzten Stelle um die Nummer des sogenannten modifizierten Konzepts. Je nach Vorliegen von I() vs. M() sind 2 (im ERNEST-Manual angegebene) Hilfsfunktionen, sll_inst(), sll_modkonz(), aus der Unix-Shell aufrufbar, welche den letztgenannten Parameter als ID verwenden. Damit erhält man eine Übersicht der Bewertung jedes Attributs, das auf den Ableitungsgrad Einfluß hatte (-> die Stelle einer attributiven UNZULAESSIGKEIT). Die alte, aus der UNIX-Shell aufrufbare Funktion sll_tree kann zusätzlich verwendet werden, um weitere Differenzierungen des Ableitungsgrads aus dem Eintrag abzulesen.

Der Vorteil der neuen (ebenfalls in ASCII-Code ausgegeben) Liste besteht vor allem, in der besseren Übersicht der Struktur des Konzeptgraphen als Produkt inkrementeller Verarbeitung. Es wird sichtbar gemacht, durch welche Teilbäume der Konzeptgraph aufgespannt wird. Die Analysephasen werden bezüglich der Bindung der Kontextkonzepte besser sichtbar. Die volle Auflistung aller inhaltlich zu unterscheidenden Teilbäumen würde zu viel Redundanz enthalten. Unterdrückte oder auch wiederkehrende Teilstrukturen sind anhand der Eintragsnummer zu identifizieren.

Das folgende Bild zeigt die durch Rechts-Einrückung dargestellte Einbindung eines Eintrags in seinen strukturhöheren Eintrag, hier der komplett aufgeführten Wurzelbäume in den pragmatischen Verbrahmen. Die auf der rechten Seite extra hervorgehobene linksbündige Positionierung von Einträgen drückt die Rolle als Wurzel eines eigenen Teilbaums aus, der in keinen höheren Eintrag eingebunden ist. Die Reihenfolge in der Vertikalen drückt die Entstehungszeit in der inkrementellen Verarbeitungsfolge aus.

```
   späterer Konzeptgraph                    früherer
  ------------------------------------|----------------
  <1517> I(P_VR_FAHREN) 57             |
         <1511> I(P_REISENDER) 50      |   <1511> ...
                <..>                   |          <..>
         <1513> I(P_ABFAHRTSZEIT) 52   |   <1513> ...
```

Auch der die Einbindung in den semantischen Verbrahmen ausdrückende Teilbaum wird komplett aufgeführt (-> Redundanz). Der an das Auskunftskonzept gebundene Gesamtgraph wird entsprechend der Verarbeitungslogik in Phase III, die nur noch mit den pragmatischen Wurzeln befaßt ist, auf dieselbigen beschränkt.

