

AD-A101 562

INTEGRATING SYNTAX SEMANTICS AND DISCOURSE DARPA
NATURAL LANGUAGE UNDERST. (U) UNISYS CORP PAOLI PA
PAOLI RESEARCH CENTER D DAHL ET AL. 14 MAY 87

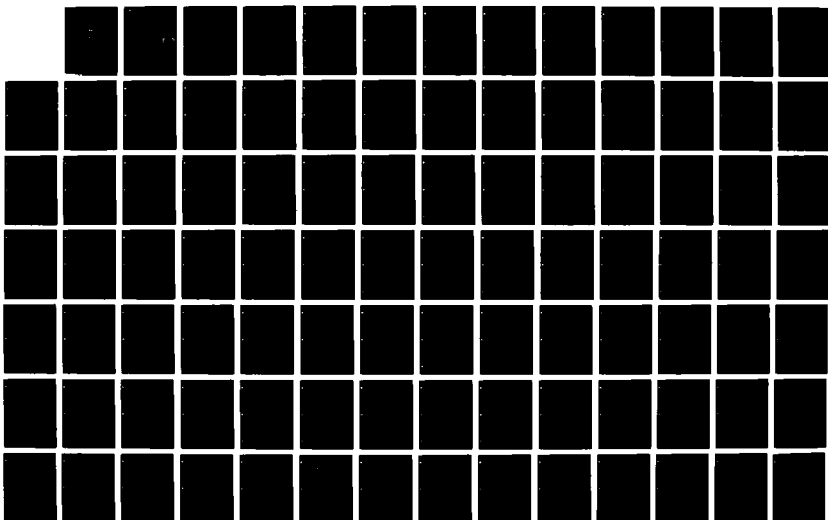
1/3

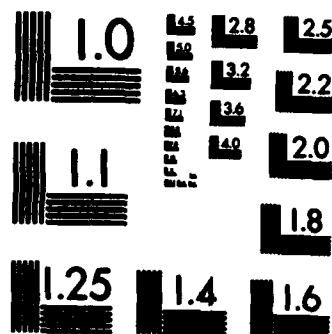
UNCLASSIFIED

N00014-85-C-0012

F/G 5/7

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A181 562

DTIC FILE COPY

12

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D STATUS REPORT
Unisys/Defense Systems

DTIC
ELECTE
JUN 04 1987

Volume II -- APPENDICES

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

Deborah Dahl, John Dowding, Lynette Hirschman,
François Lang, Marcia Linebarger, Martha Palmer,
Rebecca Passonneau, Leslie Riley

May 14, 1987

87 6 1 049

- (1) Construction of a qualitative model of a starting air compressor;
- (5) Use of the qualitative model to simulate normal and faulty behavior described by natural-language phrases. ←

Because this research has been conducted jointly at Unisys (Paoli) and NYU (New York), the two groups have focused on different modules within the system, merging each module into the system as it becomes stable. The Unisys effort has focused primarily on the semantic and pragmatic modules, while the NYU effort has been focused on development of a qualitative domain model (starting air compressors), use of this model in resolving noun phrase references, and qualitative reasoning about the fault-diagnosis process. The Unisys work, as specified in the original Statement of Work, has been conducted primarily in Prolog, to take advantage of the the excellent match between Prolog and the requirements for building a natural language processing system. This has resulted in a system with good performance and ready portability to a variety of hardware: the system runs on Vax, Sun, Explorer and Xerox Lisp Machines. The NYU work has been in Lisp and their completed system will be delivered in CommonLisp.

2. Objectives

→ The overall objective of the natural-language understanding work is to demonstrate capabilities for ^{single queries} understanding information contained in free narrative. This understanding can be demonstrated in several ways: simulation of →

events described in the narrative (as done by the current Proteus system); summarization of events described in the narrative (also done by the current Proteus system); or creation of database or knowledge base updates, to add information to an existing database or knowledge base. Such an understanding depends on the application of many sources of information, including syntactic, semantic, and pragmatic information, as well as detailed information about the specific domain in question. ~~The work described in~~ this report ~~has~~ focused on several critical research issues in building Proteus:

- Portability, supported by clear factoring of domain-independent and domain-specific information, and by a collection of tools to support creation of the various modules;
- Modularity, supporting incremental development of a large-scale system and permitting a division of labor between Unisys and NYU;
- Integration of multiple sources of information, to provide search focus during parsing and convergence on a correct interpretation of individual sentences (and ultimately of the entire discourse);
- Robustness, provided by a broad-coverage grammar, integration of multiple knowledge sources to detect inconsistent information, and feedback to the user to provide help in diagnosing missing or incorrect information;
- ^{and a} development environment for the construction of a large-scale natural-language processing system, including tools for debugging, testing, updating, and tailoring the system to different types of development.

**INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM**

**R&D STATUS REPORT
Unisys/Defense Systems**

Volume II -- APPENDICES

**Deborah Dahl, John Dowding, Lynette Hirschman,
François Lang, Marcia Linebarger, Martha Palmer,
Rebecca Passonneau, Leslie Riley**

May 14, 1987

TABLE OF APPENDICES

- Appendix A -- An Overview of the PUNDIT Text Processing System
- Appendix B -- Recovering Implicit Information
- Appendix C -- Focusing and Reference Resolution in PUNDIT
- Appendix D -- A Dynamic Translator for Rule Pruning in Restriction Grammar
- Appendix E -- Determiners, Entities, and Contexts
- Appendix F -- Verb Taxonomy
- Appendix G -- Conjunction in Meta-Restriction Grammar
- Appendix H -- A Prolog Structure Editor
- Appendix I -- Designing Lexical Entries for a Limited Domain
- Appendix J -- A Computational Model of the Semantics of Tense and Aspect
- Appendix K -- Nominalisations in PUNDIT
- Appendix L -- Situations and Intervals
- Appendix M -- The Interpretation of Tense in Discourse
- Appendix N -- Report on an Interaction between the Syntactic and Semantic Components
- Appendix O -- Improved Parsing Through Interactive Acquisition of Selectional Patterns
- Appendix P -- Grammatical Coverage of the CASREPs



RE: Distribution Statement
 Approved for Public Release. Distribution
 Unlimited.
 Per Dr. Alan Meyrowitz, ONR/Code 1133

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By <i>per phone</i>	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

APPENDIX A

An Overview of the PUNDIT Text Processing System

This paper by Lynette Hirschman, Deborah Dahl, John Dowding, Francois Lang, Marcia Linebarger, Martha Palmer, Leslie Riley, and Rebecca Schiffman was presented at the Penn Linguistics Colloquium, Philadelphia, February, 1987.

The PUNDIT Natural Language Processing System¹

**Lynette Hirschman, Deborah Dahl,
John Dowding, Francois Lang,
Marcia Linebarger, Martha Palmer,
Leslie Riley, Rebecca Schiffman**

**Paoli Research Center,
UNISYS Defense Systems**

P.O. Box 517, Paoli, PA 19301

TELEPHONE NUMBER: (215) 648-7554

¹This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research, and in part by National Science Foundation contracts DCR-8202397 and DCR-85-02205, as well as by independent R&D funding from System Development Corporation, now part of Unisys Defense Systems. AP-PROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

1. Introduction

This paper describes the PUNDIT system for processing natural language inputs. PUNDIT, written in Prolog, is a highly modular system consisting of distinct syntactic, semantic, and pragmatic components. Each component draws on one or more sets of data, including a lexicon, a broad coverage grammar of English, semantic verb decompositions, rules mapping between syntactic and semantic constituents, and a domain model. PUNDIT has been developed cooperatively with the NYU PROTEUS system. These systems are funded by DARPA as part of the work in natural language understanding for the Strategic Computing Battle Management Program. Modularity, careful separation of declarative and procedural information and separation of domain specific and domain independent information all contribute to an overall system which is flexible, extensible and portable. See Figure 1 for the overall design of the system.

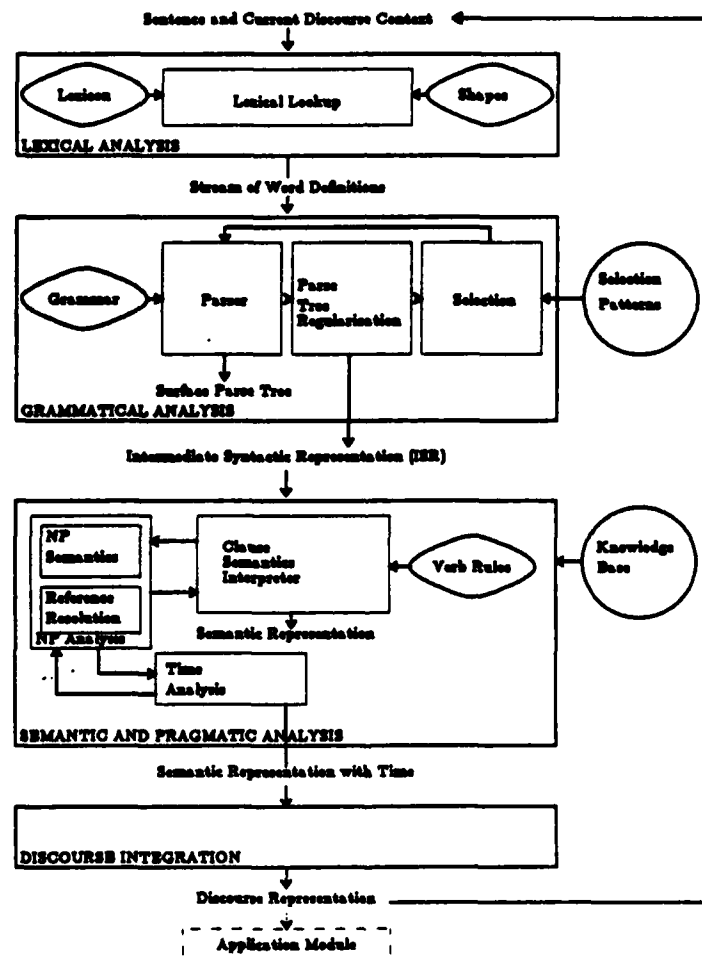


Figure 1 -- Organisation of PUNDIT

2. Current Application Domains

PUNDIT has been used as part of two different applications. The first is a text processing application and the second is as a natural language front end for a database query system. In developing these applications, the basic natural language analysis components have remained the same. Application modules have been developed which take the general representation of the meaning of a text which PUNDIT produces and create an output tailored for the specific application. For the text processing application, the application module produces a summary of the text. For the database query application, the application module produces a logic query. This flexibility of the basic PUNDIT system has two important consequences. New applications can be developed by designing a new application module, without changing the basic system, and in addition, extensions to the basic linguistic coverage which are motivated by the need to deal with problems in one application will also be available in the other applications. In addition, several applications can be combined in a single multipurpose system.

2.1. Maintenance Report Domain

In this application, Navy CASREP messages (CASualty REPorts), PUNDIT was applied to a Navy battle management domain focused on force readiness. In the CASREPS application, the system processes the remarks field of messages about failures in starting air compressors (*sacs*). The system creates a set of logical forms to represent the message content, and from this set it then generates a brief tabular summary of the major problems and findings (see Figure 2).

2.2. Naval Ship Database Domain

We have also built an interface for processing English queries to a database. For this application, the system accepts English queries and produces a logic form which is passed to a logic/database interface. The interface translates the query into QUEL and sends it to an INGRES database residing on a different machine; the interface then returns the results from

failure of sac. loss of air pressure resulted in slow gas turbine start. troubleshooting revealed normal sac lube oil pressure. erosion of impellor blade tip evident. received new sac.

Status of Sac:

Part: sac State: inoperative

Finding:

Part: air pressure State: lowered

Damage:

Part: blade tip State: eroded

Finding:

Agent: ship's force State: has new sac

Figure 2 -- Sample CASREP and Automatically Generated Summary

the query for display to the user. Only the back-end production of the logical form is unique to this application. The changes to the rest of the system consist of the modifications required for any new domain application, namely the domain-specific information in the lexicon and the domain model.

3. The Syntactic Component

In Pundit, syntactic analysis yields two syntactic descriptions of the sentence. One is a very detailed surface parse tree, and the other is a more regularised parse tree called the *Intermediate Syntactic Representation*. The surface structure parse tree is produced while performing a careful syntactic analysis and is used to construct the less detailed ISR, which is a more appropriate input to semantics and selection.

3.1. The Grammar

The grammar is written in the Restriction Grammar framework [Hirschman1982,Hirschman1985], which is a logic grammar with facilities for writing and maintaining large grammars. Restriction Grammar is a descendent of Sager's string grammar [Sager1981]. The grammar consists of a set of context-free BNF definitions (currently numbering approximately 80), augmented by restrictions (approximately 35). The restrictions enforce context-sensitive well-formedness constraints and, in some cases, apply optimisation strategies to prevent unnecessary structure-building. The grammar can either be *interpreted* or *translated*, or a mixture of both. It uses a top-down left-to-right parsing strategy, augmented by dynamic rule pruning for efficient parsing [Dowding1986]. In addition, it uses a meta-grammatical approach to generate definitions for a full range of co-ordinate conjunction structures [Hirschman1986]. The current grammar covers declarative sentences, questions, sentence fragments², sentence adjuncts, conjunction, relative clauses, complex complement structures, and a wide variety of nominal structures, including compound nouns, nominalised verbs and embedded clauses.

3.2. Intermediate Syntactic Representation

The syntax processor uses the rules of the grammar to produce a detailed surface structure parse for each sentence. This surface structure is converted into an "intermediate syntactic representation" (ISR) which regularises the syntactic parse. That is, it eliminates surface structure detail not required for semantic analysis. This regularisation is done by annotating each rule in the grammar with an expression in the *translation rule language*. These expressions tell the interpreter how to combine the ISR from the children of a node into the ISR of their parent.

An important part of regularisation involves mapping fragment structures onto canonical verb-subject-object patterns, with missing elements flagged. For example, the two fragment consists of a tensed verb + object as in *Replaced spindle motor*. Regularisation of this fragment, for example, maps the two syntactic structure into a verb+subject+object structure. The semantic and pragmatic components provide a semantic filler for the missing subject using general pragmatic principles and specific domain knowledge [Palmer1986].

4. Clause analysis

In order to produce a semantic representation of a clause, the verb is first decomposed into a semantic predicate representation appropriate for the domain. The arguments of the predicates constitute the SEMANTIC ROLES of the verb, which are similar to cases [Palmer1985]. For

²The rules for fragments enable the grammar to parse the "telegraphic" style characteristic of message traffic, such as *disk drive down*, and *has select lock*.

example, *fail* decomposes into *become inoperative*, with *patient* as its only semantic role.³ In this domain the semantic roles include: *actor*, *agent*, *experiencer*, *goal*, *instigator*, *instrument*, *location*, *patient*, *reference_pt*, *source* and *theme*. A dramatically different domain such as children's birthday parties would require much more redefinition of semantic roles and decomposition rules. However, the algorithm that fills the roles in the verb decompositions stays the same across different domains. The algorithm for the processing of nominalisations, which is described below, is equally domain independent.

Semantic roles can be filled either by a syntactic constituent or by reference resolution from defaults or contextual information. We have categorised the semantic roles into three classes, based on how they are filled [Palmer1988]. Semantic roles such as *theme*, *actor* and *patient* are syntactically OBLIGATORY, and must be filled by surface constituents. Semantic roles are categorized as semantically ESSENTIAL when they must be filled even if there is no syntactic constituent available.⁴ In this case they are filled pragmatically, making use of reference resolution, as explained below. The default categorization is NON-ESSENTIAL, which does not require that the role be filled.

The algorithm makes use of syntactic mapping rules that associate types of syntactic constituents with semantic roles. For example, the rule *instigator(I) <- subj(I)* indicates that the subject is a likely filler for the instigator role. These rules can be tailored to account for particular verb idiosyncrasies, but for the most part they are fairly general, and port readily from one domain to similar ones. There are also selection restriction rules associated with the semantic roles. These are more domain specific, since they are heavily dependent on the exact sense of the verb that is relevant.

Nominalisations are processed very similarly to clauses, but with a few crucial differences, both in linguistic information accessed and in the control of the algorithm. With respect to the linguistic information, the decomposition of a nominalisation is the same as its corresponding verb, but the mapping rules differ, since syntactically a nominalisation is a noun phrase. For example, where a likely filler for the *patient* of *fail*, is the syntactic subject, a likely filler for the *patient* of *failure* is an *of* prepositional phrase. The processing of nominalisations is discussed in detail in [Dahl1987].

5. Pragmatics Components

5.1. Reference Resolution

Reference resolution is called by the clause semantics interpreter when clause semantics is ready to instantiate a role with a specific referent. Reference resolution finds the referent of a noun phrase, and also finds other semantic relationships among the entities mentioned in texts. Some features of the reference resolution component include treatments of: pronouns (including zeroes) and *one*-anaphora using a syntax-based focusing algorithm [Dahl1986, Dahl1984], definite and indefinite noun phrases, as well as noun phrases without determiners, *implicit associates* [Dahl1986, Dahl1987.....], conjoined noun phrases, nominal references to events and situations first mentioned in clauses [Dahl1987], non-specific noun phrases [Palmer1988], and referents not mentioned explicitly [Palmer1988].

Processing for pronominals; that is, pronouns, *one*-anaphora, and zeroes, selects the referent from a list of entities in focus. The referents for full noun phrases and implicit referents are selected from all of the entities that have been mentioned in the discourse. If there is no previously mentioned referent for a definite noun phrase, or for one without a determiner, the

³There are domain specific criteria for selecting a range of semantic roles. The criteria which we have used are described in [Schiffman1986].

⁴We are in the process of defining criteria for categorising a role as ESSENTIAL. It is clearly very domain dependent, and relies heavily on what can be assumed from the context.

system associates the entity with one in focus. This processing is described in more detail in [Dahl1986]. Indefinite noun phrases are assumed to introduce new referents. While this is not strictly correct [Dahl1987.....], it seems to be sufficient for this domain. After a referent is found, control returns to the clause semantics interpreter.

5.2. Time Analysis

PUNDIT's temporal analyser determines the temporal properties of all the situations mentioned in each input sentence. It recognises three types of situations: states, processes and events, and derives distinct types of representations of their temporal structures indicating what time intervals the situations hold over. It also identifies the temporal location of a situation with respect to the time at which the text was produced and any other explicit times mentioned in the sentence associated either with other situations (*The pump failed when the engine seized*) or with clock/calendar times (*The pump failed at 0800 hours*).

To determine the temporal structure of a predication, PUNDIT needs two kinds of semantic input: the decomposition produced by the semantic analyser and the surface tense and aspect markings (tense, taxis⁵ and grammatical aspect.)⁶ Specific components of the decomposition represent the inherent temporal properties (aspect) of the predicate. The way in which these aspectual components interact with surface tense and aspect markings is semantically regular. The aspectual class of a predicate depends on the domain specific semantics. The time of the text production, used in interpreting the temporal location of a situation, is contextual or pragmatic information. Thus the temporal analyser uses semantic and domain specific input in a semantically regular way to derive information relative to the current context. The modularity of the PUNDIT system is a great advantage to the temporal analyser as the following sample analysis illustrates.

6. Portability

6.1. Automatic Generation of Selectional Restrictions

Since the ISR regularises syntactic patterns into a canonical form, we were able to write a program to analyse the ISR and examine the syntactic patterns as they are being generated. This program is called after the BNF grammar has assembled an NP or a complete sentence and constructed its ISR.

The program presents to the user a syntactic pattern from the ISR and asks about the validity of that pattern. The user can then accept that pattern, allowing parsing to proceed, or reject it (if the parse being constructed is wrong), causing the parser to backtrack.

For example, the correct parse for "Retained oil sample for future analysis" has an elided subject, "retained" as the verb, and "oil sample" as its direct object: "[Somebody] retained [the] oil sample for future analysis". An incorrect parse, however, might analyse the sentence as a serocopula fragment, with "retained oil sample" as subject: ("[The] retained oil sample [was] for future analysis"). In this case, when the program asks the user about the adjective/noun pattern "retained sample", the user would reject that pattern and fail the parse being constructed.

This interactive program greatly reduces the number of incorrect parses generated as well as the number of BNF grammar rules tried, because the user can immediately fail any (partial) parse containing an invalid syntactic pattern. More importantly, however, the information about good and bad syntactic patterns can be stored and classified as good or bad, so that no pattern is ever presented more than once. The system can thus interactively acquire domain-specific semantic information, and effectively bootstrap into a completely new domain.

⁵ *Tense* (Jakobson, 1957) refers to the semantic effect of the presence or absence of the perfect auxiliary.

⁶ Grammatical aspect is signalled by the presence or absence of the progressive suffix -ing.

The mechanism as currently implemented deals only with patterns containing lexical items. However, we are currently investigating methods to generalise the program by using information in the domain isa hierarchy to construct semantic class patterns: In our domain model, OIL is a type of FLUID. If the user accepts the noun/noun pattern "oil sample", he/she would then be asked for which X such that X isa FLUID is "X sample" a good noun/noun pattern.

6.1.1. The Development Environment

We have developed a set of tools that makes writing, modifying and testing code easier and more efficient, as well as making it easier to port Pundit to a new domain. These include: switches (to tailor the operation of PUNDIT differently depending on what one is working on), the Prolog Structure Editor (a general structure editor that makes it easy to traverse and edit Prolog terms), a lexical entry procedure (that assists in the arduous task of creating new lexical entries), and a semantic rule editor (that assists in the creation and consistency of semantic rules).

6.2. Conclusion

PUNDIT is an extremely modular natural language processing system. This modularity allows clean coordination of a large number of components performing separate tasks, such as the handling of implicit information, including syntactically elided and semantically inferred. This modularity also contributes to the extensibility and flexibility of the system both in terms of new applications and new domains of discourse.

REFERENCES

[Dahl1984]

Deborah A. Dahl, The Structure and Function of One-Anaphora in English, PhD Thesis; (also published by Indiana University Linguistics Club, 1985), University of Minnesota, 1984.

[Dahl1986]

Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[Dahl1987]

Deborah A. Dahl, Determiners, Entities, and Contexts, To be presented at Tinlap-3, Las Cruces, New Mexico, January 7-9, 1987.

[Dahl1987]

Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Schiffman, Nominalisations in PUNDIT, submitted to the 25 Annual Meeting of the Association for Computational Linguistics, Stanford University, Stanford, California, July 1987.

[Dowding1986]

John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, Logic-Based Systems Technical Memo No. 43, Paoli Research Center, System Development Corporation, 1986.

[Hirschman1982]

L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.

[Hirschman1985]

L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Hirschman1986]

L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming*, 1986.

[Palmer1985]

Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[Palmer1986]

Martha S. Palmer, Deborah A. Dahl, Rebecca J. Schiffman , Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, to be presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[Sager1981]

N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[Schiffman1986]

Rebecca J. Schiffman, Designing Lexical Entries for a Limited Domain, Logic-Based Systems Technical Memo No. 42, Paoli Research Center, System Development Corporation, April, 1986.

APPENDIX B

Recovering Implicit Information

This paper by Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau} Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, was presented at the 24th Annual Meeting of the Association for Computational Linguistics, New York, June, 1986. It describes the communication between the syntactic, semantic, and pragmatic modules that is necessary for making implicit information explicit.

RECOVERING IMPLICIT INFORMATION

Martha S. Palmer, Deborah A. Dahl, Rebecca J. Schiffman, Lynette Hirschman,
Marcia Linebarger, and John Dowding
Research and Development Division
* SDC - A Burroughs Company
P.O. Box 517
Paoli, PA 19301 USA

ABSTRACT

This paper describes the SDC PUNDIT, (Prolog UNDERstands Integrated Text), system for processing natural language messages.¹ PUNDIT, written in Prolog, is a highly modular system consisting of distinct syntactic, semantic and pragmatics components. Each component draws on one or more sets of data, including a lexicon, a broad-coverage grammar of English, semantic verb decompositions, rules mapping between syntactic and semantic constituents, and a domain model.

This paper discusses the communication between the syntactic, semantic and pragmatic modules that is necessary for making implicit linguistic information explicit. The key is letting syntax and semantics recognize missing linguistic entities as implicit entities, so that they can be labelled as such, and reference resolution can be directed to find specific referents for the entities. In this way the task of making implicit linguistic information explicit becomes a subset of the tasks performed by reference resolution. The success of this approach is dependent on marking missing syntactic constituents as elided and missing semantic roles as ESSENTIAL so that reference resolution can know when to look for referents.

¹ This work is supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research. APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

1. Introduction

This paper describes the SDC PUNDIT² system for processing natural language messages. PUNDIT, written in Prolog, is a highly modular system consisting of distinct syntactic, semantic and pragmatics components. Each component draws on one or more sets of data, including a lexicon, a broad-coverage grammar of English, semantic verb decompositions, rules mapping between syntactic and semantic constituents, and a domain model. PUNDIT has been developed cooperatively with the NYU PROTEUS system (Prototype Text Understanding System). These systems are funded by DARPA as part of the work in natural language understanding for the Strategic Computing Battle Management Program. The PROTEUS/PUNDIT system will map Navy CASREP's (equipment casualty reports) into a database, which is accessed by an expert system to determine overall fleet readiness. PUNDIT has also been applied to the domain of computer maintenance reports, which is discussed here.

The paper focuses on the interaction between the syntactic, semantic and pragmatic modules that is required for the task of making implicit information explicit. We have isolated two types of implicit entities: syntactic entities which are missing syntactic constituents, and semantic entities which are unfilled semantic roles. Some missing entities are optional, and can be ignored. Syntax and semantics have to recognize the OBLIGATORY missing entities and then mark them so that reference resolution knows to find specific referents for those entities, thus making the implicit information explicit. Reference resolution uses two different methods for filling the different types of entities which are also used for general noun phrase reference problems. Implicit syntactic entities, ELIDED CONSTITUENTS, are treated like pronouns, and implicit semantic entities, ESSENTIAL ROLES are treated like definite noun phrases. The pragmatic module as currently implemented consists mainly of a reference resolution component, which is sufficient for the pragmatic issues described in this paper. We are in the process of adding a time module to handle time issues that have arisen during the analysis of the Navy CASREPS.

2. The Syntactic Component

The syntactic component has three parts: the grammar, a parsing mechanism to execute the grammar, and a lexicon. The grammar consists of context-free BNF definitions (currently numbering approximately 80) and associated restrictions (approximately 35). The restrictions enforce context-sensitive well-formedness constraints and, in some cases, apply optimization strategies to prevent unnecessary structure-building. Each of these three parts is described further below.

² Prolog UNDERstands Integrated Text

2.1. Grammar Coverage

The grammar covers declarative sentences, questions, and sentence fragments. The rules for fragments enable the grammar to parse the "telegraphic" style characteristic of message traffic, such as *disk drive down*, and *has select lock*. The present grammar parses sentence adjuncts, conjunction, relative clauses, complex complement structures, and a wide variety of nominal structures, including compound nouns, nominalized verbs and embedded clauses.

The syntax produces a detailed surface structure parse of each sentence (where "sentence" is understood to mean the string of words occurring between two periods, whether a full sentence or a fragment). This surface structure is converted into an "intermediate representation" which regularizes the syntactic parse. That is, it eliminates surface structure detail not required for the semantic tasks of enforcing selectional restrictions and developing the final representation of the information content of the sentence. An important part of regularization involves mapping fragment structures onto canonical verb-subject-object patterns, with missing elements flagged. For example, the *two* fragment consists of a **tensed verb + object** as in *Replaced spindle motor*. Regularization of this fragment, for example, maps the *two* syntactic structure into a verb+subject+object structure:

verb(replace),subject(X),object(Y)

As shown here, **verb** becomes instantiated with the surface verb, e.g., *replace* while the arguments of the **subject** and **object** terms are variables. The semantic information derived from the noun phrase object *spindle motor* becomes associated with **Y**. The absence of a surface subject constituent results in a lack of semantic information pertaining to **X**. This lack causes the semantic and pragmatic components to provide a semantic filler for the missing subject using general pragmatic principles and specific domain knowledge.

2.2. Parsing

The grammar uses the Restriction Grammar parsing framework [Hirschman1982,Hirschman1985], which is a logic grammar with facilities for writing and maintaining large grammars. Restriction Grammar is a descendent of Sager's string grammar [Sager1981]. It uses a top-down left-to-right parsing strategy, augmented by dynamic rule pruning for efficient parsing [Dowding1986]. In addition, it uses a meta-grammatical approach to generate definitions for a full range of co-ordinate conjunction structures [Hirschman1986].

2.3. Lexical Processing

The lexicon contains several thousand entries related to the particular sub-domain of equipment maintenance. It is a modified version of the LSP lexicon with words classified as to part of speech and subcategorized in limited ways (e.g., verbs are subcategorized for their complement types). It also handles

multi-word idioms, dates, times and part numbers. The lexicon can be expanded by means of an interactive lexical entry program.

The lexical processor reduces morphological variants to a single root form which is stored with each entry. For example, the form *has* is transformed to the root form *have* in *Has select lock*. In addition, this facility is useful in handling abbreviations: the term *swp* is regularized to the multi-word expression *waiting for part*. This expression in turn is regularized to the root form *wait for part* which takes as a direct object a particular part or part number, as in *is swp 2155-6147*.

Multi-word expressions, which are typical of jargon in specialized domains, are handled as single lexical items. This includes expressions such as *disk drive* or *select lock*, whose meaning within a particular domain is often not readily computed from its component parts. Handling such frozen expressions as "idioms" reduces parse times and number of ambiguities.

Another feature of the lexical processing is the ease with which special forms (such as part numbers or dates) can be handled. A special "forms grammar", written as a definite clause grammar[Pereira1980] can parse part numbers, as in *awaiting part 2155-6147*, or complex date and time expressions, as in *disk drive up at 11/17-1236*. During parsing, the forms grammar performs a well-formedness check on these expressions and assigns them their appropriate lexical category.

3. Semantics

There are two separate components that perform semantic analysis, NOUN PHRASE SEMANTICS and CLAUSE SEMANTICS. They are each called after parsing the relevant syntactic structure to test semantic well-formedness while producing partial semantic representations. Clause semantics is based on Inference Driven Semantic Analysis [Palmer1985] which decomposes verbs into component meanings and fills their semantic roles with syntactic constituents. A KNOWLEDGE BASE, the formalization of each domain into logical terms, SEMANTIC PREDICATES, is essential for the effective application of Inference Driven Semantic Analysis, and for the final production of a text representation. The result of the semantic analysis is a set of PARTIALLY instantiated semantic predicates which is similar to a frame representation. To produce this representation, the semantic components share access to a knowledge base, the DOMAIN MODEL, that contains generic descriptions of the domain elements corresponding to the lexical entries. The model includes a detailed representation of the types of assemblies that these elements can occur in. The semantic components are designed to work independently of the particular model, and rely on an interface to ensure a well-defined interaction with the domain model. The domain model, noun phrase semantics and clause semantics are all explained in more detail in the following three subsections.

3.1. Domain Model

The domain currently being modelled by SDC is the Maintenance Report domain. The texts being analyzed are actual maintenance reports as they are called into the Burroughs Telephone Tracking System by the field engineers and typed in by the telephone operator. These reports give information about the customer who has the problem, specific symptoms of the problem, any actions take by the field engineer to try and correct the problem, and success or failure of such actions. The goal of the text analysis is to automatically generate a data base of maintenance information that can be used to correlate customers to problems, problem types to machines, and so on.

The first step in building a domain model for maintenance reports is to build a semantic net-like representation of the type of machine involved. The machine in the example text given below is the B4700. The possible parts of a B4700 and the associated properties of these parts can be represented by an *isa* hierarchy and a *haspart* hierarchy. These hierarchies are built using four basic predicates: *system*, *isa*, *hasprop*, *haspart*. For example the system itself is indicated by *system(b4700)*. The *isa* predicate associates TYPES with components, such as *isa(spindle^motor,motor)*. Properties are associated with components using the *hasprop* relationship, are are inherited by anything of the same type. The main components of the system: *cpu*, *power_supply*, *disk*, *printer*, *peripherals*, etc., are indicated by *haspart* relations, such as *haspart(b4700,cpu)*, *haspart(b4700,power_supply)*, *haspart(b4700,disk)*, etc. These parts are themselves divided into subparts which are also indicated by *haspart* relations, such as *haspart(power_supply, converter)*.

This method of representation results in a general description of a computer system. Specific machines represent INSTANCES of this general representation. When a particular report is being processed, *id* relations are created by noun phrase semantics to associate the specific computer parts being mentioned with the part descriptions from the general machine representation. So a particular B4700 would be indicated by predicates such as these: *id(b4700,system1)*, *id(cpu,cpu1)*, *id(power_supply,power_supply1)*, etc.

3.2. Noun phrase semantics

Noun phrase semantics is called by the parser during the parse of a sentence, after each noun phrase has been parsed. It relies heavily on the domain model for both determining semantic well-formedness and building partial semantic representations of the noun phrases. For example, in the sentence, *field engineer replaced disk drive at 11/2/0800*, the phrase *disk drive at 11/2/0800* is a syntactically acceptable noun phrase, (as in *participants at the meeting*). However, it is not semantically acceptable in that *at 11/20/800* is intended to designate the time of the replacement, not a

property of the disk drive. Noun phrase semantics will inform the parser that the noun phrase is not semantically acceptable, and the parser can then look for another parse. In order for this capability to be fully utilized, however, an extensive set of domain-specific rules about semantic acceptability is required. At present we have only the minimal set used for the development of the basic mechanism. For example, in the case described here, *at 11/2/0800* is excluded as a modifier for *disk drive* by a rule that permits only the name of a location as the object of *at* in a prepositional phrase modifying a noun phrase.

The second function of noun phrase semantics is to create a semantic representation of the noun phrase, which will later be operated on by reference resolution. For example, the semantics for *the bad disk drive* would be represented by the following Prolog clauses.

```
[id(disk^drive,X),
  bad(X),
  def(X),      that is, X was referred to with a full, definite noun phrase,
  full_npe(X)] rather than a pronoun or indefinite noun phrase.
```

3.3. Clause semantics

In order to produce the correct predicates and the correct instantiations, the verb is first decomposed into a semantic predicate representation appropriate for the domain. The arguments to the predicates constitute the SEMANTIC ROLES of the verb, which are similar to cases. There are domain specific criteria for selecting a range of semantic roles. In this domain the semantic roles include: **agent, instrument, theme, object1, object2, symptom** and **mod**. Semantic roles can be filled either by a syntactic constituent supplied by a mapping rule or by reference resolution, requiring close cooperation between semantics and reference resolution. Certain semantic roles are categorized as **ESSENTIAL**, so that pragmatics knows that they need to be filled if there is no syntactic constituent available. The default categorization is **NON-ESSENTIAL**, which does not require that the role be filled. Other semantic roles are categorized as **NON-SPECIFIC** or **SPECIFIC** depending on whether or not the verb requires a specific referent for that semantic role (see Section 4). The example given in Section 5 illustrates the use of both a non-specific semantic role and an essential semantic role. This section explains the decompositions of the verbs relevant to the example, and identifies the important semantic roles.

The decomposition of *have* is very domain specific.

```
have(time(Per)) <-
  symptom(object1(O1),symptom(S),time(Per))
```

It indicates that a particular **symptom** is associated with a particular **object**, as in "the disk drive has select lock." The **object1** semantic role

would be filled by the disk drive, the subject of the clause, and the **symptom** semantic role would be filled by *select lock*, the object of the clause. The **time(Per)** is always passed around, and is occasionally filled by a time adjunct, as in *the disk drive had select lock at 0800*.

In addition to the mapping rules that are used to associate syntactic constituents with semantic roles, there are selection restrictions associated with each semantic role. The selection restrictions for *have* test whether or not the filler of the **object1** role is allowed to have the type of symptom that fills the **symptom** role. For example, only disk drives have select locks.

Mapping Rules

The decomposition of *replace* is also a very domain specific decomposition that indicates that an **agent** can use an instrument to exchange two objects.

```
replace(time(Per)) <-
  cause(agent(A),
    use(instrument(I),
      exchange(object1(O1),object2(O2),time(Per))))
```

The following mapping rule specifies that the **agent** can be indicated by the subject of the clause.

```
agent(A) <- subject(A) / X
```

The mapping rules make use of intuitions about syntactic cues for indicating semantic roles first embodied in the notion of case [Fillmore1968,Palmer1981]. Some of these cues are quite general, while other cues are very verb-specific. The mapping rules can take advantage of generalities like "SUBJECT to AGENT" syntactic cues while still preserving context sensitivities. This is accomplished by making the application of the mapping rules "situation-specific" through the use of PREDICATE ENVIRONMENTS. The previous rule is quite general and can be applied to every **agent** semantic role in this domain. This is indicated by the X on the right hand side of the "/" which refers to the predicate environment of the **agent**, i.e., anything. Other rules, such as "WITH-PP to OBJECT2," are much less general, and can only apply under a set of specific circumstances. The predicate environments for an **object1** and **object2** are specified more explicitly. An **object1** can be the object of the sentence if it is contained in the semantic decomposition of a verb that includes an **agent** and belongs to the *repair* class of verbs. An **object2** can be indicated by a *with* prepositional phrase if it is contained in the semantic decomposition of a *replace* verb:

```
object1(Part1) <- obj(Part1)/ cause(agent(A),Repair_event)
```

```
object2(Part2) <-
  pp(with,Part2) /
```

cause(agent(A),use(I,exchange(object1(O1),object2(Part2),T)))

Selection Restrictions

The selection restriction on an **agent** is that it must be a field engineer, and an **instrument** must be a tool. The selection restrictions on the two objects are more complicated, since they must be machine parts, have the same type, and yet also be distinct objects. In addition, the first object must already be associated with something else in a **haspart** relationship, in other words it must already be included in an existing assembly. The opposite must be true of the second object: it must not already be included in an assembly, so it must not be associated with anything else in a **haspart** relationship.

There is also a pragmatic restriction associated with both objects that has not been associated with any of the semantic roles mentioned previously. Both **object1** and **object2** are essential semantic roles. Whether or not they are mentioned explicitly in the sentence, they must be filled, preferably by an entity that has already been mentioned, but if not that, then entities will be created to fill them [Palmer1983]. This is accomplished by making an explicit call to reference resolution to find referents for essential semantic roles, in the same way that reference resolution is called to find the referent of a noun phrase. This is not done for non-essential roles, such as the **agent** and the **instrument** in the same verb decomposition. If they are not mentioned they are simply left unfilled. The **instrument** is rarely mentioned, and the **agent** could easily be left out, as in *The disk drive was replaced at 0800.*³ In other domains, the **agent** might be classified as obligatory, and then it would have to be filled in.

There is another semantic role that has an important pragmatic restriction on it in this example, the **object2** semantic role in *wait^{for}part (awp)*.

idiomVerb(wait^{for}part,time(Per)) <-
ordered(object1(O1),object2(O2),time(Per))

The semantics of *wait^{for}part* indicates that a particular type of part has been ordered, and is expected to arrive. But it is not a specific entity that might have already been mentioned. It is a more abstract object, which is indicated by restricting it to being non-specific. This tells reference resolution that although a syntactic constituent, preferably the object, can and should fill this semantic role, and must be of type **machine-part**, that reference resolution should not try to find a specific referent for it (see Section 4).

The last verb representation that is needed for the example is the representation of *be*.

be(time(Per)) <-

³Note that an elided subject is handled quite differently, as in *replaced disk drive*. Then the missing subject is

`attribute(theme(T),mod(M),time(Per))`

In this domain *be* is used to associate predicate adjectives or nominals with an object, as in *disk drive is up* or *spindle motor is bad*. The representation merely indicates that a modifier is associated with an theme in an attribute relationship. Noun phrase semantics will eventually produce the same representation for *the bad spindle motor*, although it does not yet.

4. Reference Resolution

Reference resolution is the component which keeps track of references to entities in the discourse. It creates labels for entities when they are first directly referred to, or when their existence is implied by the text, and recognizes subsequent references to them. Reference resolution is called from clause semantics when clause semantics is ready to instantiate a semantic role. It is also called from pragmatic restrictions when they specify a referent whose existence is entailed by the meaning of a verb.

The system currently covers many cases of singular and plural noun phrases, pronouns, *one-* anaphora, nominalizations, and non-specific noun phrases; reference resolution also handles adjectives, prepositional phrases and possessive pronouns modifying noun phrases. Noun phrases with and without determiners are accepted. Dates, part numbers, and proper names are handled as special cases. Not yet handled are compound nouns, quantified noun phrases, conjoined noun phrases, relative clauses, and possessive nouns.

The general reference resolution mechanism is described in detail in [Dahl1986]. In this paper the focus will be on the interaction between reference resolution and clause semantics. The next two sections will discuss how reference resolution is affected by the different types of semantic roles.

4.1. Obligatory Constituents and Essential Semantic Roles

A slot for a syntactically obligatory constituent such as the subject appears in the intermediate representation whether or not a subject is overtly present in the sentence. It is possible to have such a slot because the absence of a subject is a syntactic fact, and is recognized by the parser. Clause semantics calls reference resolution for such an implicit constituent in the same way that it calls reference resolution for explicit constituents. Reference resolution treats elided noun phrases exactly as it treats pronouns, that is by instantiating them to the first member of a list of potential pronominal referents, the **FocusList**.

assumed to fill the agent role, and an appropriate referent is found by reference resolution.

The general treatment of pronouns resembles that of [Sidner1979], although there are some important differences, which are discussed in detail in [Dahl1986]. The hypothesis that elided noun phrases can be treated in much the same way as pronouns is consistent with previous claims by [Gundel1980], and [Kameyama1985], that in languages which regularly allow zero-np's, the zero corresponds to the focus. If these claims are correct, it is not surprising that in a sublanguage that allows zero-np's, the zero should also correspond to the focus.

After control returns to clause semantics from reference resolution, semantics checks the selectional restrictions for that referent in that semantic role of that verb. If the selectional restrictions fail, backtracking into reference resolution occurs, and the next candidate on the FocusList is instantiated as the referent. This procedure continues until a referent satisfying the selectional restrictions is found. For example, in *Disk drive is down. Has select lock*, the system instantiates the disk drive, which at this point is the first member of the FocusList, as the object1 of have:

```
[event39]
have(time(time1))
  symptom(object1([drive10]),
    symptom([lock17]),
    time(time1))
```

Essential roles might also not be expressed in the sentence, but their absence cannot be recognized by the parser, since they can be expressed by syntactically optional constituents. For example, in *the field engineer replaced the motor.*, the new replacement motor is not mentioned, although in this domain it is classified as semantically essential. With verbs like *replace*, the type of the replacement, *motor*, in this case, is known because it has to be the same type as the replaced object. Reference resolution for these roles is called by pragmatic rules which apply when there is no overt syntactic constituent to fill a semantic role. Reference resolution treats these referents as if they were full noun phrases without determiners. That is, it searches through the context for a previously mentioned entity of the appropriate type, and if it doesn't find one, it creates a new discourse entity. The motivation for treating these as full noun phrases is simply that there is no reason to expect them to be in focus, as there is for elided noun phrases.

4.2. Noun Phrases in Non-Specific Contexts

Indefinite noun phrases in contexts like *the field engineer ordered a disk drive* are generally associated with two readings. In the specific reading the disk drive ordered is a particular disk drive, say, the one sitting on a certain shelf in the warehouse. In the non-specific reading, which is more likely in this

sentence, no particular disk drive is meant; any disk drive of the appropriate type will do. Handling noun phrases in these contexts requires careful integration of the interaction between semantics and reference resolution, because semantics knows about the verbs that create non-specific contexts, and reference resolution knows what to do with noun phrases in these contexts. For these verbs a constraint is associated with the semantics rule for the semantic role **object2** which states that the filler for the **object2** must be non-specific.⁴ This constraint is passed to reference resolution, which represents a non-specific noun phrase as having a variable in the place of the pointer, for example, *id(motor,X)*.

Non-specific semantic roles can be illustrated using the **object2** semantic role in *wait`for`part (awp)*. The part that is being *awaited* is non-specific, i.e., can be any part of the appropriate type. This tells reference resolution not to find a specific referent, so the referent argument of the *id* relationship is left as an uninstantiated variable. The analysis of *fe is awp spindle motor* would fill the **object1** semantic role with *fe1* from *id(fe,fe1)*, and the **object2** semantic role with *X* from *id(spindle^motor,X)*, as in *ordered(object1(fe1),object2(X))*. If the spindle motor is referred to later on in a relationship where it must become specific, then reference resolution can instantiate the variable with an appropriate referent such as *spindle^motor3* (See Section 5.6).

5. Sample Text: A sentence-by-sentence analysis

The sample text given below is a slightly emended version of a maintenance report. The parenthetical phrases have been inserted. The following summary of an interactive session with PUNDIT illustrates the mechanisms by which the syntactic, semantic and pragmatic components interact to produce a representation of the text.

1. disk drive (was) down (at) 11/16-2305.
2. (has) select lock.
3. spindle motor is bad.
4. (is) awp spindle motor.
5. (disk drive was) up (at) 11/17-1236.
6. replaced spindle motor.

5.1. Sentence 1: Disk drive was down at 11/16-2305.

As explained in Section 3.2 above, the noun phrase *disk drive* leads to the creation of an *id of the form*: *id(disk^drive,[drive1])* Because dates and names generally refer to unique entities rather than to exemplars of a general type, their *ids* do not contain a type argument: *date([11/16-*

⁴ The specific reading is not available at present, since it is considered to be unlikely to occur in this domain.

1100]),name([paoli])).

The interpretation of the first sentence of the report depends on the semantic rules for the predicate *be*. The rules for this predicate specify three semantic roles, an **theme** to whom or which is attributed a **modifier**, and the **time**. After a mapping rule in the semantic component of the system instantiates the **theme** semantic role with the sentence subject, *disk drive*, the reference resolution component attempts to identify this referent. Because *disk drive* is in the first sentence of the discourse, no prior references to this entity can be found. Further, this entity is not presupposed by any prior linguistic expressions. However, in the maintenance domain, when a disk drive is referred to it can be assumed to be part of a B3700 computer system. As the system tries to resolve the reference of the noun phrase *disk drive* by looking for previously mentioned disk drives, it finds that the mention of a disk drive presupposes the existence of a system. Since no system has been referred to, a pointer to a system is created at the same time that a pointer to the disk drive is created.

Both entities are now available for future reference. In like fashion, the propositional content of a complete sentence is also made available for future reference. The entities corresponding to propositions are given event labels; thus *event1* is the pointer to the first proposition. The newly created disk drive, system and event entities now appear in the discourse information in the form of a list along with the date.

```
id(event,[event1])
id(disk^drive,[drive1])
date([11/16-2305])
id(system,[system1])
```

Note however, that only those entities which have been explicitly mentioned appear in the **FocusList**:

FocusList: [[event1],[drive1],[11/16-2305]]

The propositional entity appears at the head of the focus list followed by the entities mentioned in full noun phrases.⁵

In addition to the representation of the new event, the pragmatic information about the developing discourse now includes information about part-whole relationships, namely that *drive1* is a part which is contained in *system1*.

Part-Whole Relationships:

```
haspart([system1],[drive1])
```

The complete representation of *event1*, appearing in the event list in the form shown below, indicates that at the time given in the prepositional phrase *at 11/16-2305* there is a state of affairs denoted as *event1* in which a particular

⁵ The order in which full noun phrase mentions are added to the **FocusList** depends on their syntactic function and linear order. For full noun phrases, direct object mentions precede subject mentions followed by all other mentions given in the order in which they occur in the sentence. See [Dahl1986], for details.

disk drive, i.e., *drive1*, can be described as *down*.

```
[event1]
  be(time([11/18-2305]))
  attribute(theme([drive1]),
    mod(down),time([11/18-2305]))
```

5.2. Sentence 2: Has select lock.

The second sentence of the input text is a sentence fragment and is recognized as such by the parser. Currently, the only type of fragment which can be parsed can have a missing subject but must have a complete verb phrase. Before semantic analysis, the output of the parse contains, among other things, the following constituent list: [subj([X]),obj([Y])]. That is, the syntactic component represents the arguments of the verb as variables. The fact that there was no overt subject can be recognized by the absence of semantic information associated with *X*, as discussed in Section 3.2. The semantics for the maintenance domain sublanguage specifies that the thematic role instantiated by the direct object of the verb *to have* must be a symptom of the entity referred to by the subject. Reference resolution treats an empty subject much like a pronominal reference, that is, it proposes the first element in the *FocusList* as a possible referent. The first proposed referent, *event1* is rejected by the semantic selectional constraints associated with the verb *have*, which, for this domain, require the role mapped onto the subject to be classified as a machine part and the role mapped onto the direct object to be classified as a symptom. Since the next item in the *FocusList*, *drive1*, is a machine part, it passes the selectional constraint and becomes matched with the empty subject of *has select lock*. Since no select lock has been mentioned previously, the system creates one. For the sentence as a whole then, two entities are newly created: the select lock ([lock1]) and the new propositional event ([event2]): *id(event,[event2]), id(select^lock,[lock1])*. The following representation is added to the event list, and the *FocusList* and *Ids* are updated appropriately.⁶

```
[event2]
  have(time(time1))
  symptom(object1([drive1]),
    symptom([lock1],time(time1)))
```

5.3. Sentence 3: Motor is bad.

In the third sentence of the sample text, a new entity is mentioned, *motor*. Like *disk drive* from sentence 1, *motor* is a dependent entity. However, the entity it presupposes is not a computer system, but rather, a disk drive. The

⁶ This version only deals with explicit mentions of time, so for this sentence the time argument is filled in with a *gensym* that stands for an unknown time period. The current version of PUNDIT uses verb tense and verb semantics

newly mentioned motor becomes associated with the previously mentioned disk drive.

After processing this sentence, the new entity **motor3** is added to the **FocusList** along with the new proposition **event3**. Now the discourse information about part-whole relationships contains information about both dependent entities, namely that **motor1** is a part of **drive1** and that **drive1** is a part of **system1**.

```
haspart([drive1],[motor1])
haspart([system1],[drive1])
```

5.4. Sentence 4: is awp spindle motor.

Awp is an abbreviation for an idiom specific to this domain, *awaiting part*. It has two semantic roles, one of which maps to the sentence subject. The second maps to the direct object, which in this case is the non-specific spindle motor as explained in Section 4.2. The selectional restriction that the first semantic role of *awp* be an engineer causes the reference resolution component to create a new engineer entity because no engineer has been mentioned previously. After processing this sentence, the list of available entities has been incremented by three:

```
id(event,[event4])
id(part,[_2317])
id(field^engineer,[engineer1])
```

The new event is represented as follows:

```
[event4]
idiomVerb(wait^for^part,time(time2))
wait(object1([engineer1]),
      object2([_2317]),time(time2))
```

5.5. Sentence 5: disk drive was up at 11/17-0800 In the emended version of sentence 5 the *disk drive* is presumed to be the same drive referred to previously, that is, **drive1**. The semantic analysis of sentence 5 is very similar to that of sentence 1. As shown in the following event representation, the predicate expressed by the modifier *up* is attributed to the theme **drive1** at the specified time.

```
[event5]
be(time([11/17-1236]))
attribute(theme([drive1]),
          mod(up),time([11/17-1236]))
```

to derive implicit time arguments.

5.6. Sentence 6: Replaced motor.

The sixth sentence is another fragment consisting of a verb phrase with no subject. As before, reference resolution tries to find a referent in the current **FocusList** which is a semantically acceptable subject given the thematic structure of the verb and the domain-specific selectional restrictions associated with them. The thematic structure of the verb *replace* includes an **agent** role to be mapped onto the sentence subject. The only **agent** in the maintenance domain is a field engineer. Reference resolution finds the previously mentioned engineer created for *awp spindle motor*, **[engineer1]**. It does not find an **instrument**, and since this is not an essential role, this is not a problem. It simply fills it in with another gensym that stands for an unknown filler, **unknown1**.

When looking for the referent of a spindle motor to fill the **object1** role, it first finds the non-specific spindle motor also mentioned in the *awp spindle motor* sentence, and a specific referent is found for it. However, this fails the selection restrictions, since although it is a machine part, it is not already associated with an assembly, so backtracking occurs and the referent instantiation is undone. The next spindle motor on the **FocusList** is the one from *spindle motor is bad*, **([motor1])**. This does pass the selection restrictions since it participates in a **haspart** relationship.

The last semantic role to be filled is the **object2** role. Now there is a restriction saying this role must be filled by a machine part of the same type as **object1**, which is not already included in an assembly, viz., the non-specific spindle motor. Reference resolution finds a new referent for it, which automatically instantiates the variable in the **id** term as well. The representation can be decomposed further into the two semantic predicates **missing** and **included**, which indicate the current status of the parts with respect to any existing assemblies. The **haspart** relationships are updated, with the old **haspart** relationship for **[motor1]** being removed, and a new **haspart** relationship for **[motor3]** being added. The final representation of the text will be passed through a filter so that it can be suitably modified for inclusion in a database.

```
[event8]
  replace(time(time3))
  cause(agent([engineer1]),
    use(instrument(unknown1),
      exchange(object1([motor1]),
        object2([motor2]),
        time(time3))))
  included(object2([motor2]),time(time3))
  missing(object1([motor1]),time(time3))
```

Part-Whole Relationships:

```
haspart([drive1],[motor3])
haspart([system1],[drive1])
```

6. Conclusion

This paper has discussed the communication between syntactic, semantic and pragmatic modules that is necessary for making implicit linguistic information explicit. The key is letting syntax and semantics recognize missing linguistic entities as implicit entities, so that they can be marked as such, and reference resolution can be directed to find specific referents for the entities. Implicit entities may be either empty syntactic constituents in sentence fragments or unfilled semantic roles associated with domain-specific verb decompositions. In this way the task of making implicit information explicit becomes a subset of the tasks performed by reference resolution. The success of this approach is dependent on the use of syntactic and semantic categorizations such as ELLIDED and ESSENTIAL which are meaningful to reference resolution, and which can guide reference resolution's decision making process.

ACKNOWLEDGEMENTS

We would like to thank Bonnie Webber for her very helpful suggestions on exemplifying semantics/pragmatics cooperation.

REFERENCES

- [Dahl1986]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.
- [Dowding1986]
John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, submitted to AAAI-86, Philadelphia, 1986.
- [Fillmore1968]
C. J. Fillmore, The Case for Case. In *Universals in Linguistic Theory*, E. Bach and R. T. Harms (ed.), Holt, Rinehart, and Winston, New York, 1968.
- [Gundel1980]
Jeanette K. Gundel, Zero-NP Anaphora in Russian. *Chicago Linguistic Society Parasession on Pronouns and Anaphora*, 1980.
- [Hirschman1982]
L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.
- [Hirschman1985]
L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.
- [Hirschman1986]
L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming*, 1986.
- [Kameyama1985]
Megumi Kameyama, Zero Anaphora: The Case of Japanese, Ph.D. thesis, Stanford University, 1985.
- [Palmer1983]
M. Palmer, Inference Driven Semantic Analysis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, D.C., 1983.
- [Palmer1981]
Martha S. Palmer, A Case for Rule Driven Semantic Processing. *Proc. of the 19th ACL Conference*, June, 1981.

[Palmer1985]

Martha S. Palmer, *Driving Semantics for a Limited Domain*, Ph.D. thesis, University of Edinburgh, 1985.

[Pereira1980]

F. C. N. Pereira and D. H. D. Warren, *Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks*. *Artificial Intelligence* 13, 1980, pp. 231-278.

[Sager1981]

N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[Sidner1979]

Candace Lee Sidner, *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*, MIT-AI TR-537, Cambridge, MA, 1979.

APPENDIX C

Focusing and Reference Resolution in PUNDIT

This paper by Deborah Dahl was presented at AAAI-86 in Philadelphia, August, 1986. It describes the syntactic focusing algorithm used in PUNDIT, and its uses in reference resolution for pronouns, elided noun phrases, *one*-anaphora, and implicit associates.

FOCUSING AND REFERENCE RESOLUTION IN PUNDIT

Deborah A. Dahl

Research and Development Division *

SDC - A Burroughs Company

PO Box 517

Paoli, PA 19301

(Presented at AAAI-86, August 11-15, 1986, Philadelphia, PA)

* Now Paoli Research Center, Unisys

ABSTRACT

This paper describes the use of focusing in the PUNDIT text processing system.* Focusing, as discussed by [Sidner1979] (as well as the closely related concept of centering, as discussed by [Grosz1983]), provides a powerful tool for pronoun resolution. However, its range of application is actually much more general, in that it can be used for several problems in reference resolution. Specifically, in the PUNDIT system, focusing is used for *one*-anaphora, elided noun phrases, and certain types of definite and indefinite noun phrases, in addition to its use for pronouns. Another important feature in the PUNDIT reference resolution system is that the focusing algorithm is based on syntactic constituents, rather than on thematic roles, as in Sidner's system. This feature is based on considerations arising from the extension of focusing to cover *one*-anaphora. These considerations make syntactic focusing a more accurate predictor of the interpretation of *one*-anaphoric noun phrases without decreasing the accuracy for definite pronouns.

* This work is supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research. APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

I BACKGROUND

A. FOCUSING

Linguistically reduced forms, such as pronouns, are used to refer to the entity or entities with which a discourse is most centrally concerned. Thus, keeping track of this entity, (the *topic* of [Gundell1974], the *focus* of [Sidner1979], and the *backward-looking center* of [Grosz1983, Kameyama1985]) is clearly of value in the interpretation of pronouns. However, while 'pronoun resolution' is generally presented as a problem in computational linguistics to which focusing can provide an answer (See for example, the discussion in [Hirst1981]), it is useful to consider focusing as a problem in its own right. By looking at focusing from this perspective, it can be seen that its applications are more general than in simply finding referents for pronouns. Focusing can in fact play a role in the interpretation of several types of noun phrases. In support of this position, I will show how focus is used in the PUNDIT (Prolog UNDERstander of Integrated Text) text processing system to interpret a variety of forms of anaphoric reference; in particular, pronouns, elided noun phrases, *one*-anaphora, and context-dependent full noun phrase references.

A second position advocated in this paper is that surface syntactic form can provide an accurate guide to determining what entities are in focus. Unlike previous focusing algorithms, such as that of [Sidner1979], which used thematic roles (for example, *theme*, *agent*, *instrument* as described in [Gruber1976]), the algorithm used in this system relies on surface syntactic structure to determine which entities are expected to be in focus. The extension of the focusing mechanism to handle *one*-anaphora has provided the major motivation for the choice of syntactic focusing.

The focusing mechanism in this system consists of two parts—a **FocusList**, which is a list of entities in the order in which they are to be considered as foci, and a focusing algorithm, which orders the **FocusList**. The implementation is discussed in detail in Section 5.

B. OVERVIEW OF THE PUNDIT SYSTEM

I will begin with a brief overview of the PUNDIT system, currently under development at SDC. PUNDIT is written in Quintus Prolog 1.5. It is designed to integrate syntax, semantics, and discourse knowledge in text processing for limited domains. The system is implemented as a set of distinct interacting components which communicate with each other in clearly specified and restricted ways.

