The CIRCUS System as Used in MUC-3

Wendy Lehnert, Robert Williams,
Claire Cardie, Ellen Riloff and David Fisher

COINS Technical Report 91-59

Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

This report contains a technical description of a sophisticated natural language processing system designed to operate on unconstrained texts describing acts of terrorism. This particular system was used in the final evaluation of the Third Message Understanding Conference (MUC-3), where it posted the highest recall rates and highest combined scores for recall and precision of all the systems under evaluation. Details are provided with respect to semantic and syntactic sentence analysis as well as the discourse-level analysis of complete texts. Of particular interest is our ability to robustly analyze texts using only a minimal dictionary (6,000 words), and without generating syntactic parse trees for individual sentences. A case-based reasoning (CBR) approach to discourse analysis is also noteworthy. In addition, this system description indicates how much domain-specific effort was needed to bring a generic sentence analyzer up to speed for a particular information extraction application.

ACKNOWLEDGEMENTS: Our participation in MUC-3 was supported by a DARPA contract administered by Meridian Aggregates Co., Contract No. MDA903-89-C-0041, the Office of Naval Research, under a University Research Initiative Grant, Contract No. N00014-86-K-0764 and NSF Presidential Young Investigators Award NSFIST-8351863.

Contents

Preface

This technical report describes the major components of the UMass/MUC-3 system as it ran during the official testing for the MUC-3 performance evaluation in May, 1991. Some portions of this report also appear in the MUC-3 Conference Proceedings (soon to be published by Morgan Kaufmann); the others have not been previously published. Everyone associated with the UMass/MUC-3 effort contributed to this system description. Claire Cardie wrote the sections on concept node definitions and other features of CIRCUS-based processing. Ellen Riloff contributed the sections on rule-based consolidation. Robert Williams described the case-based consolidation module. David Fisher supplied the sections on preprocessing, the dictionary, and dictionary construction. We are also grateful to Priscilla Coe for assembling this document and converting a variety of text files and figures to a common LaTex format.

In the interest of releasing a timely system description, we have not labored over the text to ensure polished prose or comprehensive technical coverage. We were more concerned with recording a system description before our thinking about these problems changed too much, and moreover, without prolonging our collective MUC-3 mindset beyond a much-needed but necessarily short period of recuperation.

The system described in these pages evolved from a group effort over a period of about one year. It required constant attention, disciplined devotion, and an enthusiastic commitment from everyone. In return, our MUC-3 efforts provided us with a stimulating intellectual challenge that propelled us in new research directions. In the weeks that have now passed since the final evaluation, each of us has shifted our attention to a variety of new ideas that had to be put on the "back burner" while MUC-3 was dominating our lives. We expect additional publications to be forthcoming from each of the individuals associated with the UMass/MUC-3 system, and a more thorough presentation of our theoretical foundations will be made available as those efforts materialize. In the meantime, this bare-bones system description will provide technical details for those who need to know, and a detailed snapshot of the system that participated in the final evaluation for MUC-3.

Wendy G. Lehnert
August 14, 1991

Chapter 1

Introduction and Reader's Guide

Selective concept extraction is a sentence analysis technique that simulates the human ability to skim text and extract information in a selective manner. People engage in this mode of text comprehension when they are strongly goal-oriented and are only interested in a relatively small proportion of the total information present in a text. For example, someone reading an accident report may only care about the identify of the victims. Or someone reading a technical report may only care about the way that a particular problem is handled.

We have implemented a sentence analyzer (CIRCUS) that automates selective concept extraction using computational techniques drawn from both symbolic and connectionist traditions. It is based on a strongly predictive version of preference semantics that integrates syntactic knowledge in a highly systematic and robust manner. CIRCUS produces case frame instantiations that can be designed to accommodate any representational theory deemed desirable for a given domain. CIRCUS does not presume complete dictionary coverage for a particular domain, and does not rely on the application of a formal grammar for syntactic analysis. Just as people skip entire sections of text, CIRCUS ignores entire sentences when it does not recognize any of the words as being relevant to its predefined expectations. More importantly, CIRCUS can selectively analyze those portions of a sentence that are relevant to its expectations while ignoring other parts of the sentence altogether.

The basic ideas behind CIRCUS are relatively simple and can be mastered by a bright undergraduate within a semester. It takes considerably longer to become skilled at the knowledge engineering required for a specific application of CIRCUS. We have conducted a very serious knowledge engineering experiment in conjunction with CIRCUS as participants in DARPA's MUC-3 performance evaluation for state-of- the-art natural language processors. This technical report describes the resulting system as it ran during the final MUC-3 evaluation test. The original MUC-3/CIRCUS system was implemented in Common LISP on a TI Explorer II supporting 8 MB of RAM. A second Common LISP implementation has since been written for the MAC II so the system is now available to other research labs on two platforms.

Our system description has been organized to accommodate both casual readers and readers looking for lots of technical details. For a broad overview of the system, a casual reader can go through section II (A Brief Overview of the MUC-3/CIRCUS System) and then dip into subsequent sections as interests dictate. More ambitious readers can continue on through all of Chapter III (Sentence Analysis) and/or Chapter IV (Discourse Analysis) to get more details throughout.

Although we have tried to anticipate the needs of different readers, this document is nevertheless a technical report in the truest sense of the term. We present here a technical system description without much historical motivation, pointers to related literature, or any attempt to lay out theoretical claims. As such, this document is not intended to be a self-contained summary of our MUC-3 involvement or beliefs about natural language processing. Other publications should be read in conjunction with this report in order to provide a more satisfactory picture of ongoing research in natural language processing at the University of Massachusetts.

For an overview of the MUC-3 text analysis performance evaluation, please refer to "A performance evaluation of text processing technologies" by W. Lehnert and B. Sundheim, in the fall 1991 issue of *AI Magazine*. For a more in-depth description of MUC-3, the *Proceedings for the Third Message Understanding Conference* (publisher: Morgan Kaufmann, 1991) is the definitive publication on MUC-3 containing all official test results, site reports, and system descriptions.

For more background on the CIRCUS sentence analyzer, the best introduction is "Symbolic/subsymbolic sentence analysis: Exploiting the best of two worlds" by W. Lehnert in *Advances in Connectionist and Neural Computation Theory* (eds: J. Pollack and J. Barnden, 1991). This paper is also available as a technical report from the Department of Computer Science at the University of Massachusetts (COINS TR88-99). The CIRCUS sentence analyzer was initially used to process citation sentences in technical literature. This earlier experiment with selective concept extraction is described in "Analyzing research papers using citation sentences" by W. Lehnert, C. Cardie, and E. Riloff, in the *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (1990). The LICK formalism developed to handle complex sentences with embedded clauses is also presented in "A cognitively plausible approach to understanding complex syntax" by C. Cardie and W. Lehnert in the *Proceedings of the Ninth National Conference on Artificial Intelligence* (1991).

Chapter 2

A Brief Overview of the MUC-3/CIRCUS System

2.1 Background and Motivation

In 1988 Professor Wendy Lehnert completed the initial implementation of a semantically- oriented sentence analyzer named CIRCUS [1]. The original design for CIRCUS was motivated by two basic research interests: (1) we wanted to increase the level of syntactic sophistication associated with semantically-oriented parsers, and (2) we wanted to integrate traditional symbolic techniques in natural language processing with connectionist techniques in an effort to exploit the complementary strengths of these two computational paradigms.

Shortly thereafter, two graduate students, Claire Cardie and Ellen Riloff, began to experiment with CIRCUS as a mechanism for analyzing citation sentences in the scientific literature [2]. The key idea behind this work was to extract a relatively abstract level of information from each sentence, using only a limited vocabulary that was hand-crafted to handle a restricted set of target concepts. We called this mode of language processing *selective concept extraction*, and the basic style of sentence analysis was a type of text skimming. This project provided us with an opportunity to give CIRCUS a workout and determine whether or not the basic design was working as expected.

Although CIRCUS was subject to a number of limitations, the integration of syntax and semantics appeared to work very nicely. We believed we had constructed a robust text skimmer that was semantically oriented but nevertheless able to use syntactic knowledge as needed. Projects associated with the connectionist aspect of CIRCUS took off at about this time and carried us in those directions for a while [3,4,5]. When an announcement for MUC-3 reached us in June of 1990, we felt that the MUC-3 evaluation required selective concept extraction capabilities of just the sort we had been developing. We were eager to put CIRCUS to the test.

It was clear to us that MUC-3 would require a much more ambitious and demanding appli-

cation of CIRCUS than our earlier work on citation sentences, and we fully expected to learn a great deal from the experience. We hoped to capitalize on Cardie and Riloff's previous experience with CIRCUS while identifying some new areas for ongoing research in sophisticated text analysis. In September of 1990, Robert Williams joined our MUC-3 effort as a post doc with research experience in case-based reasoning. Cardie, Riloff, and Williams provided the technical muscle for all of our MUC-3 system development and knowledge engineering. Cardie was primarily responsible for CIRCUS and dictionary design, Riloff developed the rule-based consolidation component, and Williams designed the case-based reasoning consolidation component. Although the division of labor was fairly clean, everyone worked with CIRCUS dictionary definitions and the preprocessor at various times as needed. In January of 1991, David Fisher joined the project as an undergraduate assistant who designed an interface for faster dictionary development while assisting with internal testing. Professor Lehnert assumed a leadership role but made no programming contributions to MUC-3.

2.2 System Components

Although CIRCUS was the primary workhorse underlying our MUC-3 effort, it was necessary to augment CIRCUS with a separate component that would receive CIRCUS output and massage that output into the final target template instantiations required for MUC-3. This phase of our processing came to be known as consolidation, although it corresponds more generally to what many people would call discourse analysis. We will describe both CIRCUS and our consolidation processing with examples from TST1-MUC3-0099.

TST1-MUC3-0099

(S1) LIMA, 25 OCT 89 (EFE) – [TEXT] POLICE HAVE REPORTED THAT TERRORISTS TONIGHT BOMBED THE EMBASSIES OF THE PRC AND THE SOVIET UNION.

(S2) THE BOMBS CAUSED DAMAGE BUT NO INJURIES.

(S3) A CAR-BOMB EXPLODED IN FRONT OF THE PRC EMBASSY, WHICH IS IN THE LIMA RESIDENTIAL DISTRICT OF SAN ISIDRO.

(S4) MEANWHILE, TWO BOMBS WERE THROWN AT A USSR EMBASSY VEHICLE THAT WAS PARKED IN FRONT OF THE EMBASSY LOCATED IN ORRANTIA DISTRICT, NEAR SAN ISIDRO.

(S5) POLICE SAID THE ATTACKS WERE CARRIED OUT ALMOST SIMULTANEOUSLY AND THAT THE BOMBS BROKE WINDOWS AND DESTROYED THE TWO VEHICLES.

(S6) NO ONE HAS CLAIMED RESPONSIBILITY FOR THE ATTACKS SO FAR.

(S7) POLICE SOURCES, HOWEVER, HAVE SAID THE ATTACKS COULD HAVE BEEN CARRIED OUT BY THE MAOIST "SHINING PATH" GROUP OR THE GUEVARIST "TUPAC AMARU REVOLUTIONARY MOVEMENT" (MRTA) GROUP.

(S8) THE SOURCES ALSO SAID THAT THE SHINING PATH HAS ATTACKED SOVIET INTERESTS IN PERU IN THE PAST.

(S9) IN JULY 1989 THE SHINING PATH BOMBED A BUS CARRYING NEARLY 50 SOVIET MARINES INTO THE PORT OF EL CALLAO.

(S10) FIFTEEN SOVIET MARINES WERE WOUNDED.

(S11) SOME 3 YEARS AGO TWO MARINES DIED FOLLOWING A SHINING PATH BOMBING OF A MARKET USED BY SOVIET MARINES.

(S12) IN ANOTHER INCIDENT 3 YEARS AGO, A SHINING PATH MILITANT WAS KILLED BY SOVIET EMBASSY GUARDS INSIDE THE EMBASSY COMPOUND.

(S13) THE TERRORIST WAS CARRYING DYNAMITE.

(S14) THE ATTACKS TODAY COME AFTER SHINING PATH ATTACKS DURING WHICH LEAST 10 BUSES WERE BURNED THROUGHOUT LIMA ON 24 OCT.

Complete traces of system output for TST1-MUC3-0099 are listed in Appendices A, B, and C.

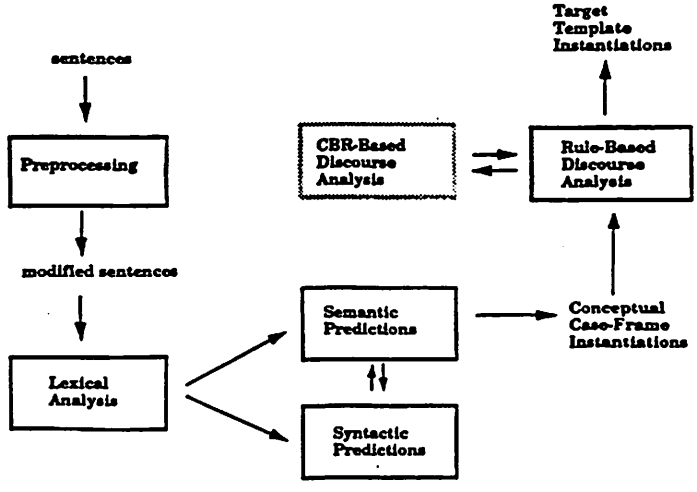

Figure 2.1: Flow Chart for the MUC-3/CIRCUS System

2.2.1 Sentence Preprocessing

To begin, each sentence is given to our preprocessor where a number of domain-specific modifications are made. (1) Dates are analyzed and translated into a canonical form. (2) Words associated with our phrasal lexicon are connected via underscoring. (3) Punctuation marks are translated into atoms more agreeable to LISP. For example, the first sentence (S1) reads:

S1: POLICE HAVE REPORTED THAT TERRORISTS TONIGHT BOMBED THE EMBASSIES OF THE PRC AND THE SOVIET UNION.

After preprocessing, we have:

S1: (POLICE HAVE REPORTED THAT TERRORISTS ON OCT_25 _89>C0 TONIGHT BOMBED THE EMBASSIES OF THE PRC AND THE SOVIET_UNION >PE)

The canonical date was derived from "tonight" and the dateline of the article, "25 OCT 89." Most of our phrasal lexicon is devoted to proper names describing locations and terrorist organizations (831 entries). 25 additional proper names are also recognized, but not from the phrasal lexicon.

2.2.2 Lexical Analysis

At this point we are ready to hand the sentence to CIRCUS for lexical processing. This is where we search our dictionary and apply morphological analysis in an effort to recognize words in the sentence. Any words that are not recognized receive a default tag reserved for proper nouns in case we need to make sense out of unknown words later. In order for any semantic analysis to take place, we need to recognize a word that operates as a trigger for a concept node definition. If a sentence contains no concept node triggers, it is ignored by the semantic component. This is one way that irrelevant texts can be identified: texts that trigger no concept nodes are deemed irrelevant. Words in our dictionary are associated with a syntactic part of speech, a position or positions within a semantic feature hierarchy, possible concept node definitions if the item operates as a concept node trigger, and syntactic complement predictions. Concept nodes and syntactic complement patterns will be described in the next section. An example of a dictionary entry with all four entry types is our definition for "dead" as seen in figure 2.2.

```
(D-WORD DEAD
    :SYNTACTIC-TYPE SPECIAL-ADJECTIVE
    :SYNTACTIC-EXPECTATIONS
        (((assign *np-flag* t
                  *predicates* (append *predicates* (list *word*))
                  *part-of-speech* 'adjective
                  *cd-form* (make-special *word*)
                  *global-cn* *cd-form*)
          (next-packet
            ((test (eq *part-of-speech* 'noun)))
            ((test (eq *part-of-speech* 'adjective))))))
    :WORD-SENSES (dead1)
    :CN-DEFS ($LEFT-DEAD$ $FOUND-DEAD$ $FOUND-DEAD-PASS$))
```

Figure 2.2: The Dictionary Definition for "DEAD"

When morphological routines are used to strip an inflected or conjugated form back to its root, the root-form dictionary definition is dynamically modified to reflect the morphological information. For example, the root definition for "bomb" will pick up a :VERB-FORM slot with PAST filling it when the lexical item "bombed" is encountered.

2.2.3 Semantic and Syntactic Predictions

Words associated with concept nodes activate both syntactic and semantic predictions. In S1 the verb "bombed" activates semantic predictions in the form of a concept node designed to describe a bombing. Each concept node describes a semantic case frame with variable slots that expect to be filled by specific syntactic constituents. The concept node definition activated by "bombed" in S1 is given in Figure 2.3.