Den Blättern der Bäume (mit verkürztem Konzeptnamen WH für H_WORTHYP) wurde zur schnelleren Erfassung der Gegenstandsbildung der assozierte Wortkandidat angehängt. In Klammern wurde die instantiierte lexikalische Subkategorie angegeben. Sie zeigt (-2) im Falle genau einer Subkategorie. Damit kann zumindest beim “manuellen Durchmustern” in OFFEN nach einer liegengebliebenen subkategoriellen Alternative gesucht werden.

```
KNOTEN (99):   Vorgaenger 97  Nachfolger: END_KNOTEN

 [0-1]ich      [2-7]morgen     [9-6]Dortmund [6-10]fahren
 [1-2]moechte  [7-8]nachmittag [8-9]nach

<1517> I(P_VR_FAHREN) 57
       <1511> I(P_REISENDER) 50
              <1512> I(S_AGENT) 47
                     <1529> I(SY_NG) 10
                            <1530> I(SY_PRON) 9
                                   <1531> I(WH) 8
                                       [ich  (1)]
       <1513> I(P_ABFAHRTSZEIT) 52
              <1514> I(S_TIME) 51
                     <1526> I(Z_ZEITANGABE) 27
                            <1527> I(Z_TAG) 18
                                   <1528> I(Z_ADV) 17
                                          <1532> I(WH) 11
                                             [morgen  (2)]
                            <1524> I(Z_ABSCHNITT) 26
                                   <1525> I(Z_ADV) 23
                                          <1533> I(WH) 22
                                        [nachmittag  (1)]
```

```
      <1515> I(P_ANKUNFTSORT) 55
                <1516> I(S_GOAL) 54
                       <1521> I(SY_PNG) 35
                              <1522> I(SY_NG) 30
                                     <1523> I(SY_NPR) 29
                                            <1534> I(WH) 28
                                             [Dortmund  (1)]
                              <1535> I(SY_PRAEP) 33
                                     <1536> I(WH) 32
                                         [nach  (1)]
        <1518> I(S_VR_FAHREN) 56
                <1519> I(SY_VG) 42
                       <1520> I(SY_VERB) 38
                              <1537> I(WH) 37
                               [fahren  (-2)]
                       <1538> I(SY_MVERB) 41
                              <1539> I(WH) 40
                              [moechte  (-2)]
                <1512> I(S_AGENT) 47
                       <1529> I(SY_NG) 10
                              <1530> I(SY_PRON) 9
                                     <1531> I(WH) 8
                                         [ich  (1)]
                <1514> I(S_TIME) 51
                       <1526> I(Z_ZEITANGABE) 27
                              <1527> I(Z_TAG) 18
                                     <1528> I(Z_ADV) 17
                                            <1532> I(WH) 11
                                              [morgen  (2)]
                                  <1524> I(Z_ABSCHNITT) 26
                                         <1525> I(Z_ADV) 23
                                                <1533> I(WH) 22
                                               [nachmittag  (1)]
<1510> I(P_FAHRPLAUSK) 58
       <1540> I(P_ABFAHRTSORT) 7
       <1513> I(P_ABFAHRTSZEIT) 52
       <1515> I(P_ANKUNFTSORT) 55
Bewertung:
 268.390 (ling.Prio 0,0) Skip-Ueb. 219 (Real-Ueb. 219)
```

Die Instantiierung des Auskunftskonzepts (fettgedruckt) drückt die erfolgreiche Analyse aus. Der inhärent eingefügte Wurzelbaum des Ankunftsortes nicht komplett aufgeführt. Die Bewertungszeile zeigt in Klammern die Maße 4,5 zur Steuerung im Sinne der Aufgabenkontrolle sowie die beiden Überdeckungsmaße. Dieser Teil ist also besonders anwendungsorientiert und für andere Domänen in der Funktion sl() zu ändern.

Literaturverzeichnis

[Aho86] A. Aho, R. Sethi, J. Ullman: *Compilers, Principles, Techniques, and Tools*, Addison-Wesley, MA, 1986.

[BC90] T. Bench-Capon: *Knowledge representation: an approach to artificial intelligence*, Academic Press, London, 1990.

[Bel73] J. Bell: *Threaded Code, Communications of the Association for Computing Machinery*, Bd. 16, 1973, S. 370–372.

[Bou94] G. Boulianne, P. Kenny, M. Lennig, D. O'Shaughnessy, P. Mermelstein: *Books on tape as training data for continuous speech recognition*, in *Speech Communication*, 1994, S. 61 – 70.

[Bra77] R. J. Brachman: *What's in a Concept: Structural Foundations for Semantic Networks*, Bolt, Beranek and Newman, Cambridge, MA, 1977.

[Bra79] R. J. Brachman: *On the Epistemological Status of Semantic Networks*, in N. V. Findler (Hrsg.): *Associative Networks*, Academic Press, New York, 1979, S. 3–50.

[Cho65] N. Chomsky: *Aspects of the Theory of Syntax*, MIT Press, Cambridge, MA, 1965.

[Cla90] W. Clancey: *Artificial intelligence and learning environments*, MIT Press, Cambridge, MA, 1990.

[Clo90] W. Clocksin, C. Mellish: *Programmieren in Prolog*, Springer-Verlag, Berlin, 1990.

[Dew75] R. Dewar: *Indirect Threaded Code, Communications of the Association for Computing Machinery*, Bd. 18, 1975, S. 330–331.