The syntactic component, Restriction Grammar, [Hirschman1982, Hirschman1985], performs a top-down parse by interpreting a set of context-free BNF definitions and enforcing context-sensitive restrictions associated with the BNF definitions. The grammar is modelled after that developed by the NYU Linguistic String Project [Sager1981].

After parsing, the semantic interpreter is called. This interpreter is based on Palmer's Inference Driven Semantic Analysis system, [Palmer1985], which decomposes verbs into their component meanings and fills their thematic roles. In the process of filling a thematic role the semantic analyzer calls noun phrase analysis on a specific syntactic constituent in order to find a referent to fill the role. Reference resolution instantiates the referent.

Domain-specific information is available in the knowledge base. The knowledge base is implemented as a semantic net containing a part-whole hierarchy and an *isa* hierarchy of the components and entities in the application domain. The current domain is that of reports of computer equipment failures. The system is being ported to reports of air compressor failures.

Following the semantic analysis, a discourse component is called which updates the discourse representation to include the information from the current sentence and which runs the focusing algorithm.

II USES OF FOCUSING

Focusing is used in four places in PUNDIT — for definite pronouns, elided noun phrases, *one*-anaphora, and implicit associates.

As stated above, reference resolution is called by the semantic interpreter when it is filling a thematic role. Reference resolution proposes a referent for the constituent associated with that role. For example, if the verb is *replace* and the semantic interpreter is filling the role of **agent**, reference resolution would be called for the surface syntactic subject. After a proposed referent is chosen for the subject, any specific selectional restrictions on the agent of *replace* (such as the constraint that the agent has to be a human being) are checked. If the proposed referent fails selection, backtracking into reference resolution occurs and another referent is selected. Cooperation between reference resolution and the semantic interpreter is discussed in detail in [Palmer1986]. The semantic interpreter itself is discussed in [Palmer1985].

A. PRONOUNS AND ELIDED NOUN PHRASES

Pronoun resolution is done by instantiating the referent of the pronoun to the first member of the **FocusList** unless the instantiation would violate syntactic constraints on coreferentiality.* (As noted above, if the proposed referent fails selection, backtracking occurs, and another referent is chosen.)

The reference resolution situation in the maintenance texts however, is complicated by the fact that there are very few overt pronouns. Rather, in

* The syntactic constraints on coreferentiality currently used by the system are very simple. If the direct object is reflexive it must be instantiated to the same referent as the subject. Otherwise it must be a different referent. Obviously, as the system is extended to cover sentences with more complex structures, a more sophisticated treatment of syntactic constraints on coindexing using some of the insights of [Reinhart1976], and [Chomsky1981] will be required.

contexts where a noun phrase would be expected, there is often elision, or a zero-np as in *Won't power up* and *Has not failed since Hill's arrival*. Zeroes are handled as if they were pronouns. That is, they are assumed to refer to the focus. The hypothesis that elided noun phrases can be treated in the same way as pronouns is consistent with previous claims in [Gundel1980] and [Kameyama1985] that in languages such as Russian and Japanese, which regularly allow zero-np's, the zero corresponds to the focus. If these claims are correct, it is not surprising that in a sublanguage like that found in the maintenance texts, which also allows zero-np's, the zero should correspond to the focus.*

B. IMPLICIT ASSOCIATES

Focusing is also used in the processing of certain full noun phrases, both definite and indefinite, which involve *implicit associates*. The term implicit associates refers to the relationship between *a disk drive* and *the motor* in examples like *The field engineer installed a disk drive. The motor failed*. It is natural for a human reader to infer that the motor is part of the disk drive. In order to capture this intuition, it is necessary for the system to relate the motor to the disk drive of which it is part. Relationships of this kind have been extensively discussed in the literature on definite reference. For example, implicit associates correspond to *inferrable* entities described by [Prince1981], the *associated use definites* of [Hawkins1978], and the *associated* type of implicit backwards specification discussed by [Sidner1979]. Sidner suggests that implicit associates should be found among the entities in focus. Thus, when the system encounters a definite noun phrase mentioned for the first time, it examines the members of the **FocusList** to determine if one of them is a possible associate of the current noun phrase. The specific association relationships (such as *part-whole*, *object-property*, and so on) are defined in the knowledge base.

This approach is also used in the processing of certain indefinite noun phrases. In every domain, there are certain types of entities which can be classified as *dependent*. By this is meant an entity which is not typically mentioned on its own, but which is referred to in connection with another entity, on which it is dependent. In the maintenance domain, for example, parts such as keyboards, and printed circuit boards are dependent, since they are normally mentioned with reference to something else, such as a disk drive, or printer.* In

* Another kind of pronoun (or zero) also occurs in the maintenance texts, which is not associated with the local focus, but is concerned with global aspects of the text. For example, the field engineer is a default agent in the maintenance domain, as in *Thinks problem is in head select area*. This is handled by defining *default elided referents* for the domain. The referent is instantiated to one of these if no suitable candidate can be found in the **FocusList**.

* There are exceptions to this generalization. For example, in a sentence like *field engineer ordered motor*, the motor on order is not part of anything else (yet). In PUNDIT, these cases are assumed to depend on the verb meaning. In this example, the object of *ordered* is categorized as *non-specific*, and reference resolution is not called. See [Palmer1986] for details.

an example like *The system is down. The field engineer replaced a bad printed circuit board*, it seems clear that a relationship between the printed circuit board and the system should be represented. These are treated in the same way as the definites discussed above.

C. ONE-ANAPHORA

PUNDIT extends focusing to the analysis of *one*-anaphora following [Dahl1984], which claims that focus is central to the interpretation of *one*-anaphora. Specifically, the referent of a *one*-anaphoric noun phrase (e.g., *the blue one, some large ones*) is claimed to be a member or members of a set which is the focus of the current clause. For example, in *Installed two disk drives. One failed*, the set of two disk drives is assumed to be the focus of *One failed*, and the disk drive that failed is a member of that set. This analysis can be contrasted with that of [Halliday1976], which treats *one*-anaphora as a surface syntactic phenomenon, completely distinct from reference. It is more consistent with the theoretical discussions of [Hankamer1976], and [Webber1983]. These analyses advocate a discourse-pragmatic treatment for both *one*-anaphora and definite pronouns.* The main computational advantage of treating *one*-anaphora as a discourse problem is that the basic anaphora mechanism then requires little modification in order to handle *one*-anaphora. In contrast, an implementation following the account of Halliday and Hasan would be much more complex and specific to *one*-anaphora.

The process of reference resolution for *one*-anaphora occurs in two stages. The first stage is resolution of the anaphor, *one*, and this is the stage that involves focusing. When the system analyzes the head noun *one*, it instantiates it with the *category* of the first set in the **FocusList** (*disk drive* in this example).** In other words, the referent of the noun phrase must be a member of the previously mentioned set of disk drives. The second stage of reference resolution for *one*-anaphora assigns a specific disk drive as the referent of the entire noun phrase, using the same procedures that would be used for a full noun phrase, *a disk drive*.

The extension of the system to *one*-anaphora provides the clearest motivation for the choice of a syntactic focus in PUNDIT. Before I discuss the kinds of examples which support this approach, I will briefly describe the relevant part of the focusing algorithm based on thematic roles which is proposed by [Sidner1979]. After each sentence, the focusing algorithm orders the elements in the sentence in the order in which they are to be considered as potential foci in the next sentence. Sidner's ordering and that of PUNDIT are compared in

* Webber's analysis in [Webber1978], is more syntactically based than [Webber1983], proposing an approach similar to Halliday and Hasan's.

** Currently the only sets in the **FocusList** are those which were explicitly mentioned in the text. However, as pointed out by [Dahl1982], and [Webber1983, Dahl1984], other sets besides those explicitly mentioned are available for anaphoric reference. These have not yet been added to the system.

Figure 1.

The feature of *one*-anaphora which motivates the syntactic algorithm is that the availability of certain noun phrases as antecedents for *one*-anaphora is affected by surface word order variations which change syntactic relations, but which do not affect thematic roles. If thematic roles are crucial for focusing, then this pattern would not be observed.

Consider the following examples:

(1) A: I'd like to plug in this lamp, but the bookcases are blocking the electrical outlets.

B: Well, can we move one?

(2) A: I'd like to plug in this lamp, but the electrical outlets are blocked by the bookcases.

B: Well, can we move one?

In both (1) and (2) the electrical outlets are the theme, which means that in a thematic-role based approach, the outlets represent the expected focus in both sentences. However, only in (1), do informants report an impression that B is talking about moving the electrical outlets. This indicates that the expected focus following (1) A is the outlets, while it is the bookcases in (1) B.*

Sidner

Theme
Other thematic roles
Agent
Verb Phrase

PUNDIT

Sentence
Direct Object
Subject
Objects of
Prepositional
Phrases

Figure 1: Comparison of Potential Focus Ordering in Sidner's System and PUNDIT

* In the case of (1), the expected focus is eventually rejected on the basis of world knowledge about what is likely to be movable, but focusing is only intended to determine the *order* in which discourse entities are considered as referents, not to determine which referent is actually correct. The referent proposed by focusing must always be confirmed by world knowledge.

Similar examples using definite pronouns do not seem to exhibit the same effect. In (3) and (4), *they* seems to be ambiguous, until world knowledge is brought in. Thus, in order to handle definite pronouns alone, either algorithm would be adequate.

(3) A: I'd like to plug in this lamp, but the bookcases are blocking the electrical outlets.

B: Well, can we move them?

(4) A: I'd like to plug in this lamp, but the electrical outlets are blocked by the bookcases.

B: Well, can we move them?

Another example with *one*-anaphora can be seen in (5) and (6). In (5) but not in (6), the initial impression seems to be that a bug has lost its leaves. As in (1) and (2), however, the thematic roles are the same, so a thematic-role-based algorithm would predict no difference between the sentences.

(5) The plants are swarming with the bugs. One's already lost all its leaves.

(6) The bugs are swarming over the plants. One's already lost all its leaves.

In addition to theoretical considerations, there are a number of practical advantages to defining focus on constituents rather than on thematic roles. For example, constituents can often be found more reliably than thematic roles. In addition, thematic roles have to be defined individually for each verb.* Furthermore, since thematic roles for verbs can vary across domains, defining focus on syntax makes it less domain dependent, and hence more portable.

III IMPLEMENTATION

A. THE FOCUSLIST AND CURRENTCONTEXT

The data structures that retain information from sentence to sentence in the PUNDIT system are the **FocusList** and the **CurrentContext**. The **FocusList** is a list of all the discourse entities which are eligible to be considered as foci, listed in the order in which they are to be considered. For example, after a sentence like *The field engineer replaced the disk drive*, the following **FocusList** would be created.

[[event1],[drive1],[engineer1]]

The members of the **FocusList** are unique identifiers that have been assigned to the three discourse entities — the disk drive, the field engineer, and the state of affairs of the field engineer's replacement of the disk drive. The **CurrentContext** contains the information that has been conveyed by the discourse so far. After the example above, the **CurrentContext** would contain three types of information:

* Of course, some generalizations can be made about how arguments map to thematic roles. However, they are no more than guidelines for finding the

(1)

Discourse id's, which represent classifications of entities. For example, **id(field^engineer,[engineer1])** means that [engineer1] is a field engineer. *

(2) **Facts** about part-whole relationships (**hasparts**).

(3) **Representations** of the events in the discourse. For example, if the event is that of a disk drive having been replaced, the representation consists of a unique identifier ([**event1**]), the surface verb (**replace(time(_))**), and the decomposition of the verb with its (known) arguments instantiated. The thematic roles involved are **object1**, the replaced disk drive, **object2**, the replacement disk drive, **time** and **instrument** which are uninstantiated, and **agent**, the field engineer. (See [Palmer1986], for details of this representation). Figure 2 illustrates how the **CurrentContext** looks after the discourse-initial sentence, *The field engineer replaced the disk drive.*

B. THE FOCUSING ALGORITHM

The focusing algorithm used in this system resembles that of [Sidner1979], although it does not use the actor focus and uses surface syntax rather than thematic roles, as discussed above. It is illustrated in Figure 3.

IV SUMMARY

This paper has described the reference resolution component of PUNDIT, a large text understanding system in Prolog. A focusing algorithm based on surface syntactic constituents is used in the processing of several different types of reduced reference: definite pronouns, *one*-anaphora, elided noun phrases, and implicit associates. This generality points out the usefulness of treating focusing as a problem in itself rather than simply as a tool for pronoun resolution.

ACKNOWLEDGMENTS

I am grateful for the helpful comments of Lynette Hirschman, Marcia Linebarger, Martha Palmer, and Rebecca Schiffman on this paper. John Dowding and Bonnie Webber also provided useful comments and suggestions on an earlier version.

References

themes of verbs. The verbs still have to be classified individually.

* **field^engineer** is an example of the representation used in PUNDIT for an idiom.

```
id(field^engineer,[engineer1]),
id(disk^drive,[drive1]),
id(system,[system1]),
id(disk^drive,[drive2]),
id(event,[event1]),
```

```
haspart([system1],[drive1]),
haspart([system1],[drive2]])
```

```
event([event1],
      replace(time(_)),
      [included(object2([drive2]),time(_)),
       missing(object1([drive1]),time(_)),
       use(instrument(_8405),
           exchange(object1([drive1]),
                     object2([drive2]),time(_))),
       cause(agent([engineer1]),
             use(instrument(_8405),
                 exchange(object1([drive1]),
                           object2([drive2]),time(_)))))]
```

Figure 2: CurrentContext after
The field engineer replaced the disk drive.

(1) First Sentence of a Discourse:

Establish expected foci for the next sentence (order FocusList): the order reflects how likely that constituent is to become the focus of the following sentence.

**Sentence
Direct Object
Subject
Objects of (Sentence-Level)
Prepositional Phrases**

(2) Subsequent Sentences (update FocusList):

If there is a pronoun in the current sentence, move the focus to the referent of the pronoun. If there is no pronoun, retain the focus from the previous sentence. Order the other elements in the sentence as in (1).

Figure 3: The Focusing Algorithm

[Chomsky1981]

Noam Chomsky, *Lectures on Government and Binding*. Foris Publications, Dordrecht, 1981.

[Dahl1982.]

Deborah A. Dahl, Discourse Structure and one-anaphora in English, presented at the 57th Annual Meeting of the Linguistic Society of America, San Diego, 1982..

[Dahl1984]

Deborah A. Dahl, The Structure and Function of One-Anaphora in English, PhD Thesis; (also published by Indiana University Linguistics Club, 1985), University of Minnesota, 1984.

[Grosz1983]

Barbara Grosz, Aravind K. Joshi, and Scott Weinstein, Providing a Unified Account of Definite Noun Phrases in Discourse. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 1983, pp. 44-50.

[Gruber1976]

Jeffery Gruber, *Lexical Structure in Syntax and Semantics*. North Holland, New York, 1976.

[Gundel1974]

Jeanette K. Gundel, Role of Topic and Comment in Linguistic Theory, Ph.D. thesis, University of Texas at Austin, 1974.

[Gundel1980]

Jeanette K. Gundel, Zero-NP Anaphora in Russian. *Chicago Linguistic Society Parasession on Pronouns and Anaphora*, 1980.

[Halliday1976]

Michael A. K. Halliday and Ruqaiya Hasan, *Cohesion in English*. Longman, London, 1976.

[Hankamer1976]

Jorge Hankamer and Ivan Sag, Deep and Surface Anaphora. *Linguistic Inquiry* 7(3), 1976, pp. 391-428.

[Hawkins1978]

John A. Hawkins, *Definiteness and Indefiniteness*. Humanities Press, Atlantic Highlands, New Jersey, 1978.

[Hirschman1982]

L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.

[Hirschman1985]

L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Hirst1981]

Graeme Hirst, *Anaphora in Natural Language Understanding*. Springer-Verlag, New York, 1981.

[Kameyama1985]

Megumi Kameyama, Zero Anaphora: The Case of Japanese, Ph.D. thesis, Stanford University, 1985.

[Palmer1985]

Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[Palmer1986]

Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, *Recovering Implicit Information*, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[Prince1981]

Ellen F. Prince, *Toward a Taxonomy of Given-New Information*. In *Radical Pragmatics*, Peter Cole (ed.), Academic Press, New York, 1981.

[Reinhart1976]

Tanya Reinhart, *The Syntactic Domain of Anaphora*, Ph.D. thesis, Massachusetts Institute of Technology, 1976.

[Sager1981]

N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[Sidner1979]

Candace Lee Sidner, *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*, MIT-AI TR-537, Cambridge, MA, 1979.

[Webber1978]

Bonnie Lynn Webber, *A Formal Approach to Discourse Anaphora*. Garland, New York, 1978.

[Webber1983]

Bonnie Lynn Webber, *So What Can We Talk About Now?*. In *Computational Models of Discourse*, Michael Brady and Robert C. Berwick (ed.), 1983.

APPENDIX D

A Dynamic Translator for Rule Pruning in Restriction Grammar

This paper by John Dowding and Lynette Hirschman has been submitted to the 2nd International Workshop on Natural Language Processing and Logic Programming, to be held in Vancouver, B.C., Canada, August 17-19, 1987.

A Dynamic Translator for Rule Pruning in Restriction Grammar¹

John Dowding and Lynette Hirschman

Paoli Research Center
Unisys Defense Systems
P.O. Box 517
Paoli, PA 19301

Submitted to the Second International Workshop on
Natural Language Understanding
and Logic Programming

Vancouver, B.C., Canada
August 17-19, 1987

ABSTRACT

This paper describes a *Dynamic Translator* for Restriction Grammar, a logic (Prolog) environment for syntactic processing of natural language text. The dynamic translator supports *Dynamic Rule Pruning*, a method of focusing search for a parse. Rule pruning permits elimination of options from BNF definitions, based on information from the partially constructed parse tree and the word input stream. It is particularly effective for pruning verb complement options based on the particular verb in the input stream. The dynamic translator combines the flexibility of an interpreter (needed for grammar development and for dynamic pruning) with the runtime performance efficiency of a translator. This combination has produced an execution mechanism that is more efficient than either an interpreter or a translator alone. The paper presents results demonstrating a 20-fold speed-up in parse times obtained via rule-pruning supported by the dynamic translator.

¹ This work is supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research; and in part by National Science Foundation contracts DCR-8202397 and DCR-85-02205.

1. Introduction

This paper describes execution strategies used by the Restriction Grammar to implement efficient syntactic processing for natural language text. These include a mechanism for dynamic rule pruning and a dynamic translator. Dynamic rule pruning uses information from the input word stream to focus the search for a correct parse. This facility is supported efficiently by the Dynamic Translator, which allows the use of an inherently interpretive mechanism within a translated system.

The Restriction Grammar [Hirschman1982a, Hirschman1985a] is the syntactic component of PUNDIT, the Unisys natural language understanding system [Palmer1986a], which includes modules for semantics [Palmer1983a] and discourse analysis [Dahl1986a], in addition to the syntactic component described here. Because the other components of PUNDIT rely on the parser, efficient parsing is a prerequisite for productive development of the entire natural language understanding system.

The Restriction Grammar belongs to the rapidly expanding class of logic grammars, including Metamorphosis Grammar [Colmerauer1978a], Definite Clause Grammar [Pereira1980a], Extraposition Grammar [Pereira1981a], Definite Clause Translation Grammar [Abramson1984a], Modifier Structure Grammar [Dahl1983a], and Gapping Grammar [Dahl1984a]. It shares with other logic grammars a set of production rules (BNF definitions) interspersed with Prolog constraints. It extends the DCG notion of "implicit" parameters to include not only the word stream, but also an automatically constructed parse tree. The Restriction Grammar differs from other logic grammars in that the constraints are restrictions on the well-formedness of the parse tree; this requires a more complex execution mechanism to support the grammar, as will be explained in the following sections. The Restriction Grammar is implemented in Prolog, which provides a highly interactive environment required for developing natural language systems. Prolog also provides great flexibility for writing alternative control structures and interpreters, and we have made extensive use of this capability in our research.

Traditionally, the execution mechanisms for logic grammars have either been interpreters or translators. A (top-down) interpreter parses a string as a phrase of a given category by choosing a grammar rule of that category, dividing the phrase into sub-phrases, and parsing those sub-phrases into the categories required by the grammar rule. Thus, at runtime the interpreter requires not only the string that is to be parsed, but also the set of grammar rules. Alternatively, the process can be broken down into two phases, the translation phase and the runtime phase. The translator takes the complete set of grammar rules and produces a set of Prolog procedures which, when called at runtime, will parse the phrase in exactly the same way that the original grammar would have. The translation phase converts the information explicit in the grammar rules into information implicit in the Prolog procedures. The translation phase requires the set of grammar rules but does not have the input string available to it, while the runtime phase after translation has access to the input string, but does not have explicit access to the original grammar rules. Due to this loss of explicit information when moving from interpretation to translation, the translated Prolog code will be more efficient than the interpreter, but the interpreter will be more flexible.

In contrast to these traditional approaches, the Restriction Grammar uses a single mechanism, the Dynamic Translator, that takes advantage of the strengths of translation and interpretation without the corresponding disadvantages. The Dynamic Translator has available to it both the input string and the grammar rules (like an interpreter), but also makes use of both a translation and a runtime phase (like a translator). In the Dynamic Translator, the translated code runs in cooperation with an interpreter to parse a sentence. Although one might thus expect that the speed of the Dynamic Translator to be intermediate between an interpreter and a translator, the Dynamic Translator is substantially faster than either. This added efficiency is gained by the use of the Dynamic Rule Pruning mechanism, an inherently interpretive device that dynamically prunes the search space.

The Dynamic Rule Pruning mechanism uses information available at run-time to reduce the number of options that must be considered for certain grammar rules. This information includes both the input word stream and the partially constructed parse tree. Reducing the number of options eliminates extraneous paths from the search space and greatly increases the efficiency of the parsing process.

2. Restriction Grammar Framework

Restriction Grammar draws its grammatical approach from earlier work of Sager, Grishman and Friedman in connection with the New York University Linguistic String Project [Sager1981a, Sager1975a, Grishman1973a]. The grammar is written in terms of context-free rules, augmented with context-sensitive restrictions stated as constraints on the shape of the partially constructed parse tree. The parsing mechanism is a standard top-down, left-to-right parser. The parse tree is constructed incrementally after the successful application of each BNF definition, using a data structure that supports free tree traversal by the restrictions [Hirschman1982a, Hirschman1985a].

Restriction Grammar differs from other logic grammar frameworks in several respects. One is that the restrictions are constraints on the well-formedness of the (surface) parse trees produced by the application of the BNF definitions. This differs from other logic grammars where contextual information is made available through parameters to the BNF definitions. Restriction Grammar isolates all context-sensitivity in the constraints, leaving the BNF definitions uncluttered by extra parameters. The restrictions are written using a special set of tree-description primitives, which can be combined into general purpose syntactically motivated routines, such as "head", "left/right adjunct", "main verb", etc.

Restrictions are written using a layered approach that makes the syntactic constructs independent of the particular implementation of tree structure. This approach has proved extremely useful in insulating the grammar from changes in the underlying execution mechanism. The lowest layer of operators consist of primitive tree relation operators (such as parent, child, left and right sibling) and operators to extract the label of a node and the lexical item associated with a node (also the lexical subclasses associated with a word). On top of this layer are a set of *restriction operators* that support various constraints on the make-up of trees. The restriction operators also include a few operators to examine the word input stream for optimisation purposes. This enables a restriction to reject certain options before the parser makes any attempt to construct them. The next layer of *routines* captures syntactic relations such as *head* of a construction, the *main verb*, or the *left/right adjunct* of a construction. Finally, restrictions are built out of the routines and the restriction operators. There are currently about 15 restriction operators, 20 routines, 30 restrictions, and 102 BNF definitions in the stable grammar without conjunction. Conjunction (treated as a meta-rule) adds another 50% to the number of grammar rules.

3. Dynamic Rule Pruning

Over the course of the past year, the coverage of our grammar has been considerably extended. These extensions include the addition of a wide range of verb complement types, conjunction (e.g., *Unit has excessive wear on inlet impellor assembly and shows high usage of oil.*), and fragmentary input (e.g., *Believe the coupling from diesel to sac lube oil pump to be sheared.*). As the size of the grammar has increased it has been critical to maintain an efficient system to support development and debugging, thus motivating the development of dynamic Rule PruningFR and the *dynamic translator*.

Dynamic rule pruning uses information from the partially parsed sentence and from the word input stream to focus the search for a correct parse. This capability has been particularly useful in pruning possible verb complements, but is also being applied to limit pre- and post-verbal sentence adjunct types, as well as complement types for nouns and adjectives. The idea of dynamic rule pruning has been adapted from its implementation in Sager's Linguistic String Parser [Sager1981a]. Our current English grammar allows for a rich verb complement structure — there are presently 34 options in the verb complement rule. In the lexicon, each verb is subcategorised for the particular type(s) of complement it can accept: e.g., noun string object (*ns/o*), direct + prepositional object (*np/p*), predicate object (*object/o*), various reduced sentential complement types (*to + verb + object = to/o*), sentential complement (*assertion*), etc.

As soon as a verb is found during parsing, the list of possible complement options can be drastically pruned to match the set of options given by the subcategorisation of that particular verb. However, this requires a dynamic interaction between the input word stream, the syntactic constructs (namely the verb), and the grammar rules. This capability has been implemented via a special dynamic prune constraint. The operator *prune* occurs within the BNF definition and replaces the normal definition, which has the form shown in Table 1 (only 19 of the 34 verb complement options are shown).

object ::=	(nstgo;	%Noun STRing Object (direct object: "eat fish")
pn;		%Preposition + Noun (prep phrase: "attend to their advice")
npn;		%Noun string + Prep + Noun ("put it on the table")
objectbe;		%Object of BE ("be late/on time/a fool")
veno;		%VEN (past participle) + Object ("have seen it")
tovo;		%TO + Verb + Object ("seems to do it")
ntovo;		%Noun + TO + Verb + Object ("want them to see it")
eqtovo;		%Equi TO + Verb + Object ("want to see it")
objtovo;		%Object + TO + Verb + Object ("helped him to fix it")
vo;		%tenseless Verb + Object ("will do it")
thats;		%THAT + Sentence ("hope that they come")
assertion;		%ASSERTION ("hope they hear")
pnthats;		%Preposition + Noun + THAT + Sentence
		("proved to her that she was right")
pnthatsvo;		%Prep + Noun + THAT + Subject + Verb + Object
		("suggested to her that she write it up")
thatsvo;		%THAT+ Subject + Verb + Object
		("suggested that she write it up")
nthats;		%Noun + THAT + Sentence ("said that they were here")
nullobj;		%NULL Object (for intransitive verbs: "I left")
sven;		%Subject + VEN (past participle: "want it finished")
dpl,		%DP (particle, as in "show off")
{w_preobj_sa}.		%{...} indicates a constraint, as in DCG's

Table 1. Object Options for Verb Complement

The prune rule in the BNF has the form:

object ::= prune(PruningProcedure,Def).

It takes two arguments: the name of a procedure to control pruning (*prune_object* in the example below), and the full set of options for the definition (the right-hand side of the BNF definition):

```
object ::= prune(prune_object,
  ((nstgo;pn;nnp;objectbe;veno;tovo;ntovo;
    eqtovo;objtovo;vo;thats;assertion;pnthats;
    pnthatsvo;thatsvo;nthats>nullobj;sven;dpl),
    {w_preobj_sa})).
```

The interpreter executes a call to *prune/2* by calling *exec_prune/5*, which returns the pruned definition. The parameters for *exec_prune* are the pruning procedure name, the original definition, the pruned definition, the starting point in the parse tree, and the remaining word stream:

exec_prune(PruningProcedure,OriginalDef,PrunedDef,StartingNode,WordStream).

Once *exec_prune* returns the pruned definition, the main interpreter loop is re-invoked with the new pruned definition. (The code for this is shown with the code for the interpreter as a whole, in Fig. 2 of the Appendix.)

The rule for **object ::= prune(prune_object,Def)**, shown above, is the rule for verb complements. It prunes the object options by locating the verb in the partial parse tree, examining the verb's subcategorisation, and using the subcategorisation to filter out inapplicable options from the full object definition. For example, the verb *replace* is subcategorised for direct object (*nstgo: replace*

A Dynamic Translator for Rule Pruning in Restriction Grammar

something), and direct + prepositional object (npms: *replace something with something*). As a result of dynamic rule-pruning, the parser tries only these two options of object, instead of all 19! The speed-up in parsing is substantial (approximately 8 fold), even for short sentences, and is particularly striking if the system retrieves *all* parses for a given input sentence.

Pruning is implemented via one basic routine, `intersect_opt` which controls the pruning of the original definition:

`intersect_opt(OptionList,OriginalDef,PrunedDef,IncludeExcludeSwitch).`

The list of options in argument 1 are included or excluded (depending on the switch setting in argument 4) from the original definition (argument 2), to produce the new definition in argument 3. This mechanism guarantees that pruning produces a subset of the original definition; it cannot introduce arbitrary new definitions.

Our main application for pruning has been to verb complements (object) options. We plan to apply pruning to restrict adjectival and noun complements as well. We are also currently implementing pruning constraints on the construction of sentence adjuncts (generally adverbial phrases and subordinate clauses), particularly in pre- and post-verb positions. Pre- and post-verb adverbial modifiers are highly constrained in the absence of commas (e.g., *the disk while being removed broke* seems questionable, but *the disk, while being removed, broke* seems better and *While being removed, the disk broke* seems good). By recognising these pre- and post-verb positions and checking for presence or absence of a comma, it is possible to prune drastically the sentence adjunct options. Another interesting application of pruning is dynamic rule-reordering based on context. This may prove to be an extremely valuable capability in tuning grammars to particular applications.

The dynamic rule-pruning facility has been a motivating factor behind the development of the dynamic translator. The rule-pruning can obviously only be done at run-time, since it depends on the particular words (e.g., verbs) in the input sentence. However, the 8-fold speed-up provided by rule pruning is too dramatic to discard in favor of a purely translated system. Our solution was to develop the *dynamic translator*, to allow a dynamic interaction where it provides greater leverage (e.g., rule pruning), but otherwise to use the translated/compiled environment for maximal efficiency. This has enabled us to speed up the system by a factor of 20.

4. The Dynamic Translator

The Dynamic Translator provides a single execution mechanism that combines the flexible and dynamic nature of an interpreter with the efficiency of translated code. The dynamic translator makes it possible to run translated code and interpreted grammar rules interleaved. This is done by determining at translation time those points in the grammar requiring the flexibility of the interpreter, and building into the translated code calls to the interpreter at those points. At present, the points in the grammar where the flexibility of the interpreter is required are those places where the dynamic rule pruning mechanism is used.

As a control mechanism, interpretation has its disadvantages. For instance, in our system, besides just interpreting the grammar rules, the control mechanism must also build a parse tree that will allow free tree traversal. Building this parse tree is costly, and this makes running our system slow compared to other styles of logic grammars. Additionally, interpreters support features to aid in grammar debugging. While these features are very helpful to the grammar writer, they typically reduce the speed of the interpreter even if the features are not being used.

The approach taken in translation is to forsake all of the debugging features introduced in the interpreter and to translate the grammar into Prolog code which can be subsequently compiled. The translated grammar rules are faster than the interpreted grammar rules for several reasons:

- As much of the tree building as possible is done at translation time, thus the translated grammar rules have to do far less work.
- The costs of the extra debugging functions introduced in the interpreter are eliminated.
- The translation produces a file of Prolog clauses which can be given to the Prolog compiler, to produce compiled Prolog running several times faster.

A Dynamic Translator for Rule Pruning in Restriction Grammar

The dynamic translator is implemented with a conventional translator and interpreter, each modified to know when calls to the other should be made. The interpreter knows to call a translated version of a grammar rule when the appropriate translated procedure is resident in Prolog and the user has set a switch that indicates that the translated mode of execution is desired (this is the default). There are situations in which a grammar writer may want to execute the grammar strictly interpreted, to take advantage of certain grammar debugging aids. At present, when the grammar rules are being translated, the translator inserts into the object code a call to the interpreter to execute a particular grammar rule only at those points where the dynamic rule pruning mechanism is called. This possibility for mutual recursion between the interpreter and the translated grammar rules results in a more flexible and efficient system.

We have also avoided a problem that might arise from using the dynamic translator, namely maintaining both interpreted and translated versions of the grammar. If the grammar writer is dynamically editing and testing grammar rules, there is the possibility that the translated code and some of the explicit grammar rules will become inconsistent. We solve this problem by providing a grammar rule editor [Riley1986a] that maintains consistency between the explicit grammar rules and their translated counterparts. After a grammar rule has been edited, the grammar rule editor will automatically retranslate and compile the revised rule.

5. Implementation

The parsing mechanisms described here, together with a broad-coverage grammar of English, constitute the syntactic component of the Unisys natural language understanding system. The system is currently implemented in Quintus Prolog 1.5 on a VAX 11/785 and on a Sun Workstation, in Xerox Quintus Prolog on Xerox Lisp Machines, in TI Prolog 1.0 on a TI Explorer, and in Symbolics Prolog 6.1 on a Symbolics 3640 Lisp Machine.

We have tested the dynamic translator on a large corpus of sentences (about 100 sentences) from the domain of equipment failure reports for starting air compressors (SACs), and compared those results with the same sentences parsed by the translated grammar without rule pruning, the interpreter with rule pruning, and the interpreter without rule pruning. In Figure 2 we have chosen a few representative sentences from this corpus, and indicate their parse times under the four different execution schemes (parse times given are the time to compute all of the parses of the sentences).

The resulting parsing statistics shown here are heavily influenced by the time taken executing restrictions. This time is constant whether running interpreted or translated code, since restrictions are Prolog code and can be compiled by the Prolog compiler without any intermediate translation. The difference between the parse times for an interpreted versus translated grammar would be much more dramatic if the time executing restrictions were ignored.

From the large corpus of sentences we have computed normalized parse times across the entire corpus. These results are given in Figure 3.² These results indicate that the translation process accounts for about a 2-3 fold speed up, and that rule pruning accounts for about an 8-9 fold speed up. The efficiency increase due to rule pruning is highly dependent on the particular grammar that we are using. A smaller grammar (or one which had fewer object options) would not show as dramatic an increase in efficiency. On the other hand, the efficiency increase due to translation is more constant across grammars.

²Normalized parse times were computed by taking the translated system with rule pruning as the base case, and comparing that parse time with the parse times for the other variations. The ratio was computed for each parse of each sentence, then averaged over each sentence. The average for the entire corpus was then computed from these sentence averages. This was done to eliminate a bias in the average towards sentences that had more parses than others.

REPRESENTATIVE SENTENCES:

1. Nr 4 sac oil pressure dropped below alarm point of 65 psig during monitoring of 1a gth.
2. Start air pressure dropped below 30 psig during monitoring of 1a gth.
3. Oil is discolored and contaminated with metal.
4. Loss of lube oil pressure during operation.
5. Investigation revealed adequate lube oil saturated with both metallic and non-metallic particles.
6. Request replacement of sac.

Sentence	No Prune/Interp	No Prune/Trans	W. Prune/Interp	W. Prune/Trans
1.	2170.63	938.016	333.7	180.4
2.	462.183	193.883	129.2	60.9
3.	77.033	27.666	9.46599	4.31699
4.	214.25	90.3658	54.283	24.417
5.	670.733	230.683	120.817	41.8669
6.	43.4839	18.083	12.566	5.2

Table 2: Timings (in cpu seconds) for all parses on sample sentences

	With Pruning	Without Pruning
Interpreter	2.35	20.43
Translator	1	8.77

**Table 3. Normalized Average Parse Times:
Pruning vs. No Pruning; Interpretation vs. Translation**

6. Research Directions

The overall goal of the Unisys Natural Language Processing group is to create a robust, portable system for processing natural language text. The Unisys system is currently being applied to the processing of texts from several domains: a set of internal maintenance reports about trouble-shooting various Unisys-supported computer systems and a set of Navy equipment casualty reports (CASREP's) for starting air compressors, as well as question-answering for a navy ships database. The natural language system accepts the text, processes the natural language information, and produces a set of database entries, knowledge base entries, or database queries capturing the information contained in the text. The database or knowledge base can then be processed for information concerning expected down-time, associations between symptoms and cause of failure, parts and/or expertise needed to repair, etc. The work on the CASREP domain is being performed jointly with a group at New York University under Ralph Grishman. The NYU group is developing a Lisp-based system.

The work on maintenance reports is driving the development of a comprehensive grammar, including a comprehensive treatment of conjunction using meta-rules[Hirschman1986a] and a syntactic and semantic treatment of sentence fragments. Elimination of spurious ambiguity is a well-known problem in parsing sentences. Sentence fragments compound the problem due to their highly degenerate structures and their analysis relies heavily on semantic constraints to reject semantically ill-formed analyses. We are now in the process of implementing a mechanism to support domain-specific selectional constraints, in order to filter out these semantically ill-formed parses. This mechanism will be tightly coupled to the parser, so that an incoherent parse is rejected as early as possible.

We are also actively involved in developing various support tools. One of these is a more sophisticated debugging environment, based on the notion of restriction relaxation. The idea is that if no analysis is obtained, we can pinpoint the cause of failure by allowing relaxation of restrictions that fail in the initial attempt to obtain a parse. A preliminary implementation is complete and is in the process of being tested.

A second area of research is in the area of interactive tools, including tools to build lexicons, to identify domain-specific selectional patterns, and to examine and compare syntactic analyses. Our goal is to build up a development environment rich enough to support the complex and time-consuming activities required to develop large-scale natural language understanding systems.

7. Acknowledgements

We would like to acknowledge the participation of a number of people involved in this research. The original implementation of the interpreter and translator were done by Karl Puder, now at Digital Equipment Corporation. The work of Marica Linebarger on extending the grammar to include fragments has been a motivating factor in the development of an efficient parser. Much of the recent work on the Pundit development environment has been done by Leslie Riley. We also appreciate helpful comments on this paper from Don McKay, Michael Freeman, Rebecca Schiffman and Deborah Dahl.

REFERENCES

- [Abramson1984a]
Harvey Abramson, Definite Clause Translation Grammars. In *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, New Jersey, Feb. 6-9, 1984, pp. 233-241.
- [Colmerauer1978a]
A. Colmerauer, Metamorphosis Grammars. In *Natural Language Communication with Computers*, L. Bok (ed.), Springer, 1978, pp. 133-189.
- [Dahl1986a]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.
- [Dahl1983a]
V. Dahl and M. McCord, Treating Co-ordination in Logic Grammars. *American Journal of Computational Linguistics* 9, No. 2, 1983, pp. 69-91.
- [Dahl1984a]
V. Dahl, More on Gapping Grammars. In *Proc. of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, Japan, 1984, pp. 669-677.
- [Grishman1973a]
R. Grishman, N. Sager, C. Rase, and B. Bookchin, The Linguistic String Parser. *AFIPS Conference Proceedings* 43, AFIPS Press, 1973, pp. 427-434.
- [Hirschman1982a]
L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.
- [Hirschman1985a]
L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.
- [Hirschman1986a]
L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming*, 1986.
- [Palmer1983a]
M. Palmer, Inference Driven Semantic Analysis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, D.C., 1983.
- [Palmer1986a]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.
- [Pereira1980a]
F. C. N. Pereira and D. H. D. Warren, Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13, 1980, pp. 231-278.
- [Pereira1981a]
F. C. N. Pereira, Extraposition Grammars. *American Journal of Computational Linguistics* 7, 1981, pp. 243-256.
- [Riley1986a]
L. Riley and J. Dowding, The Prolog Structure Editor, Logic-Based Systems Technical Memo No. 29, Paoli Research Center, System Development Corporation, January, 1986.

A Dynamic Translator for Rule Pruning in Restriction Grammar

[Sager1975a]

N. Sager and R. Grishman, The Restriction Language for Computer Grammars of Natural Language. *Communications of the ACM* 18, 1975, pp. 390-400.

[Sager1981a]

N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

8. Appendix 1. Dynamic Translator Code

Included below is the Prolog code that makes up the Dynamic Translator. Figure 1 contains the code for the translator, followed by the code for the interpreter in Figure 2. The top level procedure for the translator is `translate_rule/2`, which takes a grammar rule and produces a Prolog procedure; `translate_rule/2` calls the main recursive procedure `trans/7`, to translate the rule body; `trans/7` contains a case for each type of rule body, namely:

Conjunction
Disjunction
Rule Pruning
Restrictions
Terminals
Empty Strings
Nonterminals

Note that the clause that handles rule pruning results in a call to the procedure `prune_exec/9`, which in turn calls the interpreter to execute the pruned grammar rule.

```

/* translate_rule(+Rule,-Translation)
    translate_rule takes a grammar rule and returns its Prolog translation.
    */
translate_rule((X:=Y),(A-B)):-
    label(Parent,X),
    transPred(X,A,Parent,InWords,OutWords),
    daughter(Parent,FirstChild),
    trans(Y,B,FirstChild,Parent,InWords,OutWords),!.

/* trans(+Source,-Object,+FirstChild,-LastChild,+Parent,
    +InWords,-OutWords)
    trans takes a part of a grammar rule, and translates it into Prolog. Note
    that although FirstChild and InWords are marked "+", they will should
    still be uninstantiated when trans is called (They are "significant" logic
    variables). FirstChild is the variable (some unknown child of Parent)
    that is the location to put the next child built. NextChild is the
    location to put the next child built after this segment of the
    grammar rule. If this segment builds no children, then FirstChild =
    NextChild.
    */

%conjunction
trans((X,Y),(A,B),FirstChild,LastChild,Parent,InWords,OutWords):-!,
    trans(X,A,FirstChild,XLast,Parent,InWords,TempWords),
    trans(Y,B,XLast,LastChild,Parent,TempWords,OutWords).

%disjunction
trans((X,Y),
    (((FirstChild = XFirst,LastChild=XLast),A);
    ((FirstChild = YFirst,LastChild=YLast),B)),
    FirstChild,LastChild,Parent,InWords,OutWords):-!,
    trans(X,A,XFirst,XLast,Parent,InWords,OutWords),
    trans(Y,B,YFirst,YLast,Parent,InWords,OutWords).

%prune
trans(prune(Label,Def),
    prune_exec(Label,Def,First,Last,Parent,InWords,OutWords,0,[]),
    First,Last,Parent,InWords,OutWords):-!,
    trans(Def,_,_,_,_,_,_).

%restrictions
trans((X),((InWords=OutWords),A),Prev,Prev,Parent,
    InWords,OutWords):-!,
    transRestriction(X,A,Parent,InWords).

%terminals
trans("X,attach_word(X,Here,InWords,OutWords),
    Here,Next,InWords,OutWords):-
    label(Here,X),
    sibling(Here,Next).

%empty string
trans(",(InWords=OutWords),Prev,Prev,InWords,OutWords):-!.

%nonterminals
trans(X,A,First,Next,InWords,OutWords):-
    label(First,X),
    sibling(First,Next),
    asserts(needs_translating(X)),
    transPred(X,A,First,InWords,OutWords).

/* transPred(+Nonterminal,-Object,+Here,+InWords,OutWords)
    transPred constructs a call to a nonterminal.
    */
transPred(X,A,Here,InWords,OutWords):-
    A=..[X,Here,InWords,OutWords].

/* transRestriction(+RestrictionName,-Object,+Parent,+Words)
    transRestriction constructs a call to a restriction
    */
transRestriction(X,A,Parent,Words):-
    X=..[RestrictionName|ListOfArgs],
    atom(RestrictionName),
    append(ListOfArgs,[Parent,Words],NewListOfArgs),
    A=..[RestrictionName|NewListOfArgs].

```

Figure 1. Prolog Code for the Translator

Similar to `trans/7` in the translator is `exec/8` in the interpreter, with clauses that match those of the translator. The clauses for `execPred/8` expand nonterminals. If a translated version of the grammar rule exists, and the switch `translated_grammar_in_use` is set, then the translated version of the rule is called, rather than expanding the nonterminal via the interpreter.

```

execute(X,Tree,InWords,OutWords):-
    exec(X,Tree,_,InWords,OutWords,0,[]).

%conjunction
exec((X,Y),First,Last,Parent,InWords,OutWords,D,Anc):-!,
    exec(X,First,Next,Parent,InWords,IOtmp,D,Anc),
    exec(Y,Next,Last,Parent,IOtmp,OutWords,D,Anc).

%disjunction
exec((X;Y),First,Last,Parent,InWords,OutWords,D,Anc):-!,
    exec(X,First,Last,Parent,InWords,OutWords,D,Anc);
    exec(Y,First,Last,Parent,InWords,OutWords,D,Anc).

%prune
exec(prune(Label,Def),First,Last,Parent,InWords,OutWords,D,Anc) :- !,
    prune_exec(Label,Def,First,Last,Parent,InWords,OutWords,D,Anc).

%restrictions
exec({X},Prev,Prev,Parent,Words,Words,D,Anc):-!,
    execRestriction(X,Parent,Words).

%terminals
exec(*X,Here,Next,_,InWords,OutWords,_,D,Anc):-!,
    label(Here,X),
    attach_word(X,Here,InWords,OutWords),
    sibling(Here,Next),
    sem_info(Here,terminal:_).

%empty string
exec(,Prev,Prev,_,Words,Words,D,_,Anc):-!.

%nonterminals
exec(X,First,Next,_,InWords,OutWords,D,Anc):-!,
    label(First,X),
    execPred(X,First,InWords,OutWords,D,Anc),
    sibling(First,Next).

execPred(X,Parent,InWords,OutWords,_,_):-
    toggle(translated_grammar_in_use),
    Rule = ..[X,Parent,InWords,OutWords],
    current_predicate(X,Rule),
    !,
    call(Rule).
execPred(X,Parent,InWords,OutWords,D,Anc):-
    label(Parent,X),
    recorded(X,(X::=Y),_),
    D1 is D+1,
    daughter(Parent,FirstChild),
    exec(Y,FirstChild,_,Parent,InWords,OutWords,D1,[(X::=Y)|Anc]).

prune_exec(Label,Def,First,Last,R,InWords,OutWords,D,Anc) :-
    call(prune(Label,Def,OutDef,R,InWords)),!,
    exec(OutDef,First,Last,R,InWords,OutWords,D,Anc);
    exec(Def,First,Last,R,InWords,OutWords,D,Anc).

execRestriction(X,T,Words):-
    X = ..[Name|Args],
    append(Args,[T,Words],NewArgs),
    A = ..[Name|NewArgs],
    call(A).
    
```

Figure 2. Prolog Code for the Interpreter

APPENDIX E

Determiners, Entities, and Contexts

This paper by Deborah Dahl was presented at TINLAP-3, Las Cruces, New Mexico, January, 1987. It describes problems with certain indefinite noun phrases and argues that a procedure analogous to reference resolution for clauses is required to handle them.

Determiners, Entities, and Contexts

Deborah A. Dahl

Paoli Research Center
SDC/A Burroughs Company (now Unisys)

P.O. Box 517
Paoli, PA 19301

1. Introduction

I am concerned with the relationship between the forms of linguistic expressions, noun phrases in particular, and the discourse entities to which they refer.¹ That is, when does a noun phrase introduce a new referent into the discourse? My concern in particular is to specify the role that the discourse context plays in answering this question. A simple first approach to the relationship between noun phrases and discourse entities might suggest that definite noun phrases refer to entities which are assumed to be mutually known to the speaker and hearer, and indefinite noun phrases refer to entities which are not mutually known, and thus, that discourse context plays no role at all. This discussion will point out problems with this approach for both definite and indefinite noun phrases. I will describe examples where definite noun phrases are used to introduce new referents, and, conversely, where *indefinite* noun phrases do *not* introduce new referents. In the first case, the local focus structure provides a guide to recognising that a new entity is involved, and in the second case, the recognition that no new entity is introduced is based on the given/new status of propositions in the discourse.

I will begin by describing certain definite descriptions that introduce new entities. I will then describe some examples where indefinite descriptions do not introduce new entities. In each case, I will discuss some related processing issues.

I will restrict the current discussion to deal with cases where the mutual knowledge is based on the discourse context, rather than on knowledge that the speaker and hearer bring to an interaction. In the cases of indefinites, I will also restrict my discussion to sentential contexts where an indefinite *could* introduce a new entity; in other words, to *specific* contexts, as distinguished from non-specific contexts as discussed in [Prince1981].

2. Implicit Associates

The case of definite noun phrases that are intended to introduce new discourse entities has been relatively well-researched, in particular by [Hawkins1978, Hawkins1984] Hawkins points out that entities that have a slot/frame relationship with previously introduced entities often have a definite determiner. For example, in

(1) There were loud noises coming from a starting air compressor. The drive shaft was sheared.
it is possible to refer to the drive shaft with a definite noun phrase because of its relationship with the previously mentioned starting air compressor, even though the drive shaft has not been mentioned. This same relationship is described by [Prince1981] as *inferable*, and is also discussed in [Heim1982]. Because we understand the drive shaft mentioned in (1) to be not just any drive shaft but the drive shaft that is part of the air compressor mentioned in the previous sentence, a full understanding of this noun phrase must capture this relationship. The new noun phrase is *implicitly associated* with the *local focus* as described in [Dahl1986], and [Sidner1979]. In (2d),

¹ The research described in this paper was supported in part by DARPA under contract N000014-85-C-0012, administered by the Office of Naval Research, and by a post-doctoral fellowship in Cognitive Science from the Sloan Foundation. I have received helpful comments on this paper from John Dowding, Lynette Hirschman, Marcia Linebarger, Martha Palmer, Rebecca Schiffman, and Bonnie Webber.

for example, the referent for *the paper* seems to be the paper associated with the new package even though there is a previously mentioned entity which matches the noun phrase; that is, the paper in (2b).

- (2) a. A package arrived yesterday.
b. The wrapping paper was beautiful.
c. While I was admiring it, another package arrived.
d. I removed the paper.

After the focus change to *my package*, the associates of the new package seem to be preferred as referents over previously mentioned items, even if the old items had been in focus at one time. This is consistent with Sidner's algorithm.²

3. Specific Attributives

The second main point to be dealt with in this paper is that of indefinite noun phrases in specific contexts, which nevertheless fail to introduce new discourse entities. Most of those who have discussed indefinites seem to have assumed that an indefinite reference in a specific context invariably introduces a new discourse entity. This includes the discussions in [H.Clark1977] and [Heim1982]. However, there is a class of indefinites, which I have called *specific attributives* [Dahl1984], which I claim do not have this function.

Consider the example,

- (3) a. Dr. Smith told me that exercise helps.
b. Since I heard it from *a doctor*, I'm inclined to believe it.

An entity, Dr. Smith, is introduced in (3a), and an indefinite noun phrase, *a doctor*, is used in (3b). It is clear that this noun phrase is not intended to introduce a second doctor into the discussion. This is an example of a *specific attributive*. I use the term *specific* in the sense that a specific reference means that the speaker has a particular individual in mind when s/he uses the indefinite description. It is clear in (3), for example, that the speaker did not hear that exercise helps from some unspecified doctor, but from Dr. Smith.

The term *attributive*, as used by [Donnellan1971], can also be applied to these indefinites, although it was originally suggested only for definites, because the specific identity of Dr. Smith is not relevant to the predication, only Dr. Smith's attribute of being a doctor. (See [Dahl1984.] for detailed arguments about the applicability of this term.)

There are two important issues that must be dealt with in a treatment of specific attributives. First, there is the issue of recognising that the noun phrase in fact is not being used to introduce a new entity. Second, it is necessary to recognise the speaker's purpose in using an indefinite noun phrase, when a definite noun phrase would have been possible. Both of these issues have implications for language generation as well as understanding. For example, in the first case a language generator will have to decide when it is possible to use a specific attributive, and in the second case, it will have to decide whether a specific attributive would be useful in accomplishing its communicative goals.

I have previously suggested Dahl1984 that a specific attributive can be recognised by its occurrence in a proposition that is *given* as in (3), is related to a given proposition by simple entailment as in (4), or is related to a given proposition by a plausible inference, as in (5).

- (4) Mary and Bill both volunteered to walk the dog. Since at least *one person* is willing to walk the dog, we don't have a problem.

²A discussion by [Heim1982] suggests that introduction of a new entity with a definite noun phrase is a violation of a felicity condition, and is therefore to be handled by a repair or accommodation mechanism. Since accommodation mechanisms are typically triggered by the failure of normal processing, Heim's approach suggests that a failure of normal processing would have to occur before a system could recognise that a new referent was being introduced. If *normal processing* means searching through the discourse context for a referent matching the new description, then the example in (2) provides evidence against this position, since the correct processing cannot have been invoked by the failure to find a matching referent in the previous discourse.

(5) A: I'm afraid I miscalculated Jones's insulin dosage.

B: What happened?

A: He died.

B: So, a patient has finally died due to your carelessness. (inference 'Jones is a patient')

Thus, in order to determine when an indefinite introduces a new entity, it is necessary to know whether the proposition in which it occurs is *given* or *new*. For this, we need a representation of the events and situations described in the discourse, which can then be examined in order to determine when a proposition is given or new. Such a representation, of course, will be needed in any case for pronouns or full noun phrases that refer to events and situations. For example, in the PUNDIT text processing system, (described in [Palmer1986]), a representation is built for each event or situation mentioned. A noun phrase like *the failure* in (6) or *it* in (7) can then be recognised as a reference to something previously mentioned.

(6) The starting air compressor failed when the oil pressure dropped below 60 psig. *The failure* occurred during the engine start.

(7) The starting air compressor failed when the oil pressure dropped below 60 psig. *It* occurred during the engine start.

The difference in processing between (6) and (7) on the one hand and specific attributives on the other is that for the specific attributives we are saying that something analogous to reference resolution should be performed on clauses, as well as on noun phrases. That is, we want to ask whether this event has been mentioned before, or can be inferred from something that has been mentioned. If so, we can match corresponding participants so that it is possible to recognise that no new entity is being introduced.³

The second issue raised by specific attributives is the speaker's purpose in selecting an indefinite when a definite would have been possible. This seems to be related to the use of indefinites in general to serve to deemphasise the particular individual referred to while emphasising its general class. In (3), for example, it is not the fact that *this* doctor told me that exercise would help that is relevant, but rather that the person has the property of being a doctor. Notice the contrast between (3) and (8).

(8) a. Dr. Smith told me that exercise helps.

b. Since I did hear it from *the doctor* I'm inclined to believe it.

(8) suggests that there is something special about Dr. Smith in particular that makes this advice reliable, while (3) does not.

To sum up, I have discussed two categories of noun phrases which demonstrate the effects of discourse context on determining whether a new entity is introduced. Implicit associate definites introduce new entities which are related to the local focus. Specific attributives refer to previously introduced entities in given propositions. Minimally, specific attributes have to be recognised, in order to prevent the creation of an extra discourse entity, and this requires a representation of given propositions. In addition, a complete understanding of specific attributes requires a recognition of the speaker's reason for choosing an indefinite when a definite would have been possible.

References

[Dahl1984.....]

Deborah A. Dahl, The Rise of Shared Knowledge. *Penn Review of Linguistics* 8, 1984, pp. 1-14.

³This raises the issue of what discourse goals would be served by repeating something that is already given. There are probably a number of reasons to do this. Investigating them would be an interesting topic for future research.

- [Dahl1984]
Deborah A. Dahl, Recognising Specific Attributives, presented at the 59th Annual Meeting of the Linguistic Society of America, Baltimore, 1984.
- [Dahl1986]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.
- [Donnellan1971]
Keith Donnellan, Reference and Definite Descriptions. In *Semantics*, D.D. Steinberg and L.A. Jakobovits (ed.), Cambridge University Press, Cambridge, 1971.
- [H.Clark1977]
Herbert H.Clark and Eve H.Clark, . In *Psychology and Language*, Harcourt, Brace, Jovanovich, New York, 1977.
- [Hawkins1978]
John A. Hawkins, *Definiteness and Indefiniteness*. Humanities Press, Atlantic Highlands, New Jersey, 1978.
- [Hawkins1984]
John A. Hawkins, A Note on Referent Identifiability and Co-Presence. *Journal of Pragmatics*, 1984.
- [Heim1982]
Irene R. Heim, *The Semantics of Definite and Indefinite Noun Phrases*. Unpublished Ph.D. dissertation, University of Massachusetts, Amherst, MA, September 1982.
- [Palmer1986]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.
- [Prince1981]
Ellen F. Prince, Toward a Taxonomy of Given-New Information. In *Radical Pragmatics*, Peter Cole (ed.), Academic Press, New York, 1981.
- [Sidner1979]
Candace Lee Sidner, Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse, MIT-AI TR-537, Cambridge, MA, 1979.

APPENDIX F

Verb Taxonomy

This appendix by Martha Palmer gives the complete verb taxonomy for the verbs in the CASREP corpus. It then lists each verb's decomposition(s) along with the associated mapping rules and semantic class restriction rules for the semantic roles in the decompositions.

VERB TAXONOMY

ASPECTUAL

aspectual: *run_out, start, begin, occur, continue, remain;*

COMPLEX:

relational: *be, has(symptom), received(usage);*

perceptual: (requires propositional agent) *dictated, indicates, show, revealed, noted, find;*

propositional:

(degree of certainty): *believe, suspect, guarantee;*

(requires animate agent): *conducting, discovered, experienced, report, received3(report, alarm);*

causal: *activate, cause, result, due;*

(partial causality): *impact, contribute*

not_causal: *undetermined*

status: *required, request, receive, report;*

BASIC:

contact: *disconnected;*

damaged: *sheared, corroded, cracked, eroded, broken, worn, damage, seized;*

include: *change, repair, install, reinstalled, replace;*

location: *comes_from, retained, received2(part), location;*

removed: *cannibalized, replace, remove;*

primitive: *engage, disengage, open, use, pack, manufactured;*

symptom: *surging, clogging, saturated, contaminated, discoloured, overheating, wiped(bearing);*

investigative: *inspect, testing, monitoring, investigation, troubleshooting;*

maintenance: *cleaning, flushing, washing(water);*

operating: *start(motor), jacks_over, rotate, turn, operate, functional;*

inoperative: *inoperative, failed, degradation, restricts, loss(part), aborted(engine_start);*

change_measure: *adjust, maintain;*

lowered: *decrease, drop, loss2(scalar);*

raised: *increase;*

The verbs that appear in bold-face in the verb hierarchy tree are listed below in the order in which they appear in the taxonomy. Each verb is followed by its decomposition and the syntactic mapping rules and semantic class restrictions for the semantic roles in the decomposition. If the decomposition can be further decomposed, that is included as well, with the mapping rules. For example, the last verb is *reveal*,

VERB: **decrease**

followed by its decomposition,

DECOMPOSITION:

decrease

<-

becomeP(loweredP(patient(p)))

and the associated rules for filling in the semantic roles. The patient role can be filled by the subject or the object of the sentence, and the semantic class restriction is that it should be a scalar such as pressure or temperature.