We can see from this definition that a case frame with variable slots for the actor and target is predicted. The actor slot expects to be filled by an organization, the name of a recognized terrorist or generic terrorist referent, a proper name, or any reference to a person. The target slot expects to be filled by a physical target. We also expect to locate the actor in the subject of the sentence, and the target should appear as either a direct object or the object of a prepositional phrase containing the preposition "in." None of these predictions will be activated unless the current sentence is in the active voice.

```
;; X bombed/dynamited/blew_up
;; the bomb blew up in the building (tst1-0040) -emr
;; (if this causes trouble we can create a new cn for blew_up)
(define-word $BOMBING-3$
    (CONCEPT-NODE
    ':NAME '$BOMBING-3$
    ':TIME-LIMIT 10
    ':SLOT-CONSTRAINTS '(((class organization *S*)(class terrorist *S*)
                          (class proper-name *S*)(class human *S*))
                         ((class phys-target *DO*) (class phys-target *PP*)))
    ':VARIABLE-SLOTS '(
                        ACTOR (*S* 1)
                        TARGET (*DO* 1 *PP* (is-prep? '(in))))
    ':CONSTANT-SLOTS '(
                        TYPE BOMBING))
    ':ENABLED-BY  '((active))))
```

Figure 2.3: The $BOMBING-3$ Concept Node Definition

Syntactic complement predictions are managed by a separate mechanism that operates independently of the concept nodes. The syntactic predictions fill syntactic constituent buffers with appropriate sentence fragments that can be used to instantiate various concept node case frames. Syntactic predictions are organized in decision trees using test-action pairs under a stack-based control structure [6]. Although syntactic complements are commonly associated with verbs (verb complements), we have found that nouns should be used to trigger syntactic complement predictions with equal frequency. Indeed, any part of speech can trigger a concept node and associated complement predictions as needed. As we saw in the previous section, the adjective "dead" is associated with syntactic complement predictions to facilitate noun phrase

analysis. Figure 2.4 shows the syntactic complement pattern predicted by "Bombed" once morphology recognizes the root verb "bomb."

```
(((test (second-verb-or-infinitive?))    ;;  all verbs call this function
  (assign *part-of-speech* 'verb         ;;  just to be sure...reset some buffers for noun phrase collection

    *np-flag* nil *noun-group* nil  *predicates* nil *entire-noun-group* nil
    *determiners* nil *appositive* nil *gerund* nil

                                         ;;  verb assignments.

    *cd-form* (make-verb *word*)        *V* *cd-form*      *DO* nil)

  (next-packet                           ;;  next noun phrase should be the direct object

    ((test (equal *part-of-speech* 'noun-phrase))
     (assign *DO* *cd-form*))            ;;  don't predict *DO* if conjunction follows the verb,
                                         ;;   e.g., in "X was damaged and Y was destroyed",
                                         ;;  Y should NOT be *DO* of "damaged"
    ((test (equal *part-of-speech* 'conjunction))))))
```

Figure 2.4: The Verb Complement Pattern for "BOMBED"

Remarkably, Figure 2.4 displays all the syntactic knowledge CIRCUS needs to know about verbs. Every verb in our dictionary references this same prediction pattern. In particular, this means that we have found no need to distinguish transitive verbs from intransitive verbs, since this one piece of code handles both (if the prediction for a direct object fails, the *DO* buffer remains empty).

Once semantic and syntactic predictions have interacted to produce a set of case frame slot fillers, we then create a frame instantiation which CIRCUS outputs in response to the input sentence. In general, CIRCUS can produce an arbitrary number of case frame instantiations for a single sentence. No effort is made to integrate these into a larger structure. The concept node instantiation created by $BOMBING-3$ in response to S1 is given in Figure 2.5.

Some case frame slots are not predicted by the concept node definition but are inserted into the frame in a bottom-up fashion. Slots describing time specifications and locations are all filled by a mechanism for bottom-up slot insertion (e.g. the REL-LINK slot in Figure 2.5 was created in this way). Although the listing in Figure 3.5 shows only the head noun "embassies" in the target noun group slot, the full phrase "embassies of the PRC and the Soviet Union" has been recognized as a noun phrase and can be recovered from this case frame instantiation. The target value "ws- diplomat-office-or-residence" is a semantic feature retrieved from our dictionary definition for "embassy.' No additional output is produced by CIRCUS in response to S1.

S1: (POLICE HAVE REPORTED THAT TERRORISTS ON OCT_25_89 >C0 TONIGHT BOMBED THE EMBASSIES OF THE PRC AND THE SOVIET_UNION >PE)

```
***  TYPE = BOMBING
***  ACTOR = WS-TERRORIST
***  noun group = (TERRORISTS)
***  TARGET = WS-DIPLOMAT-OFFICE-OR-RESIDENCE
***  noun group = (EMBASSIES)
***  determiners = (THE)
***  REL-LINK (TIME  (OCT_25_89)))
```

Figure 2.5: CIRCUS Output for S1

It is important to understand that CIRCUS uses no sentence grammar, and does not produce a full syntactic analysis for any sentences processed. Syntactic constitutents are utilized only when a concept node definition asks for them. Our method of syntactic analysis operates locally, and syntactic predictions are indexed by lexical items. We believe that this approach to syntax is highly advantageous when dictionary coverage is sparse and large sentence fragments can be ignored without adverse consequences. This allows us to minimize our dictionaries as well as the amount of processing needed to handle selective concept extraction from open-ended texts.

Some concept nodes are very simple and may contain no variable slots at all. For example, CIRCUS generates two simple frames in response to S2, neither of which contain variable slot fillers.

Note that the output generated by CIRCUS for S2 as shown in Figure 2.6 is incomplete. There should be a representation for the damage. This omission is the only CIRCUS failure for TST1-MUC3-0099, and it results from a noun/verb disambiguation failure. The 14 sentences in TST1-MUC3-0099 resulted in a total of 27 concept node instantiations describing bombings, weapons, injuries, attacks, destruction, perpetrators, murders, arson, and new event markers.

2.2.4 Other Problems in Sentence Analysis

Special mechanisms are devoted to handling specific syntactic constructs, including appositives and conjunctions. We will illustrate our handling of conjunctions by examining two instances of "and" in S5:

S5: Police said the attacks were carried out almost simultaneously and
(1) that the bombs broke windows **and**(2) destroyed the two vehicles.

S2: (THE BOMBS CAUSED DAMAGE BUT NO INJURIES >PE)

***	TYPE = WEAPON	
***	INSTR = BOMB	(triggered by the noun "bombs")
***	TYPE = INJURY	
***	MODE = NEG	(triggered by the noun "injuries")

Figure 2.6: CIRCUS Output for S2

0. MESSAGE ID	DEV-MUC3-0099
1. TEMPLATE ID	1
2. DATE OF INCIDENT	25 OCT 89
3. TYPE OF INCIDENT	BOMBING
4. CATEGORY OF INCIDENT	TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S)	"TERRORISTS"
6. PERPETRATOR: ID OF ORG(S)	-
7. PERPETRATOR: CONFIDENCE	-
8. PHYSICAL TARGET: ID(S)	"EMBASSIES OF THE PRC AND THE SOVIETUNION" "PRC EMBASSY"
9. PHYSICAL TARGET: TOTAL NUM	PLURAL
10. PHYSICAL TARGET: TYPE(S)	DIPLOMAT OFFICE OR RESIDENCE: "EMBASSIES OF THE PRC AND THE SOVIET UNION" DIPLOMAT OFFICE OR RESIDENCE: "PRC EMBASSY"
11. HUMAN TARGET: ID(S)	-
12. HUMAN TARGET: TOTAL NUM	-
13. HUMAN TARGET: TYPE(S)	-
14. TARGET: FOREIGN NATION(S)	PEOPLES REP OF CHINA: "EMBASSIES OF THE PRC AND THE SOVIET UNION" PEOPLES REP OF CHINA: "PRC EMBASSY"
15. INSTRUMENT: TYPE(S)	*
16. LOCATION OF INCIDENT	PERU: LIMA (CITY): SAN ISIDRO (NEIGHBORHOOD)
17. EFFECT ON PHYSICAL TARGET(S)	-
18. EFFECT ON HUMAN TARGET(S)	NO INJURY OR DEATH: "-"

Figure 2.7: Our Output Template Representation for S1-S3

We recognize that and(1) is not part of a noun phrase conjunction, but do nothing else with it. A new control kernel begins after "that" and reinitializes the state of the parser. **and2** is initially recognized as potentially joining two noun phrases — "windows" and whatever noun phrase follows. However, when the verb "destroyed" appears before any conjoining noun phrase is recognized, the LICK mechanism determines that the conjunction actually joins two verbs and begins a new clause. As a result, the subject of "broke" (i.e., "the bombs") correctly becomes the subject of "destroyedThe CIRCUS System as Used in MUC-3The CIRCUS System as Used in MUC-3The CIRCUS System as Used in MUC-3The CIRCUS System as Used in MUC-3The CIRCUS System as Used in MUC-3The CIRCUS System as Used in MUC-3 as well.

2.2.5 Rule-Based Consolidation

When an entire text has been processed by CIRCUS, the list of the resulting case frame instantiations is passed to consolidation. A rule base of consolidation heuristics then attempts to merge associated case frames and create target template instantiations that are consistent with MUC-3 encoding guidelines. It is possible for CIRCUS output to be thrown away at this point if consolidation does not see enough information to justify a target template instantiation. If consolidation is not satisfied that the output produced by CIRCUS describes bonafide terrorist incidents, consolidation can declare the text irrelevant. A great deal of domain knowledge is needed by consolidation in order to make these determinations. For example, semantic features associated with entities such as perpetrators, targets, and dates are checked to see which events are consistent with encoding guidelines. In this way, consolidation operates as a strong filter for output from CIRCUS, allowing us to concisely implement encoding guidelines independently of our dictionary definitions.

A number of discourse-level decisions are made during consolidation, including pronoun resolution and reference resolution. Some references are resolved by frame merging rules. For example, CIRCUS output from S1, S2 and S3 is merged during consolidation to produce the target template instantiation found in Figure 2.7.

The CIRCUS output from S1 triggers a rule called create-bombing which generates a template instantiation that eventually becomes the one in Figure 2.7. But to arrive at the final template, we must first execute three more consolidation rules that combine the preliminary template with output from S2 and S3. Pseudo-code for two of these three rules is given in Figure 2.8.

Note also that the location of the incident was merged into this frame from S3 which triggers another bombing node in response to the verb "exploded" as shown in figure 2.9.

S3: (A CAR_BOMB EXPLODED IN_FRONT_OF THE PRC EMBASSY >CO IN THE LIMA RESIDENTIAL DISTRICT OF SAN_ISIDRO >PE)

```
==========================================================================
MERGE-WEAPON-BOMBING

IF  $weapon structure
 and the weapon is an explosive
 and a BOMBING or ATTEMPTED-BOMBING template  is on the stack in the current family
 and dates are compatible and locations are compatible

THEN
  merge instruments, dates, and locations

==========================================================================
MERGE-BOMBING-BOMBING

IF  $bombing structure
 and BOMBING template  is on the stack in the current family
 and dates are compatible and locations are compatible

THEN
  merge perpetrators, human targets, physical targets, instruments, dates, and locations
  and also ...
  if there is a MURDER template with compatible victim (on the stack in the same family)
          with no instruments or the instruments are explosives
  then
  merge perpetrators, human targets, instruments, dates, and locations with the MURDER
==========================================================================
```

Figure 2.8: Two Merging Rules from Rule-Based Consolidation

```
***     TYPE = BOMBING
***     INSTR =
  >>>       TYPE = WEAPON
  >>>       INSTR = CAR_BOMB
  >>>       REL-LINK (TARGET OBJECT (WS-DIPLOMAT-OFFICE-OR-RESIDENCE))
  >>>       REL-LINK (LOC2 OBJECT (WS-GENERIC-LOC))
***     noun group = (CAR_BOMB)
***     determiners = (A)
***     REL-LINK (TARGET  (PRC EMBASSY)))
***     REL-LINK (LOC2  (LIMA DISTRICT)))
```

Figure 2.9: CIRCUS Output for S3

Once again, the top-level REL-LINK for a location is printing out only a portion of the complete noun phrase that was captured.

Although we would say that the referent to "bombs" in S2 is effectively resolved during consolidation, our methods are not of the type normally associated with linguistic discourse analysis. When consolidation examines these case frames, we are manipulating information on a conceptual rather than linguistic level. We need to know when two case frame descriptions are providing information about the same event, but we aren't worried about referents for specific noun phrases per se.

We did reasonably well on this story. Three templates of the correct event types were generated and no spurious templates were created by the rule base. Sentences S9 through S13 might have generated spurious templates if we didn't pay attention to the dates and victims. Here is how the preprocessor handled S12:

S12: IN ANOTHER INCIDENT 3 YEARS AGO, A SHINING PATH MILITANT WAS KILLED BY SOVIET EMBASSY GUARDS INSIDE THE EMBASSY COMPOUND.

S12: (IN ANOTHER INCIDENT ON -DEC_31_80 >CO &&3 YEARS AGO >CO >CO A SHINING_PATH MILITANT WAS KILLED BY SOVIET EMBASSY GUARDS INSIDE THE EMBASSY COMPOUND >PE)

Whenever the preprocessor recognizes a date specification that is "out of bounds' (at least two months prior to the dateline), it inserts -DEC_31_80 as a flag to indicate that the events associated with this date are irrelevant. This date specification will then be picked up by any concept node instantiations that are triggered "clos" to the date description. In this case, the event is irrelevant both because of the date and because of the the victim (murdered militants arent usually relevant). Despite the fact that S10 and S13 contain no date descriptions, the case frames generated for these sentences are merged with other frames that do carry disqualifying dates, and are therefore handled at a higher level of consolidation. In the end, the two murders (S11 and S12) are discarded because of disqualifications on their victim slot fillers, while the bombing (S9) was discarded because of the date specification. The injuries described by S10 are correctly merged with output from S9, and therefore discarded because of the date disqualifier. Likewise, the dynamite from S13 is correctly merged with the murder of the militant, and the dynamite is subsequently discarded along with the rest of that template.

2.2.6 Case-Based Consolidation

The CBR component of consolidation is an optional part of our system, designed to increase recall rates by generating additional templates to augment the output of rule-based consolidation. These extra templates are generated on the basis of correlations between CIRCUS output for a given text, and the key target templates for similarly indexed texts. The CBR component uses a case base which draws from 283 texts in the development corpus, and the 100 texts from TST1 for a total of 383 texts. We experimented with a larger case base but found no improvement in performance. The case base contains 254 template type patterns based on

CIRCUS output for the 383 texts in the case base.

Each case in the case base associates a set of concept nodes with a template containing slot fillers from those concept nodes. The concept nodes are generated by CIRCUS when it analyzes the original source text. A case has two parts: (1) an incident type, and (2) a set of sentence/slot name patterns. For example, suppose a story describes a bombing such that the perpetrator and the target were mentioned in one sentence, and the target was mentioned again three sentences later. The resulting case would be generated in response to this text:

BOMBING
0: (PERP TARGET)
3: (TARGET)

The numerical indices are relative sentence positions. The same pattern could apply no matter where the two sentences occurred in the text, as long as they were three sentences apart.

Cases are used to determine when a set of concept nodes all contribute to the same output template. When a new text is analyzed, a probe is used to retrieve cases from the case base. Retrieval probes are new sentence/slot name patterns extracted from the current CIRCUS output. If the sentence/slot name pattern of a probe matches the sentence/slot name pattern of a case in the case base, that case is retrieved, the probe has succeeded, and no further cases are considered.

Maximal probes are constructed by grouping CIRCUS output into maximal clusters that yield successful probes. In this way, we attempt to identify large groups of consecutive concept nodes that all contribute to the same output template. Once a maximal probe has been identified, the incident type of the retrieved case forms the basis for a new CBR-generated output template whose slots are filled by concept node slot fillers according to appropriate mappings between concept nodes and output templates.

In the case of TST1-MUC3-0099, case-based consolidation proposes hypothetical templates corresponding to 3 bombings, 2 murders, 1 attack, and 1 arson incident. Two of the bombings and the arson are discarded because they were already generated by rule-based consolidation. The two murders are discarded because of victim and target constraints, while the third bombing is discarded because of a date constraint. The only surviving template is the attack incident, which turns out to be spurious. It is interesting to note that for this text, the CBR component regenerates each of the templates created by rule-based consolidation, and then discards them for the same reasons they were discarded earlier, or because they were recognized to be redundant against the rule-based output. We have not run any experiments to see how consistently the CBR component duplicates the efforts of rule-based consolidation. While such a study would be very interesting, we should note that the CBR templates are generally more limited in the number of slot fillers present, and would therefore be hard pressed to duplicate the overall performance of rule-based consolidation.