[Eck92] W. Eckert, G. Fink, A. Kießling, R. Kompe, T. Kuhn, F. Kummert, M. Mast, H. Niemann, E. Nöth, R. Prechtel, S. Rieck, , G. Sagerer, A. Scheuer, E. G. Schukat-Talamazzini, S. B.: *EVAR: Ein sprachverstehendes Dialogsystem*, in *KONVENS 92*, Informatik aktuell, Springer-Verlag, Berlin, 1992, S. 49–58.

[Eik89] H.-J. Eikmeyer: *PROLOG*, in *7.Frühjahrsschule, KIFS-89*, Springer-Verlag, Berlin, 1989, S. 100–121.

[Eve79] S. Even: *Graph algorithms*, Pitman, Cambridge, MA, 1979.

[Fil71] J. Fillmore: *Plädoyer für Kasus (Original: The Case for Case)*, in W. Abraham (Hrsg.): *Kasustheorie*, Athenäum-Verlag, Wiesbaden, 1971.

[Fin71] N. Findler: *Künstliche Intelligenz und heuristisches Programmieren*, Springer-Verlag, Berlin, 1971.

[Fin79] N. Findler (Hrsg.): *Associative networks: representation and use of knowledge by computers*, Academic Press, New York, 1979.

[Fin92a] G. Fink, G. Sagerer, F. Kummert: *Automatic Extraction of Language Models from a Linguistic Knowledge Base*, in *Proc. European Signal Processing Conference*, Brussels, 1992, S. 547–550.

[Fin92b] G. A. Fink, F. Kummert, G. Sagerer, E. G. Schukat-Talamazzini, H. Niemann: *Semantic Hidden Markov Networks*, in *International Conference on Spoken Language Processing, 12.-16. October, 92*, Bd. 2, Banff, Alberta, Canada, 1992, S. 919–922.

[Fin93a] G. Fink, F. Kummert, G. Sagerer, E. Schukat-Talamazzini: *Speech Recognition Using Semantic Hidden Markov Networks,* in *Proc. European Conf. on Speech Communication and Technology*, Bd. 3, Berlin, 1993, S. 1571–1574.

[Fin93b] G. Fink, F. Kummert, G. Sagerer, B. Seestaedt: *Robust Interpretation of Speech,* in *Proc. European Conf. on Speech Communication and Technology*, Bd. 3, Berlin, 1993, S. 1529–1532.

[Fra78] L. Frazier, J. Fodor: *The sausage machine: A new two-stage parsing model*, *Cognition*, Bd. 6, 1978, S. 291–325.

[Fuh89] T. Fuhr: *Ein Übersetzer für symbolische Attributwertberechnungsausdrücke auf der Grundlage algebraischer Systeme,* Technical report, Universität Erlangen-Nürnberg, Erlangen, 1989.

[Fuh93] T. Fuhr, F. Kummert, S. Posch, G. Sagerer: *An Approach for Qualitatively Predicting Relations from Relations*, in E. Sandewall, C. G. Jansson (Hrsg.): *Fourth Scandinavian Conference on Artificial Intelligence – 93,* IOS Press, Amsterdam, Stockholm, 1993, S. 38–49.

[Gen89] M. R. Genesereth, N. J. Nilsson: *Logische Grundlagen der Künstlichen Intelligenz,* Vieweg-Verlag, Braunschweig, 1989.

[God92] D. Goddeau: *Using Probabilistic Shift-Reduced Parsing in Speech Recognition Systems,* in *International Conference on Spoken Language Processing, 12.-16. October, 92*, Banff, Alberta, Canada, 1992, S. 321–324.

[Gör88] G. Görz: *Strukturanalyse natürlicher Sprache*, Addison-Wesley, Bonn, 1988.

[Gri75] J. Grimes: *Network Grammars*, Bd. 1, University of Oklahoma, Cambridge, MA, 1975.

[Gri86] R. Grisham: *Computational linguistics*, Cambridge University Press, Cambridge,NY, 1986.

[Hel90] P. Hellwig: *Formal-desambiguierte Repräsentation*, Springer-Verlag, Berlin, 1990.

[Hel91] H. Hellbig: *Künstliche Intelligenz und automatische Wissensverarbeitung*, Verlag Technik, Berlin, 1991.