MAPPING AND SEMANTIC CLASS RULES:

syntax: patient <- obj

syntax: patient <- subj

semantics: patient(p) <-

find_type(p,scalar,context)

In addition, *decrease* can be decomposed further. The becomeP, a time operator is removed, and then loweredP is decomposed into two belowP predicates.

DECOMPOSITION:

becomeP(loweredP(patient(p)))

<-

loweredP(patient(p))

loweredP(patient(p))

<-

belowP(goal(f),ref_pt(r)) & belowP(goal(g),source(s2))

The new semantic roles introduced by the belowP predicates have their own mapping rules and semantic class restriction rules.

MAPPING AND SEMANTIC CLASS RULES:

syntax: goal <- pp

semantics: goal(g) <-

check_numeric(g,amount,context)

syntax: ref_pt <- pp

**semantics: ref_pt(r) <-
check_numeric(r,amount,context)**

syntax: goal <- pp

**semantics: goal(g) <-
check_numeric(g,amount,context)**

syntax: source <- pp

**semantics: source(s2) <-
check_numeric(s2,amount,context)**

VERB: be

DECOMPOSITION:

```
be
  <-
be_propertyP(mod(some^predicate^expression))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: mod <- adj
semantics: mod(some^predicating^expression) <-
true
```

VERB: believe

DECOMPOSITION:

```
believe
  <-
believeP(experiencer(e1),theme(t))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: experiencer <- subj
semantics: experiencer(e1) <-
find_type(e1,animate,context)

syntax: theme <- subj
syntax: theme <- obj
semantics: theme(t) <-
find_type(t,prop,context)
```

VERB: reveal

DECOMPOSITION:

```
reveal
  <-
  causeP(instigator(i1),becomeP(knowP(experiencer(e1),theme(t))))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: instigator <- pp
```

```
syntax: instigator <- subj
```

```
semantics: instigator(i1) <-
  find_event_type(i1,investigative,context)
```

```
semantics: instigator(i1) <-
  find_type(i1,mechanical_device,context)
```

```
syntax: experiencer <- pp
```

```
syntax: experiencer <- subj
```

```
semantics: experiencer(e1) <-
  find_type(e1,animate,context)
```

```
syntax: theme <- subj
```

```
syntax: theme <- obj
```

```
semantics: theme(t) <-
  true
```

VERB: seize

DECOMPOSITION:

```
seize  
    <-  
    becomeP(seizedP(patient(p)))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: patient <- subj  
  
syntax: patient <- obj  
  
semantics: patient(p) <-  
    find_subpart_type(p,spin_element,context)
```

VERB: erode

DECOMPOSITION:

```
erode  
    <-  
    erodedP(patient(p))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: patient <- subj  
  
syntax: patient <- obj  
  
semantics: patient(p) <-  
    find_type(p,mechanical_device,context);find_type(p,component,context)
```

VERB: replace

DECOMPOSITION:

```
replace
  <-
  causeP(agent(a1),becomeP(exchangedP(object1(o1),object2(o2))))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: agent <- pp
```

```
syntax: agent <- subj
```

```
semantics: agent(a1) <-
  find_type(a1,animate,context)
```

```
syntax: object1 <- obj
```

```
semantics: object1(o1) <-
  find_type(o1,mechanical_device,context)
```

```
syntax: object2 <- pp
```

```
semantics: object2(o2) <-
  find_type(o2,mechanical_device,context) &
  is_part_of(X,o2,context)
```

DECOMPOSITION:

```
causeP(agent(a1),becomeP(exchangedP(object1(o1),object2(o2))))
  <-
  becomeP(exchangedP(object1(o1),object2(o2)))
```

```
becomeP(exchangedP(object1(o1),object2(o2)))
  <-
  exchangedP(object1(o1),object2(o2))
```

```
exchangedP(object1(o1),object2(o2))
  <-
  missingP(theme(t),source(s2)) & includedP(theme(t),goal(g))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: theme <- subj
```

syntax: theme <- obj

**semantics: theme(o1) <-
true**

syntax: source <- pp

**semantics: source(s2) <-
is_part_of(s2,o1,context)**

syntax: theme <- subj

syntax: theme <- obj

**semantics: theme(o2) <-
true**

syntax: goal <- pp

**semantics: goal(s2) <-
is_part_of(s2,o2,context)**

VERB: start

DECOMPOSITION:

```
start
      <-
causeP(agent(a1),becomeP(operateP(actor(a2))))
```

MAPPING AND SEMANTIC CLASS RULES:

syntax: agent <- pp

syntax: agent <- subj

semantics: agent(a1) <-
find_type(a1,animate,context)

syntax: actor <- subj

syntax: actor <- obj

semantics: actor(a2) <-
find_type(a2,mechanical_device,context)

VERB: rotate

DECOMPOSITION:

```
rotate
      <-
rotateP(actor(a2))
```

MAPPING AND SEMANTIC CLASS RULES:

syntax: actor <- subj

syntax: actor <- obj

semantics: actor(a2) <-
find_type(a2,spin_element,context)

VERB: operate

DECOMPOSITION:

```
operate
  <-
  causeP(agent(a1),becomeP(operateP(actor(a2))))
```

MAPPING AND SEMANTIC CLASS RULES:

syntax: agent <- pp

syntax: agent <- subj

semantics: agent(a1) <-
find_type(a1,animate,context)

syntax: actor <- subj

syntax: actor <- obj

semantics: actor(a2) <-
find_type(a2,mechanical_device,context)

VERB: fail

DECOMPOSITION:

```
fail
  <-
  becomeP(inoperativeP(patient(p)))
```

MAPPING AND SEMANTIC CLASS RULES:

syntax: patient <- subj

syntax: patient <- obj

semantics: patient(p) <-
find_type(p,mechanical_device,context)

VERB: inspect

DECOMPOSITION:

```
inspect  
  <-  
  inspectP(actor(a2),theme(t))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: actor <- subj
```

```
syntax: actor <- obj
```

```
semantics: actor(a2) <-  
  find_type(a2,animate,context)
```

```
syntax: theme <- subj
```

```
syntax: theme <- obj
```

```
semantics: theme(t) <-  
  true
```

VERB: decrease

DECOMPOSITION:

```
decrease
  <-
  becomeP(loweredP(patient(p)))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: patient <- subj
syntax: patient <- obj
semantics: patient(p) <-
  find_type(p,scalar,context)
```

DECOMPOSITION:

```
becomeP(loweredP(patient(p)))
  <-
  loweredP(patient(p))

  loweredP(patient(p))
    <-
    belowP(goal(f),ref_pt(r)) & belowP(goal(g),source(s2))
```

MAPPING AND SEMANTIC CLASS RULES:

```
syntax: goal <- pp
semantics: goal(g) <-
  check_numeric(g,amount,context)
```

```
syntax: ref_pt <- pp
semantics: ref_pt(r) <-
  check_numeric(r,amount,context)
```

```
syntax: goal <- pp
semantics: goal(g) <-
  check_numeric(g,amount,context)
```

```
syntax: source <- pp
```

semantics: source(s2) <-
check_numeric(s2,amount,context)

APPENDIX G

Conjunction in Meta-Restriction Grammar

This paper, by Lynette Hirschman, appeared in the *Journal of Logic Programming*, Vol. 4 (299-328). It describes the handling of conjunction in Restriction Grammar, based on the use of meta-rules. The paper describes how a number of complex conjunction problems are handled, including the scoping problems for conjoined nouns, the "comma" conjunction problem, and paired conjunctions such as *both...and*.

CONJUNCTION IN META-RESTRICTION GRAMMAR

LYNETTE HIRSCHMAN

- ▷ This paper describes Meta-Restriction Grammar for parsing coordinate conjunction in English. Meta-Restriction Grammar consists of Restriction Grammar, a logic grammar implementation of Sager's String Grammar, plus a metagrammatical component that automatically rewrites "base" grammar rules into more complex rules to handle coordinate conjunction. The approach resembles Sedogbo's approach of "empty elements" or "holes." This avoids the combinatorial explosion due to backtracking in the treatment of Woods, Sager, and Dahl and McCord. Restriction Grammar is well suited to metagrammar extensions, because the absence of parameters in grammar rules facilitates the statement of metarules. The metagrammatical component generates grammar rules specifying allowable conjoinings at limited types of nodes, to reduce redundancy. Meta-Restriction Grammar represents both the surface structure and a regularized structure (via pointers to elided elements) for efficient computation of selectional restrictions. This approach is sufficiently powerful to handle a number of complex phenomena, such as conjunction with comma (as distinguished from the appositive construction), paired conjunctions such as *both ... and*, *either ... or*, and scoping of left noun modifiers under conjunction. One of the great attractions of the metagrammar approach is that the grammar can be translated and *compiled*, resulting in an efficient treatment of conjunction (parse times of 1 to 3 seconds per parse). This contrasts with the interrupt-driven approach, where an interpreter generates rules for conjoining structures on demand, making it impossible to compile the complete grammar. ◁

Address correspondence to Lynette Hirschman, System Development Corporation, P.O. Box 517, Paoli, Pennsylvania 19301.

Received 1 April 1985; accepted 28 January 1986

THE JOURNAL OF LOGIC PROGRAMMING

©Elsevier Science Publishing Co., Inc., 1986
52 Vanderbilt Ave., New York, NY 10017

0743-1006/86 \$03.50

*System Development Corporation is now Unisys Corporation

1. INTRODUCTION

Conjunction has long been a major problem for natural-language processing systems. Conjunction introduces scoping problems of adjuncts relative to the conjoined elements, problems with null (elided) elements, and generally a potential combinatorial explosion of parses. Because of these difficulties, there have been few attempts to treat the problem in its full generality.

Among the earliest (and most linguistically complete) efforts was the conjunction mechanism of the Linguistic String Parser (LSP) [10, 9]. Another early treatment was the *sysCONJ* approach of Woods [14]. In both of these systems, the goal was a general treatment of conjunction that did not require an enormous proliferation or duplication of rules in the grammar. This was done by a general interrupt mechanism activated by recognition of conjunction words, such as *and*, *but*, or the *conjunction comma*, as in *apples, oranges and bananas*. Once such a word was recognized, normal parsing was suspended; a portion of the definition under construction was dynamically copied to accommodate the conjoined structure, at which point normal parsing was resumed. The resulting structure (parse tree) reflected the scoping of the conjunction with respect to other elements in the sentence, in particular, left and right adjuncts.

The advantage of these treatments was that one general "metarule" was sufficient to generate definitions for all conjoined structures. Also, in the LSP system, the handling of conjunction was made largely transparent to the grammar writer by invoking a special set of routines that automatically detected conjoined elements and then iterated restrictions over all conjoined elements.

The interrupt-driven approach has difficulty in controlling redundancy, as well as in controlling backtracking. Due to the generality of the conjunction mechanism, many redundant parses can be generated unless types of conjoining are severely restricted. The LSP solution to controlling redundancy has been to restrict conjoining to several types of nodes and to insert various restrictions to remove other redundancies. The top-down, backtracking strategy also results in inefficient parsing, especially when parsing multiple conjunctions and conjunctions within prepositional phrases.

More recently, there have been several treatments of conjunction within the framework of logic programming [2, 13]. Both of these works are based to some degree on the earlier approaches of Sager, Raze, and Woods. The work of Sedogbo requires the grammar writer to modify each relevant grammar rule in two ways: first, a predicate *CONJ* is added to generate a conjunction option for each rule where conjunction can occur; second, each element that may be omitted under conjunction has an empty option, *hole*, added to it. The Dahl-McCord approach uses an interrupt-driven interpreter, triggered by recognition of conjunction words, to generate appropriate conjunction options.

The present paper describes a metalogic grammar influenced by the metagrammatical approach of Generalized Phrase Structure Grammar (GPSG) [4]. Meta-Restriction Grammar provides an explicit metagrammatical component that automatically generates BNF definitions to parse conjunction. It is based on the Restriction Grammar framework [6, 7], which, in turn, is a logic implementation of Sager's String Grammar [5, 12]. Meta-Restriction Grammar uses a compact metagrammatical component to rewrite certain "base" rules of the grammar into more complex rules for handling conjunction. Because Restriction Grammar does

not allow parameters within BNF definitions, it is extremely well suited to a metagrammar approach. In this respect, it extends the treatment of conjunction proposed by Sedogbo [13]: it builds conjunctions only at certain types of nodes, and it uses "null elements" ("holes") to avoid the combinatorially explosive backtracking approach of Woods, Sager, and Dahl and McCord.

An important feature of Meta-Restriction Grammar is its ability to represent both the surface structure and a regularized structure. It does this by using unification to set a pointer from a gap to the filler of the gap; this preserves the scoping relationships, but also provides fast access to the filler, so that syntactic and selectional restrictions can be computed easily. Another important advantage of the metagrammar approach is that the grammar can be translated and *compiled* [3]. This contrasts with the interrupt-driven approach, where an interpreter generates rules for conjoining structures on demand, making it impossible to compile the complete grammar.

The remaining sections of the paper will describe the implementation of Meta-Restriction Grammar and its solution to the problem of coordinate conjunction. Section 2 presents a brief overview of the Restriction Grammar implementation, followed by an introduction to the String Grammar formalism. Section 3 outlines a wide range of conjunction problems and how they are treated in Meta-Restriction Grammar, including paired conjunction such as *both...and*, comma conjunction, elision of elements under conjunction, and distribution of left and right modifiers. This is followed by a brief section on implementation and a conclusion. Four appendices show various facets of the Meta-Restriction Grammar system: Appendix A contains a listing of the BNF definitions for a medium-coverage grammar of English; Appendix B contains a description of some of the basic data structures, restriction operators, and routines underlying Restriction Grammar; Appendix C contains the metagrammar for generating conjunctions, including all of the conjunction restrictions; finally Appendix D shows some sample sentences with conjunction which were parsed with the grammar in Appendices A-C.

2. RESTRICTION GRAMMAR

Restriction Grammar is a grammar-writing framework in PROLOG. It is derived from Sager's String Grammar [5, 12], which uses context-free BNF definitions, augmented by *restrictions* or constraints on the shape of the parse tree. Aside from any theoretical considerations about logic grammars, Restriction Grammar is useful simply because a very comprehensive English grammar exists in this framework [12]. As a member of the class of logic grammars, Restriction Grammar has several characteristics that distinguish it from definite-clause grammars (DCGs) [8]. First, the parse tree is automatically constructed by the interpreter, to reflect the context-free definitions successfully applied during sentence analysis. This contrasts with DCGs, where the grammar writer is responsible for specifying a parse tree by use of parameters. (Other logic grammar formalisms, such as Modifier Structure Grammar [2], also provide automatic generation of analysis trees.) As in DCGs, the non-context-free portion of the grammar is provided by *restrictions* or constraints. Moreover, the restrictions in Restriction Grammar obtain the contextual information by examining the partially built parse tree or by inspecting the input word stream, rather than from additional parameters to the BNF definitions, as in DCGs.

There are several important advantages to eliminating explicit parameters from the BNF definitions. One advantage is the increased compactness and readability of the BNF definitions and their associated restrictions, since the BNF definitions do not become cluttered with numerous parameters (see Appendix A for the BNF portion of a running grammar covering a moderate subset of English). Although restrictions, when used in the BNF definitions, have no explicit parameters, all restrictions do, in fact, have parameters in Restriction Grammar, namely their starting point in the tree and the current word list; but the interpreter hides these from the grammar writer during the formulation of the BNF definitions. The second advantage of hiding parameters is that it simplifies the metagrammar enormously. The metagrammar which generates the new rule for conjunction is extremely compact, consisting of two rules: one for strings, and the other for head-plus-modifier structures (see Appendix B).

One drawback of Restriction Grammar is that it requires extra machinery for its execution, whereas DCGs require only a minimal interpreter supported directly in the PROLOG implementation. However, this extra machinery need not lead to a loss of performance. We have recently completed the implementation of a flexible translator for Restriction Grammar [3]. The translator converts each grammar rule into a PROLOG clause very similar to a DCG clause: the head consists of a parameterized BNF definition, while the body consists of conjunction and/or disjunction of further parameterized definitions and constraints. The resulting translated code is then compiled by the PROLOG compiler. Our flexible translator provides additional efficiency by supporting dynamic rule pruning based on the input word stream. This is done via mutual recursion between the translated code (where no dynamic interaction is required) and interpreted code (for dynamic rule pruning). The flexible translator coupled with dynamic rule pruning produces a six fold speedup over the interpreted version; parse times for most sentences are in the range of 1-3 seconds, including sentences with conjunction (see Appendix D).

A limitation on Restriction Grammar is that the parse tree reflects only the surface structure analysis. However, the Restriction Grammar execution mechanism keeps a pair of parameters for the construction of a separate semantic representation during parsing. Restriction Grammar itself makes no constraint on the type of representation constructed by the semantic component. Our present semantic representation (not discussed here) is assembled by a special set of restrictions, however, we are in the process of implementing a compositional semantics based on lambda conversion.

It is actually an advantage to decouple the semantic representation from the syntactic representation of the input. First, the syntactic structure is available for those phenomena that are influenced by surface structure (e.g., analysis of focus, or the interaction of conjunction and *wh*-constructions). Second, separation of semantic representation from syntactic analysis provides a more modular system and facilitates experimentation with alternative styles of semantic representation (e.g., predicate-logic expressions, or lambda notation).

The Restriction Language

An execution mechanism controls the application of BNF definitions and associated restrictions. Restrictions are applied at the point at which they appear in a BNF

definition, that is, after the node to the left has been constructed, and before the node to the right has been created. Each restriction imposes some constraint on the parse tree or on the incoming word stream. For restrictions which examine the parse tree, the starting point is the parent node (left-hand side of the BNF definition). No modifications to the basic mechanism have been required (for either interpreter or translator) in order to support conjunction.

Restrictions follow the layered implementation strategy of the Linguistic String Project system. *Primitive tree relations* support *restriction language operators*, which are used to build syntactically motivated routines such as *head*, *adjunct*, *main verb*, etc. The routines, together with the lower-level operators, are used to implement the actual restrictions.

The lowest level primitive tree relations are supported by a data structure of the form `link(TreeTerm, Path)`, where `TreeTerm` represents the current location (a node) in the parse tree, and `Path` is a set of directions from `TreeTerm` back to the root node. `TreeTerm` is a recursive data structure containing four fields:

`tt(Label, Child, RightSib, Word)`.

Here, `Label` is the name of the node as given by the left-hand side of the BNF definition; `Child` and `RightSib` are themselves `TreeTerms` which represent the first child and immediate right sibling of the current node in the tree, and `Word` is the word field containing the lexical item and its attributes. The `tt` (tree term) structure provides the ability to find daughter and (right) sibling nodes in a tree.

The `Path` data structure consists of the functor `up` or `left`, with two arguments, namely the node reached by going one node up or left, and the remainder of the path as the second argument:

`link(TreeTerm, up(Parent, ParentPath))`.

`link(TreeTerm, left(LeftSib, LeftSibPath))`.

The four primitive tree relations are shown in Figure 1. These relations assume that the first argument (the current location) is instantiated; during execution, the second argument is instantiated to the new location (child/right sib/parent/left sib). The daughter (down) or right sib (right) node of the current node is given by the appropriate field of the current node's `TreeTerm`; the `NewPath` from a daughter (or right sib) is returned as `up(TreeTerm, Path)` (or `left(TreeTerm, Path)`). The

FIGURE 1. Basic tree relations

```
down(link(TreeTerm, Path), link(NewTreeTerm, up(TreeTerm, Path)))
TreeTerm = tt(_NewTreeTerm, _), movevar(NewTreeTerm)

right(link(TreeTerm, Path), link(NewTreeTerm, left(TreeTerm, Path)))
TreeTerm = tt(_NewTreeTerm, _), movevar(NewTreeTerm)

up(link(_up(NewTreeTerm, NewPath)), link(NewTreeTerm, NewPath))
movevar(NewTreeTerm),
up(link(_left(NewTreeTerm, NewPath)), Parent)
movevar(NewTreeTerm), up(link(NewTreeTerm, NewPath), Parent)

left(link(_left(NewTreeTerm, NewPath)), link(NewTreeTerm, NewPath))
movevar(NewTreeTerm)
```

parent (up) or left sib (left) node is found from the Path of the current node: in the case of parent, it may be necessary to move through all siblings until the parent is reached.

These relations have a procedural flavor, inherited from the implementation of Restriction Language in the Linguistic String Project system [11]. They have been implemented in their present form in order to permit easy reuse of LSP restrictions by preserving the functionality of Restriction Language. The higher levels of Restriction Language are more syntactic in motivation and correspondingly less procedural. As our system diverges from the LSP system, we plan to replace these low-level procedural operators with their nonprocedural counterparts (e.g., child/parent, sibling).

In addition to the tree-examination primitives, there are the label(Node, Name) relation between a node and its name (given by the label field in the tree term), and the word(Node, Word) relation between a node and the word(s) it subsumes (given by the word field of the tree term). Layered on top of the basic tree relations are the restriction operators [11]. Some of these operators are described in Figure 2. Routines are built up from the elementary restriction operators and other routines; a few are described in Figure 3. The elementary restriction operators and the routines provide a modular framework and allow the grammar writer to capture important linguistic generalizations within the string grammar framework.

FIGURE 2. Primitive restriction operators.

lookahead: scans the word stream for a particular word;
empty: checks a node to test if it is empty;
ascend: ascends to the node (type) given in argument 1,
 passing through nodes (or node types) listed in argument 2,
 not passing through nodes (or node types) listed in argument 3,
 starting at the node in argument 4,
 terminating at the node in argument 5;
descend: does a breadth-first descent through the parse tree.
 with the same five arguments as *ascend*;
word: examines the current word in the word stream;
next: examines the next word in the word stream.

FIGURE 3. Syntactic routines.

core(Start, Head):
 finds the linguistic Head of the construction
 dominated by the node Start.
left_adjunct(Start, LeftAdjunct)/right_adjunct(Start, RightAdjunct):
 finds the left/right adjunct of the construction in which Start occurs
head(Start, Head):
 given that Start is within an adjunct.
 finds the Head which the adjunct modifies
get_verb(Start, Verb):
 starts from the assertion and locates the main verb under assertion
get_obj(Start, Object):
 starts from the assertion and locates the object of the main verb

String Grammar Concepts

String Grammar distinguishes two classes of structures: *exocentric* ("headless") constructions, and *endocentric* constructions (constructions with a head or center). The distinction between endocentric and exocentric constructions is taken from Harris's early work on string analysis [5], but has its root in classical structural linguistics, e.g. in Bloomfield [1].

The constituents of an *endocentric* construction form a phrase of the same category as the *head* of the construction; for example, *the old disks* is a noun phrase, whose head is the noun *disks*. The constituents of an *exocentric* construction form a phrase of a *different* category than the categories of its constituents. An assertion, for example, consists of a noun phrase plus a verb phrase, but is not of the same category as either. Similarly, a prepositional phrase is made up of a preposition and a noun phrase; it is considered to be neither a noun phrase nor a preposition but rather a distinct entity whose distribution differs from nouns and prepositions.

In string analysis, exocentric constructions are called *strings* (it is this notion of string that gives its name to the *Linguistic String Project*). A string consists of *two or more* obligatory elements plus optional string adjunct (*sa*) elements. For example, an assertion can be defined as having obligatory elements subject + verb + object, with interspersed *sa*'s; a prepositional phrase (*pn*) is also a string, with obligatory preposition *p* plus noun string object (*nsigo*):

assertion :: = sa, subject, sa, verb, sa, object, sa.

pn :: = *p, nsigo. (* indicates a terminal symbol)

In Restriction Grammar, the endocentric constructions are called *lrx* constructions, derived from LSP terminology for left-adjunct + x (head) + right-adjunct. An *lrx* construction consists of a (possibly empty) left adjunct, a head, and a (possibly empty) right adjunct. This is used to define noun phrases, adjective phrases, and various forms of verb plus associated modifiers (tensed verb *tv* in the *ltvr* construction, present participle *vng* in the *lvng*, past participle *ven* in *lvenr*, and infinitive *v* in *lvr*). These endocentric constructions are shown in Figure 4.

Restriction Grammar, following the LSP implementation, provides a mechanism for grouping nodes together into syntactically motivated *types*. Some of the major types are shown in Figure 5. These are atomic (terminal) elements, strings, *lrx* constructions, adjuncts (*adjset*), and left/right adjuncts (*ladjset/radjset*).

Using the type definitions and the related notions of *lrx* and *string* structures, we can define a number of additional routines which capture basic linguistic relationships. For example, the routine *core* finds the head of an *lrx* construction; *get_verb* finds the main verb under an assertion; and *get_obj* finds the complement of the main verb under an assertion. These last two operations are nontrivial because the

FIGURE 4. *lrx* constructions.

lnr :: = ln, nvar, rn.	(where nvar becomes a noun or pronoun)
lar :: = la, *adj, ra.	(* indicates a terminal symbol)
ltvr :: = lv, *tv, rv.	
lvng :: = lv, *vng, rv.	
lvenr :: = lv, *ven, rv.	
lvr :: = lv, *v, rv.	

```

type(atom(d,n,int,null,t,adj,ven,pro,rv,nullobj,v,ving,q,nulln,nullwh,w)).
type(string(assertion,pa,apa,nrep,thats,tovo,veno,vingo,venpass)).
type(adjset(sa,ln,lv,rn,rv,la,ra)).
type(ladjset(ln,lv,la)).
type(radjset(rn,rv,ra)).
type(lxr,lxr,lvr,lvr,lvngx,lvrnx,lar)).

```

FIGURE 5. Selected type lists.

grammar handles all verbs, including auxiliaries, uniformly in terms of strict subcategorization for possible complement types. Thus a verb such as *have* has as its complement types the past-participle construction (*veno* = *past participle* + *object*), as well as the direct-object (*nstgo*) construction. Similarly, the verb *be* has the progressive construction (*vingo* = *present participle* + *object*) as a complement type, and also passive (*venpass* = *past participle* + *passive object*), as well as *objectbe*, which contains predicate adjectives, predicate nominals and predicate adverbials. The subcategorization is applied to allow only appropriate objects for each verb. The result is a very uniform handling of verb complements, but also a "nested" complement structure, where an *object* node may well contain a participial form of a verb (the main verb of the sentence) and its complement. Appendix A shows the BNF definitions for such objects containing participial forms of the verb.

The distinction between endocentric and exocentric constructions is central to String Grammar and is reflected in the specifics of the grammar described here. However, our general approach to conjunction is largely independent of String Grammar theory: if a particular theory categorizes certain constituents differently (e.g., prepositional phrases as endocentric, with the preposition seen as a type of case marker), this could be accommodated with minor changes in BNF definitions and specific constraints; it would not affect the general conjunction mechanism.

3. TREATMENT OF CONJUNCTION

The goal of our treatment of coordinate conjunction is to provide a compact, efficient, linguistically motivated treatment of conjunction. Ideally, this treatment should be transparent to the grammar writer, so that treatment of conjunction can be separated from a statement of general language rules.

The overall approach of Meta-Restriction Grammar resembles closely that taken by Sedogbo [13]. In Sedogbo's treatment, a conjoined structure is accommodated by duplicating the entire preconjunction structure following the conjunction. To account for the elision or reduction that may take place under conjunction, certain elements are designated as null elements ("holes" in Sedogbo's terminology). This eliminates the problem, for example, of separately generating three distinct rules to account for reduced subject, reduced verb, or reduced object, in the cases shown in Figure 6. (Parses for these sentences are shown in Appendix D.)

Meta-Restriction Grammar uses a variant on this approach. Rather than copying the conjoined structure, the metagrammar copies the definition. This means that all options and restrictions of the original definition are available for the conjunct. A "gap" created by reduction under conjunction appears as a special null element, *nullc*, in the parse. Each gap keeps a record of the corresponding explicit element by

The field engineer replaced the board and adjusted the disk drive.

missing subject in second clause

The field engineer installed a board and the supervisor a disk drive.

missing verb in second clause

The field engineer has installed and the supervisor has adjusted the disk drive.

missing object in first clause

FIGURE 6. String conjunction examples.

setting a pointer to this element;¹ the pointer is kept in the word field of the tree term associated with the gap. This makes the implicit "fillers" of the gaps available for subsequent semantic and syntactic restrictions, but distinguishes these implicit elements from elements explicitly present in the input word stream.

Preservation of scoping information is critical to a correct treatment of conjunction. For example, in the sentence *Everyone takes the bus or drives a car*, the subject of *takes the bus and drives a car* is *everyone*, for purposes of agreement, selection, etc. However, it is important that there be only one copy of *everyone*, with logical scope over the conjoined assertions: *for all persons X, X takes the bus or X drives a car*. The responsibility of the syntactic component is to preserve this scope information while applying all syntactic constraints. The syntax uses the pointer from filler to gap, together with a special restriction, to generate (via backtracking) all syntactically consistent permutations of these scoping relations; it is left to semantics to determine which is the correct scoping of adjuncts.

Proper integration of restrictions and conjunction is important. The LSP system employed an elegant resumption mechanism [9] to handle the interaction of restrictions with conjunction. A special routine detected the existence of conjuncts and placed them on a *resumption stack*; when one conjunct had been examined, the stack was popped to yield the next conjunct, and the restriction was reapplied. The Meta-Restriction Grammar approach does not provide automatic resumption for conjunction at this time. However, by setting pointers to the implicit information, it does provide a more regularized structure for subsequent restrictions. For example, by modifying the routine *core* to look in the word field of a null node for implicit information, all restrictions involving the head of a construction work equally well on explicit or implicit information. This illustrates the modularity of the approach and the advantage of using general routines (such as *core*) to access information in appropriate structures.

As discussed in the previous section, Restriction Grammar distinguishes two classes of structures: exocentric constructions or *strings*, and endocentric constructions or *lx* constructions. Since these raise somewhat different problems, we will discuss them separately.

Conjoining of Strings

A string consists of two or more obligatory elements and optional string adjunct (*sa*) elements. Conjunction within a string is handled by a metarule (shown in Figure 7) that allows the optional addition of a conjunct at the end of the string.

¹Of course in a logic program this merely involves unifying an uninstantiated variable in the node representation of the gap to the tree term representing the explicit element.

```

generate_conj_str(STRING):-
    retract(STRING :: = Rule),
    assert((STRING :: = [either], Rule, [or], sa, STRING)),
    assert((STRING :: = Rule, ((conj_wd, sa, STRING,
        (wconj3), (wconj4), (wnullObj)),
        null))), !.

```

FIGURE 7. Metarule to generate conjunction in strings.

followed by a new string and some additional constraints specific to conjunction:

```

assertion :: = [either], sa, subject, sa, verb, sa, object, sa, [or], sa, assertion.
assertion :: = sa, subject, sa, verb, sa, object, sa,
    ((conj_wd, sa, assertion, (wconj3), (wconj4), (wnullObj)),
    null).

```

The rule `generate_conj_str` shown in Figure 7 is called by a rule which instantiates the variable `STRING` for each definition of type *string*; `generate_conj_str` removes the original version of `assertion` and replaces it by the two rules, for two different cases of conjunction. The metarule also adds the conjunction-specific restrictions (`wconj3`), (`wconj4`), and (`wnullObj`), which are explained in Figure 9.

In addition to replacing nonconjunction rules with rules containing conjunction, there is an additional set of definitions required for handling conjunction. These are shown in Figure 8; the restrictions in these definitions are explained in Figure 9. These include a definition for the conjunction itself (`conj_wd`), which can be either a conjunction or a comma followed by a conjunction. The term *spword* ("special word") used as the conjunction word class is terminology inherited from the LSP's original interrupt-driven mechanism, which recognized "special words" to trigger the interrupt. The *spword*s include *and*, *or*, *but*, *as well as*, etc. The remaining definitions in Figure 8 provide for elision of elements under string conjunction via the `null` option. These definitions also contain restrictions to control elision.

There are several issues peculiar to the conjunction of string structures. One is that the required elements of a (nonconjoined) string are nonempty. However, under conjunction, one of these required elements (*subject*, *verb*, *object*) may be reduced, as illustrated by the sentences of Figure 6. This general approach applies equally to conjunction under the complex objects, accounting for sentences such as:

The field engineer hopes to install the drive and to replace the board.

The field engineer plans to install but not to adjust the head.

The supervisor has installed and the field engineer will adjust the disk.

FIGURE 8. Additional BNF definitions to handle conjunction.

```

subject :: = (dnullsub), null, (wnullsub)
verb :: = (dnullverb), null, (wnullverb)
object :: = (dnullobj), null
null :: =
conj_wd :: = [','], *spword
conj_wd :: = (dconj2), *spword

```

dconj2:
if present word is comma, allows conjunction only if
there is a "real" conjunction ahead, skipping over the
next word (to avoid taking comma as conjunction in ", and").

wconj3:
checks that if *conj_wd* is comma, then there is a noncomma
conjunct ahead.

wconj4:
if verb is nullc, then both subject is not nullc, and object is not empty.
and it is within a conjunction

dnullsubj:
checks to make sure subject is under a conjunction

wnullsubj:
allows nullc in subject if in a conjunction;
also sets pointer to subject tree from previous conjunct.

dnullverb:
checks to make sure verb is under a conjunction

wnullverb:
allows verb to be nullc if in a conjunction;
also sets pointer to verb tree from previous conjunct

dnullobj:
checks to make sure that the next word is a conjunction

wnullObj:
if object is nullc,
then locates the main verb & checks that it is compatible with the object;
sets pointer to explicit object tree in following conjunct

FIGURE 9. String conjunction restrictions.

Restrictions are generally divided into two classes: *disqualify* restrictions (whose names begin with a *d*); and *well-formedness* restrictions (whose names begin with a *w*). *Disqualify* restrictions apply before a node is built, to determine whether the appropriate environment exists for applying a particular rule. *Well-formedness* restrictions apply after a node and all its children have been completed. These restrictions check consistency, for example, between a verb and its object, or subject verb agreement. In the case of conjunctions, they have another important function, namely setting the pointer of the *nullc* (gap) node to the corresponding explicit (filler) information. The set of restrictions for string conjunction is shown in Figure 9.

Conjoining *lxr* Structures

Conjunction of *lxr* nodes presents a different set of problems. The *lxr* nodes are characterized by having one essential element (the head) and left and right adjuncts which may be empty or filled. The principal problem in handling conjoined *lxr* elements is to indicate the proper distribution of the adjuncts over the conjoined elements. The phrases in Figure 10 illustrate this problem.

Because adjuncts may be empty in the normal course of parsing, we chose not to generate special *nullc* elements for them, but simply to let them take on null values when reduced under conjunction. Thus *nullc* is an option reserved for essential elements in a string. Adjuncts take on the value null, which can be updated later to include a pointer to the elided information, if it turns out that they are reduced

- the last replacement and adjustment*
 = the last replacement and the last adjustment
 distribution of adjuncts over both conjuncts
- she has had the measles and mumps*
 = she has had the measles and she has had mumps
 no distribution of article over second conjunct
- they quickly replaced and adjusted the controller.*
 = they quickly replaced the controller and
 they quickly adjusted the controller
 distribution of adverb over both conjoined verbs
- due to a bad circuit board or wire in the disk drive*
 = due to a bad circuit board or due to a bad wire ?? in the drive
 ambiguous: in the drive may distribute over one or both conjuncts.

FIGURE 10. Examples of conjoined *lxr* structures.

under conjunction. This avoids having to decide prematurely whether an element is empty because it is really empty, or empty because it has been reduced under conjunction.

The examples of Figure 10 illustrate that conjunction over *lxr* constructions can often be ambiguous from a strictly syntactic point of view. The constructions can only be disambiguated by the use of semantic information. Therefore the job of the syntactic component is to generate *all* possible readings, so that the semantic component can select the correct one. Ideally, of course, the syntactic and semantic components are interleaved, so that the choice can be made as soon as possible, to avoid a combinatorial explosion of parses. However, the current treatment of *lxr* conjunctions avoids some of the combinatorial explosion of the LSP or SYSCONJ treatments. The LSP and SYSCONJ treatments produced distinct surface structures corresponding to each distinct distribution of adjuncts. This led to inefficiencies due to large amounts of backtracking when parsing conjunctions. The Meta-Restriction Grammar approach is to confine the problem of adjunct distribution to several final restrictions within the *lxr* node. Figure 11 illustrates the metagrammar rules used to generate *lxr* constructions. Note in particular that the rules governing adjunct distribution, *wconj_lx* and *wconj_rx* (described in Figure 12), are applied only when the final conjunct has been reached; at this point, they are applied once, to the entire conjoined structure. Thus if it turns out later that it is incorrect, backtracking occurs only into one of these restrictions, not into the *lxr* structure itself.

As indicated by the metagrammar rules, there are three restrictions involved in parsing the *lxr* constructions; one of them, *wconj3*, allows comma as a conjunction word only if there is also a "real" conjunction word connecting the final conjunct.

FIGURE 11. Metagrammar rules for *lxr* constructions

```
generate_conj_lxr(LXR) :-
  retract(LXR :- Rule),
  assert(LXR :- [both], Rule,
    (and[aa,LXR, (wconj_lx), (wconj_rx)])),
  assert(LXR :- Rule,
    ((conj_wd[aa,LXR, (wconj3)],
      (wconj_lx), (wconj_rx)))).
```

wconj_lx
 computes distributed elements of *lx* under conjunction;
 This procedure should compute only distinct readings, via backtracking;
 initial reading is distributed reading; backs up into local reading.
 Algorithm climbs to top of conjoined LXR pile,
 and computes distribution pairwise, traversing down the chain.

wconj_rx
 computes the distribution of the right adjunct under conjunction;
 it starts from the lowest (last) pair of conjuncts,
 assigns the higher of the two either a distributed reading,
 or a local reading;
 then finds the next right-adjunct above the current pair,
 and calls itself recursively.

FIGURE 12. Restrictions for parsing *lrx* constructions.

This rule is general to both the *lrx* and string constructions and has been discussed above. The other two have to do specifically with the distribution of left and right adjuncts.

Both *wconj_lx* and *wconj_rx* operate from the final element in a series of conjoined elements. Both operate recursively, computing distribution of adjuncts pairwise, before progressing to the next higher *lx/rx*. However, they traverse the tree in opposite directions. Since the first (higher) *lx* is explicit and the second may be implicit, the algorithm for *lx* climbs to the top of the *lrx* nest, and then traverses the structure recursively downwards. For the *rx*, it is reversed: the lower *rx* is explicit, and the preceding one may be omitted. Therefore, traversal is done from the bottom up.

Both constructions draw on the same routines: if the explicit element is empty, then there is no difference between a distributed reading and a local reading, since there is nothing to distribute, as in *cats and dogs*. In this case, nothing happens and the restriction moves on to the next pair. If the explicit element is *not* empty, then it looks at the other adjunct. If it is filled, then again, there is no distribution of adjuncts possible, as in *the controller and the head*. If, however, the other adjunct is empty, as in *the controller and head*, then there is a possibility of distributing the adjunct; this is done by setting a pointer from the word field of the reduced adjunct slot to the explicitly filled adjunct, as described for string conjoinings. The distributed reading is the first reading generated. On failure, there is a backtrack point, allowing a *local* reading to be generated; this local reading is explicitly marked by inserting a flag *local* into the word field of the empty adjunct. The local reading, however, may not always be allowed, for example, if the lower conjoined noun must share the determiner of the first noun.

The above description applies to the general case of left and right adjuncts, but not to a very important exception, namely the left noun adjunct or *ln*. This is because the *ln* is itself a string, consisting of a series of slots for various types of modifiers:

ln :: = *tpos*, *qpos*, *apos*, *npos*.

These slots are for positions for articles (*tpos*), numerical quantifiers (*qpos*), adjectives (*apos*), and compound noun modifiers (*npos*), as in *the dozen crisp doughnut holes*. Distribution of adjuncts within the *ln* must preserve proper bracket-

ing. That is, in the phrase *the dozen crisp doughnut holes and six oatmeal cookies*, the adjective *crisp* cannot be construed to modify *cookies*, because there is a quantifier *six* "blocking" this reading. Without the second quantifier, *crisp* could modify *oatmeal cookies*: *the dozen crisp doughnut holes and oatmeal cookies*. To capture this "proper bracketing" effect, the code for distribution of *ln* elements is substantially more complex, as is described below: The general restriction *wconj lx* determines whether it is dealing with a left noun modifier, in which case it invokes the specialized code in Figure 13; otherwise, it uses the general algorithm described above. Figures 14 and 15 trace the assignment of adjuncts for the phrase *the last*

FIGURE 13. Algorithm for computing distribution of left noun modifiers.

```
conj_ln:
  computes distributed elements of ln under conjunction, based
  on the following observation:
    if an element of ln (e.g., apos) is "local", then to preserve
    proper scoping, everything to its right must be local.
  Algorithm to compute whether parts of lower ln are in scope of upper ln:
  given an upper conjoined ln and a lower ln.
  1. move through lower ln and if an element is filled,
    mark everything to its right as "local";
  2. locate last element of upper ln and of lower ln;
  3. if lower element neither filled nor local,
    then if upper element is filled,
      then either set pointer to it in lower element
      and mark it and all elements to its left as distributed
      or mark it as local and
      go left in lower & upper lns and repeat step 3
      until can't go left (have assigned scope to all elements)
  This procedure should compute only distinct readings, via backtracking
```

FIGURE 14. Initial assignment of distributed adjuncts.

Trace showing distribution of adjuncts in *ln* for phrase
the last replacement and adjustment of the disk.

Reading: the last replacement of the disk and the last adjustment of the disk

```
** doing wconj_lx in LN
> > marked apos local
apos is nonnull
> > marked apos distributed
      (will mark all elements to its left as "copied")
      (NOTE: "copied" means setting a pointer from gap to filler)
> > True from upper LN after marking elements
```

```
ln
  apos
    l = = the
  apos
    adj = = last
  over
    a = = replacement
    re = = copied po
    conj_wd
      spword = = and
  last
  ln
```

```

tpos == copied the
qpos == copied null
apos == copied last
spos == tagged local
svar
  a == adjustment
ra
  pa
    p == of
    astgo
    astg
    lar
    la
      tpos
      i == the
      svar
      a == disk

```

FIGURE 14. (Continued)

FIGURE 15. Assignment of adjuncts on backtracking into *lnr*.

*Trace showing backtracking into local reading of apos in lnr for phrase:
the last replacement and adjustment of the disk.*

Reading: the last replacement of the disk and the adjustment of the disk

```

> > marked apos local
> > marked qpos local
tpos is non-null
> > marked tpos distributed
> > Tree from upper LN after marking elements

```

```

ln
  tpos
    i == the
  apos
    adj == last
svar
  a == replacement
ra == copied pa
com_wd
  spword == and
lar
  la
    tpos == copied the
    qpos == tagged local
    apos == tagged local
    spos == tagged local
  svar
    a == adjustment
ra
  pa
    p == of
    astgo
    astg
    lar
    la
      tpos
      i == the
      svar
      a == disk

```

The distinction between conjunction at the string level and conjunction at the *lxr* level allows the grammar to avoid the generation of spurious ambiguities. For example, Sedogbo comments that his grammar, unless constrained by restrictions, generates two readings for the sentence *John and Peter sleep*, one with conjunction at the noun-noun level, and one with conjunction at the level of conjoined subjects:

subject (nullc verb + nullobj) and subject verb nullobj.

Coverage

- Paired conjuncts such as *both...and* and *either...or* are handled by the appropriate alternative in the metarules.
- Comma conjunction, such as *I ate apples, oranges, and pears*, is handled by the same mechanism as other conjunction types, except that the comma is allowed as a conjunct only if it occurs in a sequence of conjunctions ending with a "real" conjunction, such as *and*.
- Distribution of left and right modifiers over conjoined head-plus-modifier (*lxr*) constructions is accounted for in full generality.
- Conjunction of *lxr* constructions is accounted for by a single meta-rule e.g.

installed and adjusted

- Conjunction at the string level, including prepositional phrases and complex objects, is handled by a single mechanism e.g. "The cat sat on the mat and they ate well and slept well and they ate..."

At the present time, Marcia Linetberger is working on extending the grammar to cover relative clauses and question conjunction with the relative clause and conjunction. In the straightforward the grammar currently handles the relative clause.

AD-A181 562 INTEGRATING SYNTAX SEMANTICS AND DISCOURSE DARPA
NATURAL LANGUAGE UNDERST. (U) UNISYS CORP PAOLI PA
PAOLI RESEARCH CENTER D DAHL ET AL. 14 MAY 87

AD-A181 562 INTEGRATING SYNTAX SEMANTICS AND DISCOURSE DARPA
NATURAL LANGUAGE UNDERST. (U) UNISYS CORP PAOLI PA
PAOLI RESEARCH CENTER D DAHL ET AL. 14 MAY 87

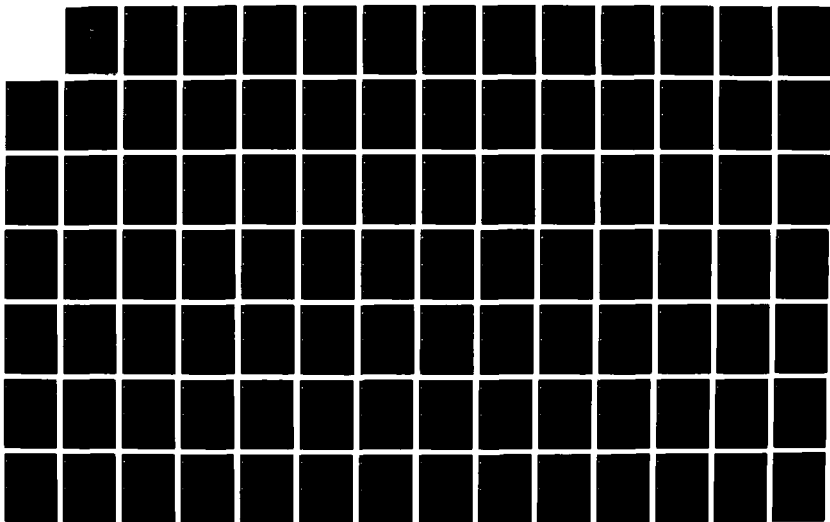
2/3

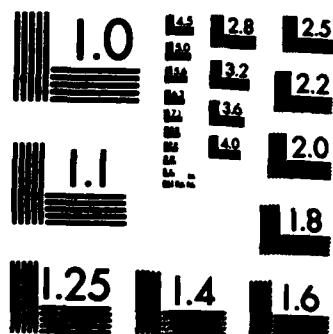
UNCLASSIFIED N00014-85-C-0012

UNCLASSIFIED N00014-85-C-0012

UNCLASSIFIED PAGE 1 RESEARCH CENTER 6 DATE 11 MAR 14 11:01 AM
N00014-85-C-0012 F/G 5/7

UNCLASSIFIED PAGE:1 RESEARCH CENTER 9 DATE: 11 JUL 24 1971 BY NL
N00014-85-C-0012 F/G 5/7





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

1. Conjoined assertions with reduced object in relative clause:
The disk which he installed and she repaired has failed.
2. Conjoined assertions with reduced subject in relative clause:
The disk which was installed properly but was not repaired regularly has been removed.
3. Conjoined *ltvr* in relative clause:
The disk which he repaired and installed has failed.
4. Conjoined *vemo* in relative clause:
The disk she has repaired but not installed is old.
5. Conjoined *wh* question with reduced subject:
What has been installed but has not been repaired?
6. Conjoined *wh* question with reduced subject:
Who has installed or will soon install the disk?
7. Conjoined *yesnoq* with reduced object:
Has he installed or does he intend to install the disk?

FIGURE 16. Coverage of conjunction interacting with relative clauses and questions.

There are two limitations to the current coverage of conjunction: first, we have not yet investigated distribution of sentence adjuncts at the string level; and second, conjunction of subparts of certain complex objects is not yet handled (e.g., *the car drove through and completely demolished the window*). Implementations for both of these have been sketched out and do not appear to impose any major problems.

4. IMPLEMENTATION

The current system is implemented in Quintus PROLOG running on a VAX 11/785 under Berkeley 4.2 Unix. The system also runs in PROLOG on a Symbolics 3640. The parsing times shown in Appendix D are for the translated, compiled grammar plus metagrammar listed in Appendices A and C, running in Quintus PROLOG on a moderately loaded VAX 11/785. This grammar includes some 60 BNF definitions and some 30 restrictions.

5. CONCLUSION

The previous sections have described a broad-coverage treatment of coordinate conjunction within a metagrammatical framework. The emphasis has been on parsing strategies to generate all possible readings. Clearly, the parsing strategy must be coupled with semantic strategies to choose the correct reading, in case of ambiguities. However, the parsing strategy has been carefully designed to eliminate spurious redundancies, as well as to handle some difficult constructions rarely mentioned in the literature (comma as a conjunction, scoping of left noun modifiers, parsing of appositives, treatment of paired conjunctions such as *both ... and*, *either ... or*). The metagrammar treatment offers an unusually compact and efficient method of capturing a wide range of conjunction phenomena.

APPENDIX A. LISTING OF BNF DEFINITIONS AND TYPES

```

/* root node */
rootnode(sentence).

/* bnf definitions */
sentence :: = center, (['.']; ['?']).
center :: = assertion.
tense :: = (lv, *w, rv); null.
assertion :: = sa, subject, { wsel1 }, sa, verb, { wagree }, sa, object, sa.
sa :: = *d; *int; pn; null.
null :: = .
pn :: = *p, nstgo, commaopt.
commaopt :: = ['.']; null.
subject :: = nstg; ( { dquest4 }, nullwh ); { there }.
nullwh :: = .
nstg :: = (lnr, { wcount }); nrep.
lnr :: = ln, { wln }, nvar, { noun_agree }, rn.
nvar :: = *n; *pro; ( { dn2 }, nulln, { wn1 } ).
nulln :: = .
ln :: = tpos, qpos, apos, npos, { np_agree }.
tpos :: = *t; whln; null.
whln :: = ( { whose }; { which }; { what }; howqastg ).
howqastg :: = [how], ( [much ]; { many } ), ( ([of], *t ); null ).
qpos :: = *q; null.
apos :: = *adj; *ven; null.
npos :: = nnn; null.
nnn :: = { dn1 }, ( *n; ( *n, nnn ) ).
rn :: = pn; vingo; venpass; null; appos.
appos :: = ['.'], nstg, ['.'].
nrep :: = { what }, subject, sa, verb, { ww1 }, sa.
verb :: = ltr; lvr.
ltr :: = lv, *tv, rv.
lvingr :: = lv, *ving, rv.
lvenr :: = lv, *ven, rv.
lvr :: = lv, *v, rv.
lv :: = *d; null.
rv :: = pn; *d; null.
object :: = ( dverbobj ), ( nstgo; pn; npn; ( { dsel6 }, objectbe ); veno; nullobj; tovo ),
             { wverbobj }.
nstgo :: = nstg; ( { dwh1 }, nullwh ).
npn :: = nstgo, pn.
objectbe :: = astg; nstg; pn; vingo; venpass; ( { dwh2 }, nullwh ).
astg :: = lar.
lar :: = la, *adj, ra.
la :: = *d; null.
ra :: = pn; null.
vingo :: = { dsel5 }, lvingr, sa, object, sa.
venpass :: = { dsel4 }, lvenr, { wpassobj1 }, sa, passobj, sa.
veno :: = { dsel4 }, lvenr, sa, object, sa.

```

```

nullobj :: = ".
thats :: = [that], assertion.
tovo :: = [to], lvr, sa, object, sa.
passobj :: = (nullobj; pn), { wpassobj2 }.

/* lists */
type(atom, [d, n, int, null, t, adj, ven, pro, tv, nullobj, v, ving, q, nulln, nulle, nullwh, w]).
type(string, [assertion, pn, npn, nrep, thats, tovo, veno, vingo, venpass]).
type(stgseg, [assertion, tovo, vingo]).
type(adjset, [sa, ln, lv, rn, rv, la, ra]).
type(ladjset, [ln, lv, la]).
type(radjset, [rn, rv, ra]).
type(lxr, [lnr, ltr, lvr, lvingr, lvenr, lar]).
type(verbal, [lvingr, lvenr, lvr, ltr, verb]).
type(conj_word, [conj_wd, 'and', 'or']).

```

APPENDIX B. ELEMENTARY DATA STRUCTURES, OPERATORS, AND ROUTINES

% DATA STRUCTURES

```

tt(Label, Child, RightSib, Word).
link(TreeTerm, up(TreeTerm, Path)).
link(TreeTerm, left(TreeTerm, Path)).

```

% MOVEMENT OPERATORS

```

down(link(TreeTerm, Path), link(NewTreeTerm, up(TreeTerm, Path))):-
    TreeTerm = tt( _, NewTreeTerm, _, _), nonvar(NewTreeTerm).

right(link(TreeTerm, Path), link(NewTreeTerm, left(TreeTerm, Path))):-
    TreeTerm = tt( _, _, NewTreeTerm, _), nonvar(NewTreeTerm).

up(link( _, up(NewTreeTerm, NewPath)), link(NewTreeTerm, NewPath)):-
    nonvar(NewTreeTerm), !.
up(link( _, left(NewTreeTerm, NewPath)), Parent):-
    nonvar(NewTreeTerm), up(link(NewTreeTerm, NewPath), Parent).

left(link( _, left(NewTreeTerm, NewPath)), link(NewTreeTerm, New Path)):-
    nonvar(NewTreeTerm).

```

% RESTRICTION OPERATORS

```

lookahead: scans the word stream for a particular word;
empty: checks a node to test if it is empty;
ascend: ascends to the node (type) given in argument 1,
        passing through nodes (or node types) listed in argument 2,
        not passing through nodes (or node types) listed in argument 3,
        starting at the node in argument 4,
        terminating at the node in argument 5:

```

descend: does a breadth-first descent through the parse tree,
with the same five arguments as *ascend*;
wordl: examines the current word in the word stream;
nextl: examines the next word in the word stream.

% ELEMENTARY ROUTINES

element(NodeNames,Start,End):
searches for a node of type *NodeNames*
(a list of node names or the name of a *type* of node)
among the children of *Start*, storing the node in *End*;
l(NodeNames,Start,End)/r(NodeNames,Start,End):
searches for a node of type *NodeNames* to the left/right of *Start*;
last_element(Start,End):
searches for last child under *Start* and stores it in *End*;
last(Start,End):
searches for last sibling (or coelement) under *Start*, storing it in *End*.

% SYNTACTIC ROUTINES

core(Start,Head):
finds the linguistic *Head* of the construction
dominated by the node *Start*.
left_adjunct(Start,LeftAdjunct)/right_adjunct(Start,RightAdjunct):
finds the left/right adjunct of the construction in which *Start* occurs.
head(Start,Head):
given that *Start* is within an adjunct,
finds the *Head* which the adjunct modifies.
get_verb(Start,Verb):
starts from the assertion and locates the main verb under assertion.
get_obj(Start,Object):
starts from the assertion and locates the object of the main verb.

% CONJUNCTION ROUTINES

checkNotEmpty(Node):
checks that a node contains neither an actual word,
nor a copy of a word implicit under conjunction;
copyNullc(EmptyNode,FilledNode)
1. if *FilledNode* has an entry in its word field that is a tree term
(it has already been restored under conjunction—see 2),
then it is stored in the word field of *EmptyNode*;
2. If the word field of *FilledNode* is a word (the normal case),
then the tree term associated with *FilledNode* is copied into
the word field of *EmptyNode*.

APPENDIX C. METAGRAMMAR FOR CONJUNCTION

% META-RULES FOR GENERATING CONJUNCTION STRINGS

```
:-op(1200,xfx,::=).
:-op(950,xfx,:).
:-op(500,fx,*).
```

```

gen_conj :- of_type(lxr,LXR), generate_conj_lxr(LXR),fail.
gen_conj :- of_type(string,STRING), generate_conj_str(STRING),fail.
gen_conj.

generate_conj_lxr(LXR):-
    retract((LXR :: = Rule)),
    assert((LXR :: = [both],Rule,[and],sa,LXR,{wconj_lx},{wconj_rx})),
    assert((LXR :: = Rule,
                ((conj_wd,sa,LXR,{wconj3}):
                 {wconj_lx},{wconj_rx}))),!.

generate_conj_str(STRING):-
    retract((STRING :: = Rule)),
    assert((STRING :: = [either],Rule,[or],sa,STRING)),
    assert((STRING :: = Rule,((conj_wd,sa,STRING,
                                {wconj3},{wconj4},{wnullcObj}):
                                null))),!.

of_type(Type,Member):-
    type(Type,List),!.isin(Member,List).

```

% CONJUNCTION DEFINITIONS

```

subject :: = {dnullsubj},nullc,{wnullsubj}.
verb :: = {dnullverb},nullc,{wnullverb}.
object :: = {dnullobj},nullc.
nullc :: = .
conj_wd :: = ['.', *spword.
conj_wd :: = {dconj2}, *spword.

```

% CONJUNCTION RESTRICTIONS

```

/* dnullsubj
    checks if in scope of conjunction by looking for conjunction
    to the left of the parent (assertion) node
*/
dnullsubj(S,W):-up(S,Assert),l(conj_word,Assert,CW).
/* dnullobj
    checks if in scope of conjunction by checking that next word
    is conjunction word (has attribute "spword")
*/
dnullobj(_S,W):-
    wordl(W,_X:[_Root,spword:_Y]).
/* dnullverb
    checks that verb is within conjunction scope by ascending to nearest
    assertion and checking that it is preceded by a conjunction
*/
dnullverb(S,W):-
    ascend(assertion,_adjset,S,Assert),l(conj_word,Assert,CW).

```

```

/* wnullsubj
    allows subject to be null if in a conjunction;
    also fills in word field with subject tree.
*/
wnullsubj(S,_) :-
% test that immediate node is conj_word
    up(S,Assert),k(conj_word,Assert,Conj),
% locate value of subject.
    k(subject,Conj,Subj), down(Subj,SubjValue),
% locate nullc and set pointer from nullc to subject into word field
    down(S,Nullsubj),copyNullc(Nullsubj,SubjValue).

/* wnullverb
    allows verb to be null if in a conjunction;
    also sets pointer in word field to verb tree.
*/
wnullverb(S,_) :-
% test that immediate node is conj_word
    up(S,Assert),k(conj_word,Assert,Conj),
% check that subject is not nullsubj
    ((element(subject,Assert,Subj),!,not(element(nullc,Subj,_Nullc)))):true),
% locate core of verb, to set pointer from nullverb
    get_verb(Conj,V),down(V,VValue),
% locate nullc and set pointer to verb
    down(S,Nullverb),copyNullc(Nullverb,VValue).

/* wnullcObj
    if object is nullc, then locate the main verb;
    make sure that it is compatible with the object;
    set pointer to explicit object in word field of nullc object.
*/
wnullcObj(S,W) :-
% if object is nullc
    element([object,passobj],S,Obj),down(Obj,Nullc),
    test(nullc,Nullc,Nullc),!,
% then locate the verb
    get_verb(S,V),
% find the conjoined explicit object
    last_coelement(Obj,LowS),
    get_obj(LowS,LowObj),down(LowObj,ObjType),
% make sure that the explicit object goes with the verb
    checkVerbObj(ObjType,V),
% set pointer to it from the word field of the nullc object
    copyNullc(Nullc,ObjType),
wnullcObj(_,_).

/* dconj2
    if present word is comma, allows conjunction only if, skipping
    next word (to avoid taking comma as conjunction in ", and").
    there is a "real" conjunction ahead.