2.3 MUC-3 Test Results

We believe that the score reports we obtained for TST2 provide an accurate assessment of our systems capabilities insofar as they are consistent with the results of our own internal tests conducted near the end of phase 2. The required TST2 score reports indicate that our system achieved the highest combined scores for recall (51%) and precision (62%) as well as the highest recall score of all the MUC-3 systems under the official MATCHED/MISSING scoring profile.

We ran one optional test in addition to the required test for TST2. The optional run differs from the required run in only one respect, an alteration to our consolidation module. The consolidation module contains all procedures that translate parser output into target template instantiations. The complete consolidation module includes a case-based reasoning (CBR) component that makes predictions about the target output based on a portion of the development corpus. For our optional run, we executed a modified version of consolidation that does not include this CBR component. We predicted that the absence of the CBR component would pull recall down but push precision up (looking at MATCHED/MISSING only). This trade-off prediction was confirmed by the required and optional TST2 score reports. (Please consult Appendix F for our required and optional test score summaries).

The source of our recall/precision trade-off can be found by examining the actual, spurious and missing counts for template-ids. When we run with CBR, we generate 215 actual templates as opposed to 137 actual templates without CBR. Most of these extra templates are spurious (64), but some are correct (14). The extra CBR templates increase our recall by reducing the number of missing templates from 16 to 6, while lowering our precision by raising the number of spurious templates from 44 to 108. The net effect of the CBR component on TST2 is a 4% gain in recall and a 3% loss of precision.

All of our system development and testing took place on a Texas Instruments Explorer II workstation running Common Lisp with 8 megabytes of RAM. It took about 1.5 hours to process the TST2 texts (without traces). No effort had been made to optimize run-time efficiency. Shortly after the final TST2 evaluation we found a way to reduce runtimes by about 40%.

2.4 System Development

Almost all of our MUC-3 effort has been knowledge engineering in one form or another. We can further categorize this effort in terms of (1) dictionary construction, and (2) discourse analysis. Dictionary construction received somewhat more attention than discourse analysis, with both relying heavily on examples from the development corpus. Overall, we estimate that roughly 30-40% of the development corpus was analyzed for the purposes of either dictionary construction or discourse analysis by the end of phase 2.

Because we are working with a domain-specific dictionary, we construct our lexicon on the basis of examples in the development corpus. Virtually all of our dictionary construction is done by hand. We examine texts from the corpus in order to identify critical verbs and nouns

that organize information relevant to the domain. Then we create syntactic and semantic predictions based on these instances with the expectation that similar linguistic constructs will be encountered in other texts as well. Our dictionary is effective only to the extent that we can extrapolate well on the basis of the examples we've seen.

Our TST2 dictionary contained 5407 words and 856 proper names (mostly locations and terrorist organizations). 1102 dictionary entries were associated with semantic features, and 286 entries operated as concept node triggers (CIRCUS cannot produce any output unless it encounters at least one concept node trigger). 131 verbs and 125 nouns functioned as concept node triggers. Our semantic feature hierarchy contained 66 semantic features. Although CIRCUS operates without a syntactic sentence grammar, it did exploit syntactic knowledge in the form of 84 syntactic prediction patterns, with 12 of these doing most of the work. CIRCUS also accessed 11 control kernels for handling embedded clause constructions [7].

Our version of discourse analysis took place during consolidation, when output from the CIRCUS sentence analyzer was examined and organized into target template instantiations. This translation from CIRCUS output to MUC-3 templates was handled by a rule base containing 139 rules. Consolidation errors could effectively destroy perfectly good output at the level of sentence analysis, so our overall performance was really only as good as our consolidation component. One of our ongoing problems was in trying to evaluate the performance of CIRCUS and the performance of consolidation independently. We never did manage to tease the two apart, but we are confident that both components would benefit from additional knowledge engineering.

Serious consolidation development could not really get underway until we had a large number of texts to examine along with internal scoring runs based on the development corpus. Although our earliest opportunity for this was November, dictionary deficiencies delayed substantial progress on consolidation until February or March. It was impossible to know how well consolidation was operating until CIRCUS could provide consolidation with enough input to give it a fighting chance. The consolidation rule base was generated by hand and modified upon inspection, with rapid growth taking place during phase 2. The number of consolidation rules almost doubled between TST1 and TST2.

We estimate that our time spent (measured in person/years) on technical development for MUC-3 was distributed as follows:

alterations to CIRCUS	.35
case-based discourse analysis	.15
corpus development	.25
dictionary construction	.75
rule-based discourse analysis	.50
test runs & other misc. technical	.25
	2.25 person/years

This estimate assumes that our graduate research assistants were working 30 hrs/wk on MUC-3, although it is notoriously difficult to estimate graduate student labor. General alterations to CIRCUS included morphological analysis, conjunction handling, noun phrase recognition, embedded clause handling, and machinery for some special constructions like appositives. These alterations to CIRCUS and the CBR component are all domain-independent enhancements. All other effort should be categorized as domain-specific.

2.5 Domain-Independent Advances

Prior to MUC-3, we had no experience with consolidation-style processing, so consolidation provided us with many opportunities to explore new problem areas. For example, we can locate pronominal referents both within sentences and across sentence boundaries 73% of the time (based on an analysis of pronouns in the relevant texts of the development corpus and TST1). However, these heuristics are limited to four pronouns and there are only 130 instances of these pronouns in the texts analyzed. We examined the role of pronoun resolution with internal test runs, and came to the conclusion that this particular problem has little impact on overall recall or precision.

A more compelling innovation for consolidation was first proposed in March, when we began to experiment with the CBR component. The CBR component allows our system to augment its final template output based on known correlations between CIRCUS output and target template encodings found in the development corpus. It performs this analysis using a case base of 254 template patterns drawn from the 100 TST1 texts along with 283 development corpus texts.

Case-based consolidation generates additional templates that might have been missed or dismissed during the rule-based analysis, and thereby reduces the number of missing templates. Because the CBR component effectively operates to counterbalance omissions made by rule-based consolidation, we expect that the gain in recall due to CBR will eventually diminish as the system becomes more comprehensive in its domain coverage. Even so, the prospects for applying CBR techniques in NLP are open-ended, and deserve further attention. This preliminary attempt to bring CBR into natural language processing is one of two original advances made during the course of our work on MUC-3.

The other significant advance was made very early on while we were assessing the robustness of the CIRCUS sentence analyzer and making some final adjustments to CIRCUS. We were generally concerned about scaling up with respect to complex syntax, and thinking about ways that CIRCUS might approach syntactically complex sentences in a principled manner. At that time we discovered a formalism for embedded clause analysis, Lexically Indexed Control Kernels (aka LICKs). LICKs describe syntactic and semantic interactions within CIRCUS as it interprets embedded clauses. This formalism makes it relatively easy to see how CIRCUS handles an embedded clause, and has made it possible for us to talk about this aspect of CIRCUS much more effectively. In fact, a paper was written during MUC-3 relating embedded clause analysis by CIRCUS to experimental results in psycholinguistics [7]. In that paper we

argue that CIRCUS provides a cognitively plausible approach to complex syntax.

2.6 Up Against the Wall: Are We There Yet?

The major limiting factor in our TST2 performance was time. We are confident that significant improvements in recall could be made if we had more time to do more knowledge engineering. We would also predict higher precision scores although our precision percentages have grown at a much slower rate than our recall percentages, based on a comparison of official test scores for TST1 and TST2.

We tend to think of our system in three major pieces: (1) the CIRCUS sentence analyzer, (2) rule- based consolidation, and (3) case-based consolidation. Because the CBR component is truly optional, the primary responsibilities fall on CIRCUS and rule-based consolidation. We know that both of these components make mistakes, but we have not been able to separate them well enough to say which one is the weaker one. As with all knowledge-based systems, an assessment of these components is also confounded by the fact that we are working with incomplete knowledge bases. Both the dictionary and the consolidation rule base incorporate domain knowledge, and we have thus far analyzed less than 50% of the MUC-3 development corpus in our knowledge engineering efforts.

As one might expect, our best internal test runs are those that include texts we have analyzed for the purposes of constructing our dictionary and consolidation rules. For example, on May 13 we ran the TST2 version of our system on TST1, and posted recall-precision scores of 66-68 running with CBR, and 62-73 running without CBR (for MATCHED/MISSING). It is heartening to contrast this with our phase 1 test results for TST1 which were 28-59 (no CBR component was available for phase 1 testing). Roughly 20% of the TST1 texts were analyzed between February and May, so the substantial improvement in both recall and precision on TST1 can be only partially attributed to knowledge engineering based on the TST1 texts. A complete analysis of all the TST1 texts would provide us with a better estimate of a performance ceiling that is not confounded by inadequate knowledge engineering.

As far as future system development goes, we cannot conclude at this time that any one of our system components requires redesign or major alterations. We would like to exploit more of the corpus for the sake of knowledge engineering to get a better sense of what we can do when incomplete knowledge is not a factor. Only then can we hope to isolate limitations that need to be addressed by innovations in system design.

One limitation that applies more to our system development than our system itself, is the hand-coding of dictionary definitions and consolidation rules. It would be highly advantageous for us to automate certain aspects of this or at least design an intelligent interface to speed the work of our knowledge engineers. We did manage to use the CBR component as a tool to direct us to useful texts during dictionary construction, and this application of the CBR component was both very welcome and very effective (albeit rather late in the MUC-3 timetable). In any event, we would clearly benefit from intelligent interfaces or more ambitious machine learning

strategies to help facilitate the knowledge engineering effort that is so central to our whole approach.

To sum up, we are confident that the performance of CIRCUS has not yet reached its limit. Unfortunately, it is not possible to say anything about where our ultimate upper bound lies. We hope to pursue this question by participating in future performance evaluations.

2.7 Conclusions

As we explained at the beginning of this paper, CIRCUS was originally designed to investigate the integration of connectionist and symbolic techniques for natural language processing. The original connectionist mechanisms in CIRCUS operated to manage bottom-up slot insertion for information found in unexpected (i.e. unpredicted) prepositional phrases. Yet when our task orientation is selective concept extraction, the information we are trying to isolate is strongly predicted, and therefore unlikely to surface in a bottom-up fashion. For MUC-3, we discovered that bottom-up slot insertion was needed primarily to handle only dates and locations: virtually all other relevant information was managed in a predictive fashion. Because dates and locations are relatively easy to recognize, any number of techniques could be successfully employed to handle bottom-up slot insertion for MUC-3. Although we used the numeric relaxation technique described in [1] to handle dates and locations, we consider this mechanism to be excessively powerful for the task at hand, and it could readily be eliminated for efficiency reasons in a practical implementation.

Although our score reports for TST2 indicate that our system is operating at the leading edge of overall performance for all MUC-3 systems, we nevertheless acknowledge that there are difficulties with our approach in terms of system development. It would take us a lot of hard work (again) to scale up to this same level of performance in a completely new domain. New and inexperienced technical personnel would probably require about 6 months of training before they would be prepared to attempt a technology transfer to a new domain. At that point we would estimate that another 1.5 person/years of effort would be needed to duplicate our current levels of performance in a new domain. Although these investments are not prohibitive, we believe there is room for improvement in the ways that we are engineering our dictionary entries and rule-based consolidation components. We need to investigate strategies for deducing linguistic regularities from texts and explore available resources that might leverage our syntactic analysis. Similar steps should be taken with respect to semantic analysis although we are much more skeptical about the prospects for sharable resources in this problem area.

Although we have had very little time to experiment with the CBR consolidation component, the CBR approach is very exciting in terms of system development possibilities. While the rule-based consolidation component had to be crafted and adjusted by hand, the case base for the CBR component was generated automatically and required virtually no knowledge of the domain or CIRCUS per se. In fact, our CBR module can be transported with minor modification to any other MUC-3 system that generates case frame meaning representations for sentences. As a discourse analysis component, this module is truly generic and could be moved into a new

domain with simple adjustments. The labor needed to make the CBR component operational is the labor needed to create a development corpus of texts with associated target template encodings (assuming a working sentence analyzer is already in place). It is much easier to train people to generate target templates for texts than it is to train computer programmers in the foundations of artificial intelligence and the design of large rule bases. And the amount of time needed to generate a corpus from scratch is only a fraction of the time needed to scale up a complicated rule base. So the advantages of CBR components for discourse analysis are enticing to say the least. But much work needs to be done before we can determine the functionality of this technology as a strategy for natural language processing.

Having survived the MUC-3 experience, we can say that we have learned a lot about CIRCUS, the complexity of discourse analysis, and the viability of selective concept extraction as a technique for sophisticated text analysis. We are encouraged by our success, and we are now optimally positioned to explore exciting new research areas. Although our participation in MUC-3 has been a thoroughly positive experience, we recognize the need to balance intensive development efforts of this type against the somewhat riskier explorations of basic research. We would not expect to benefit so dramatically from another intensive performance evaluation if we couldn't take some time to first digest the lessons we have learned from MUC-3. Performance evaluations can operate as an effective stimulus for research, but only if they are allowed to accompany rather than dominate our principal research activities.

Chapter 3

Sentence Analysis

3.1 Preprocessing in CIRCUS

Before CIRCUS can process an input text it must be transformed into a form which is more palatable to LISP. This is done by the preprocessing component of CIRCUS. In addition to transforming the text into a list of strings to be processed, the preprocessor also performs a number of tasks in service of parsing the sentences. These tasks include:

- Morphological analysis and definition generation.
- Translation of numbers and dates to a canonical form.
- Translation and definition of possessives.
- Phrasal substitution.
- Setting variables used by CIRCUS.

Once these tasks have been completed CIRCUS can parse the input text.

3.1.1 Dates

All references to dates are transformed into a canonical form MMM_DD_YY. Phrases like *today*, *yesterday*, and *10 days ago* are converted to absolute dates with the date of the news wire text as the base. So, if the wire date were June 1, 1990, yesterday would transform to MAY_31_90. References to dates which are more than three months previous to the wire date are considered to be irrelevant in the MUC-3 domain, and so are transformed to the constant old date DEC_31_80, which flags any incident which the date refers to as irrelevant. Once the date has been transformed a definition is created for the symbol.

3.1.2 Numbers

Numbers composed of digits are prefixed with "&&" and a definition is generated. This is done to allow the definitions to be recorded. Numbers which are words, such as *one*, are left unchanged.

3.1.3 Possessives

Words of the form *John's*, or *Munoz'*, are transformed into `@JOHN@S` and `@MUNOZ@`. A possessive definition is then generated. The at sign "@" is used to signal that the word is a possessive, this knowledge is used during the parse of the sentence. As with the numbers, this is used to create acceptable symbols for the definitions.

3.1.4 Punctuation

Punctuation marks are converted to LISP atoms, which are predefined in the punctuation lexicon. Brackets '[' and ']' are used as a surface cue for identifying irrelevant text. Bracketed phrases are removed during preprocessing.

3.1.5 Morphological Analysis and Definition Generation

Each word of the input text is checked for a definition after the above transformations have been completed. If the word does not have a definition, one is generated for it. This is done in one of three ways. If the word has an identifiable root which has a definition, such as a regular plural noun, a definition is generated from that root. The same is true for the verb forms of a base verb, such as *cook*, where *cooked*, *cooks*, and *cooking* would all be derived from the definition of *cook*. Gerund and adjectival forms are also derived in this fashion. If the word does not have a defined root then a default definition is generated. The default is to use noun for the part of speech, and `ws-proper-name` for the semantic features. The third alternative is to have a definition entered interactively via the `auto-lex` function.

3.1.6 Phrasal Substitution

A number of phrases other than dates are transformed during preprocessing. These are predominately phrases concerning locations. The remainder are domain specific phrases which have been identified as items which could be eliminated without altering the understanding of the text.

Part of Speech	Number
Special	32
Pronoun	18
Noun	2614
Triggering Noun	132
Special Noun	3
Noun-Org	182
Adjective	1089
Special Adjective	8
Adverb	237
Special Adverb	15
Determiner	6
Preposition	44
Special Preposition	3
Auxiliary	10
Copular	8
Verb	1517
Special Verb	11
Gerund	40

Figure 3.1: MUC-3 Lexicon by Part of Speech

3.2 Dictionary Construction

3.2.1 Word Definition for MUC-3

The MUC-3 project required the engineering of a large lexicon, which included both general and domain specific lexical items. A total of 5971 separate words were defined, with 5407 unique (that is to say 564 words had more than one part of speech assigned). Figure 3.1 shows the breakdown of the MUC-3 lexicon by part of speech.