[Hil91] B. Hildebrand: *ERNEST-Manual, Version 1.7*, Handbuch, Technische Fakultät der Universität Erlangen-Nürnberg, Bielefeld-Erlangen, 1991.

[Hil93a] B. Hildebrand: *Struktur und Bedeutung temporaler Konstituenten bei Anfragen über Bahnfahrten*, Technical report, Technische Fakultät der Universität Erlangen-Nürnberg, Bielefeld, 1993.

[Hil93b] B. Hildebrandt, G. Fink, F. Kummert, G. Sagerer: *Modelling of Time Constituents for Speech Understanding*, in *Proc. European Conf. on Speech Communication and Technology*, Bd. 3, Berlin, 1993, S. 2247–2250.

[Hir79] D. Hirschberg, A. Chandra, D. Sarwate: *Computing connected components on parallel computers, CACM*, Bd. 22, Nr. 8, 1979, S. 461–464.

[Hol75] K. Holzkamp: *Sinnliche Erkenntnis: historischer Ursprung u. gesellsch. Funktion d. Wahrnehmung*, Athenaeum Fischer, Frankfurt a.M., 1975.

[Hop88] J. Hopcroft, J. Ullman: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*, Addison-Wesley, Bonn, 1988.

[Kan80] T. Kanade: *SURVEY Region Segmentation: Signal vs. Semantics*, *Computer Graphics and Image Processing*, 1980.

[Kim73] J. Kimball: *Seven priciples of surface structure parsing in natural language*, *Cognition*, Bd. 2/1, 1973, S. 15–47.

[Kla91] H. Klaeren: *Vom Problem zum Programm*, Teubner, Stuttgart, 1991.

[Kov84] V. Kovalevski: *Discrete topology and contour definition*, *Pattern Recognition Letters*, Bd. 2, 1984, S. 281–288.

[Kov86] V. Kovalevski, B. Seestaedt: *Image segmentation by multi-resolution smoothing and zero-crossing*, Technical report 86-73-02, Central Institute of Cybernetics, Berlin, GDR, 1986.

[Kuh79] T. Kuhn: *Die Struktur wissenschaftlicher Revolutionen*, Suhrkamp, Frankfurt a.M., 1979.

[Kuh91] T. Kuhn, S. Kunzmann, F. Kummert, M. Mast, H. Niemann, E. Nöth, R. Prechtel, S. Rieck, A. Reißer, G. Sagerer, E. G. Schukat-Talamazzini, U. J.: *Ein System zur Interpretation einer Äußerung: Methoden*, in *Plenarvorträge und Fachbeiträge der 17. Gemeinschaftstagung der Deutschen Arbeitsgemeinschaft für Akustik*, Bd. Teil B von *Fortschritte der Akustik*, Bochum, 1991, S. 1081–1084.

[Kuh94] T. Kuhn: *Die Erkennungsphase in einem Dialogsystem*, Dissertation, Universität Erlangen-Nürnberg, Erlangen, 1994.

[Kum87] F. Kummert, H. Niemann, G. Sagerer, S. Schröder: *Werkzeuge zur modellgesteuerten Bildanalyse und Wissensakquisition – Das System ERNEST*, in M. Paul (Hrsg.): *GI – 17. Jahrestagung Computerintegrierter Arbeitsplatz im Büro*, Springer-Verlag, Berlin, 1987, S. 556–570.

[Kum88] V. Kumar, D. S. Nau, L. N. Kanal: *A General Branch-and-Bound Formulation for And/Or Graph and Game Tree Search*, in L. N. Kanal, V. Kumar (Hrsg.): *Search in Artificial Intelligence*, Springer-Verlag, New York Berlin Heidelberg London Paris Tokyo, 1988, S. 91–130.

[Kum91a] F. Kummert, G. Sagerer: *Integrierte daten- und erwartungsgesteuerte Analyse gesprochener Sprache*, in B. Radig (Hrsg.): *Mustererkennung 91, 13. DAGM-Symposium München, Informatik-Fachberichte*, Springer-Verlag, Berlin, 1991, S. 103–110.