```

```

*/
dconj2(_S,':': X[MoreWords]) :-!,
    nextl(MoreWords,Next),
    lookahead(spword,[andstg,orstg],Next,_Out).
dconj2(_S,_W).

/* wconj3
    checks that if conjstg is comma. then there is a real conjunction ahead.
*/
wconj3(S,W):-
    (element(conj_word,S,C),last_element(C,CWD),not(word(CWD,''))),!,
    (element(lxr,S,LXR),!,wconj3(LXR,W));
    (element(string,S,String),wconj3(String,W)).

/* wconj4
    if verb is nullc, then both subject is not nullc, and object not empty
    and it is within a conjunction
*/
wconj4(S,_):-
    l(conj_word,S,C),!,
    element(verb,S,V),((element(nullc,V,_Nullc),!,element(subject,S,Subj),
    not(empty(Subj))), element(object,S,Obj), not(empty(Obj))):true).
wconj4(_,_).

/* wconj_lx
    computes distributed elements of lx under conjunction:
    this procedure should compute only distinct readings, via backtracking:
    initial reading is distributed reading: backs up into local reading.
    Algorithm climbs to top of conjoined LXR pile,
    and computes distribution pairwise, traversing down the chain.
*/
wconj_lx(LXR,_):-
    topLX(LXR.TopLX),
    last_coelement(TopLX.NextLXR),
    ((test(lxr,NextLXR.NextLXR),!,
    nl,print(' * found next lxr'),
    element(ladset,NextLXR.LowLX),conj_lx(TopLX.LowLX)):
    true).
topLX(LXR.TopLX):-
    l(ladset,LXR,UpLX),!,up(UpLX.NewLXR),topLX(NewLXR.TopLX).
topLX(LXR.TopLX):-
    element(ladset,LXR.TopLX).
conj_lx(UpLX.LowLX):-
    test(ln,UpLX,UpLX),!,
    conj_ln(UpLX.LowLX).
conj_lx(UpLX.LowLX):-
    ((checkNullValue(UpLX),!),distrib_adj(UpLX.LowLX)),
    ((next_pair(LowLX.NewLowLX),!,conj_lx(LowLX.NewLowLX)):true).

```

```

% conj_ln
%   computes distributed elements of ln under conjunction, based
%   on the following observation:
%       if an element of ln (e.g., apos) is "local", then to preserve
%       proper scoping, everything to its right must be local.
%   Algorithm to compute whether parts of lower ln are in scope of upper ln:
%       given an upper conjoined ln and a lower ln,
%       1. move through lower ln and if an element is filled.
%           mark everything to its right as "local";
%       2. locate last element of upper ln and of lower ln;
%       3. if lower element neither filled nor local,
%           then if upper element is filled,
%               then either set pointer to it in lower element
%                   and mark it and all elements to its left as distributed
%                   or mark it as local and
%                   go left in lower & upper lns and repeat step 3
%                   until can't go left (have assigned scope to all
%                                           elements).
%   This procedure should compute only distinct readings. via backtracking.
%

conj_ln(UpLN, LN) :-
    down(LN, LNelement), mark_ln(LNelement),
% get last elements of upper and lower lns
    last_element(UpLN, ULast),
    last_element(LN, Last),
    mark_distrib(Last, ULast),
    do_next_conj(LN).
do_next_conj(LN) :-
    next_pair(LN, LowLN), !, conj_ln(LN, LowLN).
do_next_conj(_LN).
next_pair(LN, LowLN) :- !,
    r(lxr, LN, LowLXR),
    element(ladjset, LowLXR, LowLN).
/* if copy filled, then don't do anything */
/* otherwise get next elements of Main and Copy and repeat, */
/* or succeed if can't go left */
mark_distrib(Copy, Main) :-
    not(empty(Copy)), !,
    ((left(Copy, NewCopy), !, left(Main, NewMain)),
    mark_distrib(NewCopy, NewMain));
    true.)
% if Copy is empty and unmarked, try to get from upper ln ...
mark_distrib(Copy, Main) :-
    not_filled(Copy), down(Copy, LV),
    not(checkNullValue(Main)),
    down(Main, UV),
    copyNullc(LV, UV),
    mark_left_distr(Copy, Main).

```

```

% else mark local.
mark_distrib(Copy,Main):-
    mark_local(Copy),
    ((left(Copy,NextCopy),!,left(Main,NextMain),
      mark_distrib(NextCopy,NextMain));
     true).

mark_left_distr(Copy,Main):-
    left(Copy,Next),!,down(Next,LV),left(Main,UNext),down(UNext,UV),
    copyNullc(LV,UV),
    mark_left_distr(Next,UNext).
mark_left_distr(_Copy,_Main).

not_filled(E):-
    empty(E), not(checkWdField(E,local)).
mark_in(E):-
    not(empty(E)),!,mark_right(E).
mark_in(E):-
    right(E,Next),!,mark_in(Next).
mark_in(_E).

mark_right(E):-
    right(E,Next),!,mark_local(Next),mark_right(Next).
mark_right(_E).

mark_local(E):- down(E,Value), getWdField(Value,local),!.
mark_local(_). % if there is already a word. no need to mark local

copyNullc(Node,Tree):-
    makeNullc(Node,Tree).

/* wconj_rx
   computes the distribution of the right adjunct under conjunction;
   it starts from the lowest (last) pair of conjuncts,
   assigns the higher of the two either a local reading. or a
   distributed reading; then finds the next right-adjunct above
   the current pair, and calls itself recursively.
*/
wconj_rx(LXR,_):-
    % get upper and lower RX
    l(radjset,LXR,UpRX),!, element(radjset,LXR,RX).
    % either value is null, or choose distrib or local reading (distrib_adj)
    % in either case, go to next LXR and reexecute
    ((checkNullValue(RX),!); distrib_adj(RX,UpRX)).
    up(UpRX,UpLXR), wconj_rx(UpLXR,_).
wconj_rx(_LXR,_).

% 1. succeed in case Copy is not empty
% 2. mark distributed—will fail if Copy is not empty

```

```
% 3. otherwise, mark local—this will fail if Copy not empty
distrib_adj(_Main,Copy):-
    not(checkNullValue(Copy)),!.
distrib_adj(Main,Copy):-
    mark_distrib_adj(Main,Copy).
distrib_adj(_Main,Copy):-
    mark_local(Copy),!.
mark_distrib_adj(Main,Copy):-
    down(Copy,Null),
    down(Main,Value),copyNullc(Null,Value).
```

APPENDIX D. COVERAGE OF SENTENCES WITH CONJUNCTION

Parse times are for the translated, compiled grammar in Quintus PROLOG on a VAX 11/785 running first (second) parse and to termination.

Sentence: *The field engineer replaced the board and adjusted the disk drive.*

Lexicon lookup completed

(The resulting diagram is shown in Figure 17.)

remaining words: []
runtime: 2.266 sec.
more? y

no more parses
runtime: 4.91599 sec.

Sentence: *The last replacement and adjustment of the drive took an hour.*

Parse 1

Left adjunct fully distributed; right adjunct distributed:

Paraphrase: *The last replacement of the drive and the last adjustment of the drive took an hour.*

Runtime: 1.9 sec.

(The resulting diagram is shown in Figure 18.)

Parse 2

Right adjunct is local, left adjunct is fully distributed:

Paraphrase: *The last adjustment of the drive and the last replacement took an hour.*

Cumulative runtime: 3.314 sec.

Parse 3

Right adjunct distributed, the distributed, but *apos* = last local:

Paraphrase: *The last replacement of the drive and the adjustment of the drive took an hour.*

Cumulative runtime: 5.666 sec.

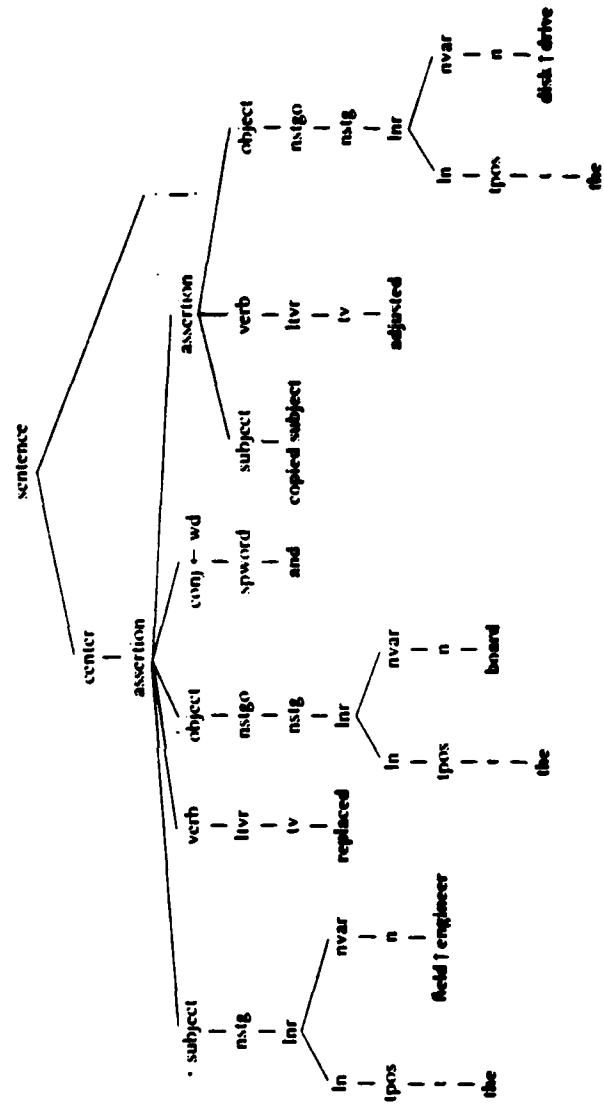


FIGURE 17

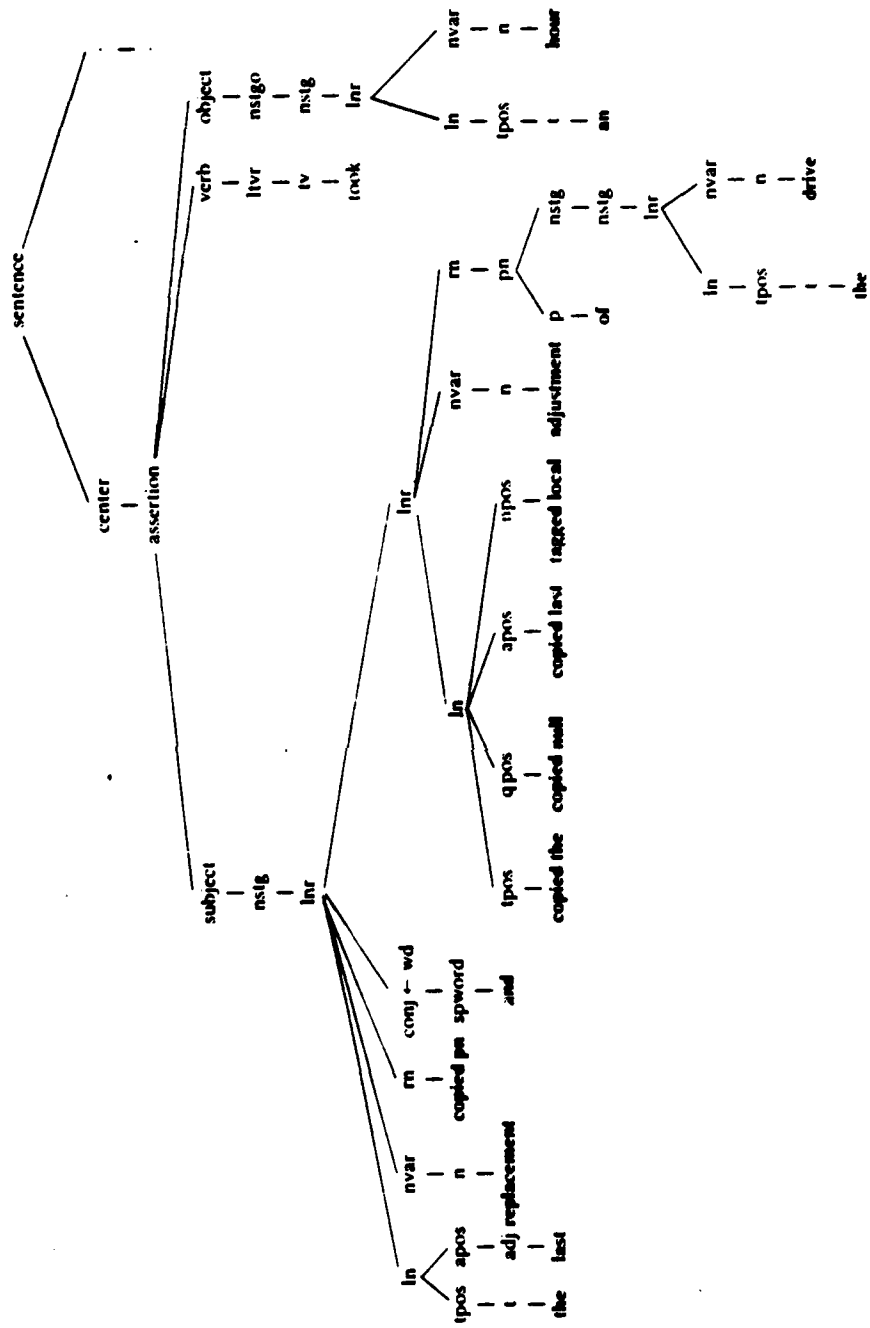


FIGURE 18

Parse 4

Right adjunct local, *the* distributed, but *apos* = *last* local:

Paraphrase: *The adjustment of the drive and the last replacement took an hour.*

Cumulative runtime: 6.64 sec.

The remaining two parses are not shown; they place the prepositional phrase of *the drive* at the sentence level (with *last* distributed or local).

Sentences Handled by the Conjunction Mechanism

- (1) Conjunction in assertion with reduced verb (2.0/4.4 sec):
The field engineer installed a board and the supervisor a drive.
- (2) Conjunction in assertion with reduced object (2.3/4.8 sec):
The field engineer has installed and the supervisor has adjusted the drive.
- (3) Comma conjunction in the noun phrase (1.6/6.9 sec):
The field engineer installed a disk, the board and a controller.
- (4) Comma conjunction or appositive in noun phrase (2 parses: 2.1/6.5/10.8 sec):
The field engineer replaced the disk, an old model, and a board.
- (5) Both/and conjunction in the noun phrase (1.5/3.4 sec):
Both the field engineer and the supervisor adjusted the drives.
- (6) Either/or conjunction in assertion (2.0/4.2 sec):
Either the board was replaced or they have adjusted the head.
- (7) Comma conjunction in assertion with reduced subject (2.7/6.3 sec):
The field engineer replaced the board, adjusted the controller, and installed a new drive.
- (8) Both/and conjunction in noun phrase, and conjunction in assertion (2.3/4.5 sec):
Both the disk and the head were replaced and a new motor was installed.
- (9) Conjunction of participles in prenominal adjective position (1.5/4.2 sec):
The repaired and adjusted drive is working.
- (10) Conjoined prepositional phrase (1.9/6.9 sec):
The boards of the controller and of the cpu were replaced.
- (11) Conjoined noun phrase under preposition (2 parses: 1.5/3.6/7.6 sec):
The boards of the controller and the cpu were installed.
- (12) Conjoined adjective in predicate position (0.9/2.0 sec):
The drive is old and worn.
- (13) Conjoined quantifier (1.3/3.0 sec):
The field engineer repaired two or three disks.
- (14) Conjoined complex object (2.0/4.3 sec):
She attempted to adjust the disk and to replace the motor.

- (15) Conjoined participle or tensed verb (2 parses: 1.9/3.9/5.5 sec):
The field engineer has repaired and adjusted the drive.

This paper has profited greatly from comments by Deborah Dahl, John Dowding, Marcia Linebarger, Donald McKay, Martha Palmer, Naomi Sager, and Rebecca Shiffman, as well as Michael McCord and several anonymous reviewers. Much of the original implementation of Restriction Grammar was done by Karl Puder, currently at Digital Equipment Corporation. John Dowding has been responsible for subsequent maintenance of the system, including much of the recent work on the dynamic translator, as well as numerous enhancements to the system implementation and interfaces. Marcia Linebarger is primarily responsible for the treatment of relative clauses and questions, and their integration with the conjunction mechanism. Margaret Heineman is responsible for the trees in Appendix D.

REFERENCES

1. Bloomfield, L., *Language*, Holt, Rinehart and Winston, New York, 1933.
2. Dahl, V. and McCord, M., Treating Co-ordination in Logic Grammars, *Amer. J. Comput. Linguistics* 9(2):69-91 (1983).
3. Dowding, John and Hirschman, Lynette, Flexible Efficient Parsing in Restriction Grammar, SDC Technical Report, March 1986.
4. Gazdar, G., Unbounded Dependencies and Co-ordinate Structure, *Linguistic Inquiry* 12: 155-184 (1981).
5. Harris, Z., *String Analysis of Sentence Structure*, The Hague, 1962.
6. Hirschman, L. and Puder, K., Restriction Grammar in Prolog, in: M. Van Caneghem (ed.), *Proceedings of the First International Logic Programming Conference*, Association pour la Diffusion et le Développement de Prolog, Marseilles, 1982, pp. 85-90.
7. Hirschman, L. and Puder, K., Restriction Grammar: A Prolog Implementation, in: D. H. D. Warren and M. Van Caneghem (eds.), *Logic Programming and its Applications*, 1985.
8. Pereira, F. C. N. and Warren, D. H. D., Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* 13:231-278 (1980).
9. Raze, C., A Computational Treatment of Coordinate Conjunction, *Amer. J. Comput. Linguistics*, microfiche 52, 1976.
10. Sager, N., Syntactic Analysis of Natural Language, in: *Advances in Computers*, Academic, New York, 1967, pp. 153-188.
11. Sager, N. and Grishman, R., The Restriction Language for Computer Grammars of Natural Language, *Comm. ACM* 18:390-400 (1975).
12. Sager, N., *Natural Language Information Processing: A Computer Grammar of English and Its Applications*, Addison-Wesley, Reading, Mass., 1981.
13. Sedogbo, C., A Meta Grammar for Handling Coordination in Logic Grammars, in: *Proceedings of the Conference on Natural Language Understanding and Logic Programming*, Rennes, France, Sept. 1984, pp. 137-150.
14. Woods, W. A., An Experimental Parsing System for Transition Network Grammars, in: R. Rustin (ed.), *Natural Language Processing*, Algorithmics Press, New York, 1973, pp. 145-149.

APPENDIX H

A Prolog Structure Editor

This paper, by Leslie Riley and John Dowding, appeared as Paoli Research Center Technical Report No. 29, January, 1986. It describes a structure editor designed to facilitate the editing of specialized constructs, such as grammar rules and lexical entries.

Prolog Structure Editor

Leslie Riley
John Dowding

LBS Technical Memo 29
January 1986

The Prolog Structure Editor is a general structure editor written in Prolog. It is intended to make it easy to edit Prolog terms by allowing the user to edit a term by traversing its internal structure. As used in the Natural Language group, the Prolog Structure Editor allows you to edit grammar rules, word definitions in the lexicon, and arbitrary Prolog clauses. You may invoke the editor on one of these three types of structures by using one of the following top-level procedures:

```
edit_rule(Key)
edit_word(Word)
edit_clause(Functor)
```

The *edit_rule* procedure takes as its argument the name of a non-terminal. It then allows you to edit all of the grammar rules that define that non-terminal. In order to maintain consistency between the grammar rules and their translated versions, when you have completed editing the set of rules the editor will ask if you want to re-translate them.

The *edit_word* procedure takes as its argument a word from the lexicon. It then allows you to edit the definitions of that word and all of its morphological variants.

The *edit_clause* procedure takes as its argument the name of some Prolog procedure. The editor finds all clauses with that head and returns them as a set of clauses to be edited. While this option is only of limited use in Quintus Prolog (because the procedure being edited must be declared dynamic), in Symbolics Prolog it will allow you to edit any Prolog procedure.

Once you have called one of the three procedures that invoke the editor, you will enter the top level. This level is distinguished from lower levels in that you are not actually editing a Prolog term, but editing a set of terms. At this level you can perform operations on the set of clauses in the procedure like retracting an old clause, or asserting a new clause. ¹

¹ These changes to the database are not actually recorded until the editing session is finished.

The editor will report at every level what kind of structure you are editing. The kinds of structures that the editor knows about are:

A *Set* of clauses (top level only)

A *List* of terms

A *Conjunction* of terms (actually any infix right-associative operator)

A *Complex Term* (A functor followed by some number of arguments)

An *Atom*

It will then display the functor of the term (if appropriate) and the members of the term. Following are some examples:

Editing a Set of Rules

Rule1: objectbe::=(astg;nstg;pn),{sem_rep(append)}

Rule2: objectbe::=(vingo;venpass),{sem_rep(copy)};{dwh2},nullwh

Editing a term

Functor: ::=

Argument 1: objectbe

Argument 2: (vingo;venpass),{sem_rep(copy)};{dwh2},nullwh

Editing conjoined terms

Functor: ,

Term 1: {dwh2}

Term 2: nullwh

There are two types of commands that you can give to the structure editor: *Movement* commands and *Editing* commands. At every level in the editor, you are stationed at some Prolog term (except at the top level, when you are stationed at a set of terms). There are then two kinds of movement commands: downward movement and upward movement. A downward movement command is simply an integer that specifies which of the arguments of the current term you wish to move down to, from 1 to N (You can sometimes move to the 0th item, if the term has a functor then it is considered the 0th argument). At any term, only one direction is up, so the command 'u' will move you up one level in the structure. For convenience, the command 't' (for 'top') will move you to the top level.

An editing command is one which actually modifies the structure of the term that you are stationed at. The editing commands that are currently supported are:

delete

Specified by 'd<integer>'. This command deletes the named term.

insert-after

Specified by 'i<integer>'. This command inserts a new term after the mentioned term. 'i0' will make the new term the first argument. You will be prompted for the term that is to be inserted. As this new term is a Prolog term, you will have to end your input with a period.

replace

Specified by 'r<integer>'. This command replaces the specified term in the place of the mentioned term. You can sometimes replace the 0th item, if you want to change the functor of some complex term.

These commands are also available:

downward-movement

Specified by <integer>. Moves down to the *Nth* term of the current term.

move-up

Specified by 'u'. Moves up to the term that contains the current term.

go-to-top

Specified by 't'. Moves to the top level.

abort

Specified by 'a'. Ends the editing session and does not save any of the changes made!

print

Specified by 'p'. This command prints the structure of the current term. This command should only be used at the end of a command line.

help

Specified by '?'. This command prints out a listing of the available commands. It should only be used at the end of a command line.

If you command the editor to insert a whole rule, word, or clause, it will print out an entry from the set you are editing (to serve as a "template" of the type of entry you want to create) and ask you to edit that entry to form the new entry that you want to insert. (As a warning, do not insert a structure and then delete it. Instead, wait until the end of the editing session, and answer "no" to the query, "do you want to add ...?") If you try to replace a rule or clause, the editor moves to that rule or clause and asks you to replace each part individually (this is supposed to save you keystrokes).

To end the editing session, you must be at the top-most structure (a set of rules, a set of words, a set of clauses). At that level, type 'u' (or 't'), and the editor will ask you if you want to save the changes that you have made.

A SAMPLE EDITING SESSION:

| ?- edit_rule(objectbe).

Editing a Set of Rules

Rule1: objectbe::=(astg;nstg;pn),{sem_rep(append)}

Rule2: objectbe::=(vingo;venpass),{sem_rep(copy)};{dwh2},nullwh

Command: 2

/ edit the second rule*

Editing a term

Functor: ::=

Argument 1: objectbe

Argument 2: (vingo;venpass),{sem_rep(copy)};{dwh2},nullwh

Command: 2

/ move to the second argument*

Editing conjoined terms

Functor: ;

Term 1: (vingo;venpass),{sem_rep(copy)}

Term 2: {dwh2},nullwh

Command: 2 r2

/ replace the second argument of the second term*

Replace the term: nullwh

.with what Prolog term: stuff.

Editing conjoined terms

Functor: ,

Term 1: {dwh2}

Term 2: stuff

Command: t

/ go to the top level*

Editing a Set of Rules

Rule1: objectbe::=(astg;nstg;pn),{sem_rep(append)}

Rule2: objectbe::=(vingo;venpass),{sem_rep(copy)};{dwh2},stuff

Command: t

/ go to the top again, i.e., finish editing this rule
('t' and 'u' have the same effect at this level)*

Do you want to replace: objectbe::=(vingo;venpass),{sem_rep(copy)};{dwh2},nullwh

with: objectbe::=(vingo;venpass),{sem_rep(copy)};{dwh2},stuff

Enter 'y' or 'n': n

If you have changed any grammar rules, you will have to either:

- 1. Retranslate this rule.**
- 2. Switch the grammar to run interpreted only.**
- 3. Do nothing (and risk inconsistency!).**

Please enter 1, 2, or 3: 3.

yes

| ?- edit_word(replace).

Editing a set of words with the same root

Word 1: {(replace,root:replace,[v:[12],tv:[12,plural],12:{objlist:[nstgo,pn:[pval:[with]],npn:[pval:[with]]]])}

Word 2: {(replaces,root:replace,[tv:[12,singular]])}

Word 3: {(replaced,root:replace,[tv:[12,past],ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]])}

Word 4: {(replacing,root:replace,[ving:[12]])}

Command: 3

/ edit the third word*

Editing a term

Functor: :

Argument 1: replaced

Argument 2: root:replace

Argument 3: [tv:[12,past],ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]]

|

Command: 3

/ move to the third argument (a list)*

Editing a list

Element 1: tv:[12,past]

Element 2: ven:[14]

Element 3: 14:[12,pobjlist:[nullobj,pn:[pval:[with]]]]

Command: 11

/ insert an element into the list after the first element of the list*

What Prolog term should be inserted: stuff.

Editing a list

Element 1: tv:[12,past]

Element 2: stuff

Element 3: ven:[14]

Element 4: 14:[12,pobjlist:[nullobj,pn:[pval:[with]]]]

Command: u

/ go up one level*

Editing a term

Functor: :

Argument 1: replaced

Argument 2: root:replace

Argument 3: [tv:[12,past],stuff,ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]]]

Command: t

Editing a set of words with the same root

Word 1: :(replace,root:replace,[v:[12],tv:[12,plural],12:[objlist:[nstgo,pn:[pval:[with]],npn:[pval:[with]]]])

Word 2: :(replaces,root:replace,[tv:[12,singular]])

Word 3: :(replaced,root:replace,[tv:[12,past],stuff,ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]])

Word 4: :(replacing,root:replace,[ving:[12]])

Command: u

Do you want to replace: :(replaced,root:replace,[tv:[12,past],ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]])

with: :(replaced,root:replace,[tv:[12,past],stuff,ven:[14],14:[12,pobjlist:[nullobj,pn:[pval:[with]]]])

Enter 'y' or 'n': n

yes

| ?- edit_word(control).

Editing a set of words with the same root

Word 1: :(control,root:control,[n:[11,singular],v:[12],tv:[12,plural],11:[nonhuman,h-change,h-norm],12:[objlist:[1],notnsbj:[2],vmanner,h-change,h-norm],1:[nstgo,nsvingo,vingofn],2:[ntime1]])

Word 2: :(controlled,root:control,[tv:[12,past],ven:[14],14:[objlist:[1],notnsbj:[2],vmanner,pobjlist:[3],h-change,h-norm],3:[nullobj]])

Word 3: :(controlling,root:control,[ving:[12]])

Word 4: :(controls,root:control,[n:[11,plural],tv:[12,singular]])

Command: l4

/ insert a word after the fourth word*

Here is a word of the type that you want to create.

Edit it to make the new word.

Editing a term

Functor: :

Argument 1: control

Argument 2: root:control

Argument 3: [n:[11,singular],v:[12],tv:[12,plural],11:[nonhuman,h-change,h-norm],12:[objlist:[1],notnsbj:[2],vmanner,h-change,h-norm],1:[nstgo,nsvingo,vingofn],2:[ntime1]]

Command: r1

/ replace the first argument, in this case,
the word to be defined*

Replace the term: control

with what Prolog term: controller.

Editing a term

Functor: :

Argument 1: controller

Argument 2: root:control

Argument 3: [n:[11,singular],v:[12],tv:[12,plural],11:[nonhuman,h-change,h-norm],12:[objlist:[1],notnsubj:[2],vmanner,h-change,h-norm],1:[nstgo,nsvingo,vingofn],2:[ntime1]]

Command: r8

/ replace the third argument, in this case, the definition list*

Replace the term: [n:[11,singular],v:[12],tv:[12,plural],11:[nonhuman,h-change,h-norm],12:[objlist:[1],notnsubj:[2],vmanner,h-change,h-norm],1:[nstgo,nsvingo,vingofn],2:[ntime1]]

with what Prolog term: [n:[11,singular],11:[human]].

Editing a term .

Functor: :

Argument 1: controller

Argument 2: root:control

Argument 3: [n:[11,singular],11:[human]]

Command: t

Editing a set of words with the same root

Word 1: :(control,root:control,[n:[11,singular],v:[12],tv:[12,plural],11:[nonhuman,h-change,h-norm],12:[objlist:[1],notnsubj:[2],vmanner,h-change,h-norm],1:[nstgo,nsvingo,vingofn],2:[ntime1]])

Word 2: :(controlled,root:control,[tv:[12,past],ven:[14],14:[objlist:[1],notnsubj:[2],vmanner,pobjlist:[3],h-change,h-norm],3:[nullobj]])

Word 3: :(controlling,root:control,[ving:[12]])

Word 4: :(controls,root:control,[n:[11,plural],tv:[12,singular]])

Word 5: :(controller,root:control,[n:[11,singular],11:[human]])

Command: t

Do you want to add the word: :(controller,root:control,[n:[11,singular],11:[human]])).

Enter 'y' or 'n': y

yes

APPENDIX I

Designing Lexical Entries for a Limited Domain

Technical Memo No. 42, by Rebecca J. Passonneau, documents the general methods used in designing the verb decompositions and mapping rules for new domains.

DESIGNING LEXICAL ENTRIES FOR A LIMITED DOMAIN

April, 1986

Rebecca J. Passonneau
SDC--A Burroughs Company
Paoli, PA
Technical Memo No. 42

1. Introduction

This report outlines procedures for building domain specific lexical entries for the PUNDIT natural language system at SDC. The lexical entries are designed for utilisation in inference-driven semantic analysis (Palmer, 1984). The procedures for constructing the lexical entries take advantage of recent works in linguistic semantics (cf. References Cited, esp. Dowty, 1979; Foley and Van Valin, 1984; Levin, 1985; Levin and Rappaport, 1985; Rappaport and Levin, 1985; and Talmy, 1978a, 1978b, 1985) without being constrained by any particular linguistic theory. Of particular utility is a section in Foley and Van Valin (1984) entitled "The Semantic Structure of the Clause" in which they draw on the work of Gruber (1965), Jackendoff (1976) and Dowty (1979). Their aim is to provide a set of general tools for the semantic analysis of the verb system of any language. The generality of their approach makes it appropriate not only for different languages but also for domain-specific sub-languages.

This is the first report in a series of two on designing lexical entries. It gives an overview of the general methods for constructing lexical entries regardless of the domain. A subsequent report will focus on specific semantic issues pertaining to the current domain application of PUNDIT. This domain consists of Navy casualty reports (casreps) describing failures in ship-board starting air compressors (sacs).

2. General

The lexical entries consist of predicate logic clauses which represent word meaning and thematic structure in a single decomposition. Currently, two classes of words are given lexical entries: 1) those that serve as predicates (excluding predicate nominals¹) i.e., verbs, adjectives and prepositions, and 2) deverbal nouns and other nouns which take arguments.² Predicating expressions can be classified on the basis of similarities of meaning and thematic structure, and the similarities can then be captured by assigning similar predicate structures to classes of expressions. The predicate structures comprising the lexical entries for the casreps contain three types of abstract elements: basic semantic predicates (primitives), thematic roles, and aspectual operators.

The three elements of a lexical decomposition are all represented as predicate-argument terms embedded in a semantic tree structure, but they have distinct functions. The thematic role predicates, e.g., agent and patient, are the leaves of the semantic tree whose arguments are constituents of surface structure (e.g., subject, direct object). Thus each role type has an associated set of possible mappings to surface structure (e.g., an agent can be realised as a subject or as the object of a *by* phrase). Thematic roles are in turn the arguments of superordinate

¹Nominals occur in a variety of predicational uses, e.g., equational sentences (e.g., *Scott is the author of Waverly*) and sentences expressing type relations (e.g., *A persimmon is a (type of) fruit*). One way to represent such sentences would be to fill in a variable of a pre-defined predicate provided in the knowledge domain: e.g., *author(Scott, Waverly)*, and *isa(persimmon, fruit)*.

²Chomsky (1970) gives a short list of nouns with various complements, many but not all of which would fall into the category of nouns with thematic structure. Levi (1978) relates the complements of such nouns to 'semantically based case relations' (p. 27). Nouns in the current domain which take arguments include those classified as 'percepts', e.g., *color as in color of oil*; also, those classified as 'scalars', e.g., *pressure as in tube oil pressure of 85 psi*.

semantic predicates, the semantic primitives in terms of which the lexical content of a predicate is represented. The aspectual operators represent the temporal structure of a predicating expression and are necessarily superordinate to one or more semantic primitives.

Decomposition Structure of BREAK:

- a) Semantic roles appear in italics
- b) Semantic predicates are capitalised
- c) Aspectual operator appears in boldface

CAUSE(*agent*(*_*),become(BROKEN(*patient*(*_*))))

While lexical entries are necessarily domain specific, there are general principles which can guide the determination of all three components.

Lexical content, thematic structure and inherent aspect can be distinguished conceptually, but have complex (lattice-like) interdependencies. Regardless of which type of semantic component motivates the preliminary classification of expressions in a domain, the sub-classes will cut across categories. For example, agents are associated with two distinct aspectual classes, activity and event predications. Thus, arriving at a semantic classification of a set of predicating expressions is a cyclic rather than linear task.

3. Basic Semantic Predicates

Given an existing knowledge base, the domain specific semantic primitives could be selected to accord with relations specified in the knowledge base. In the absence of an a priori set of semantic relations, semantic classes can be chosen by grouping predicating expressions on the basis of general meaning classes, e.g., verbs indicating change of location (*move*), manner of motion (*slide*), change of physical state (*melt*), cognition (*suspect*), and so on. The actual decompositions within a class of expressions would depend on how accurately the meaning of the expressions must be represented. Thus selecting the semantic primitives for a domain depends largely on the application.

4. Aspect

Talmy provides a concise definition of aspect as 'the pattern of distribution of action through time' and observes that a particular aspectual content is generally part of the inherent meaning of a verb, though this inherent meaning can be modified by grammatical elements with aspectual meaning. Representing aspect in lexical entries makes it possible to appropriately interpret tense, grammatical aspect (i.e., progressive) and temporal phrases. The number of aspectual distinctions proposed in analyses of lexical aspect varies, depending on the language being investigated and the predilections of the investigator, but the minimal set consists of the distinction between stative and non-stative predications, and for the latter, between activities and events (change-of-state or change-of-location predications). Stative predications denote states of affairs which persist throughout some period of time during which there is no change or activity, i.e., the truth of the predication can be determined by sampling the state of affairs at a single point in time. Activity predications also denote states of affairs which persist for some period of time but differ from statives in that some activity or process is ensuing such that there is change from moment to moment. Event predications denote a transition to a new state of affairs, e.g., into a new physical state (*The ice melted*) or to a new location (*The ship arrived in port*).

4.1. Diagnostics and defining criteria

A variety of semantic criteria and sentence frames have been proposed to distinguish between aspectual classes (cf. Dowty, 1979). Since only three aspectual classes are implemented in PUNDIT, identifying two of them—statives and events—is sufficient. Activity predications are then predications which are neither states nor events.

Statives

- a) cannot be referenced with *do it* (not applicable with passive voice)

Event: *The oil sometimes ignites;
it does it when the oil pressure is too high.*

State: * *The oil is sometimes dark in color;
it does it when the oil pressure is too high.*

- b) cannot occur in pseudo-clefts: *what X did was Y*

Event: *What the oil did was ignite.*
State: * *What the oil did was be dark.*

- c) nominalization of whole VP cannot be subject of *occur, take place*

Event: *The oil's igniting occurs too frequently.*
State: * *The oil's being dark takes place twice a day.*

Events

- a) the past participle of change of state (event) predicates can be used adjectivally; e.g., the surface sentence "NP is V-ed" is more likely to be interpreted as a stative predication than as an event expressed in the passive voice

NP is [activity verb]-ed tends to be interpreted as
a recurrent event: *The engine is [usually] operated*

NP is [event verb]-ed tends to be interpreted as
a current state: *The engine is [now] corroded*

- b) a sentence in the past tense entails that the patient or theme is in a new state or new location

New location: *The ship arrived in port at 1300 hours.*
Entails: *The ship is in port as of 1300 hours*

- c) past progressive predication does not entail the simple past

Activity predication:
The engineer was operating the machinery.
Entails:
The machinery operated.

Event predication:
The crew was installing a new engine

Does not entail:

The crew installed a new engine.

4.2. Representation

Following Dowty (1979) and Foley and Van Valin (1984), the aspectual meaning of predating expressions is represented in part in their decompositional structure. Event decompositions contain a become predicate. The resulting state or location of an event verb is embedded directly beneath the become predicate, e.g., *fail* is represented as *become(failed(_))*. Currently, distinguishing states from activities is not done via an aspectual operator. In the current domain, stative predications (excluding those treated as "transparent" predicates, e.g., cognition verbs) are those whose main verb is *be* or *have* (e.g., *be inoperative*; *have wear*). All other non-event verbs are activities. For domains with a more heterogeneous class of stative predications, an aspectual operator (e.g., Dowty's *do*) could be added to activity decompositions to distinguish them from statives in future implementations.

More fine grained treatments of lexical aspect distinguish between types of activities and types of events. For example, Talmy (1985) classifies activities into full-cycle (*strike*), multiplex (*breathe*), and steady-state (*sleep*). His distinction between full-cycle and steady-state corresponds roughly to the more familiar terminological distinction between punctual and non-punctual verbs. A full-cycle predication can be transformed into a multiplex when a duration is associated with the activity. The duration adverbial forces an interpretation of repeated instances throughout the duration (e.g., *someone struck the gong*=one strike-gong event; versus *someone struck the gong for three hours*=repeated strike-gong events). Because such distinctions can affect the interpretation of adverbial expressions, future domain applications might benefit from a fine-grained typology of activities. In the current application, activities are not subcategorized.

Causation is generally treated in discussions of aspect because causal predications are necessarily temporally complex: an activity of one participant causes a resulting state or activity in another participant. In other words, the logical structure of a causative verb can be represented as *cause(predicate1(agent(_)),predicate2(role(_))*. Predicate1 generally, if not always, falls into the aspectual class of activities, whereas predicate2 may be either an activity or a simple event. The crucial component of the first term in a *cause* predicate is the agent semantic role. For notational simplicity, *agent(_)* can be substituted for *predicate1(agent(_))* without obscuring the distinction between the two aspectually distinct types of causatives. The general decompositional structure for causatives resulting in an activity is thus: *cause(agent(_),Pred(actor(_)))* (e.g., *someone operated the sac* <- *cause(agent(_),operate(actor(sac)))*). Causatives resulting in a new state or location are represented as: *cause(agent(_),become(Pred(patient(_))))* or *cause(agent(_),become(Pred(theme(_),location(_))))* (e.g., *the drive shaft sheared the driven gear* <- *cause(actor(drive shaft),become(sheared(patient(driven gear)))* where *become* is embedded in the decomposition). Aspectual operators also have relevance to thematic structure as will be shown in the following section.

5. Thematic structure

There is no a priori set of thematic roles with fixed criteria for assigning the arguments of a predication to one or another role type. However, there are gross regularities in the lexicon pertaining to 1) the number of arguments a verb takes in various uses (e.g., transitive/intransitive uses of the same morphological form), 2) the syntactic relations between the verb and its arguments, 3) and the interpretation of how an argument participates in the

state, activity or event expressed in the predication. All three factors contribute to the analysis of thematic structure. The following discussion outlines a procedure for assigning thematic structure.

The distinction between stative and event predications and the discussion of causation provide a starting point for determining thematic structure in the following ways. First, all event predications, by definition, contain stative predications within them, i.e., all event predications are either of the form *become(stative)*, if intransitive (e.g., *the sac failed*), or *cause(X,become(stative))* if transitive (e.g., *the operator disengaged the sac*). The aspectual operator *become* doesn't change the thematic structure of a predicate. In contrast, the *cause* predicate is both an indication of causative meaning and of the presence of an agent thematic role. There is thus a regular relationship between the thematic structure and valency of a stative predication (*NP1 be X*), a simple-event whose result is the stative predication (*NP1 become X*), and the related causative-event (*NP2 cause NP1 become X*). For any stative, there may or may not be a corresponding intransitive predication: cf. *the cup is broken/the cup broke* versus *the drive shaft is lubricated/*the drive shaft lubricated*. Further, the event and stative predications may or may not make use of morphologically related forms. A first pass at determining the set of thematic roles associated with the predications used in a particular domain can be accomplished by examining triplets of stative/simple-event/causative-event predicates on the one hand, and pairs of simple activity/causative-activity predicates on the other.

5.1. Predications with Patient/Theme Arguments

A large number of event predications fall into one of two classes: state-change or location-change. The argument said to undergo a change of state is conventionally a patient while one said to undergo a change of location is conventionally a theme. The state-change state predicates typically have only the patient role while location-change predicates typically involve at least one location role (e.g., source and/or goal). Further, both patients and themes tend to be subjects of simple event predication and direct objects of causative events. Corresponding to these two types of event predications are two types of stative predications specifying the current state or current location of an entity. The two types of stative predications, which tend to be of the form *NP is Adj* or *NP is locative-PP*, have the same semantic roles as their corresponding event predications. The following chart schematically represents the three aspectual types—stative, simple-event and causative-event—of the two semantic classes—location and physical state:

Stative predication:

Physical state: "the shaft is dry"

<- dry(patient(shaft))

Location: "metal particles are in the oil"

<- in(theme(particles),location(oil))

Simple event:

Physical state: "the pump seized"

<- become(seized(patient(pump)))

Location: "the ship arrived at the port"

<- become(at(theme(ship),location(port)))

Causative event:

Physical state: "the operator disengaged the sac"

<- cause(agent(operator), become(disengaged(patient(sac))))

Location: "the operator disconnected the shaft from the hub"

<- cause(agent(operator),
become(disconnected(theme(shaft),location(hub))))

Fig 1. Six abstract semantic types

Other roles in addition to agent, patient, theme and location are sometimes associated with stative and event predications. For example, a causative event verb may have an instrument role, depending in part on whether an inanimate entity can be the subject of the causative transitive, as in *the hammer broke the cup*. As mentioned above, change of location verbs may have source or goal roles. Whether to incorporate an instrument role, or to substitute source or goal for location, depends in part on what arguments can appear in surface structure and on the set of semantic primitives appropriate for the domain. For example, the location argument of *disconnect* is more precisely a source as evidenced by the possibility of a *from* prepositional phrase alongside the impossibility of a *to* phrase:

*the operator disconnected the shaft from/*to the hub*

Other change of location verbs may take both goal and source, or only goal:

the ship went from the harbor to the open sea

*the operator attached the shaft to/*from the hub*

Both sources and goals are types of locations. Their contribution to lexical meaning can be captured by the choice of thematic roles or by the choice of semantic primitives. Thus the location argument of *disconnect* could be represented as a source: *disconnect(theme,source)*. Alternatively, the meaning captured by the source role, viz. that the theme is no longer at some source location, could be represented by embedding a location role in the negation of an *at* predicate:

disconnect <- become(not(at(theme(_),location(_)))).

Similarly, the logical structure of *the ship went from the harbor to the open sea* could be represented in a relatively flat, or inferentially shallow structure, as in:

move(theme(ship),source(harbor),goal(sea)).

Alternatively, the lexical decomposition process could be carried a step further to incorporate the logical inferences represented below (cf. Foley and Van Valin, pp. 51ff):

at(theme(ship),location(sea)),

not(*at*(theme(ship),location(harbor))).

This is a very simple illustration of how the set of thematic roles for a domain interacts with the set of primitive semantic predicates, which in turn depends on the desired output structures. The choice between implementing only a location role for a domain, or all three location, source and goal roles, also affects the set of surface structure mappings for locative arguments.

5.2. Actor predications

An activity predication minimally requires an argument which is the entity performing an act or engaged in some process, here called the *actor*. Thus actors are generally animate entities, or inanimate entities which have a source of energy or motive force. Examples of activity predications taking only an actor argument are:

```
the woman sneezed <- sneeze(actor(woman))
the wind blew    <- blow(actor(wind))
the wheel turned <- turn(actor(wheel))
```

Some activity predications of this form also have transitive/causative uses and in effect have two actor roles, a causing actor and an experiencing actor. The former is designated an *agent*, as in:

```
someone turned the wheel <- cause(agent(someone),turn(actor(wheel))).
```

The verb *turn* illustrates a relationship between a univalent activity predicate and its corresponding bivalent causative. Not all bivalent activity predicates are causatives in this sense. There are some transitive activity verbs whose direct object argument is not an actor, but rather, a passive participant, e.g., a theme as in:

```
someone kicked the wall <- kick(actor(someone),theme(wall)).
```

In sum, most activity predicates can be classified as one of the three following types:

Activity predication:

Univalent: Pred(actor(_))

Bivalent causative: Cause(agent(_),Pred(actor(_)))

Bivalent non-causative: Pred(actor(_),theme(_))

or: Pred(actor(_),location(_)).

Fig. 2. Four abstract semantic types

6. Summary of simple predicate types

The following chart, which amalgamates Figs. 1 and 2 above, schematises classes of predicates by valency, general thematic type and aspectual class.

Stative predication:

state Pred(patient(_))

location Pred(theme(_),location(_))

Simple-event predication:

change of state become(Pred(patient(_)))

change of location become(Pred(theme(_),location(_)))

Causative-event predication:

Physical state: cause(agent(_),become(Pred(patient(_))))

Location: cause(agent(become(Pred(theme(_),location(_)))))

Activity predication:

univalent Pred(actor(_))

bivalent,

non-causative Pred(actor(_),theme(_))

or Pred(actor(_),location(_))

causative cause(agent(_),Pred(actor(_)))

Fig. 3. Ten abstract semantic types

Patients and themes are both associated with stative and simple event predications: patients are associated with predicates characterising the physical state of some entity (or state-change)

while Themes, together with locations, are associated with predicates describing the location of some entity (or location-change). Patients and themes are also alike in having similar surface structure realisations; both are subjects of stative or intransitive predications or direct objects of transitive-causatives. The presence of a become operator in a decomposition changes the aspect of a predicate from stative to simple-event without changing the valency. Actors are associated with activity predicates, which may be inherently intransitive or transitive. For transitive activity predications, the second argument is likely to be a location or a theme. The agent role invariably indicates a causative predication, of which there are two aspectual types: causative-events and causative-activities. In a causative-event, the agent causes some entity to enter a new state or location; in a causative-activity, the agent causes some entity to engage in a new activity. Often a causative predication and the corresponding simple-event or activity are expressed by the same morphological form (cf. *turn*).

As shown above, the thematic roles built into a decomposition reflect in part the aspectual properties and valency of a surface predicate as well as the distinction between state-change and location-change meaning. It has been briefly observed that in addition, each thematic role has certain prototypical surface realisations. These are reviewed in greater detail in the next section.

7. Mappings from thematic structure to surface structure

The most salient arguments of a predicing expression are those appearing as clausal subjects and direct objects. Predicing expressions can also occur in noun phrases, e.g., adjectives and prepositional phrases. The following chart summarises the typical surface realisations in both noun phrases and basic clauses of the thematic roles reviewed above, except for location. As the earlier discussion of the verb *disconnect* suggests, some change of location verbs are inherently directional (*disconnect from*/**to*; compare *put on* versus *take off*). Others are not, and thus take a wide variety of locative complements (e.g., *move to/from/by*; *pass in/out/by*). Motion verbs (in English, cf. Talmy) tend to incorporate manner and cause as well as simple motion (e.g., *stand*, *bounce*, *hang*, *twist*, *pull* and so on). For these and other reasons, the surface realisations of location arguments are more idiosyncratic than the other arguments reviewed above. Discussion of location arguments will be postponed.

CHART OF THEMATIC ROLE TO SURFACE STRUCTURE MAPPINGS

AGENT IS REALIZED AS:

- 1) Possessive determiner of gerund/nominalisation:
 'the engineer's replacement of the sac'
 'the engineer's replacing the sac'
- 2) Subject of finite or non-finite clause:
 'the engineer replaced/replacing the sac'
- 3) PP obj of 'by' in a passive:
 'the sac was replaced by the engineer'

PATIENT IS REALIZED AS:

- 1) Noun modifying a nominalisation:
 'sac disengagement'
 'impeller blade tip erosion'
- 2) PP obj of 'of', where head is gerund, nominalisation or related noun:
 'disengaging of sac'
 'disengagement of sac'
 'erosion of impellor blade tip'
- 3) Possessive determiner of gerund/nominalisation:
 "sac's disengagement"
- 4) Head of NP where left modifier is adj or pple
 requiring patient role:
 'broken tooth'
 'burnt odor'
- 5) Subject of copula/passive S:
 'gear teeth are broken'
 'oil is discolored'
- 6) Direct object of transitive: 'the operator broke the sac'
- 7) Subject of intransitive, if it exists: : 'the gear tooth broke'

THEME IS REALIZED AS:

- 1) PP obj of 'of' for nominalisation/gerund:
 'disconnection of coupling'
 'color of oil'
- 2) Head of NP whose left modifier is a pred requiring a theme:
 'packed drive shaft'
 'disconnected shaft'
- 3) Subject of copula/passive S:
 'drive shaft was packed'
 'shaft was disconnected'
- 4) Dobj of causative tr.:

'someone packed the drive shaft'
'someone disconnected the diesel hub'

ACTOR IS REALIZED AS:

- 1) PP obj of 'of' for gerund or nominalisation:
'sounding of alarm'
'rotation of drive shaft '
- 2) Possessive determiner of gerund/nominalisation:
'the alarm's sounding'
- 3) Noun modifying a nominalisation:
'engine operation'
- 4) Subject of intransitive:
'the alarm sounded'
'the drive shaft rotated'
- 5) Subject of passive S:
'drive shaft was rotating'
'engine was operated'
- 6) Dobj of causative:
'someone sounded the alarm'

References Cited

- Dowty, David R. 1979. Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's PTQ. Dordrecht: D. Reidel.
- Foley, William and Van Valin, R. 1984. Functional Syntax and Universal Grammar. Cambridge: Cambridge University Press.
- Levin, Beth, ed. 1985. Lexical semantics in review. Lexicon Project Working Papers no. 1. Center for Cognitive Science. Cambridge, MA: MIT.
- Levin, Beth and Malka Rappaport. 1985. The formation of adjectival passives. Lexicon Project Working Papers no. 2. ms.
- Rappaport, Malka and Beth Levin. 1985. A case study in lexical analysis: the locative alternation. ms.
- Talmy, Leonard. 1985. Lexicalisation patterns: semantic structure in lexical forms. In Language Typology and Syntactic Description, vol. 3: Grammatical Categories and the Lexicon, pp. 57-151. Edited by Timothy Shopen. Cambridge: Cambridge University Press.
- Talmy, Leonard. 1978a. Figure and ground in complex sentences. In Universals of Human Language, vol. 4, pp. 627-49. Edited by Charles Ferguson and Edith Moravcsik. Stanford: Stanford University Press.
- Talmy, Leonard. 1978b. Relations between subordination and coordination. In Universals of Human Language, vol. 4, pp. 489-513. Edited by Charles Ferguson and Edith Moravcsik. Stanford: Stanford University Press.

APPENDIX J

A Computational Model of the Semantics of Tense and Aspect

Technical Memo No. 43, by Rebecca J. Passonneau, gives an overview of PUNDIT's temporal component. It describes the goals of the analysis, the algorithm it uses, and how the output is represented. A revised version of this paper has been submitted to the ACL to be included in a special issue on events and time.

A COMPUTATIONAL MODEL OF THE SEMANTICS OF TENSE AND ASPECT¹

Rebecca J. Passonneau

December 17, 1986

SDC/A Burroughs Company

Technical Memo No. 43

1. Introduction

This paper describes the temporal analysis component of the Pundit text-processing system.² Pundit's temporal component is designed to extract temporal information about real-world situations from short message texts. This involves three complementary analyses. First, Pundit identifies references to states-of-affairs with real time reference. Secondly, it determines the temporal structure of the predicated situation, or the manner in which it evolves through time. Lastly, Pundit analyses the temporal ordering of the real-time states-of-affairs with respect to the time of text production or to the times of other states-of-affairs. These three kinds of information are derived from the lexical head of a predication (verbal, adjectival or nominal), the surface verbal categories of tense, progressive and perfect, and finally, temporal connectives such as *before*, *after* and *when*. Each of these components of temporal meaning is assigned an informal context-dependent compositional semantics. A fundamental premise of the approach to temporal semantics taken here is that the several sentence elements contributing temporal information can and should be analysed in tandem in order to determine the times for which predications are asserted to hold. This is accomplished by means of a model of the semantics of time which incorporates both aspect and tense logic.

The short texts processed by the current version of Pundit are CASREPS (CASualty REPorts), messages which describe equipment failures aboard navy ships. The CASREPS domain has turned out to be an appropriate one for implementing a temporal component to analyse the time information given explicitly in each sentence of a text. Much of the important temporal information in the CASREPS can be extracted through semantic analysis of the individual sentences. CSREPS are diagnostic reports consisting of simple declarative sentences presenting a cumulative analysis of the current status of a particular piece of equipment. Within one sentence, several different states-of-affairs may be mentioned, linked together by explicit temporal connectives such as *before* and *after*. Because the texts under investigation contain only one sentence type and have a simple rhetorical structure, a lot of temporal information can be extracted even though the pragmatic dimensions of speech act type and inter-sentential temporal relations are ignored. Pundit's temporal component analyses the interactions among the several types of temporal information contained within the declarative sentence. This is accomplished through a general rule-driven computation of the various input values of tense, progressive and perfect along with a well-motivated aspectual decomposition of lexical items.

2. Temporal Information

One premise of the present work is that accurate computation of the temporal semantics of the verb and its categories will provide the necessary foundation for interpreting a wide range of temporal adverbials. However, the task of modelling the semantic contribution of the verb and its categories is a complex one. One source of complexity is that temporal information is distributed in a variety of lexical and grammatical elements. Further, the distinct lexical and grammatical elements are not univocal. As the extensive linguistic literature on tense and

¹This work was supported by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research.

²Pundit is an acronym for Prolog Understanding of Integrated Text. It is a modular system, implemented in Quintus Prolog, with distinct syntactic, semantic and pragmatic components. For descriptions of these components, cf. Dowding, Hirschman, Palmer, Dahl.

aspect demonstrates, the precise temporal contribution of any one surface category of the verb is contingent upon the co-occurring categories and upon the inherent meaning of the verb (cf. Mourelatos, Dowty, Vlach). Hence, even a preliminary solution to the computational problems of interpreting temporal information in natural language requires a broadly based analysis of the relevant semantic interdependencies. This paper proposes a solution to the computational task of extracting temporal information from simple declarative sentences based on separating temporal analysis into three tasks, each of which requires access to a selected portion of the temporal input.

Section § 2 of the paper explains in detail the computational goal addressed here. This goal is pursued in two phases. Phase 1 is to interpret the temporal information contributed by a predication (e.g., the verb and its grammatical categories). Phase 2 is to interpret temporal adverbials. The predication itself contains three distinct kinds of temporal information which are analysed separately. The first step of phase 1 is to determine whether the predication refers to a specific situation that takes place in actual time (cf. § 2.1); the second is to analyse how a specific situation evolves through time (cf. § 2.2); finally, the third is to find where a situation is ordered with respect to other known times (cf. § 2.3). Deriving these three pieces of information makes it possible to represent times and situations in a manner which could facilitate the interpretation of several kinds of temporal adverbials. Section § 2.4 illustrates a few types of temporal adverbials, including adverbial clauses and phrases introduced by temporal connectives (e.g., *when*, *before*, *after*). At present, connectives are the only class of temporal adverbials which Pundit processes.

The third section (§ 3) describes the input to the temporal component which is required for computing the temporal information derived from the verb and its grammatical categories on the one hand, and from temporal adverbials on the other. Then section § 4 demonstrates algorithmically how the temporal output is computed on the basis of the input it receives. One of these outputs, the reference time of a predication, pertains to the interpretation of relational temporal adverbials, i.e., adverbials which relate the time of a situation to another time (e.g. *The ship was refuelled yesterday*). Temporal connectives, for example, relate the time of a syntactically superordinate predication to a subordinate predication. The latter may consist of a clause (e.g., *after the sac had failed*) or a nominalisation (e.g., *after the failure of the sac*). Section § 5 illustrates how the reference time participates in the temporal interpretation of temporal adverbial clauses introduced by connectives such as *before*, *after* and *when*. For the sake of brevity, the treatment of temporal adverbs with noun phrase complements is not described in this paper.

2.1. The Predication

As mentioned above, three kinds of temporal information are associated with the predication. For tensed clause predications,³ the verb and its tense and aspect markings contribute information regarding the temporal specificity of the predication, its temporal structure and its temporal ordering relations. The following three subsections describe these three kinds of information in turn.

2.1.1. Specific temporal reference

Pundit's temporal component is designed to represent temporal information about specific situations. Specific situations are those which are asserted to have already occurred, or to be occurring at the time when a text is produced. This excludes, e.g., situations mentioned in

³Various types of tenseless predications are processed by Pundit's temporal component, including nominalisations and certain clausal modifiers of noun phrases (e.g., *Pressure decreasing below 80 psig caused the pump to fail.*). However, this paper focuses on the simpler case of tensed clauses.

modal, intensional, negated or frequentative contexts.⁴ The first task performed by the temporal analyzer is thus to determine whether a predication denotes an actual state-of-affairs. Sentence 1), for example, reports that a particular pump participated in a particular event at a specific time.

1) The lube oil pump seized.

Sentences 2) and 3), on the other hand, report types of recurrent events.

2) The lube oil pump seized whenever the engine jacked over.

3) The lube oil pump seizes.