What is a Word?

When CIRCUS parses a sentence, it processes a `list` of `atoms` which are the words and punctuation of the text. It is necessary that a *word* have a *definition* in order for CIRCUS to parse the sentence. Words which are undefined are assigned a default definition during preprocessing.

Phrase Recognition

CIRCUS recognizes 312 phrases, 63 of which involve transformations from one set of words to another (possibly empty) set of words. This is used primarily for standardizing time and date references. The remainder involve transformations of a group of words to a single unit which is composed of the group of words connected by underscores. For example:

burned to death ⇒ burned_to_death

where the phrase burned_to_death is defined as a word. Most of the phrases are used for recognizing the names of the organizations in the MUC-3 domain.

What is a Definition?

Part of Speech

Every part of speech carries with it a set of syntactic expectations. CIRCUS uses 12 basic definitions for these expectations, each defined for a single part of speech. These are sufficient for 99% of the MUC-3 corpus, with the remaining *special* words discussed in Section 2.3.1.

adjective Words like *big, bigger, biggest.*

adverb Words like *slowly, quickly.*

auxiliary The helping verbs, such as *have.*

copular A refinement of auxiliary, helping forms of *to be.*[1]

determiner Words like *the, a, an.*

gerund The "ing" words, the present participle.

noun Things.

noun-org Things which are also organizations (domain specific).

triggering-noun Things which also trigger concept nodes (domain specific).

preposition Words like *of, by, to.*

verb Actions.

unknown None of the above.

[1] Although there is a fascinating explanation for the genesis of this category name, the margin is much too small to contain it.

```
(((assign *part-of-speech* 'preposition)
  (next-packet
    ((test (equal *part-of-speech* 'gerund)))
    ((test (equal *part-of-speech* 'noun-phrase))
     (assign *part-of-speech* 'prep-phrase
             *last-pp* *pp*
             *cd-form* (make-pp 'to)
             *saved-word* nil
             *PP* *cd-form*))
    ((test (equal *part-of-speech* 'verb))
     (assign *aux* nil))
    ((test (equal *part-of-speech* 'aux))))))
```

Figure 3.2: Special McEli Definition for *to*

Morphological Analysis

The task of building the lexicon for MUC-3 has been aided by the use of a morphological analysis component, which identifies the root of a word, and returns the root's definition. This allows there to be a single definition for a regular verb, the infinitive. Each of the tenses can be derived from the root. For example from the verb *rain* the following definitions can all be derived:

- rains
- rained
- raining

The same methods allow regular plurals to be derived, such as *babies* from *baby*.

Derived definitions carry the word-senses of the root, and in some cases the concept-nodes. An exception to this is the derivations which produce gerunds.

3.2.2 Word Recognition

Special Words

In the MUC-3 lexicon 72 words required a McEli definition be engineered specifically for them. Figure 3.2 shows the definition for the preposition *to*, which can be compared with the generic definition in Figure 3.3. These were needed to service the individual expectations of those words.

Special words are identified through direct observation of the parser output.

```
(((assign  *saved-word* *word*
           *noun-group* nil
           *noun-group-cns* nil
           *predicates* nil
           *entire-noun-group* nil
           *determiners* nil
           *part-of-speech* 'preposition)
  (next-packet
    ((test (equal *part-of-speech* 'noun-phrase))
      (assign *part-of-speech* 'prep-phrase
              *last-PP* *PP*
              *cd-form* (make-pp *saved-word*)
              *saved-word* nil
              *PP* *cd-form*))
    ((test (member *part-of-speech* '(preposition prep-phrase verb)))))))
```

Figure 3.3: Generic McEli Definition for a Preposition

```
(D-WORD RIOT
   :SYNTACTIC-TYPE noun
   :WORD-SENSES (ws-entity))
```

Figure 3.4: Simple Definition

Mundane Words

Over 90% of the MUC-3 lexicon consists of mundane words, which are defined solely for their part of speech and word senses.

3.2.3 AUTO-LEX: A Tool for Building Dictionaries

Defining words during the preprocessing of the text has been automated, making it easier to identify words which need to be added to the lexicon, and to enter their definitions. This has been done via the function AUTO-LEX. The simple type of definitions (Figure 3.4), ones without *syntactic-expectations* or *cn-defs*, are the ones which can be created during preprocessing of the text. Skipping a word produces the default definition of syntactic-type **noun** with word-senses **ws-proper-name**.

It may seem that the power of this tool would be limited, because of the kind of words which could be defined, but recall that this kind constitutes almost 94% of the MUC-3 lexicon.

AUTO-LEX originally provided a sequence of menus, allowing the user to enter a definition for an undefined word through the following steps:

```
(D-WORD DEAD
   :SYNTACTIC-TYPE SPECIAL-ADJECTIVE
   :SYNTACTIC-EXPECTATIONS
   (((assign *np-flag* t
             *predicates* (append *predicates* (list *word*))
             *part-of-speech* 'adjective
             *cd-form* (make-special *word*)
             *global-cn* *cd-form*)
     (next-packet
        ((test (eq *part-of-speech* 'noun)))
        ((test (eq *part-of-speech* 'adjective))))))
   :WORD-SENSES (dead1)
   :CN-DEFS ($LEFT-DEAD$ $FOUND-DEAD$ $FOUND-DEAD-PASS$))
```

Figure 3.5: Complex Definition

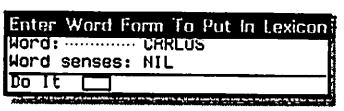

Figure 3.6: Original Auto Lex Definition Menu

1. The first menu (Figure 3.6) displayed the word to be defined in an edit field, which allowed the morphological root to be entered and the *word-senses* slot, for assigning the semantic features.

2. The second menu presented the choices for the *part-of-speech*. Also present on this menu was the option **skip** which caused AUTO-LEX to use the default definition.

While this did speed up the writing of the definitions, relative to the time it would take to do them completely by hand, there were some problems to be addressed. The context in which the word appeared was not shown, so it was necessary to find the location in the text manually in the editor. This was necessary for disambiguating the parts of speech which the word could take. This approach also required that the user know the complete spelling of all of the word-senses, including words like *ws-diplomatic-office-or-residence*, with 66 total different word senses to be concerned with.

To address these issues, and to provide a friendlier interface, AUTO-LEX was modified, with the objective of allowing a maximum return on the time spent on defining new words. The great majority of the words in the MUC-3 lexicon have definitions which are simply composed of a part of speech and a list of word senses. Only nouns, and a few adjectives, have any word senses in the list, and of those, most are of type entity. AUTO-LEX was modified to reflect these properties of the domain.

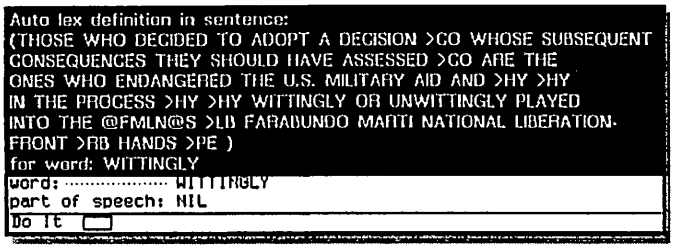

Figure 3.7: Auto Lex Definition Menu

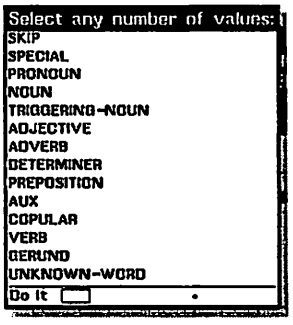

Figure 3.8: Part of Speech Selection Menu

The first issue to be addressed was that of context. The format of the function was altered to display the undefined word and the sentence in which it occurred without any menu. The user would then be prompted to choose whether to skip the word entirely, using the default definition, or to go ahead and use the menus to enter a definition. This reduced the number of times a menu had to be presented by at least half, and solved the problem of not being able to skip a definition until the second menu. The MUC-3 domain contains over 1000 proper names, and they were the most common type of word to be encountered when using AUTO-LEX.

Having passed the proper names, AUTO-LEX is then ready to enter a new definition. The process requires three menus; the edit menu (Figure 3.7), the part of speech menu (Figure 3.8), and the word sense menu (Figure 3.9). The edit menu displays the sentence in its title bar, eliminating the need for most accesses to the editor. The procedure is as follows:

- Enter the morphological root in the word field, editing the original word form the sentence.
- Select the part of speech field, which presents the menu of all of the possible parts of speech. One or more is selected.
- Complete selections and exit the menu. If the part of speech selected is noun, triggering noun, or adjective, then the word sense menu is presented, allowing selection of one or more word senses, with a default value of **ws-entity**.

With these menus it became possible to select the most specific appropriate word sense, using the documentation of the word senses menu to guide the user up and down the hierarchy. Having the sentence displayed at the top of the menu was generally sufficient for deciding the appropriate part of speech and word sense for each word.

Dictionary construction is a formidable task, especially for a domain as large as MUC-3. AUTO-LEX provides the capability to take most of the work out of defining the large percentage of relatively unimportant (those defined just for their part of speech and word sense) words. This allows the time to be spent on handcrafting the definitions for the special words, which don't conform to the generic syntactic expectations patterns, and the concept nodes themselves.

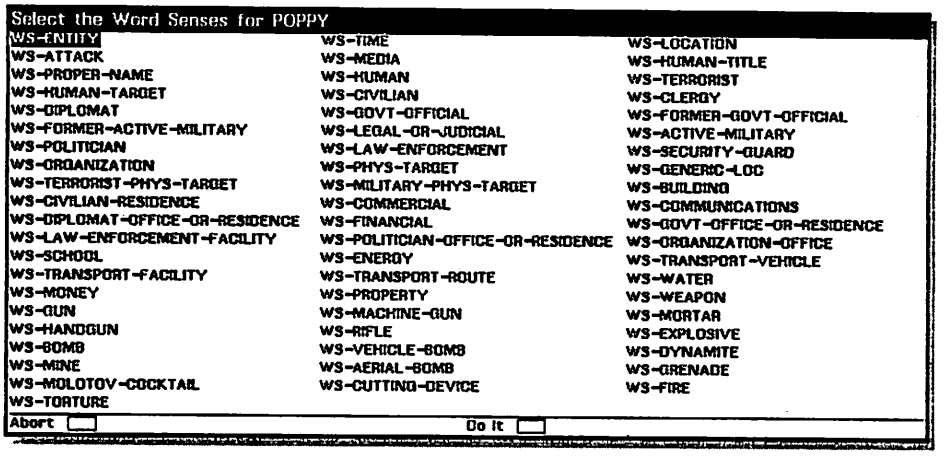

Figure 3.9: Word Senses Selection Menu

3.3 Concept Node Definitions

This section first describes the differences between the original CIRCUS concept node definitions and those used for MUC. We then discuss the five major classes of concept nodes. Each type is discussed in more detail in a separate subsection. The section concludes with some general hints for defining new concept nodes. Throughout the section, we assume that the reader is familiar with the CIRCUS parser. For a basic introduction to CIRCUS, see [1].

3.3.1 Differences from CIRCUS Concept Nodes

Concept node definitions for MUC take on a slightly different form than the original CIRCUS concept nodes. The difference is in the syntax and semantics of the soft slot constraints. Instead of the single Lisp predicates originally used as soft constraints (e.g., fn1, fn2, fn3a, etc.), the soft constraints for each variable slot in a MUC concept node definition consist of a list of references to the semantic class hierarchy.[2] (The semantic feature hierarchy is listed in [nlp.muc.lexicons]class-hierarchy.)

```
(define-word $KIDNAP-2$
   (CONCEPT-NODE
     ':NAME '$KIDNAP-2$
     ':TIME-LIMIT 10
     ':SLOT-CONSTRAINTS '(((class organization *PP*)
```

[2]Although none of the MUC concept nodes use generic lisp functions as soft constraint predicates,there is still a mechanism for employing this type of soft constraint. See Tony's (Tony Reish) functions TSC-2 and TSC-FUNC in [nlp.muc.circus]abstr.lisp.

```
                        (class terrorist *PP*)
                        (class proper-name *PP*)
                        (class human *PP*))
                       ((class human *PP*)
                        (class proper-name *PP*)))
':VARIABLE-SLOTS '(ACTOR (*PP* (is-prep? '(by)))
                   VICTIM (*PP* (is-prep? '(of))))
':CONSTANT-SLOTS '(TYPE KIDNAPPING)
':ENABLED-BY '((active-or-passive)
               (*S*-is-kidnapping?))
))
```

In the example above, there are two references to the semantic hierarchy in the soft constraints for the VICTIM slot. The *(class human *PP*)* constraint tests that at least one of the semantic features associated with the head noun of *PP* has ws-human as an ancestor. The *(class proper-name *PP*)* constraint for the VICTIM slot tests that at least one of the semantic features associated with the head noun of *PP* has ws-proper-name as an ancestor.

The semantics of the soft slot constraints has also been modified for the MUC system. A filler satisfies the soft constraints for a slot if **any** predicate in the slot constraint list succeeds. A prepositional phrase would satisfy the soft constraints of the VICTIM slot, for example, if its head noun was either of class human or class proper-name, or was any descendant of these classes.

3.3.2 Concept Node Classes

MUC concept node definitions can be categorized into the following implicit taxonomy of concept node types:

1. verb-triggered
   (a) active
   (b) passive
   (c) active-or-passive
2. noun-triggered
3. adjective-triggered
4. gerund-triggered
5. threat and attempt concept nodes

I will briefly discuss each type below, but it should be noted that the taxonomy represents generic classes of concept node types. Most of the concept nodes created for MUC began as

one of these generic types, but evolved over time into a concept node that looks quite different from its original incarnation.

Active Verb-triggered Concept Nodes

Concept nodes triggered by verbs in active voice require very simple concept node definitions. These concept nodes are triggered by the presence of a specific verb and are enabled only when that verb is in the active voice. The concept node definition typically sets up predictions for finding the ACTOR in *S* and the VICTIM or PHYSICAL-TARGET in *DO*. Active verb-triggered concept node definitions exist for essentially all verbs important to the MUC domain, e.g., kidnap, kill, murder, bomb, detonate, massacre, etc. The $KIDNAP$ concept node below is an example of a concept node definition triggered by an active verb. It handles kidnappings for clauses like: "The ELN guerillas kidnapped Castellar." Both "kidnap" and "abduct" have $KIDNAP$ in the :cn-defs field of their McEli definition.

```
(define-word $KIDNAP$
   (CONCEPT-NODE
     ':NAME '$KIDNAP$
     ':TIME-LIMIT 10
     ':SLOT-CONSTRAINTS '(((class organization *S*)
                          (class terrorist *S*)
                          (class proper-name *S*)
                          (class human *S*))
                         ((class human *DO*)
                          (class proper-name *DO*)))
     ':VARIABLE-SLOTS '(ACTOR (*S* 1)
                        VICTIM (*DO* 1))
     ':CONSTANT-SLOTS '(TYPE KIDNAPPING)
     ':ENABLED-BY '((if (eq 0 (reduced-relative)) 1 0)
                    (active))))
```

This concept node definition checks that the Actor is an organization, terrorist, proper name, or human while the Victim is expected to be a human or proper name.

In addition, note the ENABLED-BY clause. It makes two tests before enabling the concept node. First, the parser shouldn't be in a reduced relative clause. Second, the verb should be in active voice. The function ACTIVE (in [nlp.muc.circus]mucsem.lisp) checks that the verb is in active voice and also makes a series of tests on the verb phrase before enabling the concept node. It tests, for example, that:

- the verb is in past tense (e.g., "kidnapped")
- any auxiliary preceding the verb is of the correct form (e.g., "had kidnapped" is fine, but "was kidnapped" indicates the passive voice and so ACTIVE should return 0)

- the verb is not in the infinitive form (e.g., "to kidnap")
- the verb is not preceded by "being"
- the sentence is not actually describing a threat or an attempt (these events trigger a different set of concept nodes)
- the negation mode buffer (i.e., *negation*) is nil (e.g., "did not kidnap" shouldn't trigger a kidnapping concept node)
- the future/subjunctive mode buffer (i.e., *future*) is nil (e.g., "will/would kidnap" shouldn't trigger a kidnapping concept node)

Each of these features was determined to be a generic requirement for enabling concept nodes triggered by active verbs. Any of the requirements can be overridden by a keyword argument. So, for example, to allow "to kidnap" to trigger $KIDNAP$, the call to ACTIVE in the ENABLED-BY clause should be changed from *(active)* to *(active :check-infinitive nil :check-past nil)*.