[Kum91b] F. Kummert, G. Sagerer, E. Nöth, H. Niemann: *Kontrolle eines homogenen sprachverstehenden Systems*, in *Plenarvorträge und Fachbeiträge der 17. Gemeinschaftstagung der Deutschen Arbeitsgemeinschaft für Akustik*, Bd. Teil B von *Fortschritte der Akustik*, Bochum, 1991, S. 1089–1092.

[Kum92a] F. Kummert: *Flexible Steuerung eines sprachverstehenden Systems mit homogener Wissensbasis*, Bd. 12 von *Dissertationen zur Künstlichen Intelligenz*, Infix, Sankt Augustin, 1992.

[Kum92b] F. Kummert, G. Fink, G. Sagerer: *Robuste Verarbeitung fehlerhafter Segmentierungsergebnisse*, in S. Fuchs, R. Hoffmann (Hrsg.): *Mustererkennung 92, 14. DAGM-Symposium Dresden, Informatik aktuell*, Springer-Verlag, Berlin, 1992, S. 269–273.

[Kum92c] F. Kummert, G. Sagerer, H. Niemann: *A Problem–Independent Control Algorithm for Image Understanding*, in *11th International Conference on Pattern Recognition*, Bd. I, The Hague, 1992, S. 297–301.

[Kum93a] F. Kummert, E. Littmann, A. Meyering, S. Posch, H. Ritter, G. Sagerer: *A Hybrid Approach to Signal Interpretation Using Neural and Semantic Networks*, in *Mustererkennung 93, 15. DAGM-Symposium Lübeck*, Springer-Verlag, Berlin, 1993, S. 245–252.

[Kum93b] F. Kummert, E. Littmann, A. Meyering, S. Posch, H. Ritter, G. Sagerer: *Recognition of 3D-Orientation from Monocular Color Images by Neural Semantic Networks*, *International Journal of Pattern Recognition and Image Analysis*, Bd. 3, Nr. 3, 1993, S. 311–316.

[Kum93c] F. Kummert, H. Niemann, R. Prechtel, G. Sagerer: *Control and Explanation in a Signal Understanding Environment*, *Signal Processing, special issue on 'Intelligent Systems for Signal and Image Understanding'*, Bd. 32, 1993, S. 111–145.

[LA93] T. M. Lavie A.: *GLR* - An Efficient Noise-skipping Parsing Algorithm for Context-free Grammars*, in *Third International Workshop on Parsing Technologies*, 1993, S. 123–134.

[Lea94] J. Leavitt, D. Lonsdale, A. Franz: *A Reasoned Interlingua for Knowledge-based Machine Translation*, *Proceedings of the Canadian Artificial Intelligence Conference*, 1994.

[Leo73] A. Leontjew: *Probleme der Entwicklung des Psychischen*, Athenaeum Fischer, Frankfurt a.M., 1973.

[Les88] A. Lesgold: *Was ist intelligenter computerunterstützter Unterricht?*, in H. Mandl (Hrsg.): *Wissenspsychologie*, Psychologie-Verl.-Union, 1988, S. 554–569.

[Lev79] H. J. Levesque, J. Mylopoulos: *A Procedural Semantics for Semantic Networks*, in N. V. Findler (Hrsg.): *Associative Networks*, Academic Press, New York, 1979, S. 93–121.

[Loe81] R. Loeliger: *Threaded Interpretive Languages*, BYTE Books, Mc Grew-Hill, Peterborough, 1981.

[Mac76] A. Mackworth: *Consistency in Networks of Relations*, *Artificial Intelligence*, Bd. 8, 1976, S. 77–98.

[Mar80] M. Marcus: *A Theory of Syntactic Recognition for Natural Language*, Bd. 1, MIT Press, Cambridge, MA, 1980.

[Mar82] D. Marr: *Vision - a Computational Investigation into the Human Representation and Processing of Visual Information*, Freemann, San Francisco, 1982.

[Mas92] M. Mast, R. Kompe, F. Kummert, H. Niemann, E. Nöth: *The Dialog Module of the Speech Recognition and Dialog System EVAR*, in *International Conference on Spoken Language Processing, 12.-16. October, 92*, Bd. 2, Banff, Alberta, Canada, 1992, S. 1573–1576.