In example 2), it is the adverb *whenever* which indicates that the main clause refers to a recurrent type of event rather than to a specific event situated at a particular time. In example 3), it is the inherent aspect of the verb *seize* in combination with the present tense which provides that information. Aspect is the inherent semantic content of a lexical item pertaining to the temporal structure of the situation it refers to, and thus plays a major role in computing temporal information. Before looking for temporal adverbials, Pundit's temporal component inspects the verb for its inherent lexical aspect and its grammatical tense and aspect markings in order to exclude sentences like 3). This check to determine whether a particular predication denotes a specific situation is carried out by Module 1 of the Predication Algorithm as described in section § 4.1.

2.1.2. Temporal structure of specific situations

If a sentence is presumed to denote an actual state-of-affairs on the basis of a quick check of the verb and its surface marking, then the temporal structure of the denoted situation is computed. Accurate representation of the temporal structure of a situation is a prerequisite for interpreting the relative temporal location of a situation and for interpreting temporal adverbials. Situations are classified on the basis of their temporal structure into three types: states, processes and transition events. Each situation type has a distinct temporal structure comprised from a set of basic temporal components and features. The basic components of temporal structure are intervals while the features associated with them are stativity versus activity, and boundedness.

Very briefly, a state is a situation which holds over some stative interval of time. State predications evoke inherently unbounded stative intervals, although a temporal bound could be provided by an appropriate temporal adverbial (e.g., *The pressure was normal until the pump seized*). In general, temporal adverbials can modify an existing component of temporal structure or add components of temporal structure. A process is a situation which holds over an active interval of time. Active intervals can be unbounded or unspecified for boundedness, depending on the grammatical aspect of the predication. A transition event is a complex situation consisting of a process leading up to a new state or process. Its temporal structure is thus an active interval followed by—and bounded by—a new active or stative interval. As will be pointed out in this section, the boundary between the two intervals, referred to here as a transition bound, is a feature associated with the temporal structure of transition events, rather than a real component of time. This section briefly reviews how the three types of situation and their temporal structures are represented. Later, in sections §§ 3.3 and 4.2, we will see how the type of situation associated with a predication is determined from the inherent lexical aspect of the verb and its grammatical aspect.

⁴The latter are not currently handled in the Pundit system. Predications embedded in any one of these contexts do not directly denote specific situations but rather denote types of situations which, e.g., might occur, have not occurred, or tend to occur. Treatment of these contexts awaits the development of a representation which distinguishes between specific situations which hold for some real time and types of situations which hold for some potential time.

2.1.2.1. Situation Types and Time Arguments

A state is a situation holding for some interval of time in which there is no change from moment to moment. The interval associated with a state is referred to as a stative interval. Sentence 4) is an example of a typical stative predication. Note that the lexical head of the verb phrase is the adjective *low*.

- 4) The pressure is low.
 Situation type: state
 Temporal unit: stative interval

When a stative predication is recognized as such, the time component associates a stative interval with the predication to indicate that the predication holds over a homogeneous interval of time.

A process also has duration. The interval associated with a process, referred to as an active interval, contrasts with a stative interval in that there is change from moment to moment, but without an abiding change of state. Sentence 5) illustrates a typical predication denoting a process.

- 5) The gear is turning.
 Situation type: process
 Temporal unit: active interval

The distinction between stative and active intervals, or between states and processes, will prove useful in interpreting manner adverbials indicating rate of change, e.g., *slowly* and *rapidly*. Since stative predications denote the absence of change over time, they cannot be modified by rate adverbials.

The third type of unit, the transition bound, is associated with events which denote a transition to a new state-of-affairs, as in 6) and 7). A transition bound must, by definition, be a transition between an initial active interval and an ensuing active or stative interval associated with a new state-of-affairs. Examples 6) and 7) illustrate two types of transition events, one resulting in a new state, and one resulting in a new process.

- 6) The lube oil pump has seized.
 Situation type: transition event
 Temporal units: active interval + transition bound + stative interval
- 7) The engine started.
 Situation type: transition event
 Temporal units: active interval + transition bound + active interval

A transition bound is a convenient abstraction for representing how transition events are perceived and talked about. Since a transition event is one which results in a new state-of-affairs, there is in theory a point in time before which the new situation does not exist, and subsequent to which the new situation does exist. A transition bound, however, is a theoretical construct not intended to correspond to an empirically determined time. The components of temporal structure proposed here are intended to provide a basis for deriving what is said about the relative ordering of situations and their durations, rather than to correspond to physical reality.

2.1.2.2. Representing states-of-affairs

A specific state-of-affairs is represented in Pundit as a predicate structure identifying the type of situation. Each state, process or event representation has three arguments: a unique identifier of the state-of-affairs, the semantic decomposition, and the time argument. The sentence in 5) is repeated here along with its state-of-affairs representation.

- 5) The gear is turning.
 Situation type: process

Temporal unit: active interval

State-of-affairs: `process([turn1], turnPactor([gear1]), period([turn1]))`

Label: `process`
 Unique identifier: `[turn1]`,
 Semantic decomposition: `turnPactor([gear1])`,
 Time Argument: `period([turn1])`

The same pointer is used to identify both the event and its time argument because it is the actual time for which a situation holds which uniquely identifies it as a specific situation. The situation mentioned in sentence 5) is represented as a process whose name is `[turn1]`. The `[turn1]` process occurs for a specific time, i.e., for the duration of `period([turn1])`. The semantic decomposition represents the process predicate along with its participant(s).⁵

The three types of situations are distinguished both on the basis of the state-of-affairs predicate, or label, and the time argument. States receive a state label and processes receive a process label. Both are assigned period time arguments, but a period in the context of a state representation corresponds to a stative interval while a period in the context of a process representation corresponds to an active interval. Transition events have a more complex temporal structure which requires multiple states-of-affairs representations.

There are three components of the temporal structure of a transition event: an initial process interval leading up to a transition, the moment of transition, and the interval associated with the new, resulting state-of-affairs. That there are these three distinct temporal components of transition events can be illustrated by the following sentences in which different types of time adverbials pertain to the three temporally distinct parts of the predicated event.

- 8) It took 5 minutes for the pump to seize.
- 9) The pump seized precisely at 14:04:01.
- 10) The pump was seized for 2 hours.

The temporal measure phrase *5 minutes* in 8) above applies to the interval of time during which the pump was in the process of *seizing*. The clock time in 9) corresponds to the moment when the pump is said to have made a transition to the new state of *being seized*. Finally, the measure phrase in 10) corresponds to the interval associated with the new state. In theory, then, one could represent the full temporal structure of a transition event as three contiguous states of affairs: an initial process (e.g., *seizing*) leading up to a transitional event (e.g., *becoming seized*) followed by a new state of affairs (e.g., *seized*) (cf. Dowty, 1986, p. 43). In practice, Pundit explicitly represents only the latter two components of transition event predications: the moment (transition bound) associated with an event of becoming, and a period associated with a resulting situation. The event of becoming is represented as an event predicate with a moment time argument. The new situations resulting from transition events receive a period time argument and either a process or state label, depending on the temporal structure of the resulting situation. This representation has been found to be adequate for the current application.

⁵Note that semantic predicates like `turnP` are distinguished from English words by the final P.

Type of Predication	State-of-affairs Label	Time Argument	Example
State	state	period	The pressure is low
Process	process	period	The gear is turning
Transition event	event	moment	The pump has seized.
result	state		<i>pump is seized</i>
	or process	period	

Representing the full temporal structure of transition event predications would eventually make it possible to interpret all the types of temporal adverbials found in examples 8) through 10) above. In general, the temporal structure of situations not only constrains the interpretation of temporal adverbials, but also of tense and the perfect, as will be shown in the next section. The function of tense and the perfect is to locate something in time, or to provide ordering information. The next section will show that tense and the perfect pertain to a specific abstract component of temporal structure referred to as the event time.

2.1.2.3. Relational Component of Temporal Structure (Event Time)

Pundit's temporal component employs a Reichenbachian analysis of tense but provides a more complete and more accurate representation of situations. It shares with Reichenbach's framework the notion that a situation can be located in time in terms of three abstract times: the time of the situation (event time), the time of speech/text production (speech time), and the time with respect to which relational adverbials are interpreted (reference time).

Pundit's system diverges from Reichenbach's in the relation of the event time to the temporal structure of a situation. Reichenbach did not distinguish between the two. For example, Reichenbach characterised a past event as one which preceded the speech time in its entirety. For Pundit, the event time is only a single component of the full temporal structure of a situation. Consequently, after determining the type of a situation along with its time arguments, Pundit identifies a component of the temporal structure which can serve as the event time. The choice of the event time depends on a property of intervals referred to here as boundedness.

The temporal structures associated with stative, process and transition event predications all have different inherent values of boundedness, depending on the inherent aspect of the predicate, its grammatical aspect (i.e., the presence or absence of the progressive suffix *-ing*), and the interaction between the two. The boundedness of the different situation types can be illustrated by comparing how tense is interpreted in each case.

Predications denoting states and process have duration, as evidenced by the interpretation of durational adverbial phrases of the form *for X*, where *X* is a time measure (e.g., *10 minutes*) (cf. Vendler).

- 11) The pressure was low for 10 minutes.
state situation
- 12) The gear was turning for 10 minutes.
process situation

The past tense in examples 11) and 12) above locates part of the respective situations in the past. However, both the state mentioned in 11) and the process mentioned in 12) are unbounded, meaning that the tense does not necessarily locate the whole interval associated with the situation. Examples 13) and 14) illustrate that there is no contradiction in asserting that a past state or process extends up to the present.

- 13) The pressure was low and is still low.
- 14) The gear was turning and is still turning.

Similarly, a present tense state or process assertion may be followed without contradiction by an assertion that the situation extends back into the past (cf. 15 and 16).

15) The pressure is low and has been low for three hours.

16) The gear is turning and has been turning for 5 minutes.

In the context of predications referring to states or processes, tense cannot be interpreted as pertaining to the location of the full temporal extent of the situation. The sentence in 17), for example, asserts that there is at least one moment within the stative interval coinciding with the present for which $\text{lowP}(\text{patient}([\text{pressure1}]))$ is true.

17) The pressure is low.

Since the assertion would be true both in case that the pressure was low only at an instant coincident with the present or over an interval containing such an instant, the interval is said to be inherently unbounded. The moment of time within an unbounded interval to which tense applies is referred to as the event time. The representation of the event time assigned to situations holding over unbounded intervals is discussed in section § 4.2.2.

Predications denoting states always have the same temporal structure: unbounded stative intervals. A predication denotes a state if the predicate is inherently stative, regardless of the presence or absence of the progressive. Inability to occur with the progressive suffix is often cited as a test for stative verbs (cf. Vendler), but Dowty (1979) notes a set of examples of locative predications in the progressive which denote locative states (e.g., *The socks are lying under the bed*). Such statives occur in Pundit's current application domain (cf. example 46 in § 4.2.1). Predicates denoting cognitive states or behavioral states have often been classified as statives but occur in the progressive with reference to a cognitive or behavioral process (cf. *I am thinking good thoughts*; *My daughter is being very naughty*). Although such verbs do not appear in the current domain, they would be treated differently from verbs which are unequivocally stative.

The progressive in combination with verbs whose inherent aspect is eventive, i.e., in the process or transition event class, always denotes a process and the associated active interval is unbounded. However, non-progressive process verbs denote processes over an active interval which is unspecified for boundedness. It is characterised as unspecified because the event time, the time located by the non-perfect tenses, may be within the interval or may be at the inception or the close of the interval. A comparison of simple past process predications with past progressive process predications in the context of two types of temporal adverbials exemplifies the relative indeterminacy of the former. The *at* phrase in 18) and 19) specifies a clock time. In the context of the progressive process verb in 18), that clock time is interpreted as falling within the active interval of *turning* but in 19), where the verb is not progressive, 12:04 can be interpreted as falling at the inception of the process or as roughly locating the entire process.

18) The gear was turning at 12:04.

19) The gear turned at 12:04.

The durational adverbial phrase appearing in 20) and 21) not only specifies a duration, but also implies an endpoint (cf. Vendler). Since progressive process predications are unbounded, 20) cannot be interpreted as denoting a specific situation with an actual time; rather, it seems to refer to an activity that was supposed to take place 5 minutes from some previously specified time. In contrast, 21) does denote a specific situation and the endpoint of the five minute duration is interpreted as coincident with the end of the *turn* process. Example 22) is a reasonable paraphrase of 21).

20) The gear was turning in 5 minutes.

21) The gear turned in 5 minutes.

22) It took 5 minutes for the gear to turn.

Non-progressive forms of process verbs exhibit a wide variation in the interpretation of what part of the temporal structure is located by tense. The influencing factors seem to be pragmatic in nature, rather than semantic. The solution taken here is to characterise the event time of such predications as having an unspecified relation to the active interval associated with the denoted process.

Non-progressive transition event verbs denote transition event situations, i.e., temporally complex situations in which there is a transition to a new state-of-affairs. The simple past places the moment of transition in the past. Transitional moments are dimensionless, hence a simple past transitional event predication followed by an assertion of its continuation into the present is contradictory, as illustrated in 23).

23) The lights went off and they're still going off.

The second clause in 23) cannot refer to the same event mentioned in the first clause; rather, the sentence is interpreted as referring to more than one event. In sum, the event time of a transition event situation is equated with the moment of transition.

2.1.3. Reference time

The final component of time associated with a predication is the time with respect to which relational adverbials like *now*, *yesterday* and so on are interpreted. In the absence of the perfect, the reference time is identical with the event time, as in 24) and 25).

24) The pressure is normal now.

25) The pressure was low yesterday.

In the perfect tenses, the reference time and event time are distinct. The event time of both the present and past perfect predications in 26) and 27) is past, i.e., the moment of *failure* is in the past.

26) The pump has now failed.

27) The pump had failed when the gear began to turn.

With the present perfect, it is the reference time which is in the present, as shown in 26) by the admissibility of the adverb *now*, which also refers to the present. As 27) illustrates, with the past perfect the event time, or moment of failure, precedes the reference time, i.e., the time specified by the *when* clause.

2.2. Adverbial Modification

Temporal adverbials occur in many of the sample sentences cited above. While the primary function of the examples is to show how adverbial modification can provide evidence for properties of the temporal structure and temporal ordering associated with various types of predications, they also illustrate different types of temporal adverbials. It is assumed that temporal adverbials can be analysed in terms of the same temporal components and ordering relations that apply to the analysis of situations. However, computation of the modification relations between temporal adverbials and predications is too complex to provide a full discussion here. Instead, three types of modification will be briefly mentioned. It has already been pointed out that there is a class of adverbs modifying the manner in which situations evolve through time, e.g., rate adverbials like *slowly* and *rapidly*. Temporal adverbs can also specify durations, e.g., *for X*, where *X* is a temporal measure phrase. Or, they can specify relations between times. Relational adverbs like *now* and *yesterday* relate the reference time of a predication to an implicit time, which in the case of *now* is the present. Relational adverbs like *before* and *after* relate the time of the predication they modify to an explicitly mentioned time, i.e., the event time associated with their syntactic complements, as in 28) and 29).

28) The pump failed after the decrease in pressure.

29) The engine seized before the alarm sounded.

Finally, it should be pointed out that many adverbials combine relational and durational elements. The durational adverbial in *X* where *X* is a temporal measure phrase not only specifies a duration, but relates the endpoint of this duration to some other time, e.g., the time at which the utterance is produced, as in 30).

30) The lights will go off in 10 minutes (i.e., from now).

Temporal connectives like *before* and *after* can combine with temporal measure phrases to yield complex adverbials specifying both a duration and a relation, as in 31).

31) The engine seized five minutes before the alarm sounded.

One of the goals addressed here is to provide temporal models of situations in such a way as to represent the intervals and relational elements of time which can be modified by temporal adverbials. As a first step towards this goal, the interpretation of simple temporal connectives has been implemented in Pundit's time component. The algorithm for interpreting sentences with temporal connectives is presented in section § 5.

3. Input to the Temporal Component

Pundit's time component performs its analysis after each sentence has been parsed and the semantic analysis of each predication structure in the sentence has been completed. In other words, time analysis is performed not only on clauses, but also on nominalisations (*failure of sac*) and certain noun modifiers. For ease of presentation, we will focus here only on the temporal analysis of tensed clauses.

The input to the time component for each tensed clause includes not only the surface verb and its tense and aspect markings, but also the decomposition structure produced by analysing the verb and its arguments. In effect, the time component receives a list of the following form:

[[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

Before explaining how this input participates in the algorithmic interpretation of a sentence's temporal semantics, each element of the list will be described in turn.

3.1. Verbal Categories

As indicated in boldface, the first element in the input list to the time component is itself a list of the verb's surface categories.

[[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

This list contains three pieces of binary information. The tense parameter is either past or present.⁶ If the progressive suffix (*-ing*) or perfect auxiliary (*have*) is present in the original sentence, there is a corresponding element in the list of verbal categories. Absence of the progressive or perfect markers is indicated by omission from the list. The presence versus the absence of the progressive contributes information about the temporal structure of a predication. The presence versus the absence of the perfect indicates whether the reference time must be distinguished from the event time. The way in which these pieces of information are used will be described after reviewing the remainder of the input.

3.2. Three Orders of Verbs

The second element in the input list to the time component is the surface verb.

[[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

⁶ Sentence fragments such as *erosion of blade tip evident* are processed as tenseless variants of full sentences. The time component uses special rules to assign a default tense interpretation to different types of fragments. These default rules are not described here.

For the sake of completeness, the computational function of including the surface verb will be briefly described. However, this information pertains to the treatment of more complex cases of the model. Since the focus of this paper is on explicating the basic model, the following description is intended only to indicate that the model has been extended to cover verbs whose semantic structure contains temporal information of a different order than the inherent temporal structure of a situation. After a brief description of three orders of temporal verbs, the discussion will return to explication of the input required for implementing the basic model.

Many verbs can be classified on the basis of what type of real-world situation they denote (cf. Vendler, Dowty). In section § 2.3 above, three types of situations were introduced. However, there is another set of pertinent distinctions of an entirely different order from the classification of situations into states, processes and events. For example, the semantic content of some verbs is almost entirely temporal in nature. Such verbs do not directly denote possible situations in the world, but rather contribute temporal information about a situation or situations mentioned as arguments to the verb. In 32), the subject of the sentence, *the failure*, denotes an event, but the verb *occur* does not denote a new situation. It provides tense and aspect information for the interpretation of the nominalisation *failure*.

32) The failure occurred during the decrease in pressure.

In other words, the temporal information contained in 32) is very similar to that contained in 33):

33) Something failed during the decrease in pressure.

A pragmatic difference between the two sentences is that in 32) it's not necessary to mention what failed whereas in 33), *fail* must have a subject, which in this case is an indefinite pronoun. Other verbs in this class are *follow*, *precede*, *continue*, *happen* and so on. Because these verbs contribute primarily temporal information, they are given the descriptive label of *aspectual verbs*.

It is easy to see that the analysis of aspectual verbs must be implemented somewhat differently from verbs like *fail* which directly denote situations. In a sentence like 33), the relevant temporal information is contained in the verb and its tense and aspect marking alone. In contrast, the temporal information in 32) pertaining to the *fail* event is distributed not only in the verb and its tense and aspect markers, but also in its subject. Temporal analysis of sentences like 33) must be performed not only at the main clause level, but also at the level of embedded propositions. In essence, analysis of aspectual verbs is of a different order. Verbs like *fail* are classified as first-order verbs while the so-called aspectual verbs are classified as second-order verbs.

Pundit's temporal component also handles a third class of verbs, classified as third-order. A third-order verb denotes a real-world situation, but its arguments are other situations. Consequently, the verb may contribute temporal information about the arguments as well as about the situation it denotes. The verb *result* illustrates this type. Sentence 34) asserts the existence of a result state-of-affairs; the result relationship holds between an instigating situation mentioned in the noun phrase *loss of air pressure*, and a resulting situation mentioned in the noun phrase *failure*.

34) Loss of air pressure resulted in failure.

Additionally, the meaning of *result* includes the temporal information that the instigating situation (the *loss*) precedes the resulting situation (the *failure*). A full temporal analysis of sentences like 34) requires two steps. The first is to analyse the temporal structure of the situation denoted by the verb. The second is to draw the correct temporal inferences about the verb's propositional arguments. Such verbs combine some of the properties of both first and second-order verbs and thus constitute a third order of analysis. Classifying a verb as a third-order verb drives the search for temporal inferences associated with its arguments.

In sum, there are three orders of verbs:

First order verbs: e.g., 'fail', 'operate', 'install'

verbs which denote situations and whose arguments are not propositional

Second order verbs: e.g., 'occur', 'follow', 'continue'

verbs which provide temporal information about their propositional arguments

Third order verbs: e.g., 'result'

verbs which denote situations but which also provide temporal information about their propositional arguments

This information is provided by a lookup procedure where first-order is a default value. That is, all second and third-order verbs must be explicitly classified as such in the temporal component. Although the Pundit system recognizes the distinction between first, second and third order verbs and processes the relevant temporal information in each case, the remainder of the paper will deal only with the temporal analysis of first-order verbs.

3.3. Decompositional Structure and Lexical Aspect

The fourth element in the input list is the decomposition structure produced by the semantic analysis of the verb and its arguments (cf. Palmer, Palmer et al.).

[[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

The lexical decompositions employed by Pundit represent the predicate/argument structure of verbs (or other predicating words, e.g., adjectives). In addition, following the example of Dowty (1979), the decompositions represent a verb's inherent temporal properties, or lexical aspect, which is one of the components used in computing the temporal structure of a predication. The literature on the aspectual classes provides a variety of diagnostics for determining the inherent aspect of verbs (cf. Vendler, Dowty). In Pundit's lexical entries, there are three values of lexical aspect corresponding to the three types of situations described in section § 2.1.2, viz., stative, process and transition event. This section describes how the temporal component determines the value of the lexical aspect from the decomposition.

Example 35) illustrates a typical decomposition of a stative predication.

35) The pressure is low.

Decomposition: lowP(patient([pressure1]))

The semantic analyser generates this output by finding the predicate lowP associated with the predication *be low*, and then predicating it over the entity ([pressure1]) referred to by the subject noun phrase, *the pressure*. The time component recognises this structure as a stative predication through a process of elimination. That is, this decomposition contains no information associated with processes or transition events; stative is the default value.

Example 36) illustrates the decomposition for a simple process verb. Like the stative lexical decomposition, it consists of a basic predicate, turnP, with a single thematic role, actor.

36) The gear is turning.

Decomposition: turnP(actor([gear1]))

Pundit recognises process verbs by the presence of the actor role in the decomposition structure; i.e., the actor role designates the active participant of process verbs.

The decompositions of transition event verbs are distinguished by the presence of a special predicate structure containing the aspectual operator becomeP. For example, a transition event verb such as *fail* is represented as in 37) below:

37) The engine failed.

Decomposition: becomeP(inoperativeP(patient([engine1]))

As described in the preceding section, a transition event predication is by definition one in which a process engaged in by some actor or agent leads to a new state or process. With

inchoative verbs (e.g., *fail*, as in 37), the actor of the initial process is also the patient or theme of the resulting state-of-affairs. That is, the new state (*inoperative(patient([engine1]))*) is asserted to come about without the intervention of some other actor. If the intervention of another actor is involved, the verb falls into the class of causatives and the actor of the initial process is conventionally called an agent (cf. 38):

- 38) The ship's force operated the engine.
 Decomposition: *causeP(agent([ship's force1]),
 becomeP(operatingP(actor([engine])))*)

Other decompositional analyses (cf. Dowty; Foley & Van Valin) conventionally represent the initial process of transition event verbs by associating an activity predicate (e.g., *do*) with the actor or agent of the initial process (e.g., *causeP(doP(agent(_)), becomeP(inoperativeP(patient(_))))*). The decompositions in 37) and 38) can be considered abbreviated versions of these predicate/argument structures.

The decompositions of transition event verbs illustrated in examples 37) and 38) provide a crucial piece of information used during the temporal analysis. Given a reference to a specific transition event which has already taken place, the temporal component deduces the existence of the new state-of-affairs that has come into being as a result of the event. It carries out this deduction by looking at the predicate embedded beneath the *becomeP* operator. This will be described in more detail later.

The important aspectual features of the decompositions can be summarised as follows. If the decomposition of a first-order verb contains an actor role, the verb is in the process aspectual class; if the decomposition contains a *becomeP* operator, the verb is in the transition event class; else, the verb is inherently stative.

3.4. Discourse Context

The final element in the input to the temporal component is a data structure representing the current discourse context.

[[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

The first element of this data structure is a list of unanalysed syntactic constituents. Recall that at this stage of processing, Pundit has produced a full syntactic analysis of a surface sentence (or sentence fragment; cf. fn. 6 above), and a semantic decomposition of some predication within the sentence. After the semantic analysis of a clause, the constituent list contains all those syntactic constituents which do not serve as arguments of the verb, e.g., adverbial modifiers of the verb phrase and sentence adjuncts. After the analysis of the main clause of sentence 39) for example, the constituent list would contain two unanalysed constituents, the prepositional phrase introduced by *during*, and the subordinate clause introduced by *when*.⁷

- 39) The sac failed during engine start, when oil pressure dropped below 80 psig.

This list of constituents is processed after the temporal content of a predication is analysed in the search for temporal adverbials which modify the predication (cf. § 5 below).

The data structure representing the current discourse context contains another piece of temporal information: the tense of the main clause. This information is necessary for the analysis of states-of-affairs mentioned in nominalisations and embedded tenseless clauses.

3.5. Summary of Input

The canonical case for temporal analysis consists of first-order verbs, i.e., verbs which denote real-world situations and which do not take propositional arguments. In this case, the

⁷ See, a word encountered frequently in the CASREPS, is an acronym for *starting air compressor*.

input to the temporal component consists of a list of the surface tense and aspect marking, the verb, the decomposition and the discourse context containing the list of unanalysed constituents.

[Tense, Perfect, Progressive], Verb, Decomposition, {Context}]

Before performing the computation of the temporal information distributed in these elements, the temporal component determines first that the verb is first-order by performing a lexical lookup. Then it determines the aspectual class of the verb by looking at the decomposition structure. A data structure is created which carries the four pieces of information about the verb's inherent aspect, its tense (past or present) and whether it occurred in the perfect or the progressive.

Temporal Data Structure: [Tense, Perfect, Progressive, Aspect]

The decomposition and the temporal data structure include all the necessary elements for extracting the types of temporal information described in section § 2.

Aspectual information pertains to the temporal structure of a situation. It is distributed in two components of the temporal data structure: the lexical aspect of the predication (Aspect) and its grammatical aspect (Progressive), indicated by the presence or absence of the progressive suffix *-ing*. Relational temporal information includes the temporal location of a state-of-affairs with respect to the speech time and with respect to the temporal vantage point taken by the speaker. This vantage point is the reference time for interpreting relational adverbials, like *now* and *yesterday*, and temporal connectives, like *when* and *after*. Relational information is distributed in the Tense and Perfect components of the temporal data structure. The aspectual information ([Progressive, Aspect]) is passed to Module 2 as described in section § 4.2. The relational information ([Tense, Perfect]) is passed to Module 3 as described in section § 4.3.

4. Computing the Information in the Predication

Pundit's temporal component analyses the temporal semantics of a verb and its surface categories in three phases designed to answer the following questions:

- 1) Does the predication denote a specific state-of-affairs with actual time reference?
- 2) If so, what is the temporal structure of the state-of-affairs, i.e., how does it evolve through time and how does it get situated in time?
- 3) Also, what is the temporal ordering of the state-of-affairs with respect to other times and what is the temporal vantage point from which the state-of-affairs is described?

Each question is answered by a different module which has selective access to the appropriate temporal input. The first module, corresponding to question 1), requires the full temporal data structure containing the inherent aspect of the verb, its tense, and whether it occurred in the perfect and the progressive (Tense, Perfect, Progressive, Aspect). The information in the temporal data structure can be distinguished into aspectual and relational information. The second module, which computes the temporal structure, requires the aspectual information contained in the Aspect and Progressive parameters of the temporal data structure. The third module requires the relational information contained in the Tense and Perfect parameters.

The following sections describe each module in turn, indicating both the input information to which it has access and the output which it produces. Figure 1 schematically represents the modular design of the portion of the temporal analyser devoted to the analysis of the predication. The chart illustrates that the modules are sequential, and that the output of one module may provide input for another. Notice also that the output of this first step provides input for the analysis of temporal adverbials, as described in section § 5.

4.1. Module 1: Filter Out Non-specific Situations

The present implementation of the temporal component of Pundit is designed to identify references to specific situations. Screening out predications which denote types of situations is

ALGORITHM FOR COMPUTATION OF TEMPORAL INFORMATION IN A SENTENCE

STEP 1:

COMPUTATION OF TEMPORAL INFORMATION CONTAINED IN THE VERB AND ITS GRAMMATICAL CATEGORIES

INPUT:

- a) Temporal Data Structure: [Tense, Perfect, Progressive, Aspect]
- b) Decomposition

OUTPUT:

- a) Full temporal structure of a situation with specific temporal reference
- b) Temporal ordering relations

MODULE 1: Filter input to screen out predications lacking specific temporal reference

INPUT: Temporal Data Structure, i.e., [Tense, Perfect, Progressive, Aspect]

MODULE 2: Compute the full temporal structure associated with a predication

INPUT: Aspectual Temporal Data, i.e., [Progressive, Aspect]

OUTPUT: State-of-Affairs Label; Time Arguments; Event Time

MODULE 3: Compute the temporal ordering relations associated with a predication

INPUT: Relational Temporal Data: [Tense, Perfect]

OUTPUT: Reference Time; Relations among Event Time, Reference Time and Speech Time

END STEP 1

STEP 2: COMPUTATION OF TEMPORAL INFORMATION CONTAINED IN TEMPORAL ADVERBIALS

INPUT: Temporal Structure, Event Time, Reference Time

Figure 1: Overview of temporal algorithm

the first task performed by the time component so that further temporal analysis will be performed only for predications which are presumed to denote specific situations. A predication denotes a specific situation when two criteria are satisfied. First, at least one of the verb's arguments must be interpreted as specific (cf. Mourelatos, Dowty, Vlach). Second, the situation must be asserted to hold in the real world for some specific time. Regarding the first criterion,

for example, the simple past of *fly* denotes a specific situation in 40) below, but not in 41), because the subject of the verb in 41) is generic.

40) John flew TWA to Boston.

41) Tourists flew TWA to Boston.

However, this paper does not address the interaction of the nature of a verb's arguments with the specificity of references to situations. Regarding the second criterion, predications in modal contexts (including the future; cf. 42) are excluded because their truth evaluation does not involve specific real-world times, but instead involves hypothetical or potential times.

42) The oil pressure should/may/will decrease.

Frequency adverbials like *always* may also force a temporally non-specific reading, as in 43).

43) John always flew his own plane to Boston.

The first module of the time component currently does not identify modal contexts, frequency adverbials, or generic arguments. However, it does identify predications denoting types of situations on the basis of their surface tense and aspect markings, as indicated in the temporal data structure.

If we consider only the four elements in the temporal data structure (Tense, Perfect, Progressive, and Aspect) then most predications can be assumed to denote specific situations. There are two exceptions. Both process verbs and transition event verbs when used in the simple present tense (i.e., non-progressive and non-perfect) generally denote types of situations,⁸ as exemplified in 44) and 45).

44) Number 2 sac operates at reduced capacity.

operate is a process verb

45) Oil pressure decreases below alarm point whenever sac is engaged.

decrease is a transition event verb

All four elements in the temporal data structure must be inspected to identify the two cases exemplified in 44) and 45). The sentences which can be identified as potentially denoting types of situations on the basis of the information in the temporal data structure are those which are in the present tense and which are neither perfect or progressive, and where the lexical head of the predication is not inherently stative. The following conditional statement expresses these dependencies.

IF Tense=present and Perfect=no and Progressive=no and Aspect≠stative
THEN the predicate probably denotes a type of situation;

→ stop processing temporal information

ELSE assume the predicate denotes a specific situation;

→ continue processing temporal information

In the current implementation of Pundit, predications which meet the first condition do not receive further temporal analysis. Thus the screening divides predications into two classes: those which can be assumed to denote specific situations barring disconfirming information elsewhere in the sentence (e.g., arguments of the verb, modality, frequency adverbials), and those which can be assumed to denote types of situations barring disconfirming pragmatic features of the discourse context. The time component could be developed to provide broader and more accurate coverage by adding modules to check for both types of disconfirming information.

⁸ They may denote specific situations if they can be interpreted as performatives (cf. Austin), as in this sentence, *I warn you not to cross me*, or, if they can be interpreted as a report of a presently unfolding situation, as in a sports-cast. These types of discourse, which do not come up in Pundit's domain, could be fully processed only by representing pragmatic features such as the speaker/addressee relationship. Module 1 could be effectively used to identify a set of sentences for which these pragmatic features would be relevant.

4.2. Module 2: Compute Temporal Structure

A predication passes through Module 1 of Pundit's temporal component for further temporal processing if it is presumed to denote a specific situation. Module 2 computes the first type of specific temporal information: the temporal structure evoked by a predication. Part of the temporal structure, that which Talmy described as the *pattern of distribution of action through time* (Talmy, p. 77), is represented in the situation labels and the time arguments. Another part of the temporal structure is the event time, i.e., the component of the full temporal structure which is located in time by tense and the perfect.

Temporal structure is computed at this point because it constrains the interpretation of the other two elements in the temporal data structure (Tense, Perfect). The situation labels, time arguments and event time associated with a predication can be computed entirely on the basis of the values of the two aspectual elements in the temporal data structure, the verb's inherent aspect and its grammatical aspect (Aspect, Progressive). However, in order to derive the correct representations of the situations resulting from transition events, Module 2 also needs the semantic decomposition. From this input information, i.e., the verb's inherent aspect, its grammatical aspect, and—in the case of transition events—the decomposition, Module 2 determines the temporal structure. Section § 4.2.1 describes the computation of the situation labels and time arguments. Section § 4.2.2 explains the computation of the event time.

4.2.1. Situation Labels and Time Arguments

Pundit distinguishes between three types of situations: states, processes and transition events. It determines the type of situation evoked by a predication on the basis of one or more values of the aspectual parameters. If the lexical aspect of the predicate is stative (Aspect=stative), then the Progressive parameter is irrelevant; such a predication always denotes a state. Similarly, if the lexical aspect of the predicate is process (Aspect=process), then the denoted situation is always a process. In the case of a process predication, however, the Progressive parameter affects the computation of the event time (cf. § 4.2.2). Finally, if the lexical aspect of the predicate is in the transition event class, then the type of situation depends on the progressive parameter. Progressive transition event verbs denote process situations; non-progressive transition event verbs denote transition event situations.

A predicate that is inherently stative is assumed to denote a state, regardless of whether the progressive suffix is present or not. In this case, the decomposition is not relevant. In other words, two types of input lead to state representations: stative predicates in the non-progressive aspect and stative predications in the progressive aspect. A state is represented with a state label and a period time argument. The following conditional statement summarises the relevant input and output:

IF Aspect=stative
THEN → Label=state and Time Argument=period

In the discussion of the semantics of progressive aspect with lexical statives (section § 2.1.2.3), the class of progressive statives denoting location was mentioned. An example of this phenomenon occurs in the current domain in the sentence fragment appearing in 46).

46) Material clogging strainer.

RELEVANT INPUT

a) Aspect=stative

OUTPUT

a) Label=state

b) Time Argument=period (i.e., *stative interval*)

The preceding sentence fragment is parsed with *clog* as the main verb in the progressive, with an elided *be* verb (i.e., as a tenseless version of *material is clogging strainer*). *Clog* is classified in

this domain as a stative verb, and in this context, the progressive suffix has no temporal semantic value.

If a predicate belongs to the process aspectual class (e.g., *operate*), then it denotes a process regardless of the value of the Progressive parameter. In this case, Pundit assigns a process label and a period time argument.

IF Aspect=process

THEN → Label=process and Time Argument=period

Examples 47a) and 47b) illustrate the two types of predications with process verbs that denote process situations.

47a) The diesel operated.

47b) The diesel was operating.

RELEVANT INPUT

a) Aspect=process

OUTPUT

a) Label=process

b) Time Argument=period (i.e., *active interval*)

As described above (in § 2), a process is a type of event that happens over a period of time (an active interval). There is a third case where a predication denotes a process: where the verb falls in the transition event class (e.g., *fail*) and occurs in the progressive, as illustrated in 48). In other words, in this case, both aspectual parameters are relevant input.

48) The pump is failing.

RELEVANT INPUT

a) Aspect=transition event

b) Progressive=yes

OUTPUT

a) Label=process

b) Time Argument=period (i.e., *active interval*)

Here Pundit would also assign a process label and a period argument. Recall that a period argument within a state representation represents a stative interval while a period argument within a process representation represents an active interval. In sum, there are three surface forms which denote process situations: the non-progressive form of a process verb, the progressive of a process verb, and the progressive of a transition event verb.

The last type of situation, the transition event, is denoted by a predicate which is inherently in the transition event class and does not occur in the progressive. As pointed out in the preceding paragraph, both aspectual parameters are relevant when the inherent aspect is a transition event. A transition event verb which is not in the progressive denotes a transition event situation. In this case, the decomposition input is also relevant for deriving the full temporal structure. As illustrated in 49), transition event situations are assigned two state-of-affairs representations: an event situation with a moment time argument, represented with the input decomposition, and a resulting state or process situation with a period time argument, for which a new decomposition is derived by removing the becomeP aspectual operator.

49) The pump failed.

RELEVANT INPUT

a) Aspect=transition event

- b) Progressive=no
- c) Decomposition=becomeP(inoperativeP(patient([pump1])))

OUTPUT

- a) Label=event
- b) Time Argument=moment
- c) Result Decomposition=inoperative(patient([pump1]))
- d) Result Label=state
- e) Result Time Argument=period (i.e., *static interval*)

The role of the decomposition in representing transition events is explained more fully in the following paragraphs.

The moment argument of a transition event is a transition bound, meaning that it is necessarily the onset of a new state-of-affairs. When Module 2 creates an event with a moment argument, it also creates a related representation for the new state-of-affairs. It is necessary to find an appropriate situation label, time argument, and semantic decomposition for the new state-of-affairs. It is also necessary to create a predicate specifying the temporal relationship between the transitional event moment and the following period associated with the new state.

All transition event verbs contain a state or process predicate embedded beneath the aspectual operator becomeP. The full decomposition represents the type of situation associated with the moment of transition. The portion embedded beneath becomeP is the situation type associated with the new situation. For example, the decomposition passed to the time component for sentence 49) — *The pump failed* — would be:

becomeP(inoperative(patient([pump1]))).

This decomposition appears in the representation of the transitional event, as follows:

event([fail1], becomeP(inoperative(patient([pump1]))), moment([fail1]))

The argument to becomeP is extracted to be used as the decomposition in the new state-of-affairs representation:

inoperative(patient([pump1]))

The set of semantic roles contained in the extracted decomposition is then inspected to determine if the actor role is a member. Since in this case, it is not, the new situation is assumed to be a state and represented as shown below:

state([fail2], inoperative(patient([pump1])), period([fail2]))

If actor had been one of the roles in the embedded predicate structure, the new situation would have been assumed to be a process. In this manner, the decomposition guides the selection of the label for the state-of-affairs inferred to result from the transition event.

The final piece of information to be represented here is the temporal relation between the moment associated with the transition event (e.g., moment([fail1])) and the period associated with the resulting situation (e.g., period([fail2])). The moment associated with a transition event serves as a lower bound to the period. Following Allen (1983), this is called a *start* relationship. Thus, every moment starts some period. In the case of example 49), the moment of failure starts a period in which the pump is in an inoperative state.

`start(moment([fail1]), period([fail2]))`

The two pieces of information associated with a transition event, namely the representation of the new situation and the start predicate, are both part of the final output of the temporal component.

4.2.2. Event time

The ordering relations specified by tense and the presence or absence of the perfect involve an abstract component of the temporal structure called the event time, following Reichenbach. The present treatment differs from Reichenbach's in two ways. First, his framework does not incorporate a treatment of inherent lexical aspect and its interaction with the surface categories of the verb. Secondly, his event time was either a period or a moment, and represented the entire event. By retaining the notion of event time but redefining it to be a component of the full temporal structure of a situation, it is possible to provide a simple semantics for tense and perfect while at the same time providing a more complete representation of the full temporal structure evoked by a predication.

Selecting some component of temporal structure to serve as the event time simplifies the computation of the ordering relations given by the perfect and non-perfect tenses. To understand the motivation in detail requires an examination of the semantics of tense and the perfect, which is postponed to section § 4.3. Briefly, one type of semantic information provided by the perfect and non-perfect tenses concerns the relative ordering among three semantically distinct times which Reichenbach referred to as the event time, the speech time and the reference time. An appropriate characterisation of event time makes it possible to give a simple formulation of the ordering relations specified by tense and the presence or absence of the perfect.

The three elements of temporal structure mentioned so far are stative intervals (cf. 46), repeated below), active intervals (cf. 47), and transition bounds (cf. 49).

- 46) Material clogging strainer.

```
state([clog1],
      clog(theme([material1]), location([strainers2])),
      period([clog1]))
```

- 47) The diesel operated/was operating.

```
process([operate1],
        operate(actor([diesel])),
        period([operate1]))
```

- 49) The pump failed.

```
event([fail1],
      become(inoperative(patient([pump1])),
      moment([fail1])
```

```
state([fail2],
      inoperative(patient([pump1])),
      period([fail2]))
```

```
starts(moment([fail1]), period([fail2]))
```

The relationship of event time to the temporal structure depends on a single property of the structure referred to as its boundedness (cf. § 2.1.2.3). Stative intervals are always unbounded; active intervals may be unbounded or unspecified for boundedness; the transition bound of an event constitutes a bound between some active interval (not represented here) and a following active or stative interval.

If an interval is unbounded, the event time is some moment included within the interval. It is represented as a binary predicate of the following form:

includes(period(_), moment(_))

A stative verb always gives rise to a period time argument which is, by definition, unbounded. Therefore, for a stative predication such as 46), the event time is always some arbitrary moment included within its period time argument.

46) Material clogging strainer.

includes(period([clog1]), moment([M1]))

Event Time=moment([M1])

In this case, the relevant input information contained in the list of aspectual parameters (Aspect, Progressive) is that the inherent aspect of the verb is stative.

IF Aspect=stative

THEN → Label=state and Time Argument=period

AND → includes(period, event time)

The time component generates a unique name to point to the arbitrary moment which will serve as the event time (e.g., moment([M1]), and represents its relationship to the period associated with the predication as shown above. That is, there is a type of state identified by the predicate clog which holds over some unbounded interval of time identified as period([clog1]). This period includes a moment arbitrarily identified as moment([M1]) which serves as the event time for the situation denoted by the predication in 46).

Period time arguments are not only associated with stative predications, but also with progressive or non-progressive process verbs (*the pump operated/is operating*), and with progressive transition event verbs (*the pump is failing*). The progressive always implies unboundedness, and in this respect resembles inherently stative predicates. Thus, the relationship of the event time to the time argument is the same for stative verbs (progressive or non-progressive), for progressive process verbs, and for progressive transition event verbs. The includes relation is generated for the event time of a progressive transition event verb, as in 48) repeated below.

47b) The pump is failing.

includes(period([fail1]), moment([M1]))

Event Time=moment([M1])

In other words, two types of predications evoke unbounded active intervals, progressive process verbs, and progressive transition event verbs. This can be expressed in a single conditional statement as follows:

IF Aspect=stative and Progressive=yes

THEN → Label=process and Time Argument=period

AND → includes(period, event time)

This condition indicates that the progressive has the semantic value of specifying an unbounded active interval except in the context of an inherently stative predicate.

If the process verb is not progressive, then the period associated with the predication is unspecified for boundedness. This gives rise to a different relationship between the event time and the period time argument. An interval which is unspecified for boundedness is one where the event time has an indeterminate relationship to the period time argument; i.e., it may start, end, or be included within the interval. This unspecified relationship is represented by means of the predicate has, as shown below.

47a) The diesel operated.

has(period([operate1]), moment([M1]))

Event Time=moment([M1])

The third conditional statement for computing temporal structure from the aspectual parameters is thus:

IF Aspect=process and Progressive=no
 THEN → Label=process and Time Argument=period
 AND → has(period, event time)

The event time of a transitional event is always identified with the transition bound. For sentences like 49) where the verb is a transition event verb and the surface aspect is non-progressive, a moment time argument will have been created. This time argument serves as the event time of the transition event.

49) The pump failed.
 Event Time=moment([fail1])

The last conditional statement pertaining to the computation of temporal structure and the event time is as follows:

IF Aspect=transition event and Progressive=no
 THEN → Label=event and Time Argument=moment
 AND → Event Time=moment

The next section summarises the four conditional statements reviewed above.

4.2.3. Summary of Module 2

Four sets of conditions, as illustrated in table 1 below, summarises the relationship between the two aspectual parameters and the temporal structure of a predication. As shown in the table, there are four types of temporal structure: states holding over an unbounded stative interval, processes holding over an unbounded active interval, processes holding over an unspecified active interval, and transition bounds implying a preceding active interval and following active or stative interval.

- 46) Material clogging strainers. *Time Argument*: period([C1])
 48) Pump is failing. *Time Argument*: period([F1])
 47b) The diesel operated. *Time Argument*: period([O2])
 49) The pump failed. *Time Argument*: period([F2])

TABLE 1 SUMMARY OF MODULE 2: COMPUTATION OF TEMPORAL STRUCTURE					
EG. No.	Input		Output		
	Aspect	Prog	Time Arg	Type	Event Time
46	stative	—	period([C1])	unbounded stative interval	M1 such that includes(period([C1]),M1)
47b	not stative	yes	period([F1])	unbounded active interval	M1 such that includes(period([F1]),M1)
47a	process	no	period([O2])	unspecified active interval	M1 such that has(period([O2]),M1)
49	event	no	moment([F2])	transition bound	moment([F2])

4.2. Module 3: Compute Ordering Relations

Reichenbach proposed that tense be interpreted in terms of three semantically distinct times associated with a predication: the speech time, the event time and the reference time. The speech time is the time at which the predication is produced (i.e., spoken or written). The event time, as used here, is a component of the situation referred to by a predication. The reference time is the time with respect to which one interprets relational temporal adverbials, e.g., *yesterday, now, when, before* and so on.

The four verb forms that Module 4 interprets are the simple present, the simple past, the present perfect and the past perfect. These are represented by means of two parameters in the temporal data structure: Tense (past or present) and Perfect (yes or no). The four permutations of these values of Tense and Perfect have different meanings, depending on the temporal structure of the predication. Given that there are four types of temporal structure which combine with these four tense forms, there are potentially 16 distinct semantic outputs. Instead of providing a separate formulation for each of the temporal relations among event time, reference time and speech time, event time has been chosen in such a way that its relation to the temporal structure of the predication captures most of the semantic differences pertaining to aspectual structure. This makes it possible to provide a more concise formulation of the semantics of tense and perfect.

4.2.1. Non-perfect tenses

In interpreting the perfect and non-perfect tenses, the only inputs needed are the previously computed event time and the relational parameters of the temporal data structure, Tense and Perfect. In the case of the non-perfect tenses, where Perfect=no, Tense pertains to the relation of the event time to the speech time. Speech time is assumed a priori to be identical with the time at which the report was generated. The negative value of the Perfect parameter also indicates that the reference time and the event time are identical. Because of the manner in which event time has been characterised, the ordering relations among the speech time, the reference time and the event time can be computed on the basis of two conditional statements, one for each non-perfect tense.

Very simply, the (non-perfect) present tense indicates that the event time and the speech time coincide. This is represented in the following conditional statement, showing the coincides relationship as a binary predicate.

IF Tense=present and Perfect=no,
THEN → *reference time = event time*
AND → *coincide(event time, speech time)*

The non-perfect past tense indicates that the event time precedes the speech time, that is:

IF Tense=past and Perfect=no,
THEN → *reference time = event time*
AND → *precedes(event time, speech time)*

The event time can only precede or coincide with the speech time since we are not dealing with potential or hypothetical times.

A brief review of the non-perfect tenses in the context of the different types of temporal structure will demonstrate the utility of the characterisation of event time offered here. Let us look at the present tense in the context of three types of temporal structure: unbounded intervals, unspecified intervals, and transition bounds. If the temporal structure associated with a predication is an unbounded interval, the present (non-perfect) tense locates some time within the interval coincident with the speech time. Examples 50)-53) illustrate the four types of predications with associated unbounded intervals.

- 50) The pressure is low.
non-progressive, stative

- 51) Metal particles are clogging the strainer.
progressive, stative
- 52) The pump is operating
progressive, process
- 53) The pump is failing.
progressive, transition event

In all four sentences, the type of situation denoted by the predicate is asserted to hold for some interval of unknown duration which includes the speech time. Predications involving process or transition event verbs in the simple (non-perfect, non-progressive) present tense have already been eliminated by Module 1.

Considering now the simple (non-perfect) past, if the temporal structure is an unbounded interval, then the event time is some moment within the interval and the non-perfect past tense locates it prior to the speech time, as illustrated in 54).

- 54) The pump was failing

Time Argument: period([fail1])
Temporal structure: unbounded
Event time: M1 such that includes(period([fail1], M1)
Temporal ordering: precedes(M1, speech time)

The situation is asserted to hold at some point M1 preceding the speech time, and over some interval which includes M1. The temporal structure associated with 55) is an unspecified interval.

- 55) The pump operated.

Time Argument: period([operate1])
Temporal structure: unspecified
Event time: M1 such that has(period([operate1], M1)
Temporal ordering: precedes(M1, speech time)

Here the situation is asserted to hold over some interval *period([operate1])* which has an unspecified component M1 preceding the speech time. M1 may be the onset of the period of operation, termination of this period, or some time included within the period.

The last case involves the non-perfect past in the context of a predication denoting a transition event. The event time is the transition bound, i.e., the theoretical point at which a transition to a new state-of-affairs (a *becoming*) is said to occur.

- 56) The pump failed.

Time Arguments: moment([fail1])
period([fail2]) such that starts(moment([fail1]), period([fail2]))
Temporal structure: transition bound
Event time: moment([fail1])
Temporal ordering: precedes(M1, speech time)

Sentence 56) is represented by Pundit to assert the following temporal information: there was a moment of transition at which the pump failed (moment([fail1])); this started a period in which the pump was inoperative (start(moment([fail1], period([fail2])); the moment of transition (moment([fail1])) preceded the speech time.

The examples presented above illustrate how the semantics of non-perfect present and past tense sentences can vary, depending on the temporal structure associated with the aspectual elements of the sentence. However, the interdependency among tense and aspect can be accounted for by separating the computation of the ordering relations specified by tense from the computation of the temporal structure of a predication. The rules for interpreting tense are

formulated in terms of the speech time and the event time. By first selecting an appropriate event time, and then computing the semantics of the simple tenses, a single rule for each non-perfect tense suffices to capture the semantic differences among the different present and past tense sentences illustrated above.

4.3.2. Perfect Tenses

In contrast with the non-perfect tenses, for the perfect tenses the reference time is distinct from the event time. Each of the perfect tenses specifies two ordering relations: the relation of the event time to the speech time, and of the reference time to the speech time. The possible orderings associated with the past and present perfects are highly constrained. First, the event time always precedes both the speech time and the reference time, regardless of the tense. Secondly, the reference time always follows the event time, but it may precede or coincide with the speech time. In the present perfect, it coincides with the speech time. In the past perfect, it precedes the speech time.

The ordering information provided by the perfect tenses can be formulated as a set of conditional statements. The first condition pertains to both perfect tenses; as shown below, if the predication is perfect, then the event time precedes the reference time.

IF Perfect=yes then precedes(event time, reference time)

The event time has already been instantiated in Module 3, but the reference time remains to be computed. The relation of the reference time to the speech time depends on the tense as follows; if the tense is present, the reference time is the speech time; if the tense is past, the reference time precedes the speech time.

IF Perfect=yes and Tense=present

THEN → precedes(event time, reference time)

AND → reference time=speech time

IF Perfect=yes and Tense=past

THEN → precedes(event time, reference time)

AND → precedes(reference time, speech time)

Once the reference time is instantiated, it can be passed to the module which interprets temporal connectives like *when*, *before* and *after*. In the case of the present perfect, the reference time is instantiated as the speech time. In the case of the past perfect, where the reference time is distinct from both the event time and the speech time, the time component generates a unique name to point to the arbitrary moment which will serve as the reference time of the predication. Since the function of the reference time is particularly obvious in the context of sentences with temporal adverbials, examples illustrating the computation of the reference time in perfect and non-perfect sentences is postponed to the discussion of temporal connectives.

5. Computing the Information in Temporal Connectives

5.1. Overview of Algorithm

The temporal adverbials encountered in the current domain handled by Pundit consist predominantly of phrases introduced by temporal connectives. The majority of these are instances of *when*, *before* and *after*. This section will focus on the temporal analysis of sentences containing *when* clauses. The general problem in analysing temporal connectives is to associate some time evoked by the matrix clause with some time evoked by the complement phrase. In general, connectives are represented as associating the reference time of the matrix clause with the reference time of the complement. The procedure involved in analysing a sentence the temporal relations specified by a *when* adverb (or other temporal connective) has the six steps illustrated in 57) below.

57) The sac failed when the pump seized.

Step 1: Analyse semantics of the main clause	<i>The sac failed</i>
Step 2: Find reference time of main clause (RT1)	moment([fail1])
Step 3: Recognise temporal connective	<i>when</i>
Step 4: Analyse semantics of subordinate clause	<i>the pump seized</i>
Step 5: Find reference time of subord. clause (RT2)	moment([seise1])
Step 6: Look up semantic structure of connective	coincide(RT1, RT2)

Result: coincide(moment([fail1]), moment([seise1]))

First, the temporal semantics of the main clause is analysed. One of the outputs of this analysis is the reference time of the main clause, which in this case would be represented as moment([fail1]). Then the time component finds the adverbial phrase *when the pump seized* in the constituent list which it recognises as consisting of a temporal connective (*when*) and a complement. The complement clause is sent to the semantic analyser and is returned to the time component for temporal analysis. The fourth step, the temporal analysis of the subordinate clause, yields the information that the reference time of the subordinate clause is moment([seise1]). Finally, the time component looks up the predicate structure representing the semantics of the temporal connective. *When* is represented as a binary predicate—coincide—whose first argument is the reference time of the main clause and whose second argument is the reference time of the complement clause.

The procedure illustrated above is recursive. For each predication which is analysed, the time component examines each remaining element in the constituent list. If the constituent list is empty, temporal analysis has been completed. If the next constituent is a temporal adverbial which can be analysed by the time component, it does so. If the next constituent cannot be analysed by the time component, it prints out a message indicating as such and looks for another constituent. This process continues until every constituent has been examined.

6. REFERENCES

- Allen, James F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26:11: 832-843.
- Austin, J. L. 1977. *How to Do Things with Words*.
- Dahl, Deborah A. 1986. Focusing and Reference Resolution in PUNDIT. Paper presented at AAAL Philadelphia, PA.
- Dowding, John and Lynette Hirschman. 1986. Dynamic Translation for Rule Pruning in Restriction Grammar. Paper presented at AAAL Philadelphia, PA.
- Dowty, David R. 1986. The effects of aspectual class on the temporal structure of discourse: semantics or pragmatics. *Linguistics and Philosophy* 9: 37-61.
- Dowty, David R. 1982. Tense, time adverbials and compositional semantic theory. *Linguistics and Philosophy* 5: 23-55.
- Dowty, David R. 1979. *Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's PTQ*. Dordrecht: D. Reidel.
- Foley, William and Van Valin, R. 1984. *Functional Syntax and Universal Grammar*. Cambridge: Cambridge University Press.
- Hirschman, Lynette. 1986. Conjunction in Meta-Restriction Grammar. To appear in *Journal of Logic Programming*.
- Mourelatos, Alexander P. D. 1981. Events, processes, and states. In *Syntax and Semantics*, vol 14: *Tense and Aspect*, pp. 191-212. Edited by P. J. Tedeschi and A. Zaenen. New York: Academic Press.

Palmer, Martha S.; Dahl, Deborah A.; Passonneau, Rebecca J.; Hirschman, Lynette; Linebarger, Marcia; Dowding, John. 1986. Recovering Implicit Information. Paper presented at 14th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York. August 1986.

Reichenbach, Hans. 1947. Elements of Symbolic Logic. New York: The Free Press.

Talmy, Leonard. 1985. Lexicalisation patterns: semantic structure in lexical forms. In Language Typology and Syntactic Description, vol. 3: Grammatical Categories and the Lexicon, pp. 57-151. Edited by Timothy Shopen. Cambridge: Cambridge University Press.

Vendler, Zeno. 1967. Verbs and times. Linguistics in Philosophy. New York: Cornell University Press.

Vlach, Frank. 1981. The semantics of the progressive. In Syntax and Semantics, vol 14: Tense and Aspect, pp. 271-292. Edited by P. J. Tedeschi and A. Zaenen. New York: Academic Press.

APPENDIX K

Nominalizations in PUNDIT

This paper by Deborah Dahl, Martha Palmer, and Rebecca Passonneau will be presented at the 25th Annual Meeting of the Association for Computational Linguistics in Palo Alto, July 1987.

Nominalizations in PUNDIT

Deborah A. Dahl, Martha S. Palmer, Rebecca J. Passonneau
Research and Development Division
SDC - A Burroughs Company
P.O. Box 517
Paoli, PA 19301 USA

To be presented at the 25th ACL, July 6-9, 1987, Stanford, CA

1. Introduction

In this paper we will discuss the analysis of nominalizations in the PUNDIT text processing system.¹

Syntactically, nominalizations are noun phrases, as in examples (1)-(7).

- (1) An *inspection of lube oil filter* revealed metal particles.
- (2) *Loss of lube oil pressure* occurred during *operation*.
- (3) SAC received *high usage*.
- (4) *Investigation* revealed adequate lube oil.
- (5) Request *replacement of SAC*.
- (6) *Erosion of impellor blade tip* is evident.
- (7) Unit has low output air pressure, resulting in *slow gas turbine starts*.

Semantically, however, nominalizations resemble clauses, with a predicate/argument structure like that of the related verb. Our treatment attempts to capture these resemblances in such a way that very little machinery is needed to analyze nominalizations other than that already in place for other noun phrases and clauses.

There are two types of differences between the treatment of nominalizations and that of clauses. There are those based on *linguistic* differences, related to (1) the mapping between syntactic arguments and semantic roles, which is different in nominalizations and clauses, and (2) tense, which nominalizations lack. There are also differences in *control*; in particular, control of the filling of semantic roles and control of reference resolution. All of these issues will be discussed in detail below.

Nominalizations are often addressed in treatments of noun noun compounds, because some nominalizations, as in (7) above, are noun noun compounds, and because some noun noun compounds are nominalizations. However, we have found it more useful to provide a general treatment of nominalizations, whether they are noun noun compounds or not. While we do not claim to have

¹ The research described in this paper was supported in part by DARPA under contract N000014-85-C-0012, ad-

provided an answer to the general problem of noun noun compounds, we do claim to have isolated one easily recognizable class which is amenable to the very general treatment which is described in this paper.