Passive Verb-triggered Concept Nodes

Almost every verb that has a concept node definition for its active form should also have a concept node definition for its passive voice form. The passive verb-triggered concept nodes are triggered by the presence of a specific verb and are enabled only when that verb is in the passive voice. (There is a function called PASSIVE that is the same as ACTIVE except that it ensures that the triggering verb and its auxiliaries are consistent with the passive voice.) These concept node definitions typically set up predictions for finding the ACTOR in a by-*PP* (i.e., *PP* is a prepositional phrase that begins with the preposition "by") and the VICTIM or PHYSICAL-TARGET in *S*. The $KILL-PASS-1$ concept node below is an example of a concept node definition triggered by a passive verb. It handles clauses like: "Castellar was killed by ELN guerillas." The verbs "murder," "execute," "kill," "massacre," "assassinate," "gun," "gunned_down," and "shot_to_death" all trigger this concept node because each has $KILL-PASS-1$ in the :cn-defs field of its McEli definition. It should be noted that $KILL-PASS-1$ is only enabled if *S* is not a synonym of "no one." This prohibits sentences like "No one was killed by the terrorists" from triggering a murder.

```
(define-word $KILL-PASS-1$
   (CONCEPT-NODE
     ':NAME '$KILL-PASS-1$
     ':TIME-LIMIT 10
     ':SLOT-CONSTRAINTS '(((class organization *PP*)
                          (class terrorist *PP*)
                          (class proper-name *PP*)
                          (class human *PP*))
```

```
                        ((class human *S*)
                        (class proper-name *S*)))
    ':VARIABLE-SLOTS '(ACTOR (*PP* (is-prep? '(by)))
                       VICTIM (*S* 1))
    ':CONSTANT-SLOTS '(TYPE MURDER)
    ':ENABLED-BY '((passive :check-s-no-one t))))
```

The passive concept nodes often look for slot fillers in more than one prepositional phrase. For example, both the victim and the instrument may be found in prepositional phrases: "Castellar was killed by ELN guerillas with with a knife." Because CIRCUS only has one *PP* buffer and because a concept node only freezes when all of its slots are filled (or at the end of a clause or when its time limit expires), we usually create separate concept nodes for predicting each distinct PP. For example, $KILL-PASS-2$ looks very much like $KILL-PASS-1$ above and is responsible for picking up the instrument.

```
(define-word $KILL-PASS-2$
   (CONCEPT-NODE
     ':NAME '$KILL-PASS-2$
     ':TIME-LIMIT 10
     ':SLOT-CONSTRAINTS '(((class human *S*)
                          (class proper-name *S*))
                         ((class weapon *PP*)))
     ':VARIABLE-SLOTS '(VICTIM (*S* 1)
                        INSTR (*PP* (is-prep? '(by with))))
     ':CONSTANT-SLOTS '(TYPE MURDER)
     ':ENABLED-BY '((passive :check-s-no-one t))))
```

Another way to handle the multiple PP problem is to create (at least) two concept nodes that are identical except for their TIME-LIMITs. For example, we might create a $KILL-PASS-1$ and $KILL-PASS-2$ that each contain the ACTOR, VICTIM, and INST slot but have time limits of, say, 2 and 10, respectively. In theory, $KILL-PASS-1$ would pick up the first PP and $KILL-PASS-2$ would pick up the second PP. In practice, however, using the time limit to control predictive prepositional phrase attachment is difficult and unreliable. Unfortunately, a limited number of concept nodes continue to make use of this solution. They should probably be changed.

Other Verb-triggered Concept Nodes

Occasionally, concept nodes can be triggered by verbs in either the active or passive voice. For instance, the following concept node definition handles sentences with verbs in active voice (e.g., "The kidnapping occurred...") and passive voice (e.g., "The kidnapping was carried_out ..."). Creating concept nodes that look for fillers for both classes of verbs is usually dangerous, however. It is better to create separate active and passive versions of the concept node.

```
;;; triggered off of verbs like OCCURRED, TOOK_PLACE, CARRIED_OUT
(define-word $KIDNAP-2$
   (CONCEPT-NODE
     ':NAME '$KIDNAP-2$
     ':TIME-LIMIT 10
     ':SLOT-CONSTRAINTS '(((class organization *PP*)
                           (class terrorist *PP*)
                           (class proper-name *PP*)
                           (class human *PP*))
  ((class human *PP*)
                           (class proper-name *PP*)))
     ':VARIABLE-SLOTS '(ACTOR (*PP* (is-prep? '(by)))
                        VICTIM (*PP* (is-prep? '(of))))
     ':CONSTANT-SLOTS '(TYPE KIDNAPPING)
     ':ENABLED-BY '(;;active for occurred, took_place
                    ;;passive for carried_out
                    (active-or-passive)
                    (*S*-is-kidnapping?))))
```

Noun-triggered Concept Nodes

Concept nodes triggered by nouns have definitions that predict slot fillers in prepositional phrases that follow the noun. The following $MURDER$ concept node definition is triggered by the nouns "massacre," "murder," "death," "murderer," "assassination,", "killing" (in its gerund form), and "burial." It looks for the Victim in an of-PP.

```
(define-word $MURDER$
   (CONCEPT-NODE
     ':NAME '$MURDER$
     ':TIME-LIMIT 2 ;;5 was too much (nosc42, #33)
                    ;;2 was too much for 1221 -emr
                    ;;needed 2 for 1243
     ':SLOT-CONSTRAINTS '(((class human *PP*)
                           (class proper-name *PP*)))
     ':VARIABLE-SLOTS   '(VICTIM (*PP* (pp-check '(of))))
     ':CONSTANT-SLOTS   '(TYPE MURDER)
     ':ENABLED-BY '((noun-triggered)
                    (not-threat-or-plot))))
```

Note the hard constraint associated with the Victim slot. We use PP-CHECK as a hard constraint to ensure that only prepositional phrases that begin with "of" and follow the triggering noun will satisfy the hard constraint. The alternative hard constraint predicate, IS-PREP?, does not check the position of the prepositional phrase with respect to the triggering noun.

As the comments next to TIME-LIMIT suggest, establishing the correct time limit in the noun-triggered concept node definitions is often difficult. A time limit of 10 is usually a good place to start. The function NOUN-TRIGGERED in the ENABLED-BY clause(in [nlp.muc.circus]mucsem.lisp) performs the same function for noun-triggered concept nodes that the ACTIVE and PASSIVE functions perform for verb-triggered concept nodes. It confirms that some generic conditions are satisfied before a noun can enable a concept node. A call to NOUN-TRIGGERED should be in the ENABLED-BY slot for all noun-triggered concept nodes. The $MURDER$ concept node definition shown above has an additional enabling condition that disables the concept node if the sentence describes a threat or a plot (e.g., "they planned the murder of Castellar"). This test should become part of the NOUN-TRIGGERED function if it turns out to be a generic condition for enabling all noun-triggered concept nodes.

As described in the last section, we usually create separate concept nodes to pick up slot fillers that require different prepositional phrases. As a result, there exists another noun-triggered MURDER concept node definition that looks very much like the $MURDER$ definition above but picks up the Actor in a by-PP.

An important note: Be sure to create McEli definitions that have *:SYNTACTIC-TYPE triggering-noun* instead of *:SYNTACTIC-TYPE noun* when defining nouns that trigger concept nodes, i.e., nouns that have concept node definitions associated with them.

Adjective-triggered Concept Nodes

It is sometimes necessary to trigger concept nodes off of adjectives. We do this when the verb is too general to make a good trigger, e.g., "Castellar was found dead." Intuitively, we don't want "found" to trigger a "death" concept node because it shows up in too many other contexts. A "death" concept node triggered by "found" would pop up too often unless we created a very specific enabling clause to disable the concept node whenever "found" wasn't eventually followed by "dead."

In the case of "found dead," it is easier to trigger a FOUND-DEAD concept node off of the adjective and then check for the presence of specific verbs (in the ENABLED-BY clause) using the APPROPRIATE-VERB? or EXACT-VERB? functions. The following concept node definition was created for sentences like "The officials found the mayor dead when they arrived."

```
;;triggered by dead.
(define-word $FOUND-DEAD$
   (CONCEPT-NODE
     ':NAME '$FOUND-DEAD$
     ':TIME-LIMIT 2
     ':SLOT-CONSTRAINTS '(((class human *DO*)
                          (class proper-name *DO*)))
     ':VARIABLE-SLOTS '(VICTIM (*DO* 1))
     ':CONSTANT-SLOTS '(TYPE FOUND-DEAD)
     ':ENABLED-BY '((active)
```

```
                    (appropriate-verb? *V* '(found)))
    ))
```

We also trigger concept nodes off of adjectives when looking for very specific lexical items in the vicinity of the adjective. For phrases like "20 dead" and "50 wounded", for example, the number preceding the adjective should fill the victim slot in either a MURDER or INJURY concept node. Unfortunately, numbers can be either nouns or adjectives and adjectival phrases like "20 dead" and "50 wounded" can show up in any (or none) of the global syntactic buffers. As a result, defining a concept node that looks for information from the adjectival phrase in specific constituent buffers is very difficult. To solve this problem, we use special McEli definitions for some triggering adjectives (e.g., "dead"). These McEli definitions store the number that precedes the adjective in a special *NUM* buffer. We then create specialized concept nodes that are triggered by the adjective and predict slot fillers in the *NUM* buffer. An example of such a concept node definition is the $NUMBER-DEAD$ definition below. The enabling conditions indicate that the concept node is only enabled if a number precedes the adjective.

```
;; "...NUM DEAD..."
(define-word $NUMBER-DEAD$
   (CONCEPT-NODE
     ':NAME '$NUMBER-DEAD$
     ':TIME-LIMIT 1
     ':SLOT-CONSTRAINTS '(((class entity *NUM*)))
     ':VARIABLE-SLOTS '(VICTIM (*num* 1))
     ':CONSTANT-SLOTS '(TYPE MURDER)
     ':ENABLED-BY '((if (get-number (format nil "~A" (prev-word)))
                        1 0))))
```

Gerund-triggered Concept Nodes

Some concept nodes are designed specifically for picking up the noun phrases that follow gerunds.[3] We originally allowed gerunds to trigger the same concept nodes as the associated verb form, but the verb-triggered concept nodes turned out to be too general for the very specific processing required by gerunds. For example, the gerund-triggered concept nodes often have no enabling clause because the fact that the triggering word was parsed as a gerund is itself a reliable enabling condition. In addition, the object of the gerund is the noun phrase that follows it, but the actor cannot reliably be found in the *S* buffer as it is for active verb-triggered concept nodes.

As a result, we create special concept nodes for important gerunds like "killing," "destroying," "damaging," etc. The McEli definition for gerunds stores the noun phrase that follows the

[3]Gerunds are verb forms that act as nouns: e.g., The army was accused of *destroying* houses and *damaging* buildings.

gerund in *DO* and the concept node definition looks for its slot filler in that buffer. The following concept node definition handles phrases like "...destroying 13 business establishments...".

```
(define-word $destroy-gerund$
   (CONCEPT-NODE
     ':NAME '$destroy-gerund$
     ':TIME-LIMIT 2
     ':SLOT-CONSTRAINTS '(((class human *DO*)
                          (class proper-name *DO*)))
     ':VARIABLE-SLOTS '(VICTIM (*DO* (*do*-not-before-cn?)))
     ':CONSTANT-SLOTS '(TYPE destruction
                        EFFECT destroyed)
     ':ENABLED-BY '()))
```

Note: always use *DO*-NOT-BEFORE-CN? as a hard constraint in the gerund-triggered concept nodes to ensure that *DO* fills a slot in the case frame **only** if it follows the gerund, i.e., we don't want direct objects in existence prior to the gerund to fill slots in the gerund-triggered concept node.

Threat and Attempt Concept Nodes

THREAT and ATTEMPT concept nodes are often difficult to design because they require enabling conditions that check for both a specific event (e.g., murder, attack, kidnapping, etc.) and for indications that the event is a threat or attempt. One way to design these concept node definitions is to allow the noun or verb associated with the main event type to trigger the concept node and then create enabling conditions that look for mention of "attempts" or "threats" in an earlier clause. The following concept node, for example, handles sentences like "The terrorists intended to storm the embassy." It is triggered by the verbs "attack", "intercept", "derail", and "storm".

```
(define-word $attempted-attack-2$
   (CONCEPT-NODE
     ':NAME '$attempted-attack-2$
     ':TIME-LIMIT 5
     ':SLOT-CONSTRAINTS '(((class organization *S*)
                          (class terrorist *S*)
                          (class proper-name *S*)
                          (class human *S*))
                         ((class human *DO*)
                          (class proper-name *DO*))
                         ((class phys-target *DO*)))
     ':VARIABLE-SLOTS '(ACTOR (*S* 1)
                        VICTIM (*DO* 1)
```

```
                    TARGET (*DO* 1))
  ':CONSTANT-SLOTS '(TYPE attempted-attack)
  ':ENABLED-BY '((active :check-threat nil
                         :check-infinitive nil
                         :check-past nil)
                 (probably-an-attempt?))))
```

This concept node definition uses keyword arguments to ACTIVE so that the infinitive form of the triggering verb will enable the concept node. The function PROBABLY-AN-ATTEMPT? (in [nlp.muc.circus]mucsem.lisp) checks that an "attempt" word precedes the triggering verb by less than or equal to 10 words. Clearly, this is a hack. We chose the number 10 in a case-based fashion. For threats, we set the window to 9. Because threats and attempts occur much less frequently than real events, this method of defining concept nodes was satisfactory.

A Note Concerning Some Injury and Damage Concept Nodes

The $INJURY-2$ and $NO-INJURY-1$ concept nodes are triggered by the same lexical items and are designed so that only one of them will succeed for each triggering word. Unfortunately, the enabling conditions that allow this interaction are antiquated (and very complicated). They were designed before there was a method for checking the adjectives that precede a triggering noun. The enabling conditions of these concept nodes could now be simplified by using the function NOT-IN-PREDICATES? (in [nlp.muc.circus]mucsem.lisp). The same is true for $DAMAGE-NOUN-1$, $DAMAGE-NOUN-2$, and $NO-DAMAGE-NOUN-1$.

3.3.3 Defining New Concept Nodes

There are 3 steps to defining a concept node for a new example:

1. Look for an existing concept node that picks up slots from the correct buffers and has enabling conditions that will be satisfied by the current example.
2. If one exists, you won't have to create a new concept node. Instead, add the name of the existing concept node to the :CN-DEFS portion of the McEli definition of the triggering word.
3. Otherwise, create a new concept node definition by modifying an existing one to handle the new example. (It is best if you start out using a general concept node so that it can be made more specialized as related examples appear in the text. Generalizing a concept node is a more dangerous task.)

3.4 Prepositional Phrase Attachment

The CIRCUS parser provides a network relaxation mechanism for bottom-up insertion of prepositional phrases into the semantic case frame representation. Although the mechanism is completely general, only a very restricted form of network relaxation was required for MUC. Our goal was to restrict the mechanism so that *any* prepositional phrase relevant to the MUC domain (mainly times and locations) would attach to *all* concept nodes that occur in the clause containing the prepositional phrase. CIRCUS' data-driven mechanism for prepositional phrase attachment was restricted as follows:

- Only nouns, verbs, and adjectives that carry concept node definitions can become attachment points in the network. (In the original CIRCUS system, any noun or verb can potentially serve as an attachment point.
- We only allow noun phrases and prepositional phrases whose head nouns are tagged with the *location* word sense, the *proper-name* word sense, or any of the *phys-target* word senses to become part of the relaxation network. All other noun and prepositional phrases are ignored by the network construction algorithm.
- We initialize all preposition nodes without consulting any memory model. If the preposition node connects a prepositional phrase to a lexical item that carries a concept node definition, we initialize the preposition node with a value of 6 . In all other cases, we initialize the preposition node with a value of 0.
- We replaced CIRCUS' MAKE-NETWORK function with a specialized MAKE-MUC-NETWORK. This function allows prepositional phrases to attach to *any* noun phrase or verb phrase in the current clause, even if the prepositional phrase occurs at the beginning of the clause. E.g., "on 25 Jan" will be attached to the murder concept node in "On 25 Jan, Castellar was murdered". The original MAKE-NETWORK would not have allowed forward attachment of prepositional phrases.
- Instead of running the relaxation algorithm as each constituent in a sentence is recognized, the relaxation algorithm is only run once — at the end of each clause. This restriction merely speeds up the parser without affecting the final attachment decisions.