[May75] J. Maynard: *Dictionary of Data Processing*, Newnes-Butterworths, London, 1975.

[Med87] N. Meder: *Der Sprachspieler: d. postmoderne Mensch oder d. Bildungsideal im Zeitalter d. neuen Technologien*, Bd. 1, Janus-Presse, Köln, 1987.

[Mic83] D. Michie, J. R. (Hrsg.): *Chess skill in man and machine*, Springer, New York, 1983.

[Mil60] G. Miller, E. Galanter, K. Pribram: *Plans and the structure of behaviour*, Holt, London, 1960.

[Min69] M. Minsky, S. Papert: *Perceptrons: an introduction to computational geometry*, MIT Press, Cambridge, MA, 1969.

[Müh94] M. Mühlhäuser, B. Frommherz: *Nestor System - Course Authoring and Learning, a New Approach*, Technical report, DEC's CEC, Karlsruhe, 1994.

[Myl83] J. Mylopoulos, H. J. Levesque: *An Overview of Knowledge Representation*, in M. Brodie, J. Mylopoulos, J. V. Schmidt (Hrsg.): *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer-Verlag, New York, 1983.

[Myl89] J. Mylopoulos: *An Overview of Knowledge Representation*, in J. Mylopoulos, M. Brodie (Hrsg.): *Readings in artificial intelligence and databases*, Kaufmann, Los Altos, 1989.

[Nie90] H. Niemann, G. Sagerer, S. Schröder, F. Kummert: *ERNEST: A Semantic Network System for Pattern Understanding*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Bd. 12, 1990, S. 883–905.

[Nil82] N. J. Nilsson: *Principles of Artificial Intelligence*, Springer-Verlag, Berlin, 1982.

[Oer93] M. Oerder, H. Ney: *Word Graphs: An Efficient Interface Between Continuous-Speech Recognition and Language Understanding*, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Bd. 2, Minneapolis, 1993, S. 119–122.

[Pal75] J. Palme: *Rechenanlagen, die natürliche Sprache verstehen*, in N. Findler (Hrsg.): *Künstliche Intelligenz und heuristisches Programmieren*, Springer-Verlag, Berlin, 1975, S. 209–254.

[Pau92] D. B. Paul: *An efficient A* Stack Decoder Algorithm for Contiuous Speech Recognition with a Stochastic Language Model*, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Lincoln Laboratory, MIT, 1992, S. I–25–I–28.

[Per80] F. Pereira, D. Warren: *Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks*, *Artificial Intelligence*, Bd. 3, 1980, S. 231–278.

[Pre89] R. Prechtel: *Erklärungen für komplexe Wissensbasen*, Dissertation, Universität Erlangen-Nürnberg, Erlangen, 1989.

[Pro94] B. Profitlich: *SB-ONE: ein Wissensrepräsentationssystem basierend auf KL-ONE*, Technical report, Technische Fakultät der Universität Erlangen-Nürnberg, Saarbrücken, 1994.

[P.W85] F. P.W.: *The creative computer: machine intelligence and human knowledge*, Penguin, Harmondsworth, 1985.

[Rab78] L. Rabiner: *Digital processing of speech signals*, Bd. 1, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Rab85] L. R. Rabiner, S. E. Levinson: *A Speaker-Independent, Syntax-Directed, Connected Word Recognition System Based on Hidden Markov Models and Level Building*, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Bd. 33, 1985, S. 561–573.

[Rab94] L. Rabiner (Hrsg.): *Digital signal processing*, Bd. 1, IEEE Press reprint serie, New York, 1994.

[Rau93] H. Rautenstrauch: *Schritthaltende Generierung der n-besten Wortketten*, Technical report, techfak, Bielefeld, 1993.

[Rau94] H. Rautenstrauch, G. A. Fink, F. Kummert, G. Sagerer: *Schritthaltende Generierung von Wortgraphen*, in *Plenarvorträge und Fachbeiträge der 20. Gemeinschaftstagung der Deutschen Arbeitsgemeinschaft für Akustik*, Bd. Teil C von *Fortschritte der Akustik*, Dresden, 1994, S. 1261–1264.