2. PUNDIT

The semantic processing to be described in this paper is part of the SDC PUNDIT² system for processing natural language messages. The PUNDIT system is a highly modular system, written in Prolog, consisting of distinct syntactic, semantic and discourse components, each drawing on one or more sets of data, including a lexicon, a broad-coverage grammar of English [Hirschman1985, Hirschman1986], semantic verb decompositions, rules mapping between syntactic and semantic constituents, a domain model, and general information about time [Passonneau1986].

The next sections describe the algorithms for the semantic processing of clauses and nominalizations in detail, pointing out similarities and differences.

The semantic domain from which these examples are taken is that of reports of failures of the air compressors used in starting Navy ships, or *sac's*.

3. Clause analysis

In order to produce a semantic representation of a clause, its verb is first decomposed into a semantic predicate representation appropriate for the domain. The arguments of the predicates constitute the SEMANTIC ROLES of the verb, which are similar to cases [Palmer1985]. For example, *fail* decomposes into **become inoperative**, with **patient** as its only semantic role.³ In this domain the semantic roles include: **agent**, **instigator**, **experiencer**, **instrument**, **theme**, **location**, **actor**, **patient**, **source**, **reference_pt** and **goal**. Semantic roles can be filled either by a syntactic constituent or by reference resolution from default or contextual information. We have categorized the semantic roles into three classes, based on how they are filled [Palmer1986]. Semantic roles such as **theme**, **actor** and **patient** are syntactically OBLIGATORY, and must be filled by surface constituents. Semantic roles are categorized as semantically ESSENTIAL when they must be filled even if there is no syntactic constituent available.⁴ In this case they are filled pragmatically, making use of reference resolution, as explained below. The default categorization is NON-ESSENTIAL, which does not require that the role be filled. The algorithm in Figure 1 produces a semantic representation using this information. Each step in the algorithm will be illustrated at least once in the next section using the

ministered by the Office of Naval Research. APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

² PUNDIT UNDERstands and Integrates Text

³ There are domain specific criteria for selecting a range of semantic roles. The criteria which we have used are described in [Abramson1984].

⁴ We are in the process of defining criteria for categorizing a role as ESSENTIAL. It is clearly very domain dependent, and relies heavily on what can be assumed from the context.

following (typical) CASREPS text.

Sac failed.

Pump sheared.

Investigation revealed metal contamination in filter.

3.1. A Simple Example DECOMPOSE VERB - The first example uses the *fail* decomposition for *Sac failed: fail* <- becomeP(inoperativeP(patient(P))). It indicates that the entity filling the OBLIGATORY patient role has or will become inoperative.

FOR patient ROLE -

PROPOSE SYNTACTIC CONSTITUENT FILLER - The following mapping rule indicates that the syntactic subject is a likely filler for any patient role, including this one, as indicated by the variable to the right of the slash indicating no specific constraints.

patient(P) <- subject(P) / X

DECOMPOSE VERB;

FOR EACH SEMANTIC ROLE

CASE 1: IF THERE ARE SYNTACTIC CONSTITUENTS -
 PROPOSE SYNTACTIC CONSTITUENT FILLER
 & CALL REFERENCE RESOLUTION
 & TEST SELECTIONAL RESTRICTIONS

CASE 2: IF ROLE IS OBLIGATORY AND SYNTACTICALLY UNFILLED -
 FAIL

CASE 3: IF ROLE IS ESSENTIAL AND UNFILLED -
 CALL REFERENCE RESOLUTION TO HYPOTHESE A FILLER
 & TEST SELECTIONAL RESTRICTIONS

CASE 4: IF ROLE IS NON-ESSENTIAL AND UNFILLED -
 LEAVE UNFILLED

CALL TEMPORAL ANALYSIS ON DECOMPOSITION

Figure 1. Clause Analysis Algorithm

The mapping rules make use of intuitions about syntactic cues for indicating semantic roles first embodied in the notion of case [Fillmore1968, Palmer1981]. The mapping rules can take advantage of general syntactic cues like "SUBJECT goes to PATIENT" while still indicating particular context sensitivities. This is accomplished by making the application of the mapping rules "situation-specific" through the use of PREDICATE ENVIRONMENTS which are indicated on the right-hand side of the rule, after the "/". The predicate environment can consist of specific predicates in which the role is an argument, the other arguments to that predicate, and constraints on the fillers of those arguments. The **subject <- patient** rule is quite general and can be applied to every **patient** semantic role in this domain. A variable to the right of the slash will match any predicate environment, and so will always be true. A mapping rule that only applied to the **patient** role of *fail* could be made application specific by indicating the predicate environment, **/inoperativeP(patient(P))**. Then the rule would only apply when the **patient** being filled had the same **predicate environment**.

CALL REFERENCE RESOLUTION - Clearly, *sac* is the subject of *sac failed*, and is the likely filler of the **patient** role. At this point the semantic interpreter asks noun phrase analysis to provide a unique referent for the noun phrase subject. Since no *sacs* have been mentioned previously, a new name is created: **sac1**.

TEST SELECTION RESTRICTIONS - In addition to the mapping rules that are used to associate syntactic constituents with semantic roles, there are selection restrictions associated with each semantic role. The selection restrictions for *fail* test whether or not the filler of the **patient** role is a mechanical device. A *sac* is a mechanical device so the subject of the sentence *sac failed* maps straightforwardly onto the **patient** role, e.g., **becomeP(inoperativeP(patient(sac1)))**.

Since there are no other roles to be filled the algorithm terminates successfully at this point and the remaining steps are not applied. The next example illustrates further steps in the algorithm.

3.2. Unfilled Obligatory Roles

The next utterance in the example, *Pump sheared*, illustrates the effect of an unfilled obligatory role.

DECOMPOSE VERB -

shear <- causeP(instigator(I), becomeP(shearedP(patient(P))))

Shear is an example of a verb that can be used either transitively or intransitively. In both cases the **patient** role is filled by a mechanical device that becomes *sheared*. If the verb is used transitively, the **instigator** of the

shearing, also a mechanical device, is mentioned explicitly, as in, *The rotating drive shaft sheared the pump*. If the verb is used intransitively, as in the current example, the **instigator** is not made explicit.

FOR instigator ROLE - Working from left to right in the verb decomposition, the first role to be filled is the **instigator** role. The subject of the sentence, *pump*, is a likely filler for this role, as indicated by the following mapping rule.

instigator(I) <- subject(I) / X

Reference resolution returns **pump1** as the referent of the noun phrase. Since *pump* is a mechanical device, the selection restriction test passes.

FOR patient ROLE - There are no syntactic constituents left, so a syntactic constituent cannot be proposed and tested.

UNFILLED OBLIGATORY ROLES - The **patient** role, a member of the set of obligatory roles, is still unfilled. This causes failure, and the binding of **pump1** to the **instigator** role is undone. The algorithm starts over again, trying to fill the instigator role.

FOR instigator ROLE- There are no other mapping rules for **instigator**, and it is non-essential, so Case 4 applies and it is left unfilled.⁵

FOR patient ROLE - The following mapping rules apply to **patient**, where the first one suggests the subject, in this case, the pump, as a filler.

patient(P) <- subject(P)

patient(P) <- object(P)

Reference resolution returns **pump1** again, which passes the selection restriction of being a mechanical device. The final representation is:

causeP(instigator(I),becomeP(shearedP(patient(pump1))))).

3.3. Nominalizations as Role Fillers

The third sentence of our example, *Investigation revealed metal contamination in filter*, illustrates the filling of semantic roles by nominalizations.

DECOMPOSE VERB - The decomposition of the verb *reveal* indicates that a NON-ESSENTIAL **instigating** event causes an OBLIGATORY **theme** to be *revealed* to an NON-ESSENTIAL **experiencer**.

reveal <-

causeP(instigator(I),

becomeP(known_toP(theme(T),experiencer(E))))

⁵In other domains, the **instigator** might be an **ESSENTIAL** role and would get filled by pragmatics.

The subject, *investigation*, and the object, *contamination*, both nominalizations, are likely fillers for the *instigator* and the *theme* role, respectively. Section 4 describes the algorithm for nominalization analysis in detail, using *investigation* and *metal contamination in filter* as examples. *Investigation1* and *contamination1* are returned as the referents of the two nominalizations, allowing the following representation for *reveal* to be produced.

**causeP(instigator(investigation1),
becomeP(known_toP(theme(contamination1),experiencer(E))))**

3.4. Temporal Analysis of Tensed Clauses

After the semantic decomposition of the verb and its arguments has been completed, the resulting predicate structure is passed to the temporal analysis component along with the tense and an indication of whether the verb was in the perfect or progressive. Certain kinds of temporal adverbials are also analyzed. The temporal component determines what time the predication is asserted to hold for by analyzing the inherent temporal meaning, or aspect of the verb, in the context of its surface tense and aspect markings. Temporal analysis results in three kinds of output: an assignment of a real time to the predication, if appropriate; a representation of the temporal type of the situation denoted by the predication as either a state, a process or an event; and finally, a set of predicates about the ordering of the time of the situation with respect to other times explicitly or implicitly mentioned in the same sentence [Passonneau1986]. The input and output to the temporal component can be illustrated with the simple example sentence, *sac failed*. The input would consist of the predicate structure resulting from semantic analysis along with the information that the sentence was in the simple past:

Decomposition: **become(inoperative(patient([sac1])))**

Verb form: Past

The output would be a representation of a transitional event, corresponding to the moment of *becoming inoperative*, and a resulting state in which the *sac* is inoperative.

event([fail1], become(inoperative(patient([sac1])), moment([fail1]))

state([fail2], inoperative(patient([sac1])), period([fail2]))

As shown above, situations are represented as predicates indicating the type of situation whose arguments are the pointer to the situation, the semantic decomposition, and the time argument. The temporal output also represents the information that the [fail1] event precedes the time at which the report was filed, and that the [fail2] state starts immediately after the momentn of the [fail1] event.

4. Nominalizations

Nominalizations are processed very similarly to clauses, but with a few crucial differences, both in linguistic information accessed and in the control of the algorithm. With respect to the linguistic information, the decomposition of a nominalization is the same as its corresponding clause, but the mapping rules differ since syntactically a nominalization is a noun phrase. For example, where a likely filler for the patient of *fail*, is the syntactic subject, a likely filler for the patient of *failure* is an *of* pp. In addition, no noun phrase modifiers are syntactically obligatory. This suggests the hypothesis that syntactically obligatory roles for clause decompositions automatically become pragmatically essential roles for nominalization decompositions. This hypothesis seems to hold in the current domain; however, it will have to be tested on other domains. Secondly, because nominalizations may themselves be anaphoric, there are two separate role-filling stages in the algorithm instead of just one. The first pass is for filling roles which are explicitly given syntactically; essential roles are left unfilled. If a nominalization is being used anaphorically some of its roles may have been specified or otherwise filled when the event was first described. The anaphoric reference to the event, the nominalization, would automatically inherit all of these role fillers, as a by-product of reference resolution. After the first pass, the interpreter looks for a referent, which, if found, will unify with the nominalization representation, sharing variable bindings. This is a method of filling unfilled roles pragmatically that is not currently available to clause analysis⁶. However, it is important to fill roles with any explicit syntactic arguments of the nominalization before attempting to resolve its reference, since there may be more than one event in the context which nominalization could be specifying. For example, *failure of pump* and *failure of sec* can only be distinguished by the filler of the patient role. After reference resolution a second role-filling pass is made, where still unfilled roles may be filled pragmatically with default values in the same way that unfilled verb roles can be filled.

As with clauses, the temporal analysis of nominalizations takes place after the semantic analysis of the nominalization and its arguments. Also as with clauses, one of the inputs to the temporal analysis of nominalizations is the semantic decomposition. This input is used in determining the inherent aspectual properties of the denoted situation, i.e., whether the nominalization inherently denotes a state, a process or an event situation. The critical difference between the two cases is that a nominalization does not occur with tense. Since this is one of the necessary inputs for the temporal analysis of situations, the tense used in analysing the temporal semantics of a nominalization must come from elsewhere in the sentence. The general principle followed in the temporal analysis of nominalizations is to look for relevant temporal

⁶ Clauses can describe previously mentioned events, as discussed in [Dahl1987]. In order to handle cases like these, something analogous to reference resolution for clauses may be required. However a treatment of this has not yet been implemented in PUNDIT.

information in the superordinate constituents in which the nominalization is embedded. In its current implementation, the temporal component can only handle a restricted set of contexts.

One of the functions of the temporal component is to identify predications which denote situations with actual temporal reference, i.e., situations which are said to have occurred or to be occurring at the time of the report. Whether or not a nominalization is presumed to denote an actual situation depends on the context in which it occurs. Currently, Pundit processes nominalizations in two types of contexts.

The first type of context where Pundit assumes a nominalization to denote an actual situation is where a nominalization occurs as the prepositional object of a temporal connective (e.g., *before*, *during*, *after*) and the matrix clause denotes an actual situation. For example, in the sentence *sac lube oil pressure decreased below 60 psig after engagement*, the temporal component processes the main clause as referring to an actual event which happened in the past and which resulted in a new situation. After processing the temporal information associated with the main clause, the temporal component finds the temporal adverbial phrase *after engagement*. Because *after* is a temporal connective (i.e., it relates the times of two situations), and because the main clause has actual temporal reference, the nominalization *engagement* is assumed to have actual temporal reference. For this set of cases, the nominalization is processed using the meaning of the prepositional adverb and the tense of the main clause.

The second type of context in which a nominalization undergoes temporal analysis is where it occurs as the argument to a verb whose arguments can be inferred to denote actual situations. The verb may provide only temporal information used in the analysis of the nominalization, in which case the verb is classified as an aspectual one. For example, in the sentence *failure occurred during engine start*, the matrix verb *occur* is an aspectual verb whose argument must be a reference to a process or an event (but not a state). The tense and aspect of the matrix verb are used in analyzing the temporal semantics of its argument. Hence, the clause *failure occurred* would be processed very similarly to a clause containing the simple past tense of the related verb, i.e., *something failed*.

Another type of verb whose nominalization arguments are presumed to denote actual situations is a verb which itself denotes an actual situation that is semantically distinct from its arguments. For example, the sentence *investigation revealed metal contamination in oil filter* mentions three situations: the situation denoted by the matrix verb *reveal*, and the two situations denoted by its nominalization arguments, *investigation* and *contamination*. In this respect, *reveal* differs from *occur*; in contexts where the verb *occur* refers to an actual occurrence, the argument to *occur* (e.g., *failure*) constitutes the occurrence rather than denoting a semantically distinct situation.

If the situation denoted by *reveal* has actual temporal reference, then its arguments are presumed to as well. Thus, if *reveal* were to occur in a modal context, e.g., *a full investigation might reveal the cause of contamination*, then neither the *reveal* state, nor its propositional arguments—the *investigation* process and the *contamination* state—would be assigned real temporal reference. However, in the sample sentence given here, all three situations would be processed as having actual temporal reference. The algorithm for clause analysis specifies that the referents for the nominalizations must be found during the semantic analysis of *reveal*. The temporal analysis of the nominalization *investigation*, e.g., is part of the process of finding the referent. That is, the temporal analysis of the nominalizations *investigation* and *contamination* precedes the temporal analysis of the verb *reveal*. Very briefly, the input to the temporal component for the analysis of the nominalization consists of the semantic decomposition and the tense of the matrix verb. This will be illustrated in more detail in the next section.

4.1. Nominalisation Mapping Rules

We will use the two nominalizations from the previous example, *investigation revealed metal contamination in filter*, to illustrate the nominalization analysis algorithm. We will describe the *contamination* example first, since all of its roles are filled by syntactic constituents. The dotted line divides the algorithm in Figure 2. into the parts that are the (above the line), and the parts that differ (below the line.)

DECOMPSE VERB - *Contaminate* decomposes into a NON-ESSENTIAL instrument that contaminates an OBLIGATORY location. *contaminate* <- contaminatedP(instrument(I),location(L))

FOR instrument role - In the example, *metal* is a noun modifier of *contamination*, and *metal1* is selected as the filler of the instrument role.

FOR theme ROLE - The mapping rules for a theme that could apply are:

theme(T) <- of_pp(T)
theme(T) <- in_pp(T)

The role is filled with *filter*, the referent of *in filter*.

At this point the temporal component is called for the nominalization *metal contamination in oil filter* with two inputs. The first is the decomposition structure resulting from the semantic analysis:

DECOMPOSE NOMINALIZATION
FOR EACH SEMANTIC ROLE:

**IF THERE ARE SYNTACTIC CONSTITUENTS -
 PROPOSE SYNTACTIC CONSTITUENT FILLER
 & CALL REFERENCE RESOLUTION
 & TEST SELECTIONAL RESTRICTIONS**

CALL TEMPORAL ANALYSIS ON DECOMPOSITION
CALL REFERENCE RESOLUTION FOR NOMINALIZATION NOUN PHRASE**FOR EACH SEMANTIC ROLE:**

**IF ESSENTIAL ROLE AND UNFILLED
 CALL REFERENCE RESOLUTION TO HYPOTHESE A FILLER
 & TEST SELECTIONAL RESTRICTIONS
 ELSE LEAVE UNFILLED**

Figure 2. Nominalisation Analysis Algorithm

contaminatedP(instrument([metal1]),location([filter1]))

The second is the tense of the matrix verb, which in this case is in the simple past. Inspection of the decomposition indicates that this predicate falls in the class of statives. States are situations which persist over a period of time without change. The representation of the **contamination** situation is thus a **state** predicate with a **period** time argument included along with the unique identifier (which will be eventually be instantiated by reference resolution as [contaminat1], see the next paragraph) and the decomposition:

**state([S],
 contaminatedP(instrument([metal1]),location([filter1])),
 period([S]))**

In this context, the past tense provides the information that at least one moment within the period of contamination (**period([S])**) precedes the time at which the report was filed.

CALL REFERENCE RESOLUTION FOR NOMINALIZATION - There are no previously mentioned **contamination** events, so a new referent, **contamination1** is created. There are no unfilled roles, so the analysis is completed. The final representation is the same as above.

4.2. Filling Essential Roles

The analysis of the other nominalization, *investigation*, illustrates how essential roles are filled. The decomposition of *investigation* has two semantic roles, a NON-ESSENTIAL agent doing the investigation and an OBLIGATORY theme being investigated.⁷

investigate <- **investigateP(agent(A),theme(T))**

There are no syntactic constituents, so the mapping stage is skipped, and reference resolution is called for the nominalization. There are no previously mentioned investigative events in this example⁸, so a new referent, *investigation1* is created. At this point, a second pass is made to attempt to fill any unfilled roles.

FOR agent ROLE - Case 4 applies, and it is left unfilled.

FOR theme ROLE - The selection restriction on the theme of an *investigation* is that it must be a **damaged** component or a **damage** causing event. All of the events and entities mentioned so far, the *sac* and the *pump*, the *failure of the sac* and the *shearing of the pump* satisfy this criteria. In this case, the item in focus, *the shearing of the pump*, would be selected [Dahl1986].

The final representation is:

investigateP(agent(A),theme(shear1))

5. Other Compounds

In addition to nominalizations, PUNDIT deals with three other types of noun-noun compounds. One is the category of nouns with arguments. These include *pressure* and *temperature*, for example. They are decomposed and have semantic roles like nominalizations; however, their treatment is different from that of nominalizations in that they do not undergo time analysis, since they do not describe temporal situations. As an example, the definition of *pressure*,

pressureP(theme(T),location(L)),

specifies **theme** and **location** as roles. In the CASREPS domain, the location must be a mechanical device, and the theme must be a fluid, such as oil or air. The analysis of a noun phrase like *sac oil pressure* would fill in the location role with the *sac* and the theme role with the *oil*, resulting in the final representation, **pressureP(theme(oil1),location(sac1))**. The syntactic mapping rules for the roles permit the theme to be filled in by either a noun modifier,

⁷ In other domains, the theme can be essential, as in "Let's investigate."

⁸ If the example had been, *A new engineer investigated the pump. The investigation occurred just before the complete breakdown.*, a previously mentioned event would have been found, and the agent and theme roles would have inherited the fillers *engineer1* and *pump1* from the reference to the previous event.

such as *oil* in this case, or the object of an *of* prepositional phrase, as in *pressure of oil*. Similarly, the mapping rules for the location allow it to be filled in by either a noun modifier or by the object of an *in* prepositional phrase. Because of this flexibility, the noun phrases, *see oil pressure*, *oil pressure in see*, and *pressure of oil in see*, all receive the same analysis.

The second class of compounds, is that of nouns which do not have semantic roles. For these, a set of domain-specific semantic relationships between head nouns and noun modifiers has been developed. These include: **area of object**, for example, *blade tip*, **material-form**, such as *metal particles*; and **material-object**, such as *metal cylinder*. These relationships are assigned by examining the semantic properties of the nouns. The corresponding prepositional phrases, as in *tip of blade*, *particles of metal*, and *cylinder of metal*, have a similar analysis.

Finally, many noun-noun compounds are handled as idioms, in cases where there is no reason to analyze the semantics of their internal structure. Idioms in the CASREPS domain include *ships force*, *gear shaft*, and *connecting pin*. Our decision to treat these as idioms does not imply that we consider them unanalyzable, or noncompositional, but rather that, in this domain, there is no need to analyze them any further.

6. Previous Computational Treatments

Previous computational treatments of nominalizations differ in two ways from the current approach. In the first place, they have often treated nominalizations as one type of noun-noun compound. This viewpoint is adopted by [Finin1980, Leonard1984, Brachman 1978]. Certainly many nominalizations contain nominal premodifiers and hence, syntactically, are noun-noun compounds; however, this approach obscures the generalization that prepositional phrase modifiers in non-compound noun phrases often have the same semantic roles with respect to the head noun as noun modifiers. *Repair of engine*, for example, should receive the same semantic analysis as *engine repair*. In PUNDIT, once a noun phrase is recognized as a nominalization, the treatment of noun pre-modifiers is very similar to the treatment of prepositional phrase post-modifiers. In essence, PUNDIT's analysis is aimed at a uniform treatment of the semantic similarity among expressions like *repair of engine*, *engine repair*, and *(someone) repaired engine* rather than the syntactic similarity of *engine repair*, *air pressure*, and *metal particles*. Of the analyses mentioned above, Brachman's analysis seems to be most similar to ours in that it provides an explicit link from the nominalization to the related verb to relate the roles of the noun to those of the verb. The second way in which our approach differs from previous approaches is that PUNDIT's analysis is driven by taking the semantic roles of the nominalization and try to fill them in any way it can. Other approaches, have tended to start by fitting the explicitly mentioned arguments into the role slots. This means that PUNDIT knows when a role is not

explicitly present, and consequently can call on the other mechanisms which we have described above to fill it in.

7. Limitations

The current system has two main limitations. First, there is no attempt to build internal structure within a compound. Each nominal modifier is assumed to modify the head noun unless it is part of an idiom. For this reason, noun phrases like *impellor blade tip erosion* cannot be handled by our system in its current state because *impellor blade tip* forms a semantic unit and should be analyzed as a single argument of *erosion*. The second problem is related to the first. The system does not now keep track of the relative order of nominal modifiers. In this domain, this does not present serious problems, since there are no examples where a different order of modifiers would result in a different analysis. Generally, only one order is acceptable, as in *sac oil contamination*, **oil sac contamination*.

8. Future Directions

There are a number of open research issues which are not addressed in this version of the system. So far, all nominalizations have been treated referentially, i.e., as specifying a particular event. Many nominalizations, however, refer to other types of situations, including non-specific, generic, modal, iterative, and opaque contexts. For example, in *Unit has low output pressure, resulting in slow gas turbine starts*, no particular gas turbine start is referred to; rather, the noun phrase describes a typical behavior of the gas turbine. Another example can be seen in *Request replacement of sac*, where no specific event of replacement is referred to. This is a general problem for descriptions of events, of course, and applies to clauses as well, such as *unable to start gas turbine*, and *Pump will not turn when engine jacks over*. These contexts raise many interesting issues for reference resolution and temporal analysis. Although we have not dealt with these problems, we believe that our approach to nominalizations will allow whatever treatments we develop for clauses of these kinds to be naturally extended to nominalizations.

9. Conclusion

In this paper we have described a treatment of nominalizations in which the goal is to maximize the similarities between the processing of nominalizations and that of the clauses to which they are related. The semantic similarities between nominalizations and clauses are captured by making the semantic roles, semantic decompositions, and selectional restrictions on the roles the same for nominalizations and their related verbs. As a result, the same semantic representation is constructed for both structures. This similarity in representation in turn allows reference resolution to find referents for nominalizations

which refer to events previously described in clauses. In addition, it allows the time component to integrate temporal relationships among events and situations described in clauses with those referred to by nominalizations.

On the other hand, where differences between nominalizations and clauses have a clear linguistic motivation, our treatment provides for differences in processing. PUNDIT recognizes that the semantic roles of nominalized verbs are expressed syntactically as modifiers of nouns rather than arguments of clauses by having a different set of syntactic mapping rules. It is also true in nominalizations that there are no syntactically obligatory arguments, so the analysis of a nominalisation does not fail when there is an unfilled obligatory role, as is the case with clauses. Finally, the temporal analysis component is able to take into account the fact that nominalizations are untensed.

While there are many cases not yet covered by our system, in general, we believe this to be an approach to processing nominalizations which is both powerful and extensible, and which will provide a natural basis for further development.

REFERENCES

[Abramson1984]

Harvey Abramson, Proc. of the First Annual National Conference on Artificial Intelligence. In *Proc. 1984 International Symposium on Logic Programming*, The Armed Forces Communications and Electronics Association, RADC Griffiss Air Force Base, Rome, NY, May 6-7, 1981, pp. 233-241.

[Dahl1986]

Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAI, Philadelphia, PA, 1986.

[Dahl1987]

Deborah A. Dahl, Determiners, Entities, and Contexts, Presented at Tinlap-3, Las Cruces, New Mexico, January 7-9, 1987.

[Fillmore1968]

C. J. Fillmore, The Case for Case. In *Universals in Linguistic Theory*, E. Bach and R. T. Harms (ed.), Holt, Rinehart, and Winston, New York, 1968.

[Finin1980]

Tim Finin, The Semantic Interpretation of Compound Nominals, PhD Thesis, University of Illinois at Urbana-Champaign, 1980.

[Hirschman1985]

L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Hirschman1986]

L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* 4, 1986, pp. 299-328.

[Leonard1984]

Rosemary Leonard, *The Interpretation of English Noun Sequences on the Computer*. North Holland, Amsterdam, 1984.

[Palmer1981]

Martha S. Palmer, A Case for Rule Driven Semantic Processing. *Proc. of the 19th ACL Conference*, June, 1981.

[Palmer1985]

Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[Palmer1986]

Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[Passonneau1986]

Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect, LBS Technical Memo No. 43, Logic-Based Systems Group, Unisys, Paoli, PA, November, 1986.

APPENDIX L

Situations and Intervals

This paper, by Rebecca J. Passonneau, will be delivered at the July 1987 meeting of the ACL. It explains the temporal structures of the three situation types— states, processes and events—and shows how they are computed.

Situations and Intervals¹

Rebecca J. Passonneau

May 2, 1987

Knowledge Systems²
Defense Systems, UNISYS

P.O. Box 517
Paoli, PA 19301
USA

Summary

The PUNDIT system processes descriptions of situations and the intervals over which they hold using an algorithm that integrates *tense logic* and *aspect*. It analyzes the main verb and its tense, taxis and grammatical aspect to generate representations of three types of situations: states, processes and events. Each type has a distinct temporal structure, represented in terms of intervals having two features: stativity vs. activity, and boundedness.

Topic Area: Temporal Semantics

¹This work was supported by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research. AP-PROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

²Formerly Paoli Research Center, SDC-A Burroughs Company.

1. Introduction

This paper describes a semantics of situations and the intervals over which they hold that is neither situation semantics (Barwise and Perry, 1983) nor interval semantics (Dowty, 1979, 1982, 1986; Taylor, 1977). It is unfortunately difficult to avoid the overlap in terminology because what will be described here shares with situation semantics the starting point that predication refers to situations. It shares with interval semantics the assumption that the times involved in references to situations are intervals. The concerns addressed here, however, arise from the computational task of processing descriptions of situations in natural language text in order to represent what predicates are asserted to hold over what entities and when.

The PUNDIT text-processing system³ processes references to situations using an algorithm that integrates *tense logic* (Reichenbach, 1947) with *aspect*, or what Talmy (1985) calls the "pattern of distribution of action through time." The algorithm (Passonneau, 1986) first analyzes the temporal semantics of the main verb of a sentence (i.e., its lexical aspect) and its concomitant categories of tense, *taxis*⁴ and grammatical aspect.⁵ This analysis provides a foundation for the subsequent interpretation of temporal adverbials by explicitly representing the components of time which can be modified. This paper describes how PUNDIT represents the temporal structure of three types of situations, namely states, processes and events, how these situations are located in time, and what the computational advantages of these representations are for interpreting temporal adverbials.

Three specific goals for processing references to situations were identified. The first was to distinguish references to actual time, i.e., to specific situations that are said to have

³PUNDIT is an acronym for Prolog UNDERstanding of Integrated Text. It is a modular system, implemented in Quintus Prolog, with distinct syntactic, semantic and pragmatic components (cf. Dahl, 1986; Palmer et al., 1986).

⁴*Taxis* (Jakobson, 1957) refers to the semantic effect of the presence or absence of the perfect auxiliary.

⁵Aspect is both part of the inherent meaning of a verb (lexical aspect) and also signalled by the presence or absence of the progressive suffix *-ing* (grammatical aspect).

occurred or to be occurring, as in 1),⁶

1) Oil pressure has been slowly decreasing.

from references to types of situations which have not occurred, might occur, or tend to occur, as in 2).

2) The lube oil pump seizes.

This distinction helps determine, among other things, what proposition tense applies to. In 2), the present tense does not pertain to a specific event of the pump seizing, but rather to the tendency for this type of situation to recur. Currently, PUNDIT only processes references to specific situations associated with actual time.

The second processing goal was to closely link the times at which situations hold with the lexical decompositions of the predicates used in referring to situations. This allows PUNDIT to represent precisely what kinds of situations entities participate in and when. The basic components of time are intervals while the two features associated with them are stativity versus activity, and boundedness. The former feature pertains to the internal structure of an interval while boundedness pertains to the way in which the temporal structure associated with a situation is located in time by tense and taxis (cf. § 3.2 below).

The final goal was to represent the times for which situations hold in a sufficiently rich manner to process temporal adverbials modifying different temporal components. For example, both 3) and 4) contain an *at* phrase locating a situation with respect to a clock time.

3) Pressure was low at 08:00.

4) The pump seized at 08:00.

But, the relation of the clock time to the two situations differs. In 3), 08:00 occurs within an interval over which the state of *pressure being low* holds. In 4), it coincides with a transition

⁶PUNDIT's current application is to process short messages texts called CASREPS (CASualty REPort)s which describe Navy equipment failures. As the examples illustrate, the texts often contain sentence fragments.

from a process of the pump becoming seized to a state of the pump being seized.

2. Problems in Computing Appropriate Representations

The critical problems in the semantic analysis of references to situations and their associated times are: 1) language encodes several different kinds of temporal information, 2) this information is distributed in many distinct linguistic elements, and finally, 3) the semantic contribution of many of these elements is context-dependent and cannot be computed without looking at co-occurring elements.

These problems have been addressed as follows. A decision was made to focus on the kinds of temporal information pertaining to the goals outlined in the previous section and to temporarily ignore other kinds of temporal information.⁷ Computation of this information was then divided into relatively independent tasks, with appropriate information passed between tasks to accommodate context-sensitivities. First, the linguistic input which is both syntactically obligatory and semantically critical is computed, i.e., the verb and its categories. This happens in two stages with the aspectual information (lexical and grammatical aspect) computed first and the relational information (tense and taxis) computed second. Since adverbial modification is optional, temporal adverbs are processed not only separately from but also subsequent to the interpretation of the verb and its categories.

3. Solution: Intervals and their Features

3.1. Temporal Structure

The input used to compute the temporal structure of a situation consists of the lexical aspect of the verb and its grammatical aspect. Situations are represented as predicates identifying the type of situation as a state, process or event; they take three arguments: a unique

⁷E.g., rate (given by adverbs like *rapidly*), "patterns of frequency or habituation" (cf. Mourelatos, 1981), and so on.

identifier of the situation, the semantic decomposition, and the time argument. Example 6) shows a simple stative sentence, the type of temporal structure it evokes, its semantic decomposition, and a representation of the situation it denotes. The same pointer ([low1]) identifies both the situation and its time argument because it is the actual time for which a situation holds which uniquely identifies it as a specific situation.

- 6) Sentence: The pressure is low.
 Temporal structure: unbounded stative interval
 Decomposition: lowP(patient([pressure1]))
 Situation: state([low1], lowP(patient([pressure1])), period([low1]))

As shown, stative predications hold over unbounded stative intervals. Stative intervals are represented as a period time argument to a state representation.

Interval semantics captures the distinct temporal properties of situations by specifying a single truth conditional relation between a predication and a unique interval. In contrast, PUNDIT associates two types of features with the intervals over which the predications hold. The feature of stativity is defined here just as stative predications are defined in interval semantics: *A sentence ψ is stative iff it follows from the truth of ψ at an interval I that ψ is true at all subintervals of I* (Dowty, 1986, p. 42). But stative predications are also defined in terms of the feature of boundedness, which pertains to how stative intervals participate in temporal ordering relations, as will be shown below (§§ 3.2,3.3).

A process is a situation which holds over an active interval of time. Active intervals are represented as a period time argument to a process representation. Again, the criterion for defining the internal structure of an active interval is borrowed from interval semantics: *A sentence ψ is an activity iff it follows from the truth of ψ at an interval I that ψ is true at all subintervals of I down to a certain limit in size* (Dowty, 1986, p. 42). But in addition, it is proposed that active intervals can be unbounded or unspecified for boundedness, depending on the grammatical aspect of the predication; both progressive and non-progressive process predications have the

same situational representation (cf. 7a, b).

7)	Sentence:	a) The pump is operating.	b) The pump operated.
	Grammatical Aspect:	progressive	non-progressive
	Temporal structure:	unbounded active interval	unspecified active interval
	Decomposition:	operateP(actor([pump1]))	
	Situation:	process([operate1], operateP(actor([pump1])), period([operate1]))	

But the different values of boundedness lead to different relations between event time and temporal structure, as will be shown below.

Transition event verbs denote a transition to a new situation. Following Dowty (1979), their decompositions contain the aspectual operator *becomeP*. The aspectual operator's argument is the predicate denoting the situation which results from a transition event. PUNDIT's representations of transition events differ from other proposals⁸ by linking distinct time arguments to distinct components of the semantic decomposition. A transition event consists of a process (of becoming) leading up to a new state or process. Its temporal structure is thus an active interval followed by—and bounded by—a new active or stative interval (cf. 8).

8)	Sentence:	The engine seized.
	Temporal structure:	active interval + transition bound + stative interval
	Decomposition:	becomeP(seizedP(patient([engine1])))
	Situations:	event([seize1], becomeP(seizedP(patient([engine1])), moment([seize1])) state([seize2], seizedP(patient([engine1])), period([seize2]))
	Temporal relation:	starts(moment([seize1]), period([seize2]))

Currently, PUNDIT does not explicitly represent the initial process.⁹ The transition is represented as an event with a moment time argument (e.g., moment([seize1])), and the resulting state is represented with a period time argument (e.g., period([seize2])) which starts at the moment of transition. A transition bound (e.g., moment([seize1]) is an abstract feature rather than a real component of time. It is represented because it participates in temporal ordering relations and can be directly modified by temporal adverbials. It can be thought of as the same kind of

⁸There are some similarities between my *transition bound* and the notion of *nucleus* proposed by Steedman and Moens; Steedman, personal communication.

boundary between intervals implied by Allen's *meets* relation (Allen, 1983; 1984, esp. p. 128).

3.2. Event Time

PUNDIT employs a Reichenbachian analysis of tense which temporally locates situations in terms of three abstract times: the time of the situation (*event time*), the time of speech/text production (*speech time*), and the time with respect to which relational adverbials are interpreted (*reference time*). Reichenbach (1947) did not distinguish between the temporal structure of a situation and its *event time*. For PUNDIT, the *event time* is an abstract component of temporal structure in terms of which ordering relations are specified. It is determined on the basis of boundedness, and is always represented as a dimensionless moment.

The three values of boundedness outlined above correspond to three possible relations of *event time* to a time argument. Examples 9) through 11) illustrate these relations. If an interval is unbounded (as in 6 and 7a above), its *event time* is represented as an arbitrary moment included within the interval:

- 9) The pressure is low.
Event time: M1 *such that includes*(period([low1]),moment([M1]))

For an interval unspecified for boundedness (as in 7b above) the *event time* has a non-committal relation to the interval, i.e., it may be an endpoint of or included within the interval:

- 10) The pump operated.
Event time: M1 *such that has*(period([operate1]),moment([M1]))

The moment time argument of a transition event is its *event time*:

- 11) The pump seized.
Event time: moment([seize1])

Defining these three different relations of *event time* to temporal structure simplifies the computation of the ordering relations given by the perfect and non-perfect tenses.

*This is simply a matter of convenience given the needs of the present application domain.

3.3. Temporal Ordering Relations

The event time and the verb's tense and taxis comprise the input used in computing temporal ordering relations. The different relations of event time to the temporal structures of situations captures several important facts about the interaction of tense and aspect. For example, only unbounded intervals allow present tense, thus present is computed for examples like 6) and 7) above, but not for examples like 2).

Also, a predication denoting a past unbounded situation can be followed by a predication asserting the continuation (or cessation) of the same situation:

- 12) The pump was operating at 08:00 but is no longer operating.

This is provided for by representing the event time for 12) as a moment included within an interval of indeterminate duration. A similar assertion following a past transition event predication is contradictory:¹⁰

- 13) ?The pump sheared the drive shaft and is still shearing it.

The event time for the first conjunct in 13) is a transition bound necessarily culminating in a new situation (i.e., a state of *being sheared*). Since the transition itself is dimensionless, the second conjunct cannot refer to its persistence. A predication evoking an unspecified interval in a similar context can be interpreted analogously to either 12) or 13):

- 14) The pump operated at 08:00 and is still operating.

The non-committal relation of event time to temporal structure for such cases makes both interpretations possible. Assigning a more specific interpretation is probably pragmatic rather than semantic in nature. As we will see next, the utility of distinguishing between unbounded and unspecified process predications is especially apparent in contexts containing temporal adverbials.

¹⁰The contradiction arises because one naturally interprets *still* as indicating persistence of the same event. One could interpret such a sentence without contradiction as indicating the iteration of the same type of event.

3.4. Conclusion: Adverbial Modification

The representations described above were inspired by remarks found in the literature on tense and aspect (cf. esp. Bull, Dowty, Mourelatos, Vendler) to the effect that "the time schemata" (Vendler, p. 98) associated with different situations are crucial to the way we perceive and talk about them. One of the crucial types of evidence used in deriving the representations was the interpretation of temporal adverbials in different contexts. Consequently, one of the advantages to the representations is that they make it possible to tailor the interpretation of a temporal adverb to the temporal structure of the modified situation.

For example, specifying a different relation for the event time of an active interval, depending on grammatical aspect, yields different temporal relations between the situations described in sentences like 14)-16).¹¹

- 14) The pump failed when the engine was rotating.
transition of failure during period of rotation
- 15) The pump failed when the engine rotated.
transition of failure during OR at one endpoint of period of rotation
- 16) The engine rotated when the pump failed.
Same as 15)

Sentences like 16) are often, but not always, interpreted with the process (e.g., *rotation*) beginning at or after the transition event moment (e.g., *failure*). PUNDIT's representations of the temporal semantics of predications are explicit enough yet sufficiently non-committal to provide suitable input to a pragmatic reasoner that could decide these cases.

Acknowledgements

I would like to thank several people for their comments, encouragement and patience: Martha Palmer, Lynette Hirschman, Bonnie Webber and Debbie Dahl.

¹¹ The different relations are shown informally in the examples; the formal representations PUNDIT generates will be given in the full paper.

References Cited

- Allen, James F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23.2: 123-154.
- Allen, James. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26.11:832-843.
- Barwise, Jon and John Perry. 1983. *Situations and Attitudes*. Cambridge, Massachusetts: The MIT Press.
- Bull, William E. 1971. [1963]. *Time, Tense and the Verb*. University of California Publications in Linguistics 19. Berkeley: University of California Press.
- Dahl, Deborah A. 1986. Focusing and Reference Resolution in PUNDIT Presented at AAAI, Philadelphia, PA.
- Dowty, David R. 1986. The effects of aspectual class on the temporal structure of discourse: semantics or pragmatics? *Linguistics and Philosophy* 9: 37-61.
- Dowty, David R. 1982. Tense, time adverbials and compositional semantic theory. *Linguistics and Philosophy* 5: 23-55.
- Dowty, David R. 1979. *Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's PTQ*. Dordrecht: D. Reidel.
- Jakobson, Roman. 1971 [1957]. Shifters, verbal categories and the Russian verb. In his *Selected Writings*, Vol. 2, pp. 130-147. The Hague: Mouton.
- Mourelatos, Alexander P. D. 1981. Events, processes, and states. In *Syntax and Semantics*, vol 14: *Tense and Aspect*, pp. 191-212. Edited by P. J. Tedeschi and A. Zaenen. New York: Academic Press.
- Palmer, Martha; Dahl, Deborah A.; Schiffman, Rebecca J. [Passonneau]; Hirschman, Lynette; Linebarger, Marcia; Dowding, John. 1986. *Recovering Implicit Information*. 24th Annual Meeting of the Association for Computational Linguistics. Columbia University, New York.
- Reichenbach, Hans. 1947. *Elements of Symbolic Logic*. New York: The Free Press.
- Passonneau, Rebecca. 1986. *A Computational Model of the Semantics of Tense and Aspect*. Logic-Based Systems Technical Memo No. 43. Paoli Research Center. System Development Corporation. December, 1986.
- Talmy, Leonard. 1985. Lexicalisation patterns: semantic structure in lexical forms. In *Language Typology and Syntactic Description*, vol. 3: *Grammatical Categories and the Lexicon*, pp. 57-151. Edited by Timothy Shopen. Cambridge: Cambridge University Press.
- Taylor, Barry. 1977. *Tense and continuity*. *Linguistics and Philosophy* 1.
- Vendler, Zeno. 1967. *Verbs and times*. *Linguistics in Philosophy*. New York: Cornell University Press.
- Vlach, Frank. 1981. The Semantics of the progressive. In *Syntax and Semantics*, vol 14: *Tense and Aspect*, pp. 271-292. Edited by P. J. Tedeschi and A. Zaenen. New York: Academic Press.

APPENDIX M

The Interpretation of Tense in Discourse

This paper by Bonnie Webber will be presented at the 25th Annual Meeting of the Association for Computational Linguistics, Palo Alto, July, 1987.

THE INTERPRETATION OF TENSE IN DISCOURSE

Bonnie Lynn Webber
Department of Computer & Information Science
University of Pennsylvania
Philadelphia PA 19104-6389

Abstract

This paper gives an account of the role tense plays in the listener's reconstruction of the events and situations a speaker has chosen to describe. Several new ideas are presented: (a) that tense is better viewed by analogy with definite NPs than with pronouns; (b) that a narrative has a temporal focus that grounds the context-dependency of tense; and (c) that focus management heuristics can be used to track the movement of temporal focus.¹

1. Introduction

My basic premise is that in processing a narrative text, a listener is building up a representation of the speaker's view of the events and situations being described and of their relationship to one another. This representation, which I will call an event/situation structure or e/s structure, reflects the listener's best effort at interpreting the speaker's ordering of those events and situations in time and space. The listener's problem can therefore be viewed as that of establishing where in the evolving e/s structure to attach the event or situation described in the next clause. My claim is that the discourse interpretation of tense contributes to the solution of this problem.

This work on the discourse interpretation of tense is being carried out in the context of a larger enterprise whose goal is an account of explicit anaphoric reference to events and situations, as in Example 1.

Example 1

It's always been presumed that when the glaciers receded, the area got very hot. The Folsom men couldn't adapt, and they died out. *That's* what's supposed to have happened. *It's* the textbook dogma. But *it's* wrong. They were human and smart. They adapted their weapons and culture, and they survived.

Example 1 shows that one may refer anaphorically to structured entities built up through multiple clauses. Thus an account of how clauses arrange themselves into structures is necessary to an account of event reference.²

¹This work was partially supported by ARO grant DAA29-84-9-0027, NSF grant MCS-8219116-CER, and DARPA grant N00014-85-K-0018 to the University of Pennsylvania, and by DARPA grant N00014-85-C-0012 to UNISYS.

²Other parts of the enterprise include a general mechanism for individuating composite entities made up of ones separately introduced [20, 21] and a representation for events that allow for anaphoric reference to both particular events and situations and to abstractions thereof [16].

In this paper, I will relate the problem of building up an e/s structure to what has been described as the anaphoric property of tense [7, 11, 6, 1, 12] and of relative temporal adverbials [18]. Anaphora are expressions whose specification is context-dependent. Tense and relative temporal adverbials, I interpret as specifying positions in an evolving e/s structure. My view of their anaphoric nature is that the particular positions they can specify depend on the current context. And the current context only makes a few positions accessible. (This I will claim to be in contrast with the ability of temporal subordinate clauses and noun phrases (NPs) to direct the listener to any position in the evolving structure.)

The paper is organized as follows: In Section 2, I discuss tense as an anaphoric device. Previous work in this area has discussed how tense is anaphoric, claiming as well that it is like a pronoun. While agreeing as to the source of the anaphoric character of tense, I do not think the analogy with pronouns has been productive. In contrast, I discuss what I believe to be a more productive analogy between tense and definite noun phrases.

Previous work has focussed on the interpretation of tensed clauses in simple linear narratives (i.e., narratives in which the order of underlying events directly corresponds to their order of presentation).³ Here the most perplexing question involves when the next clause in a sequence is interpreted as an event or sequence coincident with the previous one and when, as following the previous one [4, 6, 12]. In Section 3, I show that if one moves beyond simple linear narratives, there are more options. In terms of the framework proposed here, there may be more than one position in the evolving e/s structure which can provide a context for the interpretation of tense. Hence there may be more than one position in e/s structure which tense can specify and which the new event or situation can attach to.

To model the possible contexts, I introduce a discourse-level focussing mechanism - temporal focus or TF - similar to that proposed for interpreting pronouns and definite NPs [17]. I give examples to show that change of TF is intimately bound up with narrative structure. To keep track of and predict its movement, I propose a set of focus heuristics: one Focus Maintenance Heuristic, predicting regular movement forward, two Embedded Discourse Heuristics for stacking the focus and embarking on an embedded narrative, and one Focus Resumption

³Another person currently addressing the interpretation of tense and aspect in more complex narratives is Nakhimovsky [9, 10]. Though we are addressing somewhat different issues, his approach seems very compatible with this one.

Heuristic for returning and resuming the current narrative. The need for each of these is shown by example.

In Section 4, I show that relative temporal adverbials display the same anaphoric property as simple tense.

That the interpretation of tense should be entwined with discourse structure in this way should not come as a surprise, as a similar thing has been found true of other discourse anaphora [5].

2. Tense as Anaphor

Tense does not seem *prima facie* anaphoric: an isolated sentence like "John went to bed" or "I met a man who looked like a basset hound" appears to make sense without previously establishing when it happened. On the other hand, if some time or event is established by the context, tense will invariably be interpreted with respect to it, as in:

Example 2

After he finished his chores, John went to bed.
John partied until 3am. He came home and went to bed.

In each case, John's going to bed is linked to an explicitly mentioned time or event. This linkage is the anaphoric property of tense that previous authors have described.

Hinrichs [6] and Bauerle [1], following McCawley [7] and Partee [11], showed that it is not tense *per se* that is interpreted anaphorically, but that part of tense called by Reichenbach [14] reference time.⁴ According to Reichenbach, the interpretation of tense requires three notions: speech time (ST), event time (ET), and reference time (RT). RT is the time from which the event/situation described in the sentence is viewed. It may be the same as ST, as in

present perfect: ET<RT=ST

John has climbed Aconcagua and Mt. McKinley.

simple present: ET=RT=ST

John is in the lounge.
the same as ET, as in

simple past: ET=RT<ST

John climbed Aconcagua.

simple future: ST<ET=RT

John will climb Aconcagua.
in between ET and ST, as in

past perfect: ET<RT<ST

John had climbed Aconcagua.
or following both ET and ST (looking back to them), as in

future perfect: ST<ET<RT

John will have climbed Mt. McKinley.

That it is RT that it is interpreted anaphorically, and not either ET or tense as a whole can be seen by considering Example 3.

Example 3

John went to the hospital.
He had twisted his ankle on a patch of ice.

It is not the ET of John's twisting his ankle that is interpreted anaphorically with respect to his going to the hospital. Rather, it is the RT of the second clause: its ET is interpreted as prior to that because the clause is in the past perfect tense (see above).

Having said that it is the RT of tense whose interpretation is anaphoric, the next question to ask is what kind of anaphoric behavior it evinces. In previous work, tense is claimed to behave like a pronoun. Partee [12] makes the strongest case, claiming that pronouns and tense display the same range of antecedent-anaphor linkages:

Deictic Antecedents

pro: She left me! (said by a man crying on the stoop)⁵
tense: I left the oven on! (said by a man to his wife in the car)

Indefinite Antecedents

pro: I bought a banana. I took it home with me.
tense: I bought a banana. I took it home with me.
<I took it home after I bought it.>

Bound Variables

pro: Every man thinks he is a genius.
tense: Whenever Mary phoned, Sam was asleep.
<Mary phoned at time *t*, Sam was asleep at *t*>

Donkey Sentences

pro: Every man who owns a donkey beats it.
tense: Whenever Mary phoned on a Friday, Sam was asleep.
<Mary phoned at time *t* on a Friday, Sam was asleep at *t* on that Friday>

Because of this similarity, Partee and others have claimed that tense is like a pronoun. Their account of how time is then seen to advance in simple linear narratives is designed, in part, to get around the problem that while pronouns co-specify with their antecedents, the RT of clause N cannot just co-specify the same time as the previous clause [6, 12, 4].

There is another option though: one can draw an analogy between tense and definite NPs, which are also anaphoric. Support for this analogy is that, like a definite

⁴I believe that the deictic use of pronouns is infelicitous. In this example, the speaker is distraught and making no attempt to be cooperative. It happens. But that doesn't mean that pronouns have deictic antecedents. I include the example here because it is part of Partee's argument.

⁵Hinrichs' work is discussed as well in [12].

NP, tense can cause the listener to create something new. With a definite NP, that something new is a new discourse entity [19]. With tense, I will say for now that it is a new time at which the event or situation is interpreted as occurring.⁶ If one looks at texts other than simple linear narratives, this ability becomes clear, as the following simple example shows:

Example 4

I was at Mary's house yesterday.
We talked about her brother.
He spent 5 weeks in Alaska with two friends.
Together, they made a successful assault on Denali.
Mary was very proud of him.

The event of Mary's brother spending five weeks in Alaska is not interpreted as occurring either coincident with or after the event of my conversation with Mary. Rather, the events corresponding to the embedded narrative in the third and fourth clause are interpreted at a different spatio-temporal location than the conversation. That it is before the conversation is a matter of world knowledge. In the e/s structure for the whole narrative, the tense of the third clause would set up a new position for the events of the embedded narrative, ordered prior to the current position, to site these events.

The claimed analogy of tense with pronouns is based on the similarity in antecedent-anaphor linkages they display. But notice that definite NPs can display the same linkages in two different ways: (1) the definite NP can co-specify with its antecedent, as in the a. examples below, and (2) the definite NP can specify a new entity that is 'strongly' associated with the antecedent and is unique by virtue of that association, as in the b. examples below⁷

Deictic Antecedents

The car won't start! (said by a man crying on the stoop)

Indefinite Antecedents

- a. I picked up a banana. Up close, I noticed *the banana* was too green to eat.
- b. I picked up a banana. *The skin* was all brown.

Bound Variables

- a. Next to each car, the owner of *the car* was sleeping soundly.
- b. In each car, *the engine* was idling quietly.

Donkey Sentences

- a. Everyone who wants a car must fix *the car* himself.
- b. Everyone who owns a Ford tunes *the engine* himself.

Thus the range of antecedent-anaphor behavior that Partee calls attention to argues equally for an analogy between tense and pronouns as for an analogy between tense and definite NPs.

⁶After I say more about e/s structure construction, I will be able to claim that tense can cause the listener to create a new position in e/s structure at which to ascribe the event or situation described in its associated clause.

⁷Clark & Marshall [2] are among those who have described the necessary "common knowledge" that must be assumable by speaker and listener about the association for the specification to be successful.

However, there are two more features of behavior to consider: On the one hand, as noted earlier, definite NPs have a capability that pronouns lack⁸. That is, they can introduce a new entity into the discourse that is 'strongly' associated with the antecedent and is unique by virtue of that association, as in the b. examples above. Example 4 shows that tense has a similar ability. Thus, a stronger analogy can be drawn between tense and definite NPs.

On the other hand, definite NPs have the capability to move the listener away from the current focus to a particular entity introduced earlier or a particular entity associated with it. This ability tense lacks. While tense can set up a new node in e/s structure that is strongly associated with its 'antecedent', it does not convey sufficient information to position that node precisely - for example, precisely relative to some other event or situation the listener has been told about. Thus its resemblance to definite NPs is only partial, although it is stronger than its resemblance to pronouns. To locate a node precisely in e/s structure requires the full temporal correlate of a definite NP - that is, a temporal subordinate clause or a definite NP itself, as in Example 5.

Example 5

The bus reached the Stadium, terminal for the suburban bus services. Here De Witt had to change to a streetcar. The wind had abated but the rain kept falling, almost vertically now. He was travelling to a two o'clock appointment at Amsterdam police headquarters in the center of town, and he was sure to be late.

When De Witt got to the police president's office, he telephoned his house.

[adapted from Hans Koning, *De Witt's War*]

Notice that without the "when" clause, the simple past tense of "he telephoned his house" would be anaphorically interpreted with respect to the "reaching the Stadium" event, as happening sometime after that. A new node would be created in e/s structure ordered sometime after the "reaching the Stadium" event. On the other hand, with the "when" clause, that new node can be ordered more precisely after the "reaching the Stadium" event. By association with its "antecedent" (the "travelling to the appointment" event), it can be ordered after the achievement of that event.

There is another advantage to be gained by pushing further the analogy between tense and definite NPs that relates to the problem tackled in [6, 4, 12] of how to reconcile the anaphoric nature of tense with the fact that the event or situation described in the next clause varies as to whether it is taken to be coincident with, during, before or after the event or situation described in the previous clause. This I will discuss in the next section, after introducing the notion of temporal focus.

⁸except for "pronouns of laziness" which can evoke and specify new entities through the use of previous descriptions

3. Temporal Focus

In this section, I give a more specific account of how the discourse interpretation of tense relates to e/s structure construction.

At any point N in the discourse, there is one node of e/s structure that provides a context for the interpretation of the RT of the next clause. I will call it the temporal focus or TF. There are three possibilities: (1) the RT of the next clause will be interpreted anaphorically against the current TF, (2) the TF will shift to a different node of e/s structure - either one already in the structure or one created in recognition of an embedded narrative - and the RT interpreted with respect to that node, or (3) the TF will return to the node previously labelled TF, after completing an embedded narrative, as in (2), and the RT interpreted there. These three behaviors are described by four focus management heuristics described in this section: a Focus Maintenance Heuristic, two Embedded Discourse Heuristics and a Focus Resumption Heuristic.⁹

In [21], I presented a control structure in which these heuristics were applied serially. The next heuristic would only be applied when the prediction of the previous one was rejected on grounds of "semantic or pragmatic inconsistency". I now believe this is an unworkable hypothesis. Maintaining it requires (1) identifying grounds for such rejection and (2) arguing that one can reject proposals, independent of knowing the alternatives.

I now don't believe that either can be done. It is rarely the case that one cannot come up with a story linking two events and/or situations. Thus it would be impossible to reject a hypothesis on grounds of inconsistency. All one can say is that one of such stories might be more plausible than the others by requiring, in some sense not explored here, fewer inferences.¹⁰

Thus I would now describe these heuristics as running in parallel, with the most plausible prediction being the one that ends up updating both e/s structure and the TF. For clarity in presentation though, I will introduce each heuristic separately, at the point that the next example calls for it.

3.1. Interpreting RT against TF

Before presenting the temporal focus management heuristics, I want to say a bit more about what it can mean to interpret the RT of the next clause against the current TF. This discussion points out the additional advantage to

be gained by pushing the analogy between tense and definite NPs.

As I noted above, a definite NP can specify an entity 'strongly' associated with its antecedent. One might thus consider what is 'strongly' associated with an event. One answer to this question appears in two separate papers in this volume [8, 13], each ascribing a tripartite structure to the way we view and talk about events. This structure consists of a preparatory phase, a culmination, and a consequence phase, to use the terminology of [8]. (Such a structure is proposed, in part, to give a uniform account of how the interpretation of temporal adverbials interacts with the interpretation of tense and aspect.)

Nodes in e/s structure correspond to events and situations, as the speaker conceives them. If one associates such a structure with the node labelled the current TF, then one can say that 'strongly' associated with it are events and situations that could make up its preparatory phase, culmination or consequence phase. Like a definite NP, the RT of tense may either co-specify the current TF or set up a new node in e/s structure 'strongly' associated with the TF. In the latter case, its corresponding event or situation will be interpreted as being part of one of these three phases, depending on the speaker and listener's assumed shared knowledge. Since, arguably, the most common way of perceiving the world is as an ordered sequence of events, this will increase the plausibility of interpreting the next event or situation as (1) still associated with the current TF and (2) part of the consequence phase of that event (i.e., after it). On the other hand, this 'strong association' treatment no longer limits anaphoric interpretation to "co-specify" or "right after" as in [4, 6, 12]. The event described can be anaphorically associated with the the whole event structure (Example 6a), the consequence phase (Example 6b - "right after"), or the preparatory phase (Example 6c - "before").

Example 6

a. John walked across Iowa. He thought about Mary, who had run off with a computational linguist.

b. John walked across Iowa. He crossed the state line at Council Bluffs and headed west through Nebraska.

c. John walked across Iowa. He started in Sioux City and headed east to Fort Dodge.

Deciding which of these three options holds in a given case demands an appeal to world knowledge (e.g. which actions can be performed simultaneously by a single agent). This is yet another area demanding further study and is not treated in this paper.¹¹

⁹Rohrer [15] suggests that there may exist a set of possible temporal referents, possibly ordered by saliency, among which the tense in a sentence may find its reference time, but doesn't elaborate how. That is the only thing I have seen that comes close to the current proposal.