3.4.1 Examples

The MUC system uses the relaxation algorithm for prepositional phrase attachment of times, locations, and physical targets. The following sections give some examples of each and list the prepositions responsible for the attachment.[4]

[4]In CIRCUS, only prepositions that carry word senses can become part of the relaxation network. To see the functions that define the prepositional phrase attachment constraints for each preposition in the MUC system, see the invocations of CREATE-CONSTRAINTS at the end of [nlp.muc.circus]mucsem.lisp.

Time Phrases

The prepositions ON and IN carry the *time* word sense and alert the data-driven slot insertion mechanism to attach the time phrase that follows to all concept nodes in the clause. As indicated in the following example, the slots inserted by this mechanism are labelled as REL-LINKs. Note that the preprocessor has converted the phrase "last week" to a specific range of seven days based on the date of the newswire:

```
MESA MENESES WAS ARRESTED ON MAR_29_90-APR_05_90 >CO LAST WEEK >CO
BY THE ADMINISTRATIVE DEPARTMENT OF SECURITY...

=============================================================

***          TYPE = PERPETRATOR
***          CONFIDENCE = SUSPECTED_OR_ACCUSED_BY_AUTHORITIES
***          NEW-INFO = T

***          PERPETRATOR = WS-PROPER-NAME
***          noun group = (MESA MENESES)
***          REL-LINK (TIME  (MAR_29_90-APR_05_90)))
```

Location Phrases

There are three forms for bottom-up pp-attachment of location phrases. The prepositions IN, NEAR, AT, and THROUGHOUT (defined with the *loc2* word sense) attach the location phrase that follows to all concept nodes in the clause as indicated in the following example:

```
A POWERFUL DYNAMITE_CHARGE ON DEC_20_89 >CO TODAY EXPLODED NEAR THE
AMERICAN EMBASSY IN LA_PAZ >PE

=============================================================

***          TYPE = WEAPON
***          INSTR = DYNAMITE

***          REL-LINK (TIME  (DEC_20_89)))
***          REL-LINK (LOC2  (LA_PAZ)))

=============================================================

***          TYPE = BOMBING
***          INSTR =
     >>>                  TYPE = WEAPON
```

```
     >>>                 INSTR = DYNAMITE
     >>>                 REL-LINK (TIME OBJECT (WS-TIME))
     >>>                 REL-LINK (LOC2 OBJECT (WS-LOCATION))

***          noun group = (DYNAMITE_CHARGE)
***          predicates = (POWERFUL)
***          determiners = (A)
***          TARGET = WS-DIPLOMAT-OFFICE-OR-RESIDENCE
***          noun group = (EMBASSY)
***          predicates = (AMERICAN)
***          determiners = (THE)
***          REL-LINK (TIME  (DEC_20_89)))
***          REL-LINK (LOC2  (LA_PAZ)))
```

The second form for bottom-up pp-attachment of location phrases is slightly more complicated. The preposition OF (defined with the *loc1* word sense) attaches the location phrase that follows to all concept nodes in the clause if the OF-pp was immediately preceded by an IN-pp or ON-pp. In addition, the attachment will only occur when the head noun of the IN-pp or ON-pp is not labelled with the *time*, *proper-name*, or *generic-loc* word senses. This pattern will attach the location "San Salvador" from phrases like "in the heart of San Salvador" but not from "in the department of San Salvador." The *loc2* word sense of IN is responsible for the attachment in the second case — it would attach the more specific "department of San Salvador" to all concept nodes in the clause.

```
... WHEN A CAR_BOMB EXPLODED ON OCT_31_89 >CO TODAY OUTSIDE THE
OFFICES OF THE SALVADORAN WORKERS NATIONAL UNION FEDERATION IN THE
HEART OF SAN_SALVADOR >CO THE POLICE REPORTED >PE

================================================================

***          TYPE = BOMBING
***          INSTR =
     >>>                 TYPE = WEAPON
     >>>                 INSTR = CAR_BOMB
     >>>                 REL-LINK (TIME OBJECT (WS-TIME))
     >>>                 REL-LINK (LOC1 OBJECT (WS-LOCATION))

***          noun group = (CAR_BOMB)
***          determiners = (A)
***          REL-LINK (TIME  (OCT_31_89)))
***          REL-LINK (LOC1  (SAN_SALVADOR)))
```

The third form for bottom-up pp-attachment of location phrases is responsible for insertion of location ranges, i.e., when an event occurs between two locations. The preposition BETWEEN

(defined with the *loc-betwixt* word sense) attaches the location phrase that follows (usually a conjunction of two or more noun phrases) to all concept nodes in the clause. There is a similar form for attaching time ranges, i.e., BETWEEN also carries the *time-betwixt* word sense. The following is an example of a *loc-betwixt* attachment:

```
CASTELLAR WAS KIDNAPPED SOMEWHERE BETWEEN ACHI AND SAN_SALVADOR >PE

==========================================================

***          TYPE = KIDNAPPING
***          VICTIM = WS-PROPER-NAME
***          noun group = (CASTELLAR)
***          REL-LINK (LOC-BETWIXT  ((ACHI) (SAN_SALVADOR))))
```

Physical Target Phrases

Finally, we sometimes use the bottom-up attachment mechanism to locate physical targets. The pseudo-prepositions IN_THE_BACK_OF and IN_FRONT_OF (defined with the *target* word sense) attach the physical target that follows to all concept nodes in the clause:

```
ALMOST SIMULTANEOUSLY ANOTHER BOMB EXPLODED IN_FRONT_OF A BANK IN
ANOTHER PART OF CARTAGENA...

==========================================================

***          TYPE = WEAPON
***          INSTR = BOMB
***          REL-LINK (TARGET  (BANK)))
***          REL-LINK (LOC1  (CARTAGENA)))

==========================================================

***          TYPE = ATTEMPTED-BOMBING
***          INSTR =
     >>>                  TYPE = WEAPON
     >>>                  INSTR = BOMB
     >>>                  REL-LINK (TARGET OBJECT (WS-FINANCIAL))
     >>>                  REL-LINK (LOC1 OBJECT (WS-LOCATION))

***          noun group = (BOMB)
***          predicates = (ANOTHER)
***          REL-LINK (TARGET  (BANK)))
***          REL-LINK (LOC1  (CARTAGENA)))
```

3.5 Embedded Clauses

The CIRCUS parser was designed to handle sentences with only one verb, e.g., John gave Mary a kiss. Unfortunately, the MUC corpus consists mainly of very long, information-packed sentences. Consider the following typical example:

HOWEVER, ARMED FORCES SPOKESMAN COLONEL LUIS ARTURO ISAACS *SAID* THAT THE ATTACK, WHICH *RESULTED* IN THE DEATH OF A CIVILIAN WHO *WAS PASSING BY* AT THE TIME OF THE SKIRMISH, *WAS* NOT AGAINST THE FARM, AND THAT PRESIDENT CEREZO *IS* SAFE AND SOUND.

This sentence contains five separate clauses. Understanding each clause often requires that the parser carry constituents from one clause to the next. For example, "attack" is the subject of both "resulted" and "was," and "civilian" is the subject/actor of "was passing by." In other words, the embedded clause is missing a constituent (i.e., contains a gap) and the parser has to fill in the gap with a suitable antecedent from one of the previous clauses. At the very least, CIRCUS has to recognize when it has entered a new clause and reinitialize its syntactic constituent buffers. In the above sentence, for example, the third clause (with the verb phrase "was passing by") contains a gap in the subject position that should be filled by the phrase "a civilian" (from the previous clause). Because approximately 75% of the sentences in the corpus contain more than one verb phrase, it became clear that CIRCUS would require some mechanism to systematically handle sentences with multiple clauses.

To solve this problem, we define a small number of lexically-indexed control kernels (LICKs) for processing embedded clause constructions and allow individual words to selectively trigger the LICK that will correctly handle the current clause. Each LICK is essentially a new parsing environment. When we come to a subordinate clause, the top-level parsing environment (i.e., LICK) creates a new parsing environment that takes over to process the interior clause. In other words, when a subordinate clause is first encountered, the parent LICK spawns a child LICK, passes control over to the child, and later recovers control from the child when the subordinate clause is completed.

For a full explanation of LICKs, see [7]. However, we would like to emphasize that the MUC implementation of LICKs **only simulates** the LICK processing described in [7]. Instead of using lexical closures to create a new parsing environment, we clear the McEli stack and reset the syntactic constituent buffers. In addition, we do not incorporate the semantic representation of the relative clause into the semantic representation of the entire sentence. Instead, we output the case frames produced by each clause separately. The rule-based consolidation module is then responsible for merging the concept nodes from each clause in the story into the correct MUC template representation.

Consider the following sentence:

RICCARDO ALFONSO CASTELLAR, WHO WAS KIDNAPPED LAST WEEK BY ELN GUERRILAS, WAS FOUND TODAY NEAR ACHI.

CIRCUS begins parsing the sentence in the top level control kernel or environment. When it reaches the word "who", however, its lexicon entry indicates that processing of the main clause should be temporarily suspended and a child LICK spawned to process the relative clause. The lexicon entry for "who" is also responsible for initializing all syntactic constituent buffers in the new parsing environment. Because "Riccardo Alfonso Castellar" is the antecedent of "who" and because the antecedent of "who" always fills the subject position in the subordinate clause[5], "who" initializes the child LICK *S* buffer with "Riccardo Alfonso Castellar"[6] and initializes all other constituent buffers in the child to NIL. (See the annotated trace below.)

Because the MUC implementation of LICKs only simulates the lexical closure implementation required for real LICK processing, a special *OLD-S* buffer saves the subject of the main clause (i.e., "castellar") so that the main clause subject can be reinstated after processing the wh-phrase. After initializing the child LICK buffers, the lexicon entry for "who" clears the McEli stack.[7] Again, this explicit clearing of the stack is are necessary because we only simulating the LICK mechanism in the MUC system. Finally, the next-packet of the "who" definition initializes the McEli stack expectations for the embedded clause.[8] At this point, CIRCUS begins parsing the embedded clause.

The following is an annotated trace of the LICK processing for the embedded clause in the above sentence:

```
(RICCARDO ALFONSO CASTELLAR >CO WHO WAS KIDNAPPED ON
DEC_26_89-JAN_02_90 >CO LAST WEEK >CO BY ELN GUERRILAS >CO WAS FOUND ON
JAN_09_90 >CO TODAY NEAR ACHI >PE)

Processing *START* at position 0

Processing RICCARDO at position 4
Processing ALFONSO at position 5
Processing CASTELLAR at position 6

    *S* = CASTELLAR

Processing >CO at position 7
Processing WHO at position 8
    *FROZEN-CNS* = NIL
```

[5]In the general case, the antecedent of "who" can fill the subject, object, or prepositional phrase positions in the subordinate clause. For MUC texts, however, the antecedent of "who" always becomes the subject of the embedded clause.

[6]Actually, the simulation just allows Castellar to remain in *S* for the child LICK.

[7]This happens as a side effect of setting the *clear-stack* variable.

[8]Usually, the next-packet for LICK triggers like "whom", "that", "which", etc., predict a subject-verb to follow. For "who", however, no next-packet was required since the subject of the embedded clause always acts as the gap for sentences in the MUC corpus.

```
    *PART-OF-SPEECH* = RELATIVE-PRONOUN
    *OLD-S* = CASTELLAR    =====> save the current subject
                      =====> initialize all syntactic buffers
                      =====> note that *S* is not overwritten
                      =====>  and, hence, still contains CASTELLAR
    *PREDICATES* = NIL
    *V* = NIL
    *IO* = NIL
    *DO* = NIL
    *PP* = NIL
    *AUX* = NIL
    *CP* = NIL
    *NEGATION* = NIL
    *CD-FORM* = NIL
    *GLOBAL-CN* = NIL
    *NETWORK-HISTORY* = NIL
    *NETWORK-FOOD* = NIL
    *FUTURE* = NIL
    *CLAUSE* = 2
    *APPOSITIVE* = NIL
    *CLEAR-STACK* = T =====> clear the McEli stack
                      =====> continue parsing the embedded clause
Processing WAS at position 9
Processing KIDNAPPED at position 10

enabling concept $KIDNAP-PASS$


***          activation level = 1/2          ***

***          TYPE = KIDNAPPING
***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)

Processing BY at position 8
Processing ELN at position 9
Processing GUERRILAS at position 10

  ...
```

CIRCUS realizes that it has reached the end of the embedded clause when it sees the verb phrase "was found." At this point, CIRCUS exits the child LICK and returns to the main clause LICK. Because we only simulate the movement between parsing environments for MUC, returning to the parent LICK requires resetting the *S* buffer to the value saved in *OLD-S*

and initializing other syntactic buffers.

```
  ...

Processing WAS at position 11

***          activation level = 1          ***

***          TYPE = KIDNAPPING
***          ACTOR = WS-TERRORIST
***          noun group = (ELN GUERRILAS)
***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)

Freezing Concept Frame:

***          TYPE = KIDNAPPING

***          ACTOR = WS-TERRORIST
***          noun group = (ELN GUERRILAS)
***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)

Processing FOUND at position 12
    *S* = CASTELLAR        =====> reset *S* to value in *OLD-S*
    *CD-FORM* = NIL
    *SAVED-CD-FORM* = NIL
    *PREDICATES* = NIL
    *OLD-S* = NIL
    *IO* = NIL
    *DO* = NIL
    *PP* = NIL
    *CP* = NIL
    *SAVED-WORD* = NIL
    *ENABLED-CONCEPT-NODES* = NIL
    *GLOBAL-CN* = NIL
    *CLEAR-STACK* = NIL
    *NEGATION* = NIL
    *NETWORK-HISTORY* = NIL
    *NETWORK-FOOD* = NIL
    *CLEAR-STACK* = T
    *CLAUSE* = 3
    *PART-OF-SPEECH* = VERB
    *NP-FLAG* = NIL
```

```
    *NOUN-GROUP* = NIL
    *PREDICATES* = NIL
    *ENTIRE-NOUN-GROUP* = NIL
    *DETERMINERS* = NIL
    *APPOSITIVE* = NIL
    *GERUND* = NIL
    *CD-FORM* = FOUND
    *V* = FOUND
    *DO* = NIL

Processing DEAD at position 13

enabling concept $FOUND-DEAD-PASS$

***          activation level = 1          ***

***          TYPE = FOUND-DEAD
***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)

Freezing Concept Frame:

***          TYPE = FOUND-DEAD

***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)


Processing ON at position 14
Processing JAN_09_90 at position 15
Processing >CO at position 16
Processing TODAY at position 17
Processing NEAR at position 18
Processing ACHI at position 19
Processing >PE at position 20

Parse completed. The final representation =

==========================================================================

***          TYPE = KIDNAPPING

***          ACTOR = WS-TERRORIST
***          noun group = (ELN GUERRILAS)
```

```
***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)

================================================================

***          TYPE = FOUND-DEAD

***          VICTIM = WS-PROPER-NAME
***          noun group = (RICCARDO ALFONSO CASTELLAR)
***          REL-LINK (TIME  (JAN_09_90)))

***          REL-LINK (LOC2  (ACHI)))

================================================================
NIL
```

3.5.1 Triggering LICKs

As shown in the previous example, LICKs can be triggered in two ways. First, the McEli definition of individual lexical items may indicate that the previous clause should be suspended and a child LICK spawned. The following words spawn LICKs using this method: *after, because_of, before, because, but, if, that, when, where, whether, which, while, who, whom, since, though,* and *although.* The McEli definitions of these words are responsible for clearing the stack, finding the antecedent of the triggering word (if there is one), initializing syntactic constituents in the child LICK, and initializing the McEli predictions to be in effect at the onset of the embedded clause. The following is an annotated example of the McEli definition for "but". (If the real LICK mechanism had been used, the McEli definition would simply contain a pointer to a "new-clause-LICK".)

```
(d-word but
:syntactic-type special
:syntactic-expectations
(((assign
    ;; freeze all current concept nodes and save them
    ;; in *frozen-cns*.

    *frozen-cns* (freeze-concept-nodes)

            ;; initialize constituents in child LICK --- note that
            ;; we carry *S* from the parent to *S* in the child.
    *part-of-speech* 'connective
    *predicates* '()
    *S* *S*
```

```
    *OLD-S* '()
    *V* '()
    *IO* '()
    *DO* '()
    *PP* '()
    *AUX* '()
    *CP* '()
    *negation* '()
    *cd-form* '()
    *global-cn* '()
    *network-history* '()
    *network-food* '()
    *future* '()
    *clause* (1+ *clause*)
    *appositive* '()

    ;;clear the McEli stack
    *clear-stack* t)

          ;; set up stack in child LICK --- predict a subject
  (next-packet
    ((test
               (and (equal *part-of-speech* 'noun-phrase)
                    *cd-form*
                    (not (member 'ws-time
                                 (np-record-word-senses *cd-form*)))
                    (null *V*)))
     (assign *S* *cd-form*))))))
```

There is a second, more general mechanism for creating a child LICK. Every time a verb is recognized by the McEli component, we check to see whether it is the first verb found for the current clause. If it is, we continue parsing in the current LICK. However, if the current clause already has a verb (i.e., *V* is not null), we assume that we have entered a new clause and create a child LICK to process it. The LICK spawned at "was found" in the example above was triggered by this second method.