[Ric89] M. Richter: *Prinzipien der künstlichen Intelligenz*, Teubner, Stuttgart, 1989.

[Rie89] C. Riesbeck, S. R.C.: *Inside Case-Based Reasoning*, Bd. 1, LEA, New Jersey, Los Altos, CA, 1989.

[Rit80] T. Ritter: *Varieties of Threaded Code for Language Implementation*, *BYTE*, Bd. 5, 1980, S. 206–227.

[Ros78] E. Rosch: *Cognition and Categorization*, in E. Rosch, B. Lloyd (Hrsg.): *Principles of categorization*, Erlbaum, Hillsdale, NJ, 1978.

[SAB93] G. SABAH: *Formal Languages of Labelled Graphs*, *AI Communicatiosn,EJAI*, Bd. 6, Nr. 3/4, 1993, S. 155–186.

[Sag87] G. Sagerer, F. Kummert, E. G. Schukat-Talamazzini: *Flexible Steuerung eines sprachverstehenden Systems mit Hilfe mehrkomponentiger Bewertungen*, in E. Paulus (Hrsg.): *Mustererkennung 87, 9. DAGM-Symposium Braunschweig, Informatik-Fachberichte*, Springer-Verlag, Berlin, 1987, S. 123–127.

[Sag88a] G. Sagerer, U. Ehrlich, F. Kummert, H. Niemann, E. G. Schukat-Talamazzini: *A Flexible Control Strategy with Multilevel Judgements for a Knowledge Based Speech Understanding System*, in *9th International Conference on Pattern Recognition*, Rom, 1988, S. 788–790.

[Sag88b] G. Sagerer, F. Kummert: *Knowledge Based Systems for Speech Understanding*, in H. Niemann, M. Lang, G. Sagerer (Hrsg.): *Recent Advances in Speech Understanding and Dialog Systems*, NATO ASI Series F, Vol. 46, Springer-Verlag, Berlin, 1988, S. 421–458.

[Sag90] G. Sagerer: *Automatisches Verstehen gesprochener Sprache*, Bd. 74 von *Reihe Informatik*, Bibliographisches Institut, Mannheim, 1990.

[Sag91] G. Sagerer, F. Kummert, G. A. Fink, B. Seestaedt: *Automatische Extraktion von Sprachmodellen für Hidden-Markov-Modelle aus einem semantischen Netzwerk*, in R. Hoffmann (Hrsg.): *Elektronische Sprachsignalverarbeitung*, Bd. 8 von *Studientexte zur Sprachkommunikation*, Dresden, 1991, S. 151–160.

[Sat88] P. Sato: *A common Parsing Scheme for Left-to-Right-Branching Languages*, *Computational Linguistics*, Bd. 14, Nr. 1, 1988, S. 20–30.

[See92] B. Seestaedt, F. Kummert, G. Sagerer: *Speech Understanding by Alternating Control: A Semantic Network Approach*, in *Proc. European Signal Processing Conference*, Brussels, 1992, S. 555–558.

[See93] B. Seestaedt, G. Sagerer, F. Kummert: *Robuste Links-Rechts-Verarbeitung von spontan-gesprochener Sprache*, in *Plenarvorträge und Fachbeiträge der 19. Gemeinschaftstagung der Deutschen Arbeitsgemeinschaft für Akustik*, Bd. B von *Fortschritte der Akustik*, Frankfurt am Main, 1993, S. 952–955.

[See94] B. Seestaedt, F. Kummert, G. Sagerer: *Left-to-Right Analysis of Spoken Language*, in *Workshop 'Natural Language and Speech Processing', AAAI'94*, Seattle, 1994, S. 1–6.

[Sow84] J. Sowa: *Conceptual structures: Information Processing in Mind and Machine*, Addison-Westley Publishing Company, MA, 1984.

[Sow91] J. Sowa (Hrsg.): *Principles of semantic networks: explorations in the representation of knowledge*, Morgan Kaufmann, Mateo, Calif., 1991.

[Spi94] M. Spies: *Grundzüge der Spracherkennung in einem Diktiersystem*, *Spektrum der Wissenschaften*, Bd. 3, 1994, S. 90–94.