¹⁰Crain and Steedman [3] make a similar argument about prepositional phrase (PP) attachment. For example, it is not impossible for a cat to own a telescope - e.g., by inheritance from its former owner. Thus "a cat with a telescope" is not an inconsistent description. However, it must compete with other plausible interpretations like "seeing with a telescope" in "I saw a cat with a telescope".

¹¹Mark Steedman shares responsibility for this idea, which is also mentioned in his paper with Marc Moens in this volume [8].

3.2. Focus Maintenance and Focus Movement

The following pair of examples illustrate the simplest movement of temporal focus in a discourse and its link with e/s structure construction.

Example 7a

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses.
3. Unfortunately the roses failed to cheer her up.

Example 7b

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses.
3. He picked out 5 red ones, 3 white ones and one pale pink.

Since the first two clauses are the same in these examples, I will explain them together.

With no previous temporal focus (TF) established prior to clause 1, the listener creates a new node of e/s structure, ordered prior to now, to serve as TF. TF sites the anaphoric interpretation of RT_1 , which, because clause 1 is in the simple past, also sites ET_1 . This is shown roughly in Figure 3-1.

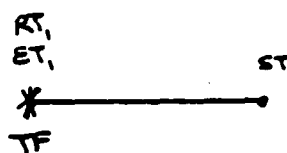


Figure 3-1: E/S structure after processing clause 1

The first heuristic to be introduced is a Focus Maintenance Heuristic (FMH).

After interpreting clause N, the new TF is the most recent TF - i.e., the node against which RT_N was interpreted.

The most recent TF is cotermporal with RT_1 . This new TF now provides a site for interpreting RT_2 . Since clause 2 is past perfect, ET_2 is interpreted as being prior to RT_2 . E/s structure is now roughly as shown in Figure 3-2.

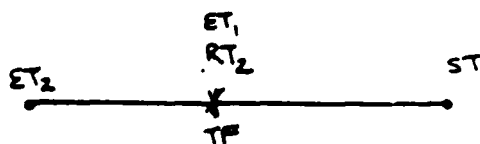


Figure 3-2: E/S structure after processing clause 2

Applying the FMH again, RT_2 is the new TF going into clause 3. Examples 7a and 7b here diverge in what subsequently happens to the TF.

In 7a, RT_3 can be anaphorically interpreted as immediately following the TF. Since RT_3 in turn directly

sites ET_3 (clause 3 being simple past), the "failing event" is interpreted as immediately following the "going over to Mary's house" event. This is shown roughly in Figure 3-3. (TF is shown already moved forward by the FMH, ready for the interpretation of the next clause, if any.)

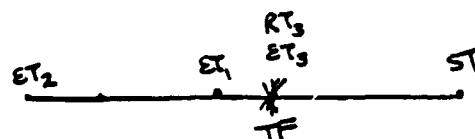


Figure 3-3: E/S structure after processing clause 7a-3

To get the most plausible interpretation of 7b - i.e., where the "rose picking" event is interpreted anaphorically with respect to the "flower shop" event - requires a second heuristic, which I will call an Embedded Discourse Heuristic. This will be EDH-1, since I will introduce another Embedded Discourse Heuristic a bit later.

If ET_N is different from $RT_N=TF$, treat utterance N as the beginning of an embedded narrative, reassign ET_N to TF (stacking the previous value of TF, for possible resumption later) and try to interpret RT_{N+1} against this new TF.

By this heuristic winning the plausibility stakes against the FMH, TF is reassigned to ET_2 (stacking the previous TF, which is sited at $RT_2=RT_1=ET_1$), and RT_3 is anaphorically interpreted as following this new TF. As before, ET_3 is sited directly at RT_3 (since simple past), so the "picking out the roses" event is viewed as immediately following the "stopping at the florist" event. This is shown roughly in Figure 3-4.

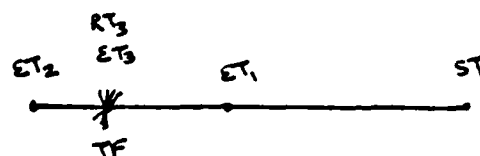


Figure 3-4: E/S structure after processing clause 7b-3

Now consider the following extension to example 7b.

Example 7c

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses.
3. He picked out 5 red ones, 3 white ones and one pale pink.
4. Unfortunately they failed to cheer her up.

First notice that clauses 2-3 form an embedded narrative that interrupts the main narrative of John's visit to Mary's. The main sequence of events that begins with clause 1 resumes at clause 4. Now consider the anaphoric interpretation of tense. Clauses 1-3 are interpreted as in Example 7b (cf. Figure 3-4). The problem comes in the interpretation of Clause 7c-4.

NO-A101 562

INTEGRATING SYNTAX SEMANTICS AND DISCOURSE DARPA
NATURAL LANGUAGE UNDERST. (U) UNISYS CORP PAOLI PA
PAOLI RESEARCH CENTER D DAHL ET AL. 14 MAY 87

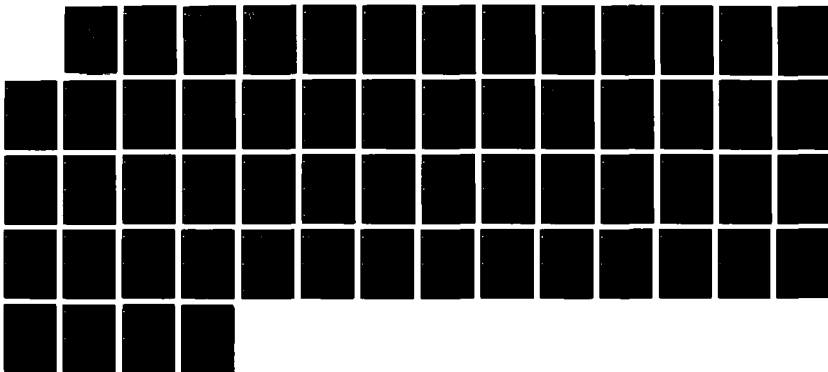
3/3

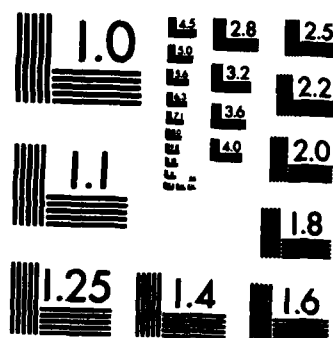
UNCLASSIFIED

NO0014-85-C-0012

F/G 5/7

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

To get the most plausible interpretation requires a third heuristic which I will call a Focus Resumption Heuristic (FRH).

At the transition back from an embedded narrative, the TF prior to the embedding (stacked by an Embedded Discourse Heuristic) can be resumed.

Using this heuristic, the previously stacked TF (sited at $RT_2=RT_1=ET_1$ - the "going to Mary's house" event) becomes the new TF, and RT_4 is interpreted as directly following it. Since clause 7c-4 is simple past, the "tailing" event is again correctly interpreted as immediately following the "going over to Mary's house" event. This is shown roughly in Figure 3-5.

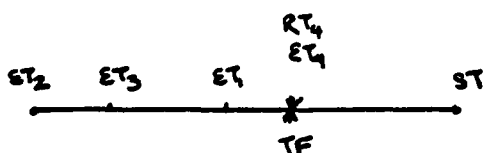


Figure 3-5: E/S structure after processing clause 7c-4

I have already noted that, like a definite NP, tense can cause the listener to create a new node in e/s structure to site its RT. What I want to consider here is the circumstances under which a reader is likely to create a new node of e/s structure to interpret RT_{N+1} , rather than using an existing node (i.e., the current TF, one associated with the previous event (if not the TF) or a previous, stacked TF).

One circumstance I mentioned earlier was at the beginning of a discourse: a reader will take an introductory sentence like Snoopy's famous first line

It was a dark and stormy night.

and start building up a new e/s structure with one node corresponding to ST and another node siting RT and ET. Generalizing this situation to the beginning of embedded narratives as well, I propose a second Embedded Discourse Heuristic (EDH-2):

If clause $N+1$ is interpreted as beginning an embedded narrative, create a new node of e/s structure and assign it to be TF. Stack the previous value of TF, for possible resumption later.

EDH-2 differs from EDH-1 in being keyed by the new clause itself: there is no existing event node of e/s structure, different from the current TF, which the embedded narrative is taken to further describe.

EDH-2 explains what is happening in interpreting the third clause of Example 4. Even though all the clauses of Example 4 are simple past, with $ET=RT$, the third clause is most plausibly interpreted as describing an event which has occurred prior to the "telling about her brother" event. EDH-2 provides the means of interpreting the tense in an embedded narrative whose events may occur either before or even after the current TF.

Example 4

1. I was at Mary's house yesterday.
2. We talked about her brother.
3. He spent 5 weeks in Alaska with two friends.
4. Together, they made a successful assault on Denali.
5. Mary was very proud of him.

Notice that the focus stacking specified in EDH-2 enables the correct interpretation of clause 4-5, which is most plausibly interpreted via the FRH as following the "telling about her brother" event.

EDH-2 is also relevant for the interpretation of NPs headed by de-verbal nouns (such as "trip", "installation", etc.). While such a NP may describe an event or situation, there may not be enough information in the NP itself or in its clause to locate the event or situation in e/s structure (cf. "my trip to Alaska" versus "my recent/upcoming trip to Alaska"). On the other hand, EDH-2 provides a way of allowing that information to come from the subsequent discourse. That is, if the following clause or NP can be interpreted as describing a particular event/situation, the original NP and the subsequent NP or clause can be taken as co-specifying the same thing. Roughly, that is how I propose treating cases such as the following variation of Example 4:

Example 8

1. I was talking with Mary yesterday.
2. She told me about her trip to Alaska.
3. She spent five weeks there with two friends, and the three of them climbed Denali.

The NP "her trip to Alaska" does not of itself cause an addition to e/s structure.¹² Rather, application of EDH-2 to the interpretation of clause 5-3 results in the creation of a new node of e/s structure against which its RT is sited. Other reasoning results in clause 3 and "her trip to Alaska" being taken as co-specifying the same event. This is what binds them together and associates "her trip to Alaska" with a node of e/s structure.

Finally, notice that there will be an ambiguity when more than heuristic makes a plausible prediction, as in the following example:

Example 9

1. I told Frank about my meeting with Ira.
2. We talked about ordering a butterfly.

It is plausible to take the second utterance as the beginning of an embedded narrative, whereby EDH-2 results in the "talking about" event being interpreted against a new node of e/s structure, situated prior to the "telling Frank" event. (In this case, "we" is Ira and me.) It is also plausible to take the second utterance as continuing the current narrative, whereby FMH results in the "talking about" event being interpreted with respect to the "telling Frank" event. (In contrast here, "we" is Frank and me.)

¹²It does, of course, result in the creation of a discourse entity [19]. The relationship I see between the listener's e/s structure and his/her discourse model is discussed in [21].

4. Temporal Focus and Temporal Adverbials

So far I have only shown that clauses containing no other time-related constructs than tense can be interpreted anaphorically against more than one site in e/s structure. Now I want to show, at least by example, that what I have proposed holds for clauses containing relative temporal adverbs as well. Relative temporal adverbials must be interpreted with respect to some other time [18]. So consider the italicized forms in the following brief texts.

John became the captain of Penn's squash team.
He was *previously* captain of the Haverford team.

John left for London *on Sunday*.
Tuesday he went to Cambridge.

Tuesday John went to Cambridge.
On Sunday, he left for London.

Previously is interpreted with respect to the previously mentioned "becoming captain" event: it was before that that he was captain at Haverford. In the second case, the adverbial *On Sunday*, given no previous link in the discourse, is interpreted with respect to ST. However, *Tuesday* is then interpreted with respect to the event of John's leaving for London: it is interpreted as the Tuesday after that event. The third case is the reverse.

What I want to show is that, as before, the same four heuristics predict the sites in e/s structure that may provide a context for a relative temporal adverbial. Consider the following.

Example 10a

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses.
3. After five minutes of awkwardness, he gave her the flowers

Example 10b

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses.
3. After 20 minutes of waiting, he left with the bouquet and fairly ran to Mary's.

I will use ADV to refer to the interpretation of the "after" adverbial. In these cases, what is sited by TF is the beginning of the interval. What in turn sites the RT of the main clause is the end of the interval.

The processing of the first two clauses is just the same as in examples 7a and b. From here, the two examples diverge.

In 10a-3, the beginning of ADV is most plausibly interpreted with respect to the TF. The end of ADV in turn provides an anaphoric interpretation point for RT₃. Since ET₃ is interpreted as coincident with RT₃ (clause 3 being simple past), the "rose giving" event is interpreted as immediately following John's getting to Mary's house. This is shown roughly in figure 4-1.



Figure 4-1: E/S structure after processing clause 10a-3

In 10b-3, the interpretation due to FMH is less plausible than that due to EDH-1. EDH-1 re-assigns TF to ET₂, where the beginning of ADV is then sited. The end of ADV in turn provides an anaphoric interpretation point for RT₃. Since ET₃ is sited at RT₃, the "leaving with the bouquet" event is sited at the end of the twenty minutes of waiting. This is shown roughly in Figure 4-2.

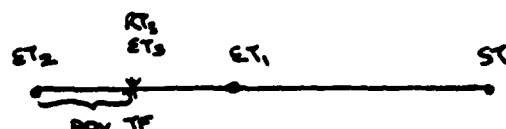


Figure 4-2: E/S structure after processing clause 10b-3

An interesting question to consider is whether a speaker would ever shift the TF as modelled by the FRH or the EDH-2, while simultaneously using a relative temporal adverbial whose interpretation would have to be linked to the new TF, as in example 11 (movement via FRH) and example 12 (movement via EDH-2).

Example 11

1. John went over to Mary's house.
2. On the way, he had stopped by the flower shop for some roses
3. He picked out 5 red ones, 3 white ones and one pale pink.
4. After 5 minutes of awkwardness, he gave her the flowers.

Example 12

1. I was at Mary's house yesterday.
2. We talked about her brother.
3. After 6 months of planning, he went to Alaska with two friends.
4. Together, they made a successful assault on Denali.
5. Mary was very proud of him.

I find both examples a bit awkward, but nevertheless understandable. Accounting for TF movement in each of them is straightforward. However, whether to attribute the awkwardness of these examples to exceeding people's processing capabilities or to a problem with the theory is grist for further study.

5. Conclusion

In this paper, I have given what I believe to be a credible account of the role that tense plays in the listener's reconstruction of the events and situations a speaker has chosen to describe. I have provided support for several new ideas: (a) that tense is better viewed by analogy with definite NPs than with pronouns; (b) that a narrative has a temporal focus that grounds the context-dependency of tense; and (c) that focus management heuristics can be used to track the movement of temporal focus. I have also identified a host of problems that require further work, including (1) how to incorporate aspectual interpretation into the model, (2) how to evaluate 'strong associations' between events and/or situations and (3) how to judge plausibility.

Acknowledgments

I would like to extend my thanks to Debby Dahl, Martha Palmer and Becky Passonneau at UNISYS for their enthusiastic support and trenchant criticism. I have also gained tremendously from discussions with James Allen, Barbara Grosz, Erhard Hinrichs, Aravind Joshi, Hans Kamp, Ethel Schuster, Candy Sidner, and Mark Steedman.

References

1. Bauerle, R.. *Temporale Deixis, temporale Frage*. Gunter Narr Verlag, Tubingen, 1979.
2. Clark, H. & Marshall, C. Definite Reference and Mutual Knowledge. In *Elements of Discourse Understanding*, A.K. Joshi, B.L. Webber & I.A. Sag, Ed., Cambridge University Press, Cambridge England, 1981, pp. 10-63.
3. Crain, S. & Steedman, M. On not being Led up the Garden Path: the use of context by the psychological syntax processor. In *Natural Language Parsing*, D. Dowty, L. Karttunen & A. Zwicky, Ed., Cambridge Univ. Press, Cambridge England, 1985, pp. 320-358.
4. Dowty, D. "The Effects of Aspectual Class on the Temporal Structure of Discourse: Semantics or Pragmatics". *Linguistics and Philosophy* 9, 1 (February 1986), 37-62.
5. Grosz, B. & Sidner, C. "Attention, Intention and the Structure of Discourse". *Computational Linguistics* 12, 3 (July-September 1986), 175-204.
6. Hinrichs, E. "Temporal Anaphora in Discourses of English". *Linguistics and Philosophy* 9, 1 (February 1986), 63-82.
7. McCawley, J. Tense and Time Reference in English. In *Studies in Linguistic Semantics*, C. Fillmore & D.T. Langendoen, Ed., Holt, Rinehart and Winston, Inc., New York, 1971, pp. 97-114.
8. Moens, M. & Steedman, M. Temporal Ontology in Natural Language. Proc. of the 25th Annual Meeting, Assoc. for Computational Linguistics, Stanford Univ., Palo Alto CA, July, 1987. This volume..
9. Nakhimovsky, A. Temporal Reasoning in Natural Language Understanding. Proc. of EACL-87, European Assoc. for Computational Linguistics, Copenhagen, Denmark, April, 1987.
10. Nakhimovsky, A. Tense, Aspect and the Temporal Structure of the Narrative. Submitted to Computational Linguistics, special issue on computational approaches to tense and aspect.
11. Partee, B. "Some Structural Analogies between Tenses and Pronouns in English". *Journal of Philosophy* 70 (1973), 601-609.
12. Partee, B. "Nominal and Temporal Anaphora". *Linguistics and Philosophy* 7, 3 (August 1984), 243-286.
13. Passonneau, R. Situations and Intervals. Proc. of the 25th Annual Meeting, Assoc. for Computational Linguistics, Stanford Univ., Palo Alto CA, July, 1987. This volume..
14. Reichenbach, H.. *The Elements of Symbolic Logic*. The Free Press, New York, 1966. Paperback edition.
15. Rohrer, C. Indirect Discourse and 'Consecutio Temporum'. In *Temporal Structure in Sentence and Discourse*, V. Lo Cascio & C. Vet, Ed., Foris Publications, Dordrecht, 1985, pp. 79-98.
16. Schuster, E. Towards a Computational Model of Anaphora in Discourse: Reference to Events and Actions. CIS-MS-86-34, Dept. of Comp. & Info Science, Univ of Pennsylvania, June, 1986. Doctoral thesis proposal..
17. Sidner, C. Focusing in the Comprehension of Definite Anaphora. In *Computational Models of Discourse*, M. Brady & R. Berwick, Ed., MIT Press, Cambridge MA, 1982, pp. 267-330.
18. Smith, C. Semantic and Syntactic Constraints on Temporal Interpretation. In *Syntax and Semantics, Volume 14: Tense & Aspect*, P. Tedeschi & A. Zaenen, Ed., Academic Press, 1981, pp. 213-237.
19. Webber, B.L. So What Can We Talk about Now? In *Computational Models of Discourse*, M. Brady & R. Berwick, Ed., MIT Press, Cambridge MA, 1982, pp. 331-371.
20. Webber, B.L. Event Reference. Theoretical Issues in Natural Language Processing (TINLAP-3), Assoc. for Computational Linguistics, Las Cruces NM, January, 1987, pp. 137-142.
21. Webber, B.L. Two Steps Closer to Event Reference. CIS-86-74, Dept. of Comp. & Info Science, Univ. of Pennsylvania, February, 1987.

APPENDIX N

Report on an Interaction between the Syntactic and Semantic Components

This report by Marcia Linebarger describes the design of an interaction between syntax and semantics to allow input from the semantic component to the parser to guide the parser to a semantically acceptable parse.

Report on an Interaction between the Syntactic and Semantic Components

Marcia Linebarger

This report describes an attempt to develop a more flexible control strategy in syntactic and semantic processing of text, by a mechanism which forces semantics to evaluate the partial products of syntactic analysis.

Currently, Pundit's control mechanism flows from the syntactic module to the semantic module; the parser operates without any reference to the semantic properties of the structures it builds. The cost of this sequential operation is that the parser frequently engages in pointless structure building, which could have been prevented by allowing the semantic module to analyze the output of the parser at some point earlier than the end of the clause.

Below we describe a restriction which calls the semantic analyzer during the syntactic parse and uses semantic information to limit the search space of the parser. Our initial efforts have been directed toward a restriction (*sv_checkpoint*) which is called at the point at which the subject and main verb have both been parsed, but before the object has been analyzed. This is a particularly felicitous moment for such interaction because (1) the parser is able to provide semantics with considerable information, i.e., the identity of the verb and one of its arguments; and (2) the parser is confronted at this point with a great variety of syntactic options, so if it is possible to abort incorrect analyses at this point we may prevent needless structure building.

This report describes the design of *sv_checkpoint*. The motivation for developing this mechanism has been primarily to test the ability of the semantic component to process partially built syntactic structures, and to point up those areas in which the communication between the two modules needs further development. Following successful implementation of this restriction, additional interactions between syntax and semantics will be explored, with particular concern for the processing of noun phrases.

Sv_checkpoint is housed in the BNF object rule, and applies during the dynamic editing of the object rule in accordance with the subcategorization features of the verb (see [9], included as Appendix D.)¹ It passes the parsed subject and verb to the semantic analyzer. The semantic analyzer attempts to assign the subject a thematic role in accordance with the verb decomposition and syntax-semantics mapping rules. The information sent back to the parser is used either to alter the parser's expectations about upcoming structure or to abort the current analysis.

¹Another possibility is to call the restriction immediately following the construction of the *lvvr* or *lvv* node, i.e., upon completion of the verb itself; this would prevent construction of intervening sentence adjuncts prior to the call to semantics, since the assertion rule permits sentence adjuncts between verb and object.

1.1. Prediction of upcoming structure

One important set of temporary ambiguities which the restriction will sometimes be able to resolve in mid-parse involves TRANSITIVITY ALTERNATION verbs which are subcategorized as both intransitive (i.e., as taking *nullobj*, in the string grammar formalism) and transitive (i.e., as taking *nstgo*). There are many such verbs; consider *melt* and *decrease*. (an asterisk indicates that the sentence is semantically or pragmatically anomalous, either generally or in the CASREP domain; all sentences below are *syntactically* well-formed.)

- (1)(a) The flame melted the ice cube. (*nstgo*)
 - (b) The ice cube melted. (*nullobj*)
 - (c) *The flame melted. (*nullobj*)
 - (d) *The ice cube melted something. (*nstgo*)
- (2)(a) The arsonists torched the building. (*nullobj*)
 - (b) The building torched easily.
 - (c) *The arsonist torched easily.
 - (d) *The building torched something.

It is clear that in these sentences the choice between the intransitive and the transitive uses of the verb can be made at any point after the subject and main verb have been analyzed. In the (a) sentences above, we know that the verb cannot be intransitive (given the anomaly of the (c) sentences), and in the (b) sentences we know at this point that the verb cannot be transitive (note the anomaly of the (d) sentences).

Sv_checkpoint allows us to utilize this information in mid-parse by analyzing the subject and verb as if they constituted a complete clause. If the clause analyzer is able to assign a thematic role to the subject, and if no obligatory roles remain unfilled, then the parser is directed to reorder the object list so that *nullobj* is the first object option tried. That is, the parser is instructed to PREFER the intransitive analysis. If, on the other hand, the clause analyzer is able to assign a thematic role to the subject but obligatory roles remain unfilled, then the parser is directed to delete *nullobj* from the list of object options; it is prevented from considering the intransitive analysis. The effect of *sv_checkpoint* in these two cases is to alter the BNF object rule itself before the parser begins to apply it.

Note that the distinction made here between PREDICTING UPCOMING STRUCTURE and FAILING THE CURRENT ANALYSIS is minimal for the cases considered above. It may be that we will collapse the two cases by limiting *sv_checkpoint* to either accepting or rejecting a given analysis. However, the distinction we are making between prediction and failure has some motivation.

Consider first the cases in which the *nullobj* option is reordered to the head of the object list. These could also be handled by having *sv_checkpoint* trigger failure out of the *nstgo* object option rather than by reordering *nullobj* so that

it is the first option considered. However, the reordering mechanism is of interest to us as an experiment in the expression of WEIGHTED PREFERENCES for (rather than outright elimination of) particular parses.

And the cases in which nullobj is deleted from the object list could equally be handled by constructing and then failing nullobj; since nullobj has no internal structure, it clearly is not expensive to build and therefore could just as well be built and then dismantled as excised from the object list. Our interest in the predictive approach arises out of the possibility that in other cases the subject-verb combination may allow us to reject a more elaborated object option even before it is built, rather than building and then rejecting this option.

A hypothetical such case is a sentence beginning *the tornado claimed...* This sentence may be completed by an *natgo*, as in *The tornado claimed three lives*; but it may not be completed by a clause, as in *The tornado claimed that three lives were lost*. This is because the two object options to correspond to utterly distinct senses of the verb. Rather than allowing the parser to attempt to build a clausal object, it may be that *sv_checkpoint* could be strengthened in the future so that when it is presented with *the tornado claimed...* it removes *assertion* and *thats* (both clausal object options) from the list of object options. This would require that the semantic component examine the possible thematic roles available to the object; and, on the basis of both general and verb-specific mapping rules, send back to the parser a list of BNF object options that may provide semantics with an acceptable argument. For example, let us say that *claim* is associated with two argument structures:

(3) *claim(agent,proposition)* - "He claims that this is true."

(4) *claim(actor, patient)* - "The disease claimed three victims."

In both cases, mapping rules will assign the first role to the syntactic subject. Ignoring, for the moment, the different roles assigned to the subject in the two argument structures, note that (3) will have semantic class restrictions imposed upon its subject, i.e., that the subject be animate; on the other hand, (4) will not impose such semantic restrictions upon its subject. Thus semantics can eliminate the FIRST argument structure above because *tornado* will fail the semantic class restriction. Since a syntactic clause (such as *that three lives were lost*) can never express a patient argument, the semantic analyzer could, at this point, eliminate from the BNF options to be considered by the parser all those options which cannot express a patient argument. This will result in the removal of *thats* (the clause object option) from the object list. Thus the parser will be prevented from even attempting to build a clause; and will be spared, for example, the construction of a huge garden path in cases such as (5) which are initially ambiguous between a noun phrase (with *member* as its head) and assertion (with *that well-known member* as its subject) analysis. And it will be prevented from generating any parse at all for the syntactically correct but semantically anomalous (6).

- (5) The tornado claimed among its victims that well-known member of parliament who was decorated several times during the war.
(Acceptable natgo)
- (6) The tornado claimed that well-known members of parliament who were decorated several times during the war had in fact been spying for the enemy.
(Anomalous thats)

Thus the distinction between prediction of upcoming structure and dismantling of a completed structure is maintained despite the fact that in a number of cases the former approach holds no advantage over the latter. The usefulness of the distinction remains an empirical issue.

We consider now a range of cases in which `sv_checkpoint` rejects a completed analysis on the basis of a call to semantics.

1.3. Aborting the current analysis

Another set of ambiguities which are sometimes resolvable during the parse involves the identification of the main verb. Main verbs are sometimes indistinguishable from passive participles or nouns. Consider first the main verb/passive participle ambiguity, which holds of *dropped*, *failed*, and *increased* in the following example.

- (7) Oil pressure dropped to 72 psi then increased to 90 psi and then failed while starting gas turbine. (Testb 13.1.1)

The entirety of (7) up to the period is analyzable syntactically as a single noun phrase (*the oil pressure which was dropped to 72 psig, was increased to 90 psi, and then was failed while starting the gas turbine...*), since all three verbs are syntactically transitive and have indistinguishable forms for *tv* (the inflected main verb) and *ven* (the past participle). This ambiguity gives rise to considerable pointless structure-building unless constrained semantically. If the *ven* analysis is considered first, then the parser misanalyzes sentences like (7) as noun phrases until the final punctuation mark is encountered. If the *tv* analysis is considered first, then other sentences will be misanalyzed until the real *tv* is encountered. And in some cases, this may not occur until a great deal of structure has been built:

- (8) Overheating expected to result in engine failure and subsequent malfunctions of number 4 sac has not occurred.

If *expected* is analyzed as the main verb with subject *overheating*, then a huge garden path is created: until the parser encounters *has*, it has no indication that

the material up to this point represents a large noun phrase (*the overheating which was expected to result in engine failure and subsequent failure of number 4 sac*); rather, it has incorrectly posited a single word subject followed by an infinitival complement containing a transitive verb with compound object (*The overheating harbored the expectation that it would result in engine failure and subsequent failure of number 4 sac*). All of this structure-building will have been in vain.

Sv_checkpoint will help to constrain this ambiguity; it allows us to try the tensed verb analysis first, and reject it immediately if it is patently anomalous. Thus the BNF grammar orders the null option of **rn** (the right adjunct of the noun phrase) before the **venpass** (passive participle phrase) option; this means that *dropped* in (7) and *expected* in (8) will be analyzed first as tensed verbs. This eliminates the massive garden path in (7), but leads the parser to misanalyze (8) in the way described above. **Sv_checkpoint**, however, will reject this analysis because there is no thematic role available for an inanimate subject of *expect*. Thus this analysis is aborted early on, and the parser backtracks into the subject, trying the **venpass** option in **rn**, the noun phrase right adjunct position.

In some cases, both the **ven** and the **tv** analyses yield a syntactically viable parse, with the result that semantics be sent a parse corresponding to a semantically/pragmatically anomalous interpretation. To consider an example from the CASREPs, the parser first misanalyzes (9) below as an assertion with tensed verb *believed*.

- (9) High lube oil temperature due to design of first flight oil cooler believed contributor to unit failure.
(Testb 26.1.5)

The only parse of (9) as an assertion must analyze *believed* as a main verb, in which the temperature is asserted to believe some proposition. However, the correct parse is as a fragment in which *believed...* is a **venpass** predicated of *temperature* (paraphraseable as *the high lube oil temperature was believed to be a contributor to the unit failure*). **Sv_checkpoint** will prevent the generation of this incorrect parse. Since the restriction is called at the point at which *temperature* has been analyzed as the subject of *believe*, it will elicit a rejection from semantics (temperature cannot harbor beliefs) and the assertion parse is aborted immediately.

And, in fact, **sv_checkpoint** should be able to abort misanalyses in a wide range of structures where a lexical item may or may not be correctly analyzed as the main verb. By sending the ambiguous item to semantics as a main verb along with its putative subject, the parser may often get early warning of an incorrect analysis.

For example, many verbs (e.g., *request*) are also analyzable as nouns. This ambiguity results in garden paths and also in syntactically correct but semantically inappropriate parses. Consider (10) and (11) below.

(10) He had the unit repair request forms cancelled by the head office. (sven)

(11) He had the secretary request some forms. (svo)

The semantically correct analysis of (10) below analyzes *request* as a noun in an sven structure (sven is a small clause consisting of subject plus passive participle phrase), paraphraseable as *He had the the forms which expressed a request for engine repairs cancelled by the head office*. However, the parser will also produce an analysis in which *request* is the main verb of an svo clause (svo is a clause with untensed verb). The anomalous interpretation of (10) is analogous to the use of *request* in (11); on this reading (10) may be paraphrased *He forced the unit repair to make a request for forms which were cancelled by the head office*.

Sv_checkpoint will allow semantics to reject the misanalysis of *request* as a v (untensed verb) in (10) immediately, since repairs do not make requests. And in a corpus with many elaborate compound nouns, such ambiguities are legion; even when they do not result in incorrect parses, they may create massive garden paths.

Thus sv_checkpoint guides the parser in several ways. It allows for the dynamic reordering and pruning of the BNF object rule in accordance with the thematic possibilities of the subject, and it allows us to abort a wide range of syntactic misanalyses resulting from the ambiguity of English verb forms.

One obvious limitation of this restriction is that it is only as effective as the semantic information which it exploits. Thus for PUNDIT to benefit significantly from the implementation of sv_checkpoint, the semantic constraints on fillers of thematic roles available to the subject will have to be developed in greater detail.

1.1. Relationship to the selection mechanism

As noted above, our purpose in developing this mechanism has been to develop greater flexibility in PUNDIT's control strategy, and to develop the interface between the syntactic and semantic components. Ultimately, however, it remains an empirical issue which method of interaction will prove the most efficient: (a) no interaction between the syntactic and semantic components prior to the completion of the parse; (b) interaction by means of selectional patterns of the sort described in section 6 and Appendix O; or (c), interaction by means of a mechanism like sv_checkpoint which transfers control to semantics during the parse. Our early experiments have suggested that (b) is more efficient than (c) as well as than (a); thus it may be more efficient to replace sv_checkpoint with selectional patterns. This will require an Intermediate Syntactic Representation that can represent information in partially constructed constituents (i.e., subject plus verb, or left adjunct plus head). An important research goal is therefore the development of closer links between selectional and semantic information, so that

selectional patterns may be exploited for the development of the semantic data base, and semantic information may be used to generate selectional patterns and to extend LEXICAL selectional patterns in accordance with verb decompositions and syntax mapping rules.

APPENDIX O

Improved Parsing Through Interactive Acquisition of Selectional Patterns

The report on "Improved Parsing", by Francois-Michel Lang and Lynette Hirschman, will be issued as a Paoli Research Center Technical Report. It describes a mechanism for collecting valid sublanguage co-occurrence patterns, and their use in pruning the search focus during parsing. Through the use of selectional patterns, the average number of parses dropped from 4.7 parses/sentence to 1.5 parses/sentence.

Improved Parsing Through Interactive Acquisition of Selectional Patterns¹

François-Michel Lang and Lynette Hirschman

Paoli Research Center
Unisys Defense Systems
P. O. Box 517, Paoli, PA 19301

(215) 648-7490

ABSTRACT

This paper presents a module of the PUNDIT natural-language system designed to facilitate porting the system to new domains, and to improve the accuracy and efficiency of the parser. The module operates interactively by querying the user about word patterns appearing in certain syntactic combinations (e.g., subject-verb-object) found in partially constructed parses. When presented with a word pattern, the user has two choices: (1) if the pattern is semantically consistent with the domain, the user accepts it, signalling a correct parse; (2) if it is semantically anomalous, the user rejects it, signalling an incorrect parse, and failing that parse. These word patterns collected by the program are stored in a pattern database which can be automatically consulted by the parser. Using this module in interactive parsing has reduced the number of parses found for the sentences in one of our corpuses from an average of 4.7 to 1.5, and decreased the time spent parsing by approximately one-third.

¹This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research, and in part by National Science Foundation contract DCR-85-02205, as well as by Independent R&D funding from System Development Corporation, now part of Unisys Defense Systems.

1. INTRODUCTION

A frequent problem encountered in parsing with large, broad-coverage grammars is that such grammars often produce a great number of parses. Our Prolog implementation of restriction grammar [Hirschman1982,Hirschman1985], which includes about 100 grammar rules and 75 restrictions, produces over 15 parses for five of the sentences in one of our corpuses. A majority of these parses, however, are incorrect because they violate some domain-specific semantic constraint. For example, two of the parses for the sentence *High lube oil temperature believed contributor to unit failure* could be paraphrased as

- (1) The high lube oil temperature believed the contributor to the unit failure.
- (2) The high lube oil temperature was believed to be a contributor to the unit failure.

but our knowledge of the domain (and common sense) tells us that the first parse is wrong, since temperatures cannot hold beliefs.

It is only because of this semantic information that we know that parse (2) is correct, and that parse (1) is not, since we cannot rule out parse (1) on syntactic grounds alone. In fact, our grammar generates the incorrect parse before the correct one, since it produces assertion parses before fragment parses. If a grammar has access to domain knowledge of this kind, however, many incorrect parses such as (1) will never be generated.

How then can we collect semantic information about a domain? One approach would involve analysing the corpus of data by hand, or perhaps even simply relying on one's intuitive knowledge of the domain in order to gather information about what relations can hold among domain entities. Several obvious drawbacks to these approaches are that they are time-consuming, error-prone, and incomplete. A more robust approach would be to develop some (semi-) automated tools designed to collect such information by interacting with a user familiar enough with the domain to distinguish good parses from bad ones.

We see now that our reasoning appears circular: Valid selectional information can only be obtained from analyses of correctly parsed structures, but our goal is to restrict the parser to these correct analyses precisely by using valid selectional information. In the example above, we need selectional information to rule out the bad parse; but it is only by knowing which parse is bad that we can obtain this information.

One way to avoid this circularity is bootstrapping into a state of increasingly complete domain knowledge. We have implemented such a bootstrapping process by incrementally collecting and storing selectional data gathered through interaction with the user [Hirschman1986]. Data collected from the user can then be used to help guide the parser to correct analyses and to decrease the search space traversed during future parsing. As the the system's semantic knowledge becomes increasingly rich, we can expect it to demonstrate some measure of learning, since it will produce fewer incorrect analyses and present fewer queries to the user about the validity of syntactic patterns.

2. A SUBLANGUAGE FOR EQUIPMENT FAILURE DIAGNOSIS

The sentence cited above, *High lube oil temperature believed contributor to unit failure*, is, of course, a sentence (-fragment) of the English language. More importantly, however, it is a sentence from a specific *sublanguage* of English. A sublanguage is a highly specialised and

often idiosyncratic form of a natural language used by specialists in a given field for communication strictly within that field. Examples of sublanguages which have been studied previously are the language of weather reports [Chevalier1978], aircraft maintenance manuals [Lehrberger1983], medical reports [Hirschman1983], and equipment-failure reports [Marsh1984].

According to Zellig Harris, who was one of the first linguists to study the use of language in restricted domains [Harris1968], a sublanguage shares the syntax of the general language,¹ but is distinguished from the general language by constraints on what words can co-occur in certain syntactic patterns, such as a subject-verb-object structure. A well-formed sentence of a sublanguage must therefore not only meet the syntactic criteria of the general language (and whatever syntactic restrictions the sublanguage imposes), but also satisfy any co-occurrence restrictions specific to that sublanguage. In a medical sublanguage, for example, the sentence *The X-ray revealed a tumor* would be well-formed, but not the sentence *The tumor revealed an X-ray*.

Our domain deals with mechanical failures of a component of a ship's engine called a *starting air compressor* (SAC). The sentences in our corpuses are from a sublanguage dealing with diagnosis of equipment failures, and consist of casualty reports (CASREPs) detailing the causes and results of SAC failures.

3. METHODOLOGY

The essential feature of our parser which facilitates the collecting of syntactic patterns is the INTERMEDIATE SYNTACTIC REPRESENTATION (ISR) produced by the syntax processor. The ISR is the result of regularising the surface syntactic structure into a canonical form of operators and arguments. Since the ISR regularises syntactic patterns into a canonical form, there are only a fairly limited number of patterns which can appear in an ISR. We have therefore been able to write a program to analyse the ISR and examine the syntactic patterns as they are generated.

A brief note about the implementation: Since the ISR is represented as a Prolog list, the program which analyses it was written as a definite-clause grammar and has the flavor of a small parser. As a sample ISR, we present the regularised representation of the obvious parse for the sentence *The field engineer repaired the broken sac* (pretty-printed for clarity):

```
[past,repair,
  [tpos(the),
    [nvar([field-engineer,singular,_])]],
  [tpos(the),
    [nvar([sac,singular,_])],
    adj([break])]]
```

At the top level, the ISR consists of the main verb (preceded by its tense operators), followed by its subject and object. The ISR of a noun phrase contains first the determiner, *tpos(the)*, then the head noun, *nvar([field-engineer,singular,_])* (the label NVAR stands for "noun or variant"), and finally any nominal modifiers, such as *adj([break])*. Note that part of the regularisation performed by the ISR is morphological, since the actual lexical items appearing in the ISR are represented by their root forms. Hence *broken* in the input sentence is regularised to *break* in the ISR, and *repaired* in the input sentence appears in the ISR simply as *repair*.

¹The allowable syntactic constructions of a sublanguage may actually be a superset of those of the general language, for example, when the sublanguage allows a telegraphic style.

The selectional pattern checker is invoked by two restrictions which are called after the BNF grammar has assembled a complete NP (and constructed the ISR for that NP), and after it has assembled a complete sentence (and constructed its ISR). The program operates by presenting to the user a syntactic pattern (either a head-modifier pattern or a predicate-argument pattern) found in the ISR, and querying him/her about the acceptability of that pattern. For each of the eleven types of patterns which the program currently generates, the following chart shows that pattern's components, an example of that pattern, and a sentence which would generate the pattern (all these sentences are naturally occurring, in that they are taken from our corpus of CASREPs, in certain cases in a somewhat simplified form):

PATTERN	COMPONENTS	EXAMPLE
(1) SVO	subject, main verb, object	inspection reveal particle
INSPECTION of lube oil filter REVEALED metal PARTICLES.		
(2) SBeO	subject, main be-verb, object	erosion be evident
EROSION of impellor blade tip IS EVIDENT.		
(3) ADJ	adjective, head*	normal pressure
Troubleshooting revealed NORMAL sac lube oil PRESSURE.		
(4) ADV	head, adverb	decrease rapid
Sac air pressure DECREASED RAPIDLY to 5.74 psi.		
(5) CONJ	conjunct, conjunction, conjunct ₂	pressure and temperature
Troubleshooting revealed normal PRESSURE AND TEMPERATURE.		
(6) NOUN	noun modifier, head	valve part
VALVE PARTS excessively corroded.		
(7) PREP	head, prep, object	disengage after alarm
DISENGAGED immediately AFTER ALARM.		
(8) PREDN	noun, predicate nominal	capability necessity
alarm CAPABILITY is a NECESSITY.		
(9) QPOS	quantifier, head noun	65 psig
oil pressure dropped below 65 PSIG.		
(10) QN	quantifier, noun, head noun	0.25 inch chip
1/4 INCH CHIPS are visible on leading edge.		
(11) NQ	noun, quantifier, head noun	number 4 sac
NUMBER 4 SAC oil pressure dropped below alarm point.		

*We use "head" throughout the chart to denote the head of the construction in which the modifier (in this case, an adjective) appears. The head can simply be thought of as that word which the modifier modifies.

Recall that part of the regularisation performed by the ISR is morphological, since each word in a pattern is represented by its root. Hence in example (4) above (the adverb pattern), we find *rapid* and not *rapidly*.

When presented with a pattern, the user can respond to the query in one of two ways,² depending on the semantic compatibility of the predicate and arguments (e.g., in the case of an SVO pattern) or of the head and modifiers (e.g., for an ADJ pattern) contained in the pattern. If the pattern presented describes a relationship that can be said to hold among domain entities (i.e., if the pattern occurs in the sublanguage), the user accepts the pattern, thereby classifying it as good. The analysis of the ISR (and the parsing of the English sentence) is then allowed to continue. If, however, the pattern describes a relationship among domain entities that is not consistent with the user's domain knowledge or with his/her pragmatic knowledge (i.e., if the pattern cannot or does not occur in the sublanguage) the user rejects it, classifying it as bad, and signalling an incorrect parse. This response causes the restriction which called the program to fail, and as a result, the parse under construction is immediately failed, and the parser backtracks.³

As the user classifies patterns into "good patterns" and "bad patterns", they are stored in a pattern database. This database is consulted before any query to the user is made, so that once a pattern has been classified as good or bad, the user is not asked to classify it again. If a pattern previously classified as bad by the user (and therefore in the DB) is encountered in the course of analysing the ISR, the program consults the database, recognises that the pattern is bad, and automatically fails the parse being assembled. Similarly, if a pattern previously recorded as good is encountered, the program will recognise that the pattern is good simply by consulting the database (and not querying the user), and allow the parsing to proceed. The algorithm is described in Figure 1 below.

4. SOME (SIMPLIFIED) EXAMPLES

4.1. SVO Patterns

Let us now consider in more detail the sentence mentioned above, *High lube oil temperature believed contributor to unit failure*. In the ISR of the correct parse for this sentence in which the verb *believe* is a passive, the atom SOMEBODY/THING is used as the placeholder for the subject, and the atom CLAUSE represents the small clause *oil [be] contributor*, which is the object. In an incorrect and somewhat amusing reading (in which the verb *believe* is active), *temperature* is the subject, and *contributor* the direct object.

Possible paraphrases for the two parses of the sentence given above (the first incorrect, the second correct) are:

- (1) The high lube oil temperature believed the contributor to the unit failure.
- (2) The high lube oil temperature was believed to be a contributor to the unit failure.

²There are actually more than two options, as we explain below.

³This explanation is actually a bit oversimplified, because failing a NOUN pattern generated by a compound nominal consisting of at least three nouns does not necessarily cause the entire parse under construction to fail, but only indicates that the pairing in that specific pattern is infelicitous. Essentially, every noun appearing in this kind of structure will form patterns in which it will be successively paired with every other noun appearing to its right until either a pattern is generated which the user accepts, or no more patterns can be generated for that noun, in which case the entire parse will then fail. For example, the compound nominal *see drive shaft* would generate the NOUN pattern *see drives*, which the user would fail, but the entire parse would not fail since the pattern *see shaft* would also be generated, and this pattern would be accepted.

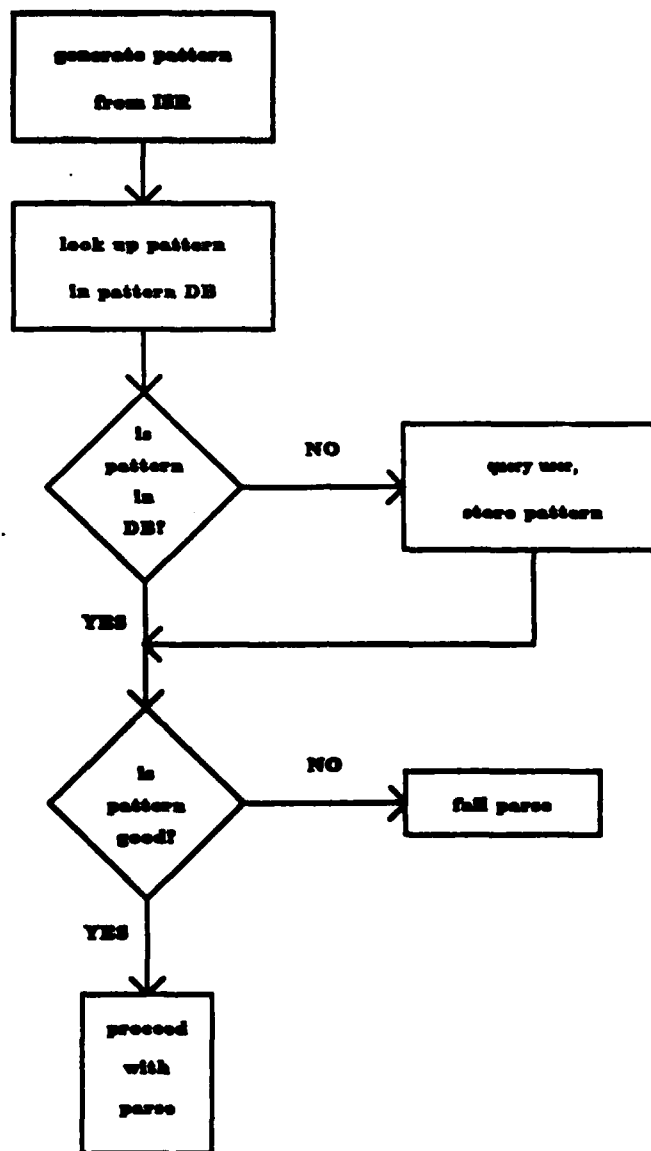


Figure 1: Selectional Pattern Algorithm

After the parser has generated the first parse for the sentence (and constructed the ISR corresponding to that parse), the ISR is passed to the selectional pattern checker, which queries the user about the SVO pattern *temperature believe contributor*.⁴ The query to the user consists of the type of pattern (e.g., SVO), and the words forming the instance in question of the

⁴In this simplified explanation, we present only the SVO patterns. In actual parsing of this sentence, however, additional patterns would be generated from the NP level. We also make the simplifying assumption that the pattern database contains no patterns relevant to this sentence.

pattern (e.g., *temperature believe contributor*). In our example, the message to the user reads

SVO : temperature believe contributor

In presenting this pattern to the user, the program is asking whether the noun *temperature* can be the subject of the verb *believes* with *contributor* as the direct object.

The user can then respond by typing either *yes* or *no*, depending on whether or not the pattern is a good one. In our example, the user would fail the pattern (by typing *no*), since in our domain model (and in the larger world) one cannot speak of temperatures believing things. Parse (1) would then fail. The parser then tries to generate another parse for the sentence, and presents the pattern

SVO : SOMEBODY/THING believe CLAUSE

asking whether it is reasonable to speak of some unspecified subject believing a proposition expressed by a clause. The user would accept this pattern, since in the domain (and, again, in the real world as well) propositions expressed by clauses can be objects of belief. Since the pattern was judged good, the parse would then proceed.

Multiple syntactic analyses of this sort, which can only be disambiguated by using semantic information, abound in our corpus because of the telegraphic and fragmentary nature of the equipment-failure reports. The ambiguity has two principal causes:

- (1) As we have seen, a sentence whose correct parse is a fragment reading can often be parsed as a full assertion as well.
- (2) Determiners are often omitted from our sentences, thus making it difficult to establish NP boundaries.

Since such syntactically degenerate sentences will generally contain fewer syntactic markers than full, non-telegraphic English sentences, they are marked by correspondingly greater ambiguity.

An especially convoluted example is found in the sentence *Experienced loss of oil pressure and self-disengagement immediately following clutch engage command*. In the correct reading, the subject is elided, the main verb is *experienced*, and the direct object is the conjunction *loss and self-disengagement* (in this parse, *immediately following clutch engage command* functions as a sentence adjunct, with *clutch engage command* as a compound nominal). However, in another reading generated by our grammar, the subject is the conjunction *experienced loss of oil pressure AND self-disengagement immediately following clutch*, the main verb is *engage*, and *command* is the direct object. This reading fails selection because *command* is not an acceptable object of the verb *engage*, which requires a machine part as an object.

4.2. PP Patterns

For an example of the use of prepositional-phrase patterns, consider the sentence *Loud noises were coming from the drive shaft during coast down*, which is syntactically ambiguous, since the prepositional phrase *during coast down* could modify either the noun *drive shaft* or the

verb *come*. (In the correct reading, the prepositional phrase modifies the verb.) In parsing this sentence, the user would be presented first with the PP pattern

PP : drive shaft during coast down

which would be rejected, since it is not correct to give mechanical parts temporal attributes. The incorrect reading would thus be blocked. The pattern checker would then query the user about the pattern

PP : come during coast down

which would be accepted, since events can take temporal modifiers. The correct reading of the sentence would therefore be allowed.

PP patterns are also essential for obtaining the correct analysis of the sentence *Loss of second installed sac*, for which there are two possible parses. In the correct analysis, the sentence is parsed as a noun string fragment; however, another reading is available in which the sentence is analysed as an assertion, with *loss of second* as the subject, *installed* as main verb, and *sac* as direct object. A paraphrase of this parse might be *The loss of a second installed the sac*. But this analysis is semantically completely anomalous for several reasons, most notably because it makes no sense to speak of the loss of a second installing a sac (or installing anything else). As with the previous example, the incorrect reading is produced first, since the parser tries assertion parses before fragment parses. In generating the assertion parse, the parser encounters the PP pattern *loss of second*, and queries the user as follows:

PP : loss of second

This pattern asks if a second can be lost in this domain, or if a domain expert would ever refer to the loss of a second. Although this pattern certainly appears plausible (and in fact, in the larger world one certainly can speak of losing precious seconds), the only entities in the SAC domain that can be lost are machine parts (as in the correct analysis of this sentence), and scalar quantities such as temperature and pressure (as in *Loss of lube oil pressure during sac failure*). Our domain does not deal with time-critical events, since repairing starting air compressors does not require split-second timing, so a reference to losing a second would never appear in an equipment-failure report. By contrast, if we were dealing with a medical sub-language, and our reports described procedures followed in an operating room or in administering emergency care, the time factor would be crucial, and losing a second could well be mentioned in a report.

The correct response to the program's query in this case is therefore to reject this pattern, causing the assertion parse to fail. Note also that if we did not reject this PP pattern for the reasons explained, the program would then present the user with the SVO pattern *loss install sac*:

SVO : loss install sac

This SVO pattern would certainly be rejected, since, as we mentioned, it is nonsensical to speak of a loss installing a sac, and so the assertion parse would fail because of the SVO pattern even if it did not because of the PP pattern.

Multiply ambiguous sentences do, of course, generate more than just the few patterns associated with the examples given above. Consider by contrast the sentence *Cannot engage sac for extended period of time due to decrease in lube oil temperature*, in which the multiple possible attachments of the four prepositional phrases generate no fewer than twenty-two readings,

of which only one is strictly correct.

5. LOCAL JUDGEMENTS

We said above that there were actually more than two possible responses that the user could make to the system's queries. We have provided the user with two other choices, called *locally good* and *locally bad*, which differ from the good and bad patterns discussed above only in that their effect is restricted to the sentence currently being parsed.

These options serve two principal functions:

- (1) Allowing the user to postpone making a global decision about the goodness or badness of a pattern (that is, not adding patterns to the (global) pattern database), while at the same time guiding the parser to a correct parse of the sentence
- (2) Dealing with a troublesome phenomenon which we call "phrasal attributes" [Hirschman1986] in which the semantic class of a complex structure (e.g., a full NP) is different from that of its syntactic head (see below). Consequently, the full phrase and its (unmodified) head will show different distribution.

"Locally good" patterns are used to deal with phrasal attributes. An example of this phenomenon taken from a medical domain is the noun phrase *stiff neck*: The semantic class of the head noun of this NP, *neck*, is something like BODY-PART, but the semantic class of the full NP *stiff neck* is not BODY-PART, but rather SYMPTOM or AILMENT. This discrepancy between the semantic classes of the full NP and of its head noun presents a difficulty in making a decision about the acceptability of patterns generated. For example, in parsing the sentence *Patient has stiff neck*, the system would present to the user the SVO pattern

SVO : patient have neck

Note that this is indeed the correct syntactic parse (in fact, probably the only one), but we do not want to assert for posterity that the SVO pattern *patient have neck* is semantically acceptable in a medical sublanguage.

This is perhaps a subtle point, but not everything that is true in a sublanguage can be said in that sublanguage. The sentence *Patient has stiff neck* is a case in point: Although it is certainly true that the patient has a neck, nobody would ever (bother to) say so because the proposition is completely uninformative. Indeed, it is one of the characteristics of a sublanguage that certain (true) information is presupposed, and never explicitly stated.

In short, the parse is good, but the pattern *patient have neck* is bad. We do not want to say the pattern is good, but saying it is bad will fail the parse, and that is not a desirable result either. Hence the appropriate response to the query about this pattern would be to tag it as "locally good", which is a sort of compromise implemented in order to allow the parse to succeed, but without entering the pattern in question into the (global) pattern database.

Our method of dealing with this phenomenon is admittedly not satisfactory. However, pending a fuller semantic treatment of NPs which allows such distinctions to be made, it at least permits the correct parse to be obtained without creating obviously bad patterns.

For an example of the phrasal-attribute phenomenon from the SAC domain, consider the

sentence *Start air pressure dropped below 30 psig*, which generates the SVO pattern *drop below psig*. The problematic NP here is *30 psig*: the semantic class of the head noun *psig* is UNIT-OF-MEASUREMENT, yet we would not say that the full NP *30 psig* is a UNIT-OF-MEASUREMENT; *30 psig* is instead an entity of the class LEVEL or perhaps THRESHOLD. The problem is that in evaluating the pattern *drop below psig*, we would realise that pressure can drop below a certain level or below a certain threshold, but it cannot drop below a unit of measurement. The solution is to tag this pattern as locally good.

The "locally bad" category serves to indicate that a certain pattern is being generated by an incorrect parse for the particular sentence, but that the pattern is not necessarily inconsistent with the domain's semantics. In fact, a pattern originally marked as "locally bad" could later be classified as good if it appears in the correct parse of another sentence. For example, the sentence from our SAC domain *Loss of oil pressure* would generate the prepositional phrase pattern

PP : loss of oil

Now although this pattern is not generated by the correct parse for this sentence (the correct parse would have instead *loss of pressure*), we probably do not want to go so far as to say that the pattern *loss of oil* cannot or would not occur in our domain. By responding that the pattern *loss of oil* is locally bad, we immediately fail the parse, but do not add the pattern to the global database of bad patterns. Again, we would not classify this pattern as *good* until we had evidence of its actual occurrence in the sublanguage (or unless a native speaker of the sublanguage asserted its validity.)

6. EXPERIMENTAL RESULTS

The experimental results we present here are based on a sample of 31 CASREP sentences from our TESTA corpus, each of which was parsed with and without the selectional mechanism. We compare results obtained without using the selectional module to results obtained with the parser set to query the user about selectional patterns (starting from an empty pattern database). The chart below summarises the results for the 31 CASREP sentences. A more detailed chart, showing the results for each sentence, is included as an appendix.

In gathering these statistics, we used a parsing procedure slightly different from our usual one in that backtracking into lexical lookup was allowed just in case no parse was found for a given set of definitions. This approach prevented generating multiple spurious parses for sentences containing idioms. Without this adjustment, for example, a sentence containing the word sequence *lube oil* would receive one parse in which *lube oil* was analysed as an idiom (i.e., the lexical item *lube oil*), and another in which *lube oil* was analysed as a compound nominal made up of the distinct lexical items *lube* and *oil*.

One of the statistics presented here is the SEARCH FOCUS, which is a measure of the efficiency of the parser in either reaching the correct parse of a sentence, or in parsing to completion. It is equal to the ratio of the number of nodes attached to the parse tree (and possibly detached upon backtracking)⁵ in the course of parsing, and the number of nodes in the completed, correct parse tree. Thus a search focus of 1.0 in reaching the correct parse would indicate that for every (branching) grammar rule tried, the first option was the correct one, or, in other words, that the parser had never backtracked.

⁵Another way to interpret this figure is that it represents the number of grammar rules tried.

Statistical Summary of TESTA Sentences

PARSING INFORMATION	W/OUT SELECTION	WITH SELECTION
# of sentences receiving a correct parse	29	31
# of sentences receiving a correct FIRST parse	17	30
# of sentences receiving more than one parse	22	8
average # of parses found* per sentence	4.66	1.45
average correct parse number	2.45	1.10
average search focus to reach correct parse	24.48	19.60
average search focus in parsing to completion	51.74	38.67
average time taken to reach correct parse	56.18	35.92
average time taken in parsing to completion	125.94	81.63

*That is, which parse, on the average, was the correct one.

SEARCH FOCUS RATIO TO CORRECT PARSE = 0.80

(i.e., ratio of search focus in reaching correct parse with selection to search focus in reaching correct parse without selection)

SEARCH FOCUS RATIO TO COMPLETION = 0.75

TIMING RATIO TO CORRECT PARSE = 0.64

TIMING RATIO TO COMPLETION = 0.65

NEW CORRECT PARSES FOUND USING SELECTION = 2

NEW CORRECT FIRST PARSES FOUND USING SELECTION = 13

7. FUTURE PLANS

There are several areas in which we have specific plans to extend the system.