The function SECOND-VERB-OR-INFINITIVE? (in [nlp.muc.circus]changes.lisp) is responsible for triggering this second class of LICKs. SECOND-VERB-OR-INFINITIVE? is called in the TEST clause of the generic McEli VERB definition. See the VERB portion of GET-SYN-DEF below:

```
(((test (second-verb-or-infinitive?))
       (assign *part-of-speech* 'verb
       *np-flag* nil
```

```
      *noun-group* nil
      *predicates* nil
      *entire-noun-group* nil
      *determiners* nil
      *appositive* nil
      *gerund* nil
      *cd-form* (make-verb *word*)
      *V* *cd-form*
      *do* nil)
      (next-packet
 ((test (equal *part-of-speech* 'noun-phrase))
                (assign *DO* *cd-form*))
 ((test (equal *part-of-speech* 'conjunction))))))
```

SECOND-VERB-OR-INFINITIVE? is called purely for its side effects and always returns true. Depending on the state of the parser, it spawns one of several different child LICKs to handle the embedded clause. It can also opt to remain in the current LICK if the current verb is the first verb in the clause. Below is code for one of the cases in SECOND-VERB-OR-INFINITIVE?.[9] This case recognizes and handles infinitive complements.

```
((or (equal (prev-word) 'to)
     (and (equal (nth (- *pos* 2) *sentence*) 'to)
          ;;12-17-90 story 28?
          ;;(added *pos* clause 1-25-91 for #78)
          (equal *part-of-speech* 'adverb)))

 (freeze-concept-nodes)

 ;;set *S* for embedded clause
 (cond ((human-np? *s*)) ;carry over human *S*
                         ;  4-18-91 (ctc) #514
       ((np-record-p *cd-form*) (reassign '*S* '*cd-form*))
       ((and (pp-record-p *cd-form*)
             (np-record-p (pp-record-np *cd-form*)))
        (reassign '*S* '(pp-record-np *cd-form*)))
       (t nil))

 ;;clear all other buffers
 (if (get-verb-syn-node (nth (- *pos* 2) *sentence*))
```

[9]This function was originally designed to handle a small number of cases. Over time, however, it has become a bit unwieldy. It now contains some very specific code and should probably be reexamined/rewritten to handle more general cases as originally intended.

```
        (clear-all-buffers :exceptions '(*S* *AUX*))
        (clear-all-buffers :exceptions '(*S*)))

    ;;increment clause count
    (reassign '*clause* '(1+ *clause*)))
```

3.6 Appositives and Conjunctions

Appositives and conjunctions are handled in two phases. First, we rely on Tony Reish's code to produce *NP-HIST*, a semi-linked list of the np-records for all noun phrases that occur in a clause. In theory, *NP-HIST* uses apposition links to connect the np-records of all noun phrases that occur in apposition to one another. It also uses conjunction links to connect the np-records of all noun phrases that occur as part of a conjunction. The following example shows the portion of *NP-HIST* that represents the appositive in "The terrorists killed Castellar, the mayor of Achi." The np-record for "Castellar" is linked via its RELATED-NP slot to the np-record for "mayor". In addition, this relationship is labelled as an appositive, i.e., the RELATED-NP-TYPE slot contains the atom APP.

```
#1=#<NP-RECORD 45341071> is a structure of type NP-RECORD
   TIME-STAMP:                  1853
   NGP-VALUE:                   (CASTELLAR)
   WORD-SENSES:                 (WS-PROPER-NAME)
   CN-LIST:                     NIL
   HEAD-NOUN:                   CASTELLAR
    ...

   ENTIRE-NOUN-GROUP:           (CASTELLAR)
   RELATED-NP-TYPE:             APP
   RELATED-NP:
        #1=#S(NP-RECORD :TIME-STAMP 1854
                        :NGP-VALUE (MAYOR)
                        :WORD-SENSES (WS-GOVT-OFFICIAL)
                        :CN-LIST NIL
                        :HEAD-NOUN MAYOR
                         ...

                        :ENTIRE-NOUN-GROUP (MAYOR)
                        :RELATED-NP-TYPE NIL
                        :RELATED-NP NIL
                        :OF-WHAT
                          #2=#S(NP-RECORD :TIME-STAMP 1855
```

```
                                        :NGP-VALUE (ACHI)
                                        ...)
    ...)
```

In contrast, note the portion of *NP-HIST* below that represents the conjunction in "The terrorists killed Castellar, his son, and two officers." The np-record for "Castellar" is linked via its RELATED-NP slot to the np-record for "his son". The np-record for "his son" is, in turn, linked via its RELATED-NP slot to the np-record for "two officers". In the case of conjunctions, however, the RELATED-NP-TYPE slot in each np-record contains the atom CONJ noting that the three noun phrases should be combined as a conjunction.

```
#1=#<NP-RECORD 27767754> is a structure of type NP-RECORD
   TIME-STAMP:                  2861
   NGP-VALUE:                   (CASTELLAR)
   WORD-SENSES:                 (WS-PROPER-NAME)
   CN-LIST:                     NIL
   HEAD-NOUN:                   CASTELLAR
    ...

   ENTIRE-NOUN-GROUP:           (CASTELLAR)
   RELATED-NP-TYPE:             CONJ
   RELATED-NP:
        #S(NP-RECORD :TIME-STAMP 2862
                     :NGP-VALUE (SON)
                     :WORD-SENSES (WS-HUMAN)
                     :CN-LIST NIL
                     :HEAD-NOUN SON
                      ...

                     :ENTIRE-NOUN-GROUP (SON)
                     :RELATED-NP-TYPE CONJ
                     :RELATED-NP
                       #S(NP-RECORD :TIME-STAMP 2863
                                    :NGP-VALUE (OFFICERS)
                                    :WORD-SENSES (WS-ACTIVE-MILITARY)
                                    :CN-LIST NIL
                                    :HEAD-NOUN OFFICERS
                                     ...
                                    :ENTIRE-NOUN-GROUP (&&2 OFFICERS)
                                    :RELATED-NP-TYPE NIL
                                    :RELATED-NP NIL
                                     ...)
                      ...)
    ...)
```

In practice, understanding appositives and conjunctions and distinguishing them from one another requires more than the syntactic analysis represented in *NP-HIST*. Phase 2 of appositive and conjunction processing uses semantic knowledge as well as specific syntactic cues to modify the appositives and conjunctions hypothesized in *NP-HIST*. The function PROCESS-APPOSITIVES-AND-CONJUNCTIONS contains the code for this phase of processing. It is invoked by PARSE after an entire sentence has been processed.[10] Phase 2 first finds and processes the appositives in *NP-HIST* and then processes any conjunctions in *NP-HIST*. We will discuss the phase 2 processing for appositives and conjunctions in the following two subsections.

3.6.1 Appositives

We will use the terms NP1 and NP2 to refer to the first and second noun phrases in an appositive. For example, in the appositive "Castellar, the mayor of Achi," NP1 = Castellar and NP2 = the mayor. Phase 2 appositive processing performs four main actions:

1. Find the full form of NP2. (E.g., NP2 should be "the mayor of Achi" instead of "the mayor")
2. Insert NP2 into the np-record for NP1 and include a marker indicating its appositive class.
3. Update the word senses of the head noun of NP1 to include information from NP2. (E.g., "Castellar" becomes a ws-govt-official as well as a proper-name.)
4. Save the appositive in the global variable *all-appositives*.

We currently recognize 3 (semantic) classes of appositives:

titles "Castellar, the mayor..." (marked with >APP)

names "his son, Kirby,..." (marked with >NAME)

locations "in Lima, Peru..." (marked with >LOC)

The following example walks through phase 2 processing for a sentence containing a "title appositive". Processing for the other classes of appositive proceeds in a similar fashion.

THE TERRORISTS KILLED CASTELLAR, THE MAYOR OF ACHI.

As shown above, *NP-HIST* for this sentence indicates that the noun phrases "Castellar" (NP1) and "Mayor" (NP2) are in apposition to one another. The function PROCESS-APPOSITIVE (called by PROCESS-APPOSITIVES-AND-CONJUNCTIONS to handle appositives) first tries to match the appositive against one of the expected appositive patterns. Each appositive pattern

[10] In fact, we only invoke the code for sentences that generate at least one concept node.

describes characteristics of NP1 and NP2. If the appositive matches none of the patterns, then it is ignored. In this case, however, the appositive matches the pattern that expects NP1 to be a human or proper name and NP2 to be a noun phrase that is not a number and is not tagged with the ws-organization or ws-entity word senses. As a result, we update the word sense of Castellar from ws-proper-name to ws-govt-official and call EXTRACT-APPOSITIVE-TEXT to find the full form of NP2.[11] For this example, EXTRACT-APPOSITIVE-TEXT correctly returns the string "Mayor of Achi" as the full text form of NP2. Next, this full form of NP2 is inserted directly into the np-record for Castellar. It is marked with the >APP marker to indicate that this is appositive is a title.[12] As a side.effect, this changes the MURDER concept node from

```
==========================================================================

***          TYPE = MURDER

***          ACTOR = WS-TERRORIST
***          noun group = (TERRORISTS)
***          determiners = (THE)
***          VICTIM = WS-PROPER-NAME
***          noun group = (CASTELLAR)

==========================================================================
```

to

```
==========================================================================

***          TYPE = MURDER

***          ACTOR = WS-TERRORIST
***          noun group = (TERRORISTS)
***          determiners = (THE)
***          VICTIM = WS-PROPER-NAME
***          noun group = (CASTELLAR >APP MAYOR OF ACHI)

==========================================================================
```

Finally, PROCESS-APPOSITIVE saves the entire appositive in the global variable *ALL-APPOSITIVES*. For this example, we add (APP (CASTELLAR) (MAYOR OF ACHI)) to

[11] In theory, we just have to extract all of the text from NP2 until the next comma. In practice, we can't rely on commas to set off the appositive, so EXTRACT-APPOSITIVE-TEXT contains a set of heuristics for locating the end of the appositive.

[12] To be consistent, we should probably use a >TITLE marker, but originally this was the only kind of appositive in MUC; hence, the >APP marker.

ALL-APPOSITIVES. This variable stores all of the appositives recognized by the parser during the current newswire. Although this variable is currently unused, rule-based consolidation could use it to augment title and location information.

3.6.2 Conjunctions

The second phase of conjunction processing is very similar to Phase 2 for appositives. The function PROCESS-APPOSITIVES-AND-CONJUNCTIONS calls PROCESS-CONJUNCTION on each pair of noun phrases in *NP-HIST* connected via the RELATED-NP slot where the RELATED-NP-TYPE slot is CONJ. The system essentially works on each conjunction from right to left. For example, PROCESS-CONJUNCTION sees the hypothesized conjunction "Castellar, his son, and two officers" in two parts. First, it tries to join "his son" and "two officers". If that succeeds, it then tries to join "Castellar" and the conjunction "((his son) (two officers))". The output of CIRCUS for the sentence "The terrorists killed Castellar, his son, and two officers" would be:

```
==========================================================

***          TYPE = MURDER

***          ACTOR = WS-TERRORIST
***          noun group = (TERRORISTS)
***          determiners = (THE)
***          VICTIM = WS-PROPER-NAME
***          noun group = ((CASTELLAR) (SON) (OFFICERS))
***          predicates = (NIL NIL (&&2))
***          determiners = (NIL (HIS) NIL)

==========================================================
```

PROCESS-CONJUNCTION takes each pair of noun phrases, NP1 and NP2, and attempts to match them against conjunction patterns. Unlike the patterns in PROCESS-APPOSITIVE, however, the patterns in PROCESS-CONJUNCTION are more like anti-patterns. We use the patterns to 1) delete and skip over noun phrases that were incorrectly marked as conjunctions in *NP-HIST*, and 2) find noun phrases that should be part of the conjunction, but were missed during Phase 1 processing. There are two main problems with the syntactic processing in Phase 1 that cause the above errors. First, combinations of appositives and conjunctions show up in *NP-HIST* as a single conjunction. For example, a phrase like "Oqueli, 45, and Gilda Flores,..." shows up in *NP-HIST* as a conjunction of three NP's: "Oqueli", "45", and "Gilda Flores". One of the patterns in PROCESS-CONJUNCTION fixes this mistake by deleting the RELATED-NP link between "45" and "Gilda Flores" and adding one between "Oqueli" and "Gilda Flores". Another pattern in PROCESS-CONJUNCTION tries to detect cases where one of the NP's in the conjunction is actually a title appositive. This pattern,

for example, would change the three-part conjunction "Oqueli, Leader of the ELN, and Gilda Flores,..." in *NP-HIST* to the conjunction "Oqueli and Gilda Flores" and the appositive "Oqueli *APPLeaderoftheELN*".

The second problem with Phase 1 processing of conjunctions is that it can't handle any conjunctions that contain a prepositional phrase. Consider, for example, the phrase "the men of San Salvador, the women of Peru, and the children". In this case, *NP-HIST* links: 1) "San Salvador" and "the women" as appositives, and 2) "Peru" and "the children" as conjunctions. PROCESS-CONJUNCTION attempts to locate this type of error and modify *NP-HIST* before creating the conjunction. After fixing any errors, PROCESS-CONJUNCTION combines NP1 and NP2 into a single noun phrase record. For example, the np-record's for "his son" (NP1 below) and "two officers" (NP2 below) in the conjunction "his son and two officers" would be joined into a single np-record (NP3 below).

```
NP1:
#S(NP-RECORD :TIME-STAMP 2862
             :NGP-VALUE (SON)
             :WORD-SENSES (WS-HUMAN)
             :CN-LIST NIL
             :HEAD-NOUN SON
              ...
             :PREDICATES NIL
             :DETERMINERS (HIS)
             :ENTIRE-NOUN-GROUP (SON)
              ... )

NP2:
#S(NP-RECORD :TIME-STAMP 2863
             :NGP-VALUE (OFFICERS)
             :WORD-SENSES (WS-ACTIVE-MILITARY)
             :CN-LIST NIL
             :HEAD-NOUN OFFICERS
              ...
             :PREDICATES (&&2)
             :DETERMINERS NIL
             :ENTIRE-NOUN-GROUP (&&2 OFFICERS)
              ... )

NP3:
#S(NP-RECORD :TIME-STAMP 2848
             :NGP-VALUE ((SON) (OFFICERS))
             :WORD-SENSES (WS-HUMAN)
             :CN-LIST NIL
             :HEAD-NOUN SON
```

```
 ...
:PREDICATES (NIL (&&2))
:DETERMINERS ((HIS) NIL)
:ENTIRE-NOUN-GROUP ((SON) (&&2 OFFICERS))
 ...)
```

The final task of PROCESS-CONJUNCTION is to notice and mark conjunctions that specify inclusion rather than straight conjunction. For example, we need to know the difference between S1:"The terrorists killed Castellar, his son, and two officers" and S2:"The terrorists killed two officers, including Castellar and his son." In the first case, there are three victims and in the second, there are two victims. The output of S2 indicates the difference:

```
==========================================================================

***          TYPE = MURDER

***          ACTOR = WS-TERRORIST
***          noun group = (TERRORISTS)
***          determiners = (THE)
***          VICTIM = WS-ACTIVE-MILITARY
***          noun group = ((OFFICERS) <INC (CASTELLAR) (SON))
***          predicates = ((&&2) NIL NIL)
***          determiners = (NIL NIL (HIS))

==========================================================================
```

Chapter 4

Discourse Analysis

4.1 An Overview

The UMass system as used for MUC-3 is composed of a conceptual sentence analyzer, CIRCUS, and a discourse analysis component called "consolidation". These two modules work together in a pipelined fashion: CIRCUS generates conceptual meaning representations for individual sentences and consolidation maps these representations onto a set of response templates. We use two distinct methods for generating templates from parser output: rule-based consolidation and case-based consolidation.