[Ste94] V. Steinbiss: *Auf der Suche nach der optimalen Wortfolge*, *Spektrum der Wissenschaften*, Bd. 3, 1994, S. 96.

[Sto91] H. Stoyan: *Programmiermethoden der künstlichen Intelligenz*, Bd. 1, Springer-Verlag, Berlin, 1991.

[Tis92] J. Tischler: *Linguistische Datenverarbeitung und historische Sprachwissenschaft - I: Die Programmiersprache AWK*, Innsbrucker Beiträge zur Sprachwissenschaft, AKAPRINT Budapest, Innsbruck, 1992.

[Tom86] M. Tomita: *Efficient Parsing for Natural Language*, Bd. 1, Kluver Academic, Hingham MA, 1986.

[Vac90] G.-U. Vack: *Programmieren mit FORTH*, Verlag Technik, Berlin, 1990.

[Vel88] B. Velichkovski: *Wissen und Handeln: kognitive Psychologie aus tätigkeitstheoret. Sicht*, VCH, Weinheim, 1988.

[Wal75] D. Waltz: *Understanding line drawings of scenes with shadows*, in P. Winston (Hrsg.): *Psychology of Computer Vision*, McGraw-Hill, New York, 1975, S. 19–91.

[War91a] W. Ward: *Understanding Spontaneous Speech: the PHOENIX System*, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Bd. 1, Toronto, 1991, S. 365–367.

[War91b] W. Ward: *Understanding Spontaneous Speech: The PHOENIX System*, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Toronto, 1991, S. 365–367.

[Win87a] P. Winston: *Artificial Intelligence*, Bd. 1, Addison-Wesley, MA, 1987.

[Win87b] P. Winston: *Künstliche Intelligenz*, Bd. 1, Addison-Wesley, Bonn, 1987.

[Wit77] L. Wittgenstein: *Philosophische Untersuchungen*, Suhrkamp, Frankfurt a.M., 1977.

[Woo70a] W. A. Woods: *Transition Network Grammar for Natural Language Analysis*, *Communications of the ACM*, Bd. 13, 1970, S. 591–602.

[Woo70b] W. A. Woods: *Transition Network Grammars for Natural Language Analysis*, *Communications of the Association for Computing Machinery*, Bd. 13, 1970, S. 591–606.

[Woo72] W. A. Woods: *The Lunar Sciences Natural Language Information System – BBN Report 2378*, Bolt, Beranek and Newman, Cambridge, MA, 1972.

[Woo75] W. A. Woods: *What's in a Link? Foundations for Semantic Networks*, in B. Bobrow, A. Collins (Hrsg.): *Representation and Understanding*, Academic Press, New York, 1975, S. 35–82.

[Woo76] W. A. Woods, M. Bates, G. Brown, others: *Speech Understanding Systems – Final Technical Progress Report*, Bd. 1-4, Bolt, Beranek and Newman, Cambridge, MA, 1976.

[Woo79] W. Woods: *Semantics for a question-answering system*, Bd. 1, Garland, Cambridge, MA, 1979.

[Woo82] W. A. Woods: *Optimal Search Strategies for Speech Understanding Control*, *Artificial Intelligence*, Bd. 18, 1982, S. 295–326.

[Woo89] W. Woods: *Knowledge Base Retrieval*, in J. Mylopoulos, M. Brodie (Hrsg.): *Readings in artificial intelligence and databases*, Kaufmann, Los Altos, 1989, S. 179–195.

[You88] S. Young, W. Ward: *Towards Habitable Systems: Use of World Knowledge to Dynamically Constrain Speech Recognition*, in *Proc. 2.Symp. Advanced Man-Machine Interface*, Hawaii, 1988, S. 30.1–30.12.

[You89] S. Young, A. Hauptmann, W. Ward, E. Smith, P. Werner: *High Level Knowledge Sources in Usable Speech Recognition Systems*, *Communications of the ACM*, Bd. 32, Nr. 2, 1989, S. 183–193.

[Zem88] H. Zemanek: *Der Geist in der Flasche: Warum der Computer nicht ausschaut*, *Informationstechnik*, Bd. 30, 1988, S. 3–10.