7.1. Semantic Class Patterns

The mechanism as currently implemented deals only with lexical patterns (i.e., patterns involving specific lexical items appearing in the lexicon). However, we are currently investigating methods to generalise the program by using information taken from the domain ~~is~~ hierarchy to construct semantic class patterns from the lexical patterns. For example, our domain model

includes the information that oil is a type of fluid. If the user accepts the noun/noun pattern *oil sample*, he/she would then be asked if one can speak of fluid samples for all fluids known in the domain. If there is some fluid *f* such that one cannot speak of an *f sample*, the system would then ask the user to identify for which of those *f* such that $\text{isa}(f, \text{fluid})$ is true one can speak of an *f sample*. (E.g., can one speak of an *air sample*? What about a *water sample*?) As before, the user's responses will be stored for reference in evaluating patterns generated by other sentences. The obvious advantage of storing semantic patterns instead of just lexical ones is the broader coverage of the former: The lexical pattern *oil sample* will be of use in guiding the parser to correct parses only of sentences containing those very words. However, if that pattern can be generalised to *fluid sample*, we then have information which can be applied to sentences containing *water sample*, *air sample*, and so on.

7.2. The User Interface

In the current implementation, the questions which the program asks the user are phrased in terms of grammatical categories, and are thus tailored to users who know what is meant by such terms as SVO and noun-noun compounds. As a result, only linguists can be reasonably expected to make sense of the questions and provide meaningful answers. Our intended users, however, are not linguists, but rather domain experts who will know what can and cannot be said in the sublanguage, but who cannot be expected to reason in terms of grammatical categories. It is a difficult problem to know just how to phrase questions designed to elicit the desired information, especially in the case of the troublesome "phrasal attribute" patterns.

The knowledge that the user must draw upon in order to answer the system's questions is usually a combination of basic commonsense knowledge (e.g., temperatures cannot hold beliefs) and domain-specific information. In certain cases, however, the user can be called upon to make fine linguistic distinctions. For example, in the sentence *Sac disengaged immediately after alarm*, does the adverb *immediately* modify the verb *disengaged*, or the prepositional phrase *after alarm*? Most users, and even trained linguists familiar with the domain, find it difficult to provide definitive answers to such questions, because there is often no definitively correct answer. In this case, the adverbial attachment would seem to be genuinely ambiguous. It would be helpful to recognise patterns which a user cannot be reasonably expected to pass judgement on, and not present such patterns to the user, perhaps allowing them to succeed by default.

7.3. Measuring the System's Learning

As more sentences are parsed and more patterns are classified, we can expect the system to grow "smarter" in the sense that it will ask the user increasingly fewer questions. Eventually, the system should reach a state of reasonably complete domain knowledge, at which time very few unknown patterns would be encountered, and the user would rarely be queried. We do not know how many sentences the pattern checker would have to parse before attaining this plateau, but an estimate would be 500 to 1000 to obtain classes of semantic patterns [Grishman1984]. We plan to measure the decrease in the frequency of queries to the user as a function of the number of sentences parsed in order to evaluate the system's learning.

7.4. Dealing with Phrasal Attributes

We do not at present have a satisfactory mechanism for detecting or correctly handling the phrasal-attribute phenomenon discussed earlier. There does not appear to be any straightforward way to discover such phrasal-attribute constructions by means of distributional analysis,

since one of the distinguishing features of these constructions is precisely that they do not occur in the expected distributional patterns. A possible course to follow would be to compare the distribution of certain nouns (e.g., *neck* or *psig*) when they appear without modifiers with their distribution when they appear in head-modifier structures. Once such constructions are detected, and the system knows, for example, that a stiff neck is not a BODY-PART but a SYMPTOM, a sophisticated semantic component could compute the semantic class of the entire phrase and attach that information to the head noun. In the case of *Start air pressure dropped below 30 psig*, the resulting semantic class pattern (disregarding the semantic class of *pressure*) would then be *pressure drop below level*, as desired, and not *pressure drop below unit-of-measurement*.

APPENDIX

Results of Parsing TESTA Sentences with and without Selection (no backtracking into lexical lookup)

CAS- REP #	PARSES		NODES			FOCUS		TIME	
	OK	TOTAL	TREE	OK	TOTAL	OK	TOTAL	OK	TOTAL
1.1.1	1	2	43	172	672	4.0	15.6	2.7	12.8
	1	1	43	172	654	4.0	15.2	3.5	12.5
1.1.3	0	1	—	—	229	—	—	—	4.7
	1	1	57	502	1255	8.8	22.0	8.0	22.8
4.1.1	1	1	129	3669	4863	28.4	37.7	37.9	74.1
	1	1	129	3669	4863	28.4	37.7	43.2	84.7
4.1.3	2	2	93	438	629	4.7	6.8	12.8	18.2
	1	1	93	438	629	4.7	6.8	10.8	17.5
5.1.2	2	4	43	305	854	7.1	19.9	7.7	20.8
	2	2	43	305	854	7.1	19.9	8.3	16.5
5.1.3	4	10	63	3709	5001	58.9	79.4	113.7	150.9
	1	4	63	3620	4917	57.5	78.0	80.1	145.3
6.1.2	1	1	73	304	798	4.2	10.9	5.2	14.6
	1	1	73	304	780	4.2	10.7	5.9	16.6
6.1.3	5	26	84	1756	13264	20.9	157.9	61.8	386.1
	1	1	84	1175	4394	14.0	52.3	32.8	122.0
9.1.1	3	4	87	1119	1300	12.9	14.9	26.4	32.8
	1	1	87	938	1115	10.8	12.8	18.2	28.9
9.1.2	1	1	99	633	1222	6.4	12.3	7.8	18.5
	1	1	99	633	1222	6.4	12.3	8.9	21.3
9.1.3	1	1	36	144	175	4.0	4.9	2.1	3.7
	1	1	36	144	175	4.0	4.9	2.6	4.5
9.1.4	4	4	99	3372	3532	34.1	35.7	79.2	84.5
	1	1	99	3253	3413	32.9	34.5	63.9	94.6
9.1.6	1	1	54	340	377	6.3	7.0	3.9	6.6
	1	1	54	340	377	6.3	7.0	5.6	8.9
21.1.1A	11	12	133	25974	35485	195.3	266.8	564.7	738.9
	1	1	133	14323	22847	107.7	171.8	304.3	495.2
21.1.1B	4	4	105	3070	6930	29.2	66.0	60.3	128.5
	1	1	105	2687	6093	25.6	58.0	51.0	141.5
22.1.1	3	4	67	1774	2639	26.5	39.4	36.4	52.4
	1	2	67	1499	2346	22.4	35.0	31.3	62.7
22.1.2	1	4	111	5244	8748	47.2	78.8	46.5	119.3
	1	2	111	4195	7699	37.8	69.4	53.1	132.0

CAS- REP #	PARSES		NODES			FOCUS		TIME	
	OK	TOTAL	TREE	OK	TOTAL	OK	TOTAL	OK	TOTAL
22.1.3	1	2	54	319	1462	5.9	27.1	5.0	31.5
	1	1	54	303	1342	5.6	24.9	5.4	25.6
23.1.2	1	1	99	388	1743	3.9	17.6	5.5	33.8
	1	1	99	388	1461	3.9	14.8	6.6	30.7
24.1.1	1	2	141	857	5599	6.1	39.7	10.9	65.6
	1	1	141	857	3915	6.1	27.8	13.0	59.0
24.1.2	2	5	66	1018	1635	15.4	24.8	21.2	34.9
	1	2	66	871	1488	13.2	22.5	13.9	32.0
25.1.1	12	21	169	11041	39255	65.3	232.3	345.1	1160.6
	1	1	169	8082	22912	47.8	135.6	134.0	419.2
25.1.2	1	1	72	700	1982	9.7	27.3	7.6	25.8
	1	1	72	700	1982	9.7	27.3	9.0	27.3
25.1.3	1	4	57	1214	3356	21.3	58.9	18.0	59.2
	1	2	57	1154	2986	20.2	52.4	19.2	54.5
28.1.1	1	2	144	1161	5093	8.1	35.4	16.2	86.9
	1	1	144	1161	5042	8.1	35.0	17.0	76.5
28.1.2	1	2	138	1096	1427	7.9	10.3	13.6	29.3
	1	1	138	1080	1411	7.8	10.2	13.5	28.2
31.1.1A	0	3	—	—	3796	—	—	—	72.7
	1	2	82	7282	13626	88.8	166.2	98.0	235.0
31.1.1B	2	4	104	4255	10583	40.9	101.8	82.9	177.0
	1	1	104	4018	7917	38.6	76.1	51.9	128.5
31.1.2	1	3	79	831	1371	10.5	17.4	17.0	36.6
	1	3	79	725	1217	9.2	15.4	16.6	35.3
31.1.3	1	1	66	274	711	4.2	10.8	4.3	12.3
	1	1	66	256	675	3.9	10.2	4.7	12.0
31.1.4	1	2	50	1031	2149	20.6	43.0	12.9	36.2
	1	1	50	1031	2149	20.6	43.0	13.3	33.9

NOTES:

- The the above chart presents the results of parsing the 31 CASREP sentences in our TESTA corpus which have well-formed ISRs containing no unresolved lambda terms. Each sentence was parsed with and without the selectional pattern checker. The statistics in the first line were obtained while parsing without the selectional mechanism, and those in the second, while parsing with the selectional mechanism.
- Columns headed by "OK" relate to the correct parse (or the search focus in reaching the correct parse); columns headed by "TOTAL" relate to the total number of parses (or the search focus in parsing to completion). The meanings of the column headings are as follows:
- PARSES OK = the number of the correct parse (if a correct parse was found)

- **PARSES TOTAL** = the total number of parses found
- **NODES TREE** = the number of nodes in the parse tree of the correct parse
- **NODES OK** = the number of nodes attached to the parse tree (and possibly detached on backtracking) in reaching the correct parse⁶
- **NODES TOTAL** = the number of nodes attached (or grammar rules tried) in parsing to completion
- **FOCUS OK** = $\text{NODES OK} / \text{NODES TREE}$, the search focus in reaching the correct parse.
- **FOCUS TOTAL** = $\text{NODES TOTAL} / \text{NODES TREE}$ the search focus in parsing to completion
- **TIME OK** = time taken (in seconds) in reaching the correct parse
- **TIME TOTAL** = time taken (in seconds) in parsing to completion

PROBLEMS and POSSIBLE IMPROVEMENTS:

In sentences testa 1.1.3 and testa 31.1.1A, substantially more nodes are attached in parsing to completion with selection than without selection, contrary to our expectations that the selection mechanism will decrease the amount of search. The reason for this increase is that the correct parse for these two sentences is a fragment, but an assertion parse is found in parsing without selection. Consequently, in parsing without selection, fragment parses are never tried (because of the XOR mechanism). However, in parsing with selection, since all assertion parses are failed, the parser tries both assertion and fragment parses, and thus traverses a far greater search space in parsing the sentences with selection than without selection.

In parsing many of the sentences which receive exactly one parse without selection (e.g., testa 4.1.1, testa 9.1.2, testa 9.1.3, testa 9.1.6, testa 25.1.2) the same number of nodes are attached (the same number of rules were tried) both with and without selection because of the efficient design of the BNF grammar. Consequently, in these sentences, the overhead of the selectional component increases the time spent parsing.

Four sentences (testa 6.1.2, testa 23.1.2, testa 31.1.2, and testa 31.1.3) received one parse both with and without selection, but traversed a smaller space in either reaching that parse or parsing to completion.

In certain sentences, even though selection rules out some parses, the same number of nodes are attached both with and without selection (e.g., testa 4.1.3, testa 5.1.2, testa 31.1.4). This unexpected behavior (it is unexpected because if some parses are ruled out, surely then fewer nodes should be attached) is caused by the very late ruling out of certain bad parses. Specifically, if a parse is not ruled out until a bad SVO pattern is encountered, the entire parse tree will have been built by then, and there will have been no reduction in the search space. It would be helpful to find a way to call selection at other times than after building an entire LNR and an

⁶As we explained above, this figure can also be interpreted as the number of grammar rules tried in reaching the correct parse.

entire center.

In the remaining 17 sentences, the selection mechanism had the expected and desired effect of reducing both the number of parses found and the search space.

Summary of search focus for these 31 sentences:

- (1) Sentences for which selection increases search in finding the correct parse (because selection fails all assertion parses and then tries fragment parses): testa 1.1.3, testa 31.1.1A (total = 2).
- (2) Sentences for which selection has no effect on the number of parses or on the search focus (because of efficiency of the grammar and late generation of SVO patterns): testa 4.1.1, testa 9.1.2, testa 9.1.3, testa 9.1.6, testa 25.1.2 (total = 5).
- (3) Sentences for which selection has no effect on the number of parses, but does reduce the search focus: testa 6.1.2, testa 23.1.2, testa 31.1.2, testa 31.1.3 (total = 4).
- (4) Sentences for which selection decreases the number of parses, but has no effect on the search focus (because of the late generation of SVO patterns): testa 4.1.3, testa 5.1.2, testa 31.1.4 (total = 3)
- (5) Sentences for which selection has the desired effect of reducing the number of parses AND the search focus: testa 1.1.1, testa 5.1.3, testa 6.1.3, testa 9.1.1, testa 9.1.4, testa 21.1.1A, testa 21.1.1B, testa 22.1.1, testa 22.1.2, testa 22.1.3, testa 24.1.1, testa 24.1.2, testa 25.1.1, testa 25.1.3, testa 28.1.1, testa 28.1.2, testa 31.1.1B (total = 17).

REFERENCES

[Chevalier1978]

M. Chevalier, *TAUM-METEO: description du systeme*. Groupe de recherche en traduction automatique, Hillsdale, NJ, 1978.

[Grishman1984]

R. Grishman, L. Hirschman, and N.T. Nhan, Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments, Paper presented at the Workshop on Transportable Natural Language Interfaces, Durham, NC, October 1984.

[Harris1968]

Zellig Harris, *Mathematical Structures of Language*. Interscience Publishers, John Wiley and Sons, New York, 1968.

[Hirschman1982]

L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.

[Hirschman1983]

L. Hirschman and Naomi Sager, Automatic Information Formatting of a Medical Sublanguage. In *Sublanguage: Studies of Language in Restricted Semantic Domains*, R. Kittredge and J. Lehrberger (ed.), Series of Foundations of Communications, Walter de Gruyter, Berlin, 1983.

[Hirschman1985]

L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Hirschman1986]

L. Hirschman, Discovering Sublanguage Structures. In *Sublanguage: Description and Processing*, R. Kittredge and R. Grishman (ed.), Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.

[Lehrberger1983]

J. Lehrberger, Automatic Translation and the Concept of Sublanguage. In *Sublanguage: Studies of Language in Restricted Semantic Domains*, R. Kittredge and J. Lehrberger (ed.), Series of Foundations of Communications, Walter de Gruyter, Berlin, 1983.

[Marsh1984]

E. Marsh, H. Hamburger, and R. Grishman, A Production Rule System for Message Summarisation. In *Proc. 1984 National Conf. on Artificial Intelligence*, Oakland University, Rochester, Michigan, 1984.

APPENDIX P

Grammatical Coverage of the CASREPs

This report by Marcia Linebarger describes the grammar, with particular emphasis on extensions for the CASREPs. It includes detailed information about parsing problems observed in the CASREP corpus.

Grammatical coverage of the CASREPS:
Summary of current status
April, 1986

Marcia Linebarger

1. Coverage of CASREPs

TOTAL OF SENTENCES: 154

Total parsed correctly: 131 (85%)

On 1st, 2nd, or 3rd parse: 109

On 1st parse: 92

On 2nd or 3rd parse: 17

On 4th or subsequent parse: 22

Total not parsed at all, or parsed incorrectly: 23

Due to ill-formed input: 9

Due to lexical scanner problems: 7

Due to inadequacies of grammar coverage: 4

Due to xor (correct reading available but not generated): 3

The figures below represent coverage of the same corpus with the lexical scanner difficulties resolved and the ill-formed input (misspellings, misspunctuations, run-on sentences) corrected. Since two of these sentences would need to be re-phrased in order to be corrected, they are simply omitted from the sentence total in the following breakdown:

TOTAL OF SENTENCES (less two): 152

Total parsed correctly: 145 (95%)

On 1st, 2nd, or 3rd parse: 120

On 1st parse: 101

On 2nd or 3rd parse: 19

On 4th or subsequent parse: 25

Total not parsed at all, or parsed incorrectly: 7

Due to inadequacies of grammar coverage: 4

Due to xor (correct reading available but not generated): 3

2. Extensions to Grammar

The extensions to the grammar required to parse this corpus include the addition of rules for fragments, objects, sentence adjuncts, and *wh*-constructions such as relative clauses.

2.1. Fragments

Approximately half of the sentences in the CASREPs are not full sentences. Nevertheless, these fragments follow quite regular patterns, and fall into one or another of four basic types: *two* (tensed sentence missing subject, as in A4.1.2, *Believe the coupling from diesel to sac lube oil pump to be sheared*); *zerocopula* (missing verb *be*, as in A6.0.0, *Part ordered*); *ntsg_fragment* (isolated noun phrase, as in B34.1.1, *Loss of oil pump pressure*); or *predicate* (isolated complement of verb *be*, as in B12.1.2, *Believed due to worn bushings*, or A.1.1.2, *Unable to consistently start nr 1b gas turbine*).

The syntax and the semantics of these elements are quite regular, and thus fragment coverage does not add significantly to the complexity of the grammar. A total of six BNF rules (out of 106 total) and 3 restrictions (out of 55 total) were added to the grammar to cover fragments; in addition, 2 BNF rules and 1 restriction were altered to accomodate fragments.

2.2. Object options

The grammar has also been extended to cover a wider range of object types, including a variety of embedded infinitivals, embedded clauses, and non-clausal predications such as *subject+object of be* (as in B26.1.5, *High lo temp due to design of first flight oil cooler believed contributor to unit failure*).

2.3. Sentence adjuncts

A rich variety of sentence adjuncts occur in the CASREPS, including a range of clausal and sub-clausal strings introduced by subordinating conjunctions (as in B20.1.1, *while engaged*) and present participles (as in B11.1.1, *causing erratic operation*). In addition, the restriction component was developed to prevent spurious ambiguities arising out of the enrichment of sentence adjunct possibilities.

2.4. Wh-expressions

Although relative clauses and other wh-expressions are rare in the CASREPs (cf. B36.1.3, *65 psi which is low lube oil alarm set point*), the grammar has also been expanded to cover these constructions and to enforce the complex restrictions on their occurrence.

3. Problems

The major remaining difficulties include the following:

3.1.1. Lexical scanner problems

Word-internal occurrences of periods, slashes, etc. are currently rejected by the lexical scanner.

3.1.2. Xor problems

The 'committed or' which controls disjunctive application of the assertion, question, fragment, and compound options is generally successful in capturing the intended parse. However, there are several sentences in the CASREP corpus in which a spurious assertion parse preempts a correct fragment parse, e.g., B26.1.5, *High lo temp believed contributor to unit failure*, where *believe* is taken as the main verb with subject *temp* and *contributor* as the object (*they believed it*), rather than as a fragment of the type *zerocopula*, where *believed* is taken as a past participle (*temp [was] believed [to be] a contributor...*).

3.1.3. Remaining grammar problems

Full and accurate coverage of the CASREPs requires further work on the grammar, including the following: finer-grained treatment of the noun phrase; restrictions on adverbs to prevent, e.g., the analysis of *very* as a sentence adverb; modification of the BNF rules to accommodate multiple sentence adjuncts; modification of conjunction rules.

CASREPS.TESTA
Summary of Parses
April, 1986

Sentences not preceded by casreps number are modifications of the original text. The rank of the correct parse is given in "Correct parse #" column. Note that these data reflect the grammar prior to the removal of sor from the fragment rule; therefore the figures for fragments do not include fragment parses subsequent to the correct one.

No.	Text	No.Parses	Times	Correct parse #
1.1.1	Starting air regulating valve failed.	5	1,3,6,8,10 (13)	4
1.1.2	Unable to consistently start nr 1b gas turbine.	1	2 (9)	1
1.1.3	Valve parts excessively corroded.	1	1 (2)	(N/G xor) 1
4.0.0	Tech assist requested.	1	2 (3)	1
4.1.1	While diesel was operating with sac disengaged, the sac lo alarm sounded.	1	10 (16)	1
4.1.2	Believe the coupling from diesel to sac lube oil pump to be sheared.	12	4,13,20,27,30,33, 37,43,49,52,56, 63,67 (69)	4
4.1.3	Pump will not turn when engine jacks over.	2	2, 4 (6)	1
5.0.0	Tech assist requested.	1	2(3)	1
5.1.1	Unable to maintain l.o. pressure to sac.	0		N/G scan
	Unable to maintain lo pressure to sac.	2	2,3 (6)	1
5.1.2	Disengaged immediately after alarm.	2	1,2 (2)	1
5.1.3	Metal particles in oil sample and strainer.	4	8,9,11,11 (15)	4
6.0.0	Part ordered.	1	2 (2)	1
6.1.1	Unable to maintain lube oil pressure to starting air compressor.	4	2,5,9,11 (36)	3
6.1.2	Inspection of lo filter revealed metal particles.	1	1 (4)	1
6.1.3	Retained oil sample and filter element for future analysis.	6	6,7,8,8,10,11 (13)	5
9.0.0a	Part fail.	1	2 (2)	1
9.0.0b	Part ordered.	1	1 (2)	1
9.1.1	Sac received high usage during two becce periods.	4	3,4,6,6 (7)	4
9.1.2	Ccs received a report that lo pressure was dropping.	2	3,5 (7)	1
9.1.3	Alarm sounded.	1	1 (1)	1

No.	Text	No.Parses	Times	Correct parse #
9.1.4	Loud noises were coming from the drive shaft during coast down.	4	9,10,12,13 (17)	4
9.1.5	Drive shaft was found to rotate freely at the ssdg end.	2	2,5 (7)	2
9.1.6	Splines were extensively worn.	1	1 (1)	1
21.0.0	Assist required.	1	1 (2)	1
21.1.1A	Nr 4 sac oil pressure dropped below alarm point of 65 psig during monitoring of 1A gth.	6	11,18,23,39,43,48 (85)	2
21.1.1B	Start air pressure dropped below 30 psig during monitoring of 1A gth.	5	7,9,15,18,21 (42)	2
21.1.2	Oil is discolored and contaminated with metal.	3	1,3,3 (5)	3
22.0.0	Tech assist requested.	1	1 (2)	1
22.1.1	Loss of lube oil pressure during operation.	3	7,8,9 (12)	1
22.1.2	Investigation revealed adequate lube oil saturated with both metallic and non-metallic particles.	0		N/G scan
	Investigation revealed adequate lube oil saturated with both metallic and non-metallic particles.	10	23,24,*25,*25,*27, 29,30,*31,*32,*33, 38,39,40,42,43,44, 45 (54)	1
22.1.3	Request replacement of sac.	1	2 (4)	1
23.0.0	Assistance required.	1	2 (2)	1
23.1.1	The low lube oil pressure alarm and compressor fail to engage the alarm activated during routine start of start air compressor.	0		N/G input
	The low lube oil pressure alarm and compressor fail to engage alarm activated during routine start of start air compressor.	27	4,8,12,25,29,.... (215)	1
23.1.2	Metallic material was discovered in lo sump and filter assembly.	1	3 (11)	1
24.0.0	Require replacement.	1	1 (1)	1
24.1.1	Loss of lube oil pressure when start air compressor engaged for operation is due to wiped bearing.	4	4,5,23,26 (29)	N/G gram

No.	Text	No.Parses	Times	Correct parse #
24.1.2	Material clogging strainers.	2	3,4 (4)	1
25.0.0	Tech assist required.	1	1 (2)	1
25.1.1	During routine start of main gas propulsion turbine, sac air pressure decreased rapidly to 5.74 psi resulting in an aborted engine start.	0		N/G scan
	During routine start of main gas propulsion turbine, sac air pressure decreased rapidly to 5.74 psi resulting in an aborted engine start.	21	67,69,71...109 (227)	14
25.1.2	Exact cause of failure unknown.	1	2 (4)	1
25.1.3	Suspect faulty high speed rotating assembly.	1	2	1
28.0.0	Return to company.	3	2,2,3 (3)	1
28.1.1	Unit has excessive wear on inlet impellor assembly and shows high usage of oil.	2	5,12 (24)	1
28.1.2	Blades are bent and 1/4 inch deep chips are visible on leading edge.	2	2,3 (5)	1
30.0.0	Tech assist requested.	1	1 (2)	1
31.1.1A	Loss of second sac of two installed sac's.	2	2,4 (10)	N/G xor
31.1.1B	Unit has low output air pressure, resulting in slow gas turbine starts.	4	16,17,22,24 (46)	2
31.1.2	Troubleshooting revealed normal sac lube oil pressure and temperature.	3	4,4,5 (6)	1
31.1.3	Erosion of impellor blade tip is evident.	1	1 (3)	1
31.1.4	Compressor wheel inducer leading edge broken.	2	4,4 (6)	1

CASREPS.TESTA
Annotations to parse summary

[1.1.1]

Note that only an adjectival reading is available for the prenominal analysis of "regulating".

[1.1.3]

Xor problem . Due to the optional intransitivity of "corrode", xor eliminates the correct zerocopula reading. However, this reading is close enough to qualify as correct.

[4.1.1]

Note that restriction {d_nullLnsr} removes rare gerund reading; {w_ving_lnr} thwarts an obscure analysis of ving as nvar.

[4.1.2]

The object is analyzable as nstgo, ntovo, or sobjbe. The latter possibility adds eight parses, but the object option sobjbe cannot be eliminated given, e.g., Testb 26.1.5 ("High LO temp .. believed contributor to unit failure").

[4.1.3]

"Over" is parsed first as an adverb preceding null object of "jacks". The most correct reading seems to be the second one, in which it is parsed as a particle; however, the sa reading is close enough to be counted as correct. If expressions such as "over" are reclassified as particles but not adverbs, in order to circumvent this, then they will have to be subcategorized for individually in the lexicon, which would lead to many false rejections of acceptable sentences.

[5.1.1]

Pn "to sac" is attached to rn in first parse (marked as correct here); but the second parse (with sa attachment of pn) seems more accurate.

[5.1.2]

In first parse, counted as correct, "immediately" is sa; perhaps the second, in which it is a left modifier of "after", is still more accurate.

[5.1.3]

I assume (without conviction) that npos "oil" should not be distributed over "strainer".

[6.1.1]

Although the third parse is listed as the correct one, the first parse is perhaps adequate: "to SAC" is attached to rn rather than sa.

[6.1.3]

The first parse differs from the correct one only in that it attaches "for future analysis" to rn rather than sa.

[9.0.0]

"Fail" is treated here as abbreviation for "failure". Or should these headers be treated as frozen expressions?

[9.1.1]

It is assumed here that the correct parse attaches "during NP" to sa and analyzes "two becce periods" as qpos + npos + nvar.

[9.1.2]

"That" is analyzable as determiner or complementizer.

[9.1.4]

"Coast down" is treated as idiom.

I assume that the most accurate parse (the fourth, counted as the correct one), attaches "from the drive shaft" to object, and "during coast down" to sa. However, the first parse might be sufficiently close, given the state of the system; it attaches the two pns to sa and rn, respectively.

[9.1.5]

Ambiguity: analysis of infinitive as sa (tovo) or passobj (correct).

[21.1.1A,B]

In the second parse, counted as correct, "below"-phrase is sa rather than object (fifth parse).

[21.1.2]

The third parse is counted as correct, but the second parse, in which "with metal" is in sa, seems adequate.

[22.1.1]

The contextually correct nstg_frag parse is generated last; However, the zerocopula parse seems adequate, and is counted correct.

[22.1.2]

Conjunction . There are some analyses of "metallic" as avar preceding nulln that seem incorrect. This should be explored.

Object type . The nstgo object analysis seems somewhat more accurate than sven analysis here; within the venpass, the most accurate parse is perhaps the one in which "with ... particles" is attached as passobj rather than as sa. But the first parse, with sa attachment of this phrase, seems adequate.

Scanner problem . The problem remains that words containing "-" and such characters fail lookup because they are not atoms.

Conjunction . In order to parse the conjoined apos, lar1 has been defined as an lxr node. This may present a problem, since lar1 lacks a right adjunct.

Six other readings generated for this sentence contain conjoined lnr with nulln head of first lnr. Perhaps nulln should be disallowed in conjuncts unless it occurs in both: "There were five *(cats) and two dogs in the park"; "old and young were present", but *"old men and young were present" is quaint at best.

[23.1.1]

Input error . It is assumed that "the" preceding "alarm" is an error.

Re corrected version: The first six parses analyze "fail to engage" as an idiom (noun). In the remaining parses, "fail" is analyzed, legitimately, as the main verb (seven parses of conjoined subject x three parses of post-verb material).

[23.1.2]

Conjunction problem . Although the correct parse is generated, there is a missing parse, with conjoined npos "sump and filter". But since it seems unadvisable to allow full lnr in nnn, it's not clear how to modify the conjunction rules to allow for this reading.

[24.1.1]

Multiple rn : In the contextually correct reading, "loss" is modified by "of lube oil pressure" and the "when"-clause. However, multiple rn's are not permitted, except in the case of pn's. A semantically close reading in which the "when"-clause is an sa is also prevented, by {wmed_sa}, which rules out such sa's between subject and verb unless set off by commas (accounting for the ill-formedness of *"Louise when I called was tired"). The closest available reading actually generated is the second one, in which the when-clause is in the rn of "pressure".

Embedded fragment: "When sac engaged" seems most accurately parsed as an sven following "when". But in standard English, "when" cannot introduce an sven ("*I left when the car repaired"). Thus it may be that this corpus requires further modifications of the bnf rules beyond simply allowing matrix

fragments. However, the optional intransitivity of "engaged" allows the material following "when" to be parsed as an assertion rather than an sven.

[24.1.2]

Perhaps an *nstg_frag* reading would be more accurate, but the first parse (zerocopula with *objectbe*—>*vingo*) seems close enough to be counted as correct. The second parse (zerocopula with *objectbe*—>*nstg*) seems more questionable; perhaps {*w_nonnull_ln*} should be strengthened to require material in *qpos* or *tpos* rather than simply *ln*. (This decision depends on judgments about acceptability of, e.g., "Sen.Jones complete idiot").

[25.1.1]

Scanner problem . The decimal point cannot currently be entered.

The long time to first parse may reflect the fact that the sentence is an extensive garden path, since the main verb "decreased" may initially be mis-analyzed as a participle in *rn*.

The parses generated prior to the correct fourteenth parse analyze the *nvar* of the subject as either "resulting" or *nulln* rather than "psi".

[28.0.0]

Third parse is questionable: *objbe* in zerocopula (analogous to "house in an uproar", or "trip to Texas, not Arizona").

[28.1.1]

First parse (counted as correct) attaches "on...assembly" to *rn*; *sa* attachment, as in second parse, might be considered the more accurate parse.

[31.1.1A]

Lexical entry procedure should be modified to generate "'s" plurals routinely for abbreviations.

Xor problem . The contextually incorrect assertion parse preempts the *nstg_frag* parse that is intended here.

[31.1.1B]

Here the attachment of the *pn* in *object* or *sa* seems important, as "result in" has an idiomatic meaning. Thus the first parse, with *sa* attachment, is not counted as correct.

[31.1.4]

"Leading edge" is entered in the lexicon as an idiom, as a result of its occurrence here in *nvar* position. ("Leading" could only be parsed as *avar*, an

impossibility here given that it follows a series of npos elements.) Occurrence in compounds seems a potential test for fixed phrases; compare this sentence with the less acceptable "*peach poisonous pits are dangerous" (vs. "peach pits are dangerous").

CASREPS.TESTB
Summary of Parses
April, 1986

Sentences not preceded by a casreps number are modifications of the original text. The rank of the correct parse is given in "Correct parse #" column. Note that these data reflect the grammar prior to the removal of xor from the fragment rule; therefore the figures for fragments do not include fragment parses subsequent to the correct one.

No.	Text	No.Parses	Times	Correct parse #
2.0.0	Replacement requested.	1	1 (2)	1
2.1.1	Loss of lube oil pressure during operation nr. 2 ssdg.	8	13,19,19,22,25, 26,30 (34)	N/G input(+scan)
2.1.2	Metal particles found in lube oil filter.	1	12	1
3.1.1	Gas turbine starting air compressor inoperative.	2		1
3.1.2	Power pack failed.	1	1 (1)	1
7.0.0	Assistance requested.	1	1 (2)	1
7.1.1	Sac had local monitoring capacity for lube oil pressure only, due to the recent failure of the sac lube oil pressure transducer.	1	10 (50)	1
7.1.2	Prior to engagement it was reported that sac lo pressure dropped to zero.	2	2,3 (8)	1
7.1.3	No metallic particles in oil filters.	2	*4,5 (6)	2
7.1.4	Borescope investigation revealed a broken tooth on the hub ring gear.	4	5,7,8,10 (11)	1
7.1.5	It is likely the lo pump has sheared.	1	2 (4)	1
7.1.6	The lo pressure and alarm capability is a necessity for operation.	4	1,2,3,4 (6)	N/G input
7.1.7	Drive shaft for sac was manufactured locally.	1	1 (3)	1
7.1.8	S/F reinstalled old sac utilising new drive shaft.	0		N/G scan
	Fe reinstalled old sac utilising new drive shaft.	3	3,5,6 (7)	3
7.1.9	On testing of sac lube oil pressure could not be adjusted above 35 psig.	3	2,6,9 (18)	3
7.1.10	Replacement sac will be required.	1	1 (2)	1

No.	Text	No.Parses	Times	Correct parse #
7.1.11	The original drive shaft, when installed, was packed utilising 60 grams of grease, when removed, on failure of sac, the drive shaft was dry and showed signs of extensive heat stress.	?	1460 ...	3
	The original drive shaft, when installed, was packed utilising 60 grams of grease.	3	3,*4,*5 (7)	1
	When removed, on failure of sac, the drive shaft was dry and showed signs of extensive heat stress.	2	11,15 (25)	1
8.0.0	Tech assist requested.	1	2 (2)	1
8.1.1	Loss of one of two starting air compressors.	?	12, ...	1
8.1.2	Low speed coupling from diesel to sac lube oil pump failed.	6	*2,*9,14,21,26, 33 (36)	4
10.0.0	Tech assist requested.	1	2 (2)	1
10.1.1	HBV failed, causing spline assy to fail causing damage to the sac.	4	3,4,7,8	1
11.0.0	Tech assist required.	1	2(2)	1
11.1.1	Compressor will not remain fully engaged causing erratic operation, surging, and a hazard to personnel and equipment.	55	8,9,10,11....115 (126)	4
12.0.0	Tech review required.	1	2 (2)	1
12.1.1	Sac lo pressure decreases below alarm point approx. seven minutes after engagement.	0		N/G scan
	Sac lo-pressure decreases below alarm point approx seven minutes after engagement.	4	3,7,13,13 (17)	2
12.1.2	Believed due to worn bushings.	4	2,2,4,5 (6)	2
13.0.0	Must be removed.	1	1 (1)	1
13.1.1	Loss of sac oil pressure dropped to 72 psi then increased to 90 psi and then failed while starting gas turbine.	0		N/G gram(+input)
	Loss of sac.	1	2 (3)	1
	Oil pressure dropped to 72 psi then increased to 90 psi and then failed while starting gas turbine.	21	38 ... 171 (187)	N/G
14.0.0	Req tech assist.	1	1 (1)	1
14.1.1	Loss of one of three start air compressors.	2	12,13	1

No.	Text	No.Parses	Times	Correct parse #
	Oil pressure has dropped to 72 psi then increased to 90 psi and then failed while starting gas turbine.	27	8 ... 175 (179)	1
14.1.2	Starting air compressor engaged for approx two minutes when lube oil pressure dropped below 65 psi alarm setting.	?	21,25,28,32,44,...(434)	5
14.1.3	Compressor could not be disengaged from either remote or local control location, for approx three minutes following low lube oil pressure alarm.	12	3,6,13,15,38,40,47, 49,71,73,80,82 (132)	1
14.1.4	Lube oil is very dark in appearance and has burnt odor.	4	*2,*4,7,9 (11)	3
15.0.0	Tech assist requested.	1	1 (2)	1
15.1.1	Reliability of third of three sac's suspect - if unit fails unable to start main propulsion gas turbines.	?	36 (58)	1
15.1.2	Color of 23699 oil indicates overheating of sac, oil pressure normal.	1	17 (20)	1
16.1.1	During normal start cycle of 1A gas turbine, approx 90 sec after clutch engagement, low lube oil and fail to engage alarm were received on the acc.	over 30	162 to 1st parse	N/G gram
16.1.2	All conditions were normal initially.	1	2 (2)	1
16.1.3	Sac was removed and metal chunks found in oil pan.	0		N/G gram
	Sac was removed and metal chunks were found in oil pan.	1	2 (5)	1
16.1.4	Lube oil pump was removed and was found to be seized.	2	3,4 (6)	1
16.1.5	Driven gear was sheared on pump shaft.	1	1 (3)	1
17.0.0	Tech evaluation req.	1	2 (2)	1
17.1.1	Loss of one of three sac's - routine visual inspection during normal engine operation revealed gear housing cracked.	2	47,56...	1

No.	Text	No.Parses	Times	Correct parse #
17.1.2	Engine secured, detailed inspection revealed large crack in gear housing on aft end and broken marmon clamp flange on surge valve outlet.	22?	2072, ...	11?
	Engine secured.	1	1 (1)	1
	Detailed inspection revealed large crack in gear housing on aft end and broken marmon clamp flange on surge valve outlet.	Over 22	215,216,...	11
18.0.0	Item cannibalized.	0		N/G input
	Item cannibalized.	1	4 (5)	1
18.1.1	Cannibalized sac for use on USS Duncan.	4	14,17,22,24	4
19.0.0	Part ordered.	1	1 (2)	1
19.1.1	Reduced capability of nr 4 sac restricts ships operation.	0		N/G input
	Reduced capability of nr 4 sac restricts ship's operation.	1	4 (9)	1
19.1.2	Extended use of nr 4 sac has resulted in periodic low lube oil pressure alarm.	3	7,16,21 (26)	2
19.1.3	Lube oil change, filter change, and adjustment of pressure regulator have had no impact on lube oil pressure.	?		4?
19.1.4	Three minutes is the maximum time nr 4 sac can be operated in a non-alarm condition.	0		N/G scan
	Three minutes is the maximum time nr 4 sac can be operated in an alarm condition.	2	4,8 (14)	1
20.0.0	Tech assist req.	1	2 (2)	1
20.1.1	During gth motor start, air pressure dropped below 30 psi and oil pressure decreased slowly to 70 psi, while engaged.	Many	102 to 1st parse	4th+
	During gth motor start, oil pressure decreased slowly to 70 psi, while engaged.	1	11 (25)	1

No.	Text	No.Parses	Times	Correct parse #
20.1.2	Metal particles found in oil sample.	2	10,12 (17)	1
26.0.0	Technical assistance requested.	1	2(2)	1
26.1.1	Reduced capacity of one of three sac's.	4+	13,14,24 ...	3
26.1.2	Cannot engage sac for extended period of time due to increased lo temp and sharp decrease in lo pressure.	30+	4 ...	4th+
26.1.3	Metal contamination in lo filter.	2	*4,6 (8)	2
26.1.4	Internal part failure.	1	2 (2)	1
26.1.5	High lo temp due to design of first flight oil cooler believed contributor to unit failure.	4	4,6,44,46 (80)	N/G xor
27.0.0	Part ordered.	1	2 (2)	1
27.1.1	Experienced loss of sac lube oil pressure and self-disengagement immediately following clutch engage command.	0		N/G xor(+scan)
	Experienced loss of sac lube oil pressure and self disengagement immediately following clutch engage command.	4	26,28,32,33 (48)	N/G (xor)
27.1.2	Sac apparently seized during clutch engagement causing input drive shaft to remain stationary while drive adapted hub on ssdg continued to rotate.	0		N/G input
	Sac apparently seized during clutch engagement causing input drive shaft to remain stationary while drive adapter hub on ssdg continued to rotate.	8	4,6,18,20,50,51, 59,60 (133)	6
27.1.3	Drive shaft sheared all internal gear teeth from drive adapter hub.	3	3,5,7 (9)	1
29.0.0	Technical assist requested.	1	2 (2)	1
29.1.1	Fct open and inspect, revealed bearing material on bottom of strainer.	0		N/G input
	Fct open and inspect revealed bearing material on bottom of strainer.	2	4,6 (12)	1
29.1.2	After flushing unit, engaged pressure dropped to 62 psig within 45 seconds of engaging sac.	16	168,171,180,181,184, 186,192,194,284,286, 298,301,304,309 (311)	8
29.1.3	Disengaged pressure satisfactory.	1	2 (3)	1
30.0.0	Technical assistance requested.	1	2 (2)	1

No.	Text	No.Parses	Times	Correct parse #
30.1.1	Loss of one of two sac's.	2	10,11 (14)	1
30.1.2	Unit has low output air pressure, resulting in slow gas turbine starts.	4	18,20,26,28 (52)	2
30.1.3	T/S revealed normal sac lube oil pressure/temperature.	0		N/G scan
	Troubleshooting revealed normal oil pressure.	1	2 (3)	1
30.1.4	Impellor blade tip erosion evident.	1	3 (5)	1
30.1.5	Sac beyond shipyard repair.	1	5 (5)	1
30.1.6	Cause of erosion of impellor blades, undetermined.	1	3 (8)	1
30.1.7	Second generation sac received on-board for installation.	5	5,7,10,12,14 (15)	2
32.1.1	Loss of 50 percent of start air capability.	1	13...	1
32.1.2	Nr 2 sac can be operated at reduced capacity.	1	2 (3)	1
32.1.3	This situation present potential over temp hasard to lm2500 during start up evolutions and further degradation of mobility.	?		N/G input
	This situation presents potential over temp hasard to lm2500 during start up evolutions and further degradation of mobility.	over 90	2...	12?
32.1.4	Difficulty began with audible pulsations in compressor outlet air pressure under steady state conditions.	8	12,15,19,23,25,28,30,32 (34)	6?
32.1.5	Cause of casualty unknown.	1	2 (3)	1
33.0.0	Request shipyard replace.	1	1 92)	1
33.1.1	Oil pressure has been slowly decreasing.	1	1 (3)	1
33.1.2	Failure occurred during engine start when oil pressure dropped below 60 psig.	4	3,4,8,9 (12)	4
33.1.3	Investigation revealed excessive fine metal particles in oil.	2	3, 4 (5)	1
34.0.0	Assistance requested.	1	1 (2)	1
34.1.1	Loss of oil pump pressure.	?	8 ...	1
34.1.2	Suspect sheared connecting pin in pump drive assembly.	2	10,11 (17)	1
34.1.3	Loss of pressure was sudden and unexpected.	1	1 (2)	1

No.	Text	No.Parses	Times	Correct parse #
34.1.4	Investigation by todd revealed sac spline input drive shaft disconnected from diesel hub.	4	11,13,15,17 (25)	4
34.1.5	Hub assembly and spline shaft erroded beyond use.	0		N/G input
34.1.5	Hub assembly and spline shaft eroded beyond use.	2	4,*8 (10)	1
34.1.6	Todd LA to replace worn hub assembly and spline shaft.	7	15,18,19,26,27,31, 31 (34)	5
35.0.0	Parts ordered.	1	1 (2)	1
35.1.1	Experienced total loss of sac lo pressure and self disengagement while conducting gte water wash.	8	55,60,67,71,84,89,94,98 (105)	2
35.1.2	Investigation revealed stripped lo pump drive gear and hub ring gear.	2	7,8 (13)	1
36.0.0	Tech assist.	1	1 (1)	1
36.1.1	A number of slow gas turbine starts has been noted recently using 13 sac.	2	4,5 (9)	1
36.1.2	A trend of increasing lube oil temperature and decreasing lube oil pressure dictated cleaning the lube oil cooler and replacing the lube oil filter as corrective maintenance.	?	212, ...	4
	A trend of increasing lube oil temperature and decreasing lube oil pressure dictated ... replacing the lube oil filter as corrective maintenance.	over 30	26...	?
	A trend of increasing lube oil temperature ... dictated cleaning the lube oil cooler ... as corrective maintenance.	8	10,*12,*13,15,46, 48,50,51 (90)	4
36.1.3	After the maintenance was accomplished, operational tests revealed low lube oil pressure (65 psi which is low lube oil alarm set point) before the required three minute sac engaged time limit had run out.	19	114,125,131,140...	4
36.1.4	The lube oil filter was opened up revealing minute metallic particles.	4	2,5,8,12 (16)	4
36.1.5	Indications are that a new lube oil pump is required.	1	3(4)	1
36.1.6	Guarantee deficiency.	1	1 (1)	1

CASREPS.TESTB
Annotations to parse summary

[2.1.1]

Scanner problem. Period in abbreviation prevents parsing.

Structure of NP . In the closest parse obtained (the second), "operation nr. 2 ssdg" is parsed inaccurately with "operation" in npos modifying the namestg. However, introduction of implicit "of" seems ill-advised as a means of coping with this non-standard input.

[2.1.2]

Adverb problem . Restriction {d_d_or_p} prevents analysis of "in" as adverb.

[7.1.1]

"Only" is parsed somewhat questionably as an adjective in rn. "Monitoring" can only be parsed prenominally as adjective, not nvar.

[7.1.3]

One might argue that the second (nstg_frag) parse (with sa attachment of the prepositional phrase) is more accurate than the first parse (in which it is attached to rn), but the first is counted as correct.

[7.1.4]

Again, one might argue that the second parse (with the prepositional phrase in sa) is more accurate than the first parse (in which it is attached to rn), but the first is counted as correct.

Note that the ambiguity of "broken" as *ven or *adj doubles the parse count.

[7.1.6]

Number agreement. The grammatical error in this sentence is not the cause of its unparsability. (Note that {wagree} has had to be relaxed at least for "be", given grammatical sentences such as "ten minutes is the limit". In fact, not only "be" allows plural subjects with singular verb; cf. "ten minutes of listening to his chatter really taxes me to the limit". It seems to be a function of the semantics of the subject rather than the verb.) Thus the error in this sentence does not prevent it from being parsed.

The sentence as it stands seems incoherent. If it is taken as "[the (correct) lo pressure] and [alarm capability]", i.e., with an implicit modifier "correct", then the correct parse is the first one. And clearly it is unlikely that the correct reading is the one paraphraseable as "the capacity for lo pressure and alarm". Another possibility, suggested by NYU, is that "and" is a

typographical error.

[7.1.7]

"Sitrep 002:" is not treated as part of the sentence proper.

[7.1.8]

Scanner problem. "/" cannot be input.

[7.1.8]

"Utilizing" could be legitimately analyzed as noun modifier in apos or rn, or (correctly here) as sentence adjunct.

[7.1.9]

I assume that on the correct parse "lube oil pressure" is the subject. The second and third parses divide up the string of nouns differently between sa and subject.

[7.1.11]

In fact, this parses as a compound; correct parse is 3rd. Time: 1,460 sec!

Punctuation error is assumed for 7.1.11. Thus the comma preceding "when" has been changed to a period, as indicated, and 7.1.11 has been broken into two clauses to test its parsability in the absence of this error.

Second clause : The second parse for this clause is the correct but contextually incorrect analysis of the object as nn rather than nstgo.

[8.1.2]

Adverb problem. "Low" is mis-analyzed as adverbial sa in first two parses.

[11.1.1]

The first three parses are correct but distribute ln incorrectly ("surging" should be local, I assume).

The massive number of parses appears to be a function of conjunction; whether there are, in fact, 55 distinct and grammatical analyses remains to be determined. In the absence of the conjoined material (that is, with the first comma and everything to its right deleted), there are only three parses.

[12.1.1]

Scanner problem. "." cannot be input.

The second two parses take "point" as the (arguably intransitive) main verb.

[13.1.1]

Punctuation error. This sentence is ungrammatical as punctuated. It has

been reanalyzed into two clauses. However, it may still be unacceptable: "failed" would seem more likely to take the *sac*, rather than the oil pressure, as its subject.

Second clause : Conjunction problems . For some reason, three assertions are not parseable in conjunction rules generated from this grammar. This forces 13.1.1b to be parsed as three *ltvr*'s, but the absence of *rv* prevents the attachment of their *pn*'s ("to 72psi", etc). Thus only the readings in which "increased" and "decreased" are past participles in *rn* remain. With the addition of "has" (see table), the correct parse is, in fact, the first parse generated.

Also, "then" (but not "and") as the conjunction allows for an incorrect reading in which a "copied nullobj" is created in the first conjunct.

[14.1.2]

Perhaps the sixth (rather than the fifth) parse is the most accurate, since it attaches "below" as object rather than *sa*. In general, *sa* attachment of subcategorized-for *pn*'s is not regarded as an error, unless the verb + *pn* form a virtual idiom.

The variety of parses arises from the different attachment possibilities of "for two minutes", the "when"-clause, "below 65 psi"; the two analyses of "65 psi alarm setting"; and the analysis of subject as gerund or *lnr*.

Also, the fact that the entire sentence can be initially misanalyzed as an *lnr* contributes to the long parsing times.

[14.1.3]

Two ambiguities in the absence of conjunction (nominal or adjectival analysis of "low", "following" as **p* or **ving*) combine with a three-way conjunction ambiguity ("remote or local" analyzed as a conjoined *adjadj* or a conjoined *lnr*, the first one headed by *nulln*). With the latter, there is the ambiguity between distributed or local scope for *tpos*, *qpos*. The correct parse is assumed to be the first, in which "low" is adjectival, "following" is a preposition, and "remote or local" is a conjoined phrase in *adjadj*.

[14.1.4]

In the parse listed as correct, "in appearance" is a sentence adjunct. However, the fourth parse, in which it is a right adjunct of the adjective "dark", is probably still more accurate.

Adverb problem . Clearly a finer-grained analysis of adverbs is necessary. In the first two parses, "very" is analyzed incorrectly as an adverb. The adverb features developed by Sager will clearly prove useful here, but there are difficulties in applying them. There is no one feature which is associated with all and only those adverbs which are acceptable in *sa* position. For example, not all adverbs which may occur in *sa* position are marked with the feature

"dsa": neither "yet" (as in "She has not eaten lunch YET") and "there" (as in "He was happy THERE") is dsa. There is a group of features one or another of which characterizes any adverb which may appear in sa; this group includes dsa, dlw, drv, drw. However, any adverb input by the SDC lexical entry procedure has an empty feature list, so a restriction limiting adverbs in sa to those bearing one of these features would require considerable lexical work. Finally, an attempt to exclude sa analyses of adverbs like "very" by forbidding adverbs with certain features (such as dla -- left adjunct of adjective) will prove too strong, since, e.g., "always" is marked with the feature dla as well as drv/drw/dlw.

[15.1.2]

I assume that "oil pressure normal" is not to be taken as part of a conjoined object of "indicate", as the color of oil would not be an indicator of oil pressure. Thus the analysis of this sentence as a compound is assumed to be correct.

First clause : Shapes needs to be developed so as to recognize part numbers for this domain. Currently, 23699 is parsed only as qpos.

[16.1.1]

This sentence presents a number of difficulties.

Grammatical error . "Alarm were received" should perhaps be "alarms were received",

Multiple sa's . The correct analysis of this sentence would seem to involve two initial sa's, something currently disallowed by the grammar. Thus "approx 90 sec after clutch engagement" is incorrectly parsed as an appos attached to "turbines".

Treatment of appositives : This points up the inadequacy of the current appos rule, which substitutes for rn and is therefore not associable with a head noun which itself contains an rn.

Conjunction. The rules do not currently allow for conjoined ln, so that "(low lube oil) and (fail to engage) alarms" cannot be correctly parsed. (And the contextually appropriate parse of "disk and sac alarms are required" cannot be generated.)

There are an extraordinarily large number of parses in which the conjunction is associated with the introductory pn in sa, the subject being "alarms".

Structure of NP . Also, the bnf rules do not currently allow for modification in npos, as in "low lube oil alarm".

This sentence clearly requires further work, because of the inadequacy of the parses obtained and the very long parsing times.

[16.1.3]

Conjunction. This sentence does not parse without addition of "were" to second conjunct. Conjunction rules do not seem to handle (verb) gapping, even without the "sloppy identity" that holds here between the overt and implicit instances of "be". ("Sac was repaired and disk replaced" is also rejected.) We could allow "and" to join conjuncts, but this seems dubious: cf. "sac was repaired - replacement of blade" vs *"sac was repaired and replacement of blade".

[17.1.1]

This is parsed as a compound, with *nstg_frag* the first element. The second parse is the more accurate one: "one" is in *qpos* modifying *nulln*. (In the first parse, "one" is the head *nvar*.) As with other fragments, only parses with the first fragment option to succeed are listed in table.

Note that *zerocopula* reading of first conjunct is ruled out by assorted heuristics (*{d_of}*, *{w_nonnull_in}*).

[19.1.1]

Punctuation. Apostrophe must be added.

[19.1.3]

Appos. The first three readings construe the second conjunct as an appositive on the first; *appos* and *null* options in *rn* should probably be re-ordered.

Conjunction. Are the twelve conjunction readings distinct possibilities? The contextually correct reading comes late because earlier readings copy the *pnpn* attached to the final conjunct ("of pressure regulator") into earlier conjuncts, while the correct reading would seem to be the local one.

[19.1.4]

Scanner problem. Word-internal dash not currently recognizable.

Wagree. Sentences such as this require that *wagree* be relaxed to allow plural subjects with "is". (Cf. "Peanut butter and pickles is a horrible combination").

[20.1.1]

Conjunction. Parsing times seem extraordinarily long for this sentence, even given its numerous unexpected conjunction ambiguities (the initial *pn* may be taken as containing three conjoined NPs; the first four readings, for example, take "start" as the first of three conjoined NPs; the next five or more take "pressure" as subject).

Upon removal of the first conjunct ("air pressure dropped below 30 psi"), a single (correct) parse is generated in 11 sec (25 to NMP), as indicated in table.

[26.1.2]

The conjunction and pn attachment possibilities in this sentence are legion, and have not all been examined; in addition, there is an ambiguity between npn (contextually inappropriate) and nstgo object analyses. (The npn object option has pval "in", as in "We engaged them in conversation".)

[26.1.5]

Xor problem. Because there is an assertion reading (with "contributor" the nstgo object of active "believed"), the correct zerocopula parse (in which "contributor" is the remanants of active subjbe) is not generated. However, selection can easily rule out this reading.

[27.1.1]

Xor problem. Because there is an assertion parse (with "engage" as main verb), the contextually correct two parse is not generated.

Re the long time to first parse: note that the analysis of "experienced" as prenominal *ven creates a severe garden path.

[29.1.1]

Input. Punctuation error.

"Open and inspect" entered as idiom in lexicon.

[29.1.2]

The extraordinarily long parsing time for this sentence needs to be investigated. (Note that it does present a considerable garden path to the parser, since the entire string "engaged sac" could be analyzed as an NP.)

The various analyses depend upon analysis of the two [ving nvar] sequences as inr or gerund (in both cases) and on pn attachment. The selection of the eighth parse as the correct one needs to be verified (accidental logout prevented closer inspection).

[30.1.1]

Structure of NP. "One" can be parsed as nvar (in first parse) or q. I mark the first parse as correct, though presumably the second is the truly correct one. Will this create difficulties for semantics?

Note that zerocopula analysis is prevented by requirement that predicate nominal have nonnull ln (compare "party a disaster" with "party disaster").

[30.1.3]

Scanner problem. Word-internal slashes not accepted.

[30.1.4]

Note that the requirement that *ln* be nonnull, {*w_nonnull_ln*}, eliminates other zerocopula analyses.

[30.1.5]

{*w_nonnull_ln*} eliminates other zerocopula readings.

[30.1.6]

Comma is now allowed post-subject in zerocopula, which may add considerably to the number of parses for zerocopulas and compounds.

[32.1.3]

Grammatical error . "Present" should have been "presents". I assume (without conviction) that "over temp" is equivalent to "overheating"; thus it is entered in the lexicon as an idiom.

The *nn* subcategorization for "present" has been removed from the lexicon, as it sounds ungrammatical to me and contributes an additional 20 parses to this sentence. However, addition of *npn* subcategorization adds parses.

The variety of parses arises from the various *pn* attachment possibilities, the ambiguity of "start up" as an idiom or noun followed by preposition; and, of course, the scope possibilities associated with conjunction.

[32.1.4]

The meaning of this sentence is unclear: are the pulsations really pulsations in *air pressure* ? The sentence as punctuated would seem to have no other analysis.

The correct analysis is assumed (without conviction) to be that in which "with" is in *pn* object of "begin" and "under" is in *sa*.

[33.0.0]

"Replace" would have to be entered as a noun to parse this header, but see 34.1.6 for consequences of this.

[33.1.1]

The gerund and *ving/nvar* readings are prevented by {*d_nullLnsr*} and {*w_ving_lnr*}.

[33.1.2]

Although the fourth parse is listed as the correct one ("when" in *sa*, "below" in object), the first parse might be adequate ("when" in *rn*, "below" in *sa*).

[34.1.4]

Although the fourth parse (seven object, "from" in pn object) is listed as the correct one, the first is perhaps adequate: nstgo object, "from" in sa.

[34.1.6]

What is "LA" here? Part of "TODD"? An abbreviated predicate of some sort? A locative phrase? It is treated here as simply *n.

The first four analyses can be eliminated if "replace" is not categorized as a noun (necessary for 33.0.0, which is perhaps a frozen phrase anyway; perhaps an elliptical tv).

[36.1.1]

Wagrec should perhaps be modified to allow for plural verbs following phrases like "a number of NP". (In this case, however, the verb is singular.)

Shapes (?): "13 sac" is parsed incorrectly as [qpos + nvar]; a more complete treatment of equipment names in this corpus is in order.

{*d_lv*} should be modified to rule out second parse in which "recently" is in lv of "using".

[36.1.2]

Note very long time (219 sec) to first parse. Correct parse is fourth.

{*d_init_sa*} disallows the reading(s) in which conjoined lnr's are flanked by vingo sa's.

"As" is (incorrectly) treated as a conjunction in certain parses because it is listed in the lexicon as a spword.

[36.1.3]

Adverb problem. Again, "low" is parsed as an adverb in sa in the first reading.

The very long parsing times need to be examined. (Note that times are shortened by adding "sac engaged time limit" as an idiom — first parse in 71 sec, 100 sec to correct parse — rather than parsing sac (oddly but not really inadequately) as an [lcda + ven].)

Also, there appear to be some duplicate parses.

[36.1.4]

The various well-formed but contextually incorrect parses generated include analysis of "up" as preposition (rather than particle), and of "revealing ..." as a gerund. (cf. "For years they talked about revealing the secret of their great wealth")

[38.1.6]

What does this mean? Xor will only allow the two reading.

END

7-87

Dtic