Our case-based reasoning (CBR) module is an optional component that may be used to augment the output of rule-based consolidation. We used the CBR component in our official MUC-3 test run because it increases recall by generating templates that rule-based consolidation may have missed. However, the increased recall comes at the expense of precision because many of the additional templates turn out to be spurious. We demonstrated this recall/precision tradeoff by doing an optional MUC-3 test run without the CBR component; as expected, our system had lower recall but better precision (see our site report in these proceedings).

4.1.1 Rule-Based Consolidation

Rule-based consolidation merges the meaning representations produced by CIRCUS into a set of response templates. This process consists of 4 phases: constructing task-specific representations, partitioning, rule-based merging, and normalization.

The CIRCUS parser produces task-independent meaning representations (*concept nodes*) for individual sentences. Concept nodes are frame-like structures that are triggered by relevant words or phrases and filled by local syntactic constituents using semantic constraints and preferences. For example, the following concept node is generated from the sentence below it:

```
TYPE = MURDER
ACTOR = (FMLN commandos)
VICTIM = (Salvadoran leftist leader Hector Oqueli Colindres)
```

"Salvadoran leftist leader Hector Oqueli Colindres was killed by FMLN commandos."

Consolidation, however, must be able to reason with task-specific knowledge so it immediately converts each concept node into a task-specific knowledge structure called a *c-structure*.[1] During this process, explicit memory objects are created for victims, physical targets, perpetrators, dates, and locations. A simple pronoun resolution algorithm also tries to locate pronominal referents in preceding sentences; if it succeeds then the referent is substituted for the pronoun in the c-structure. The following c-structure is generated from the concept node above:

```
$MURDER
   ACTOR = $Perp-1
              ID = (FMLN commandos), ORG = (FMLN),
              WORD-SENSES = (ws-terrorist ws-organization),
              CONFIDENCE = nil, NEW-INFO = nil
   VICTIM = $Victim-1
              ID = (Hector Oqueli Colindres), TITLE = (Salvadoran leftist leader),
              NATIONALITY = El Salvador, NUM = 1, TYPE = (ws-proper-name ws-politicia
              EFFECTS = (death)
```

The second phase of consolidation, *partitioning*, identifies groups of c-structures that belong to the same incident. The c-structures are divided into partitions that reflect a weak organization of the text. All c-structures within a single partition are assumed to refer to the same incident, but different partitions may or may not correspond to the same incident. Partitions are created by exploiting textual cues, including specific phrases and patterns, as well as domain-dependent heuristics. There are four classes of textual cues: **new-event-markers**[2] introduce a new incident (e.g. "meanwhile,"), **generic-event-markers** suggest a generic or irrelevant event (e.g. "wave of"), and **separate-event-markers** identify references to multiple events within a single sentence (e.g. "the day before"). Domain-dependent heuristics are also applied to infer boundaries between multiple events. For example, if a partition contains two event types that were not in the preceding partition then we infer that this partition refers to a new incident. During the partitioning phase, c-structures that represent irrelevant events are discarded and c-structures that contain certain types of summary information are removed from the merging process and put aside to be used elsewhere.

Once the c-structures have been partitioned, they are sequentially merged into a set of response templates. This merging process is guided by a rule base of 139 rules.[3] Most rules are condition-

[1] short for "consolidation structure"

[2] These include **possible-new-event-markers** that signal a context switch only under specific conditions.

[3] There is a separate subset of rules for each type of c-structure.

action pairs where the condition specifies whether a particular c-structure and template are compatible and the action dictates how to merge the c-structure into the template. There are also default rules and special rules to generate a new template instead of merging the c-structure with an existing template. As templates are created, they are pushed onto a context stack. Given a c-structure to be merged, the rules are applied to each template on the stack, in turn, until one template is found to be compatible with the c-structure or until the stack is exhausted. If a compatible template is found then the rule fires, merges the c-structure into that template, and moves the template to the top of the stack.[4] If no compatible template is found then default rules decide whether a new template should be created from the c-structure.

Each rule has its own criteria for judging whether a c-structure and template are compatible. Some rules require only that the respective dates and locations are consistent whereas other rules may also require compatible targets, victims, instruments, etc. When a rule fires, memory objects that refer to the same entity are unified as a side effect of the merging process; this involves proper name resolution, updating type and nationality information, etc.

Rule-based merging also involves maintaining families of events. Texts often contain information about multiple events that were perpetrated by the same people on the same day and in the same location. We call this a *family* of events. To keep these events together, each template is tagged with a family id number. This is where the partitions come into play. All c-structures within a partition are forced to merge with templates in the same family. For example, if the first c-structure in a partition is merged into a template in family #2, then the remaining c-structures in that partition must also be merged into templates in family #2. Therefore the first c-structure in a partition effectively commits the entire group to a particular family.

The last stage of consolidation is *normalization*. This phase integrates summary information, normalizes families to ensure that all incidents in the same family share perpetrators, dates, and locations, adds default slot fillers, and discards templates that are deemed to be irrelevant. Currently, we only deal with summary information about similar incidents that took place in multiple locations. If the rule-based merging process has not already created templates for each of these incidents, then the missing templates are generated.

4.1.2 Case-Based Consolidation

Our system also has an optional case-based reasoning (CBR) component that can be used to augment the set of response templates generated by rule-based consolidation. CBR allows us to benefit from the development corpus by examining how parser output correlated with key templates in previous texts. The CBR module currently contains 254 cases drawn from 383 texts. A case is constructed from each key template by determining which concept nodes contain slot fillers that belong in the key.[5] For example, suppose a MURDER key template

[4]Hence, this is not strictly a stack since templates are not always removed from top. However the templates are checked for compatibility from top to bottom.

[5]The current implementation only looks at the perpetrator, human target, and physical target slots in the key.

contains a perpetrator and human target that correspond to the perpetrator slot filler in a $murder concept node from sentence 1 and a victim slot filler in a $murder concept node from sentence 2, respectively. The following case would be constructed:

(MURDER (0 (perp)) (1 (victim)))

This case represents concept nodes in adjacent sentences that contain a perpetrator and victim, respectively.

The concept nodes generated by the parser are compared with the cases in the case base. Some subset[6] of cases will be retrieved. Each retrieved case recognizes a pattern of concept nodes that resulted in a key template in a previous message; therefore, it recommends that this type of template should be generated for this text. If rule-based consolidation has not already generated such a template, then the CBR module will create one from the concept nodes that retrieved the case. In this manner, CBR can suggest additional templates that rule-based consolidation may have missed or incorrectly discarded.

4.2 Rule-Based Consolidation

The "consolidation" module of our system is responsible for mapping the CIRCUS output for a text onto a set of response templates. If each text reported on a single terrorist incident, then this task would be relatively straightforward. However, many of the MUC-3 texts contain information about multiple terrorist activities. This aspect of the task adds another level of complexity to the problem. The system must be able to recognize multiple events, distinguish between them, and figure out which pieces of information belong to each event. In general, the system must be able to freely switch between the contexts of different incidents; we will refer to this problem as "discourse analysis". *Rule-based consolidation* is the primary component responsible for discourse analysis.

The nature of the MUC-3 texts requires the system to produce templates from information that is usually contained in multiple sentences that may be scattered through the text. Since the CIRCUS parser generates conceptual representations from individual sentences, consolidation is responsible for merging these local representations into templates that represent all of the relevant terrorist activities reported in the text. Rule-based consolidation relies on a set of domain-specific rules to guide this merging process; textual cues and domain-specific heuristics are used to infer boundaries between events. There are 4 phases in rule-based consolidation: constructing task-specific representations, partitioning, rule-based merging, and normalization. The next four sections discuss each phase in detail.

4.2.1 Constructing Task-Specific Representations

When CIRCUS has finished parsing a text, it passes along the resulting concept nodes to consolidation. Consolidation immediately sorts the concept nodes to recover the order in which

[6] Possibly empty if no similar cases are retrieved

they were generated by the parser.[7] Each concept node is then converted into a task-specific representation called a *c-structure* (short for "consolidation structure"). C-structures are an intermediate representation that allows consolidation to reason with task-specific knowledge. For example, the victim slot in a murder concept node is filled with a generic noun phrase, such as "Salvadoran leftist leader Hector Oqueli Colindres". But consolidation must be able to easily recognize the name, title, and nationality of the victim (i.e., "Hector Oqueli Colindres", "leftist leader", and "Salvadoran" respectively) to compare him with another victim (e.g. "the Salvadoran politician"). Therefore, consolidation first converts the task-independent concept nodes produced by CIRCUS into task-dependent knowledge structures.

There are currently 24 types of c-structures used by consolidation. Some c-structures also have subtypes, for instance to distinguish between different types of threats, so more than 24 different types may actually be represented. Each structure type has a different set of associated slots.[8] Consolidation also creates explicit memory objects for victims, physical targets, perpetrators, dates, and locations. These memory representations allow consolidation to use specialized routines to compare similar objects, e.g. to decide whether two victim objects are referring to the same individual. The procedures for instantiating these memory representations and for recognizing and merging similar objects are discussed in the following subsections. These sections are somewhat detailed so the reader may skip to Section 4.2.2 for a more high-level view of consolidation.

Consolidation structures[9]

$arson: actor p-target effect date location
$attempted-attack actor victim p-target instr-type date location
$attack: actor victim p-target instr-type date location
$attempted-bombing: actor p-target date location effect
$attempted-kidnapping: actor victim date location effect
$attempted-murder: actor victim instr-type date location
$attempted-robbery: actor victim p-target effect date location
$bombing: actor victim p-target instr-type date location effect
$clash: actor victim date location
$destruction: actor p-target instr-type effect date location
$forced-work-stoppage: actor victim p-target instr-type date location
$hijacking: actor p-target old-dest new-dest instr-type date location

[7] Each concept node has a time stamp to identify exactly when it was generated by the parser.

[8] Originally, different structure types were defined because each type seemed to require a different set of slots. Over time, however, it seems that many structures converged on a common set of slots. In retrospect, it is probably only necessary to have a single structure to represent all types of events (with an extra type slot to distinguish between them) and separate structures to represent items that are not events (e.g. perpetrators). But this is just an implementation issue.

[9] C-structures also contain slots for bookkeeping information, e.g. the words that triggered the concept node that contributed to the c-structure, whether the triggering words were plural, the sentence from which the concept node came, whether the concept node represented summary-info, etc.

$injury: victim effect mode instr-type date location
$kidnapping: actor victim date location effect
$loc-val: actor object p-target date location
$location: p-target date location
$mtrans: actor to mobj date location
$murder: actor victim instr-type date location
$perpetrator: actor date location
$robbery: actor victim p-target effect date location
$shooting: actor victim instr-type date location
$threat: actor victim p-target instr-type effect date location subtype
$weapon: actor victim instr-type date location
$marker: subtype

Victim Objects

Victims are represented as structures with the following slots:

VICTIM: id title nationality num type effects

Fillers for the **id**, **title**, **nationality**, **num**, and **type** slots are extracted directly from the noun phrase in the concept node's victim slot. The **effects** slot is filled only during rule-based merging (see Section 4.2.3). The procedures for instantiating victim structures are fairly straightforward, but two subroutines are worth mentioning.

There are two situations in which both an **id** and **title** need to be extracted from a victim slot. The first case is when an appositive immediately follows the relevant noun phrase; CIRCUS will then insert the appositive into the slot with an appropriate label so that consolidation will have access to it. For example, *"Hector Oqueli, leader of the National Revolutionary Movement"* will yield a concept node of the form:

$murder: victim = (hector oqueli >app leader of the national revoluationary movement)

In this case, the appositive is used as the victim's title and the original noun phrase as the victim's id.[10]

The second case is when there is no appositive but the noun phrase (NP) itself contains a title. For example, *"Salvadoran leftist leader Hector Oqueli Colindres"*. The following procedure separates the id from the title:

Case 1: the NP is followed by a prepositional phrase (PP) beginning with "of"
⇒ use the entire NP as the **id** (assumption is that proper names shouldn't have this attachment)

Case 2: the head noun of the NP is not a proper name
⇒ use the entire NP as the **id**

[10] If the appositive is a proper name and the original noun phrase refers to a generic person (e.g. "salvadoran leader"), then CIRCUS will insert a different label, >name, so consolidation will know to extract the id and title in the opposite fashion.

Case 3: otherwise
Divide the NP into 2 parts by scanning from right to left for the first word that is *not* a proper name (location and organization names don't count as proper names for this purpose). Use the leftmost part as the **title** and the rightmost part as the **id**.

For example:
Case 1: *troops of the 4th detachment of Gotera* ⇒ **id** = *troops of the 4th detachment of Gotera*
because "troops" is followed by a PP beginning with "of".

Case 2: *state officials* ⇒ **id** = *state officials*
because the head noun, "officials", is not a proper name.

Case 3: *Salvadoran leftist leader Hector Oqueli Colindres* ⇒
id = *Hector Oqueli Colindres* ; **title** = *Salvadoran leftist leader*
because "leader" is the rightmost word that is not a proper name.

Consolidation must also determine the **type** of the victim, e.g. civilian, government official, political figure, etc. Each word in the lexicon is tagged with semantic features. Semantic features may also be inherited by the head noun of an NP from preceding words in the NP and from words in prepositional phrases that attach to it. In most cases, consolidation only needs to retrieve the semantic features attached to the head noun to correctly identify the type of the victim. Sometimes, however, semantic feature inheritance is not always sufficient. For example, a *"leader of the Patriotic Union"* is clearly a political figure but it doesn't make sense to attach a political-figure semantic feature to any of the individual words. In situations like these, where the semantic category is dependent on a pattern or sequence of words, consolidation uses domain-specific inference rules to infer semantic features for a noun phrase.[11] For example, the procedure *infer-politician* attaches a political-figure semantic feature to any noun phrase that contains one of the following patterns:

member of party
leader of <political org>
secretary of <political org>
president of <political org>
<nationality> leader
et al.

One of the most important functions of consolidation is to recognize different references to the same object. This ability is critical to the success of our approach; the rule-based merging process will be useless if consolidation cannot accurately distinguish between different entities and correctly recognize multiple references to the same entity. In general, we judge two lists of victim objects to be compatible if and only if they share at least one victim in common. The procedure for determining whether two victim objects refer to the same person is outlined

[11] CIRCUS uses similar rules for semantic feature inheritance. In principle, there is no reason why these rules could not be applied within CIRCUS itself. In practice, however, these rules are often more easily handled by consolidation.

below.

Given a victim and a list of potential matches, we first search the list for a victim with the same id. Two names will match only if they are equal or one is a subset of the other. For instance, *Ricardo Alfonso Castellar* will successfully match *Castellar*. The victims must also have compatible effects (e.g. "death" and "injury" would not be compatible). If no victim in the list has a compatible name then we search the list for a victim with a compatible type. If both victims have proper names then we don't bother comparing types under the assumption that two victims with different names will never be referring to the same person. Otherwise, two victims will match if they have the same types or if one set of types is subsumed by the other. Type subsumption will take care of pronominal references, e.g. if *he* has the type *ws-human* and *Castellar* has the type *ws-govt-official* then they will match because *ws-human* is an ancestor of *ws-govt-official* in our semantic feature hierarchy. In addition, if one victim has a proper name but the other does not then they will match if the victim without a proper name has types that subsume those of the victim with the proper name (i.e. its types are more specific than those of the proper name). If the victim without a proper name has no types then any type of human will be compatible.

Using this criteria, if the intersection between the two lists is not empty then the lists are combined by merging victim objects that refer to the same people. For individuals, this procedure is straightforward. We search each victim in the shorter list to see if a compatible victim exists in the longer list. If it does, then we merge these two victim objects and replace the original victim with the new victim object. All victims that were in one list but not the other are added to the resulting list.

This procedure is more complicated, however, when we have references to groups of people, i.e. plural victims. For example, suppose one list contains a single victim object for *politicians* and the other list contains two objects for individual people, *Hector Oqueli* and *Gilda Flores*. In this case, both Hector and Gilda should be compatible with *politicians* and their individual victim objects should replace the single *politicians* object. The heuristics for merging plural victims are as follows: (1) if two victim objects are both generically plural (i.e. no specific numbers are known) then they will be an exact match, (2) if one is generically plural but the other has a specific number > 1 then the specific number replaces the generic plural, (3) if one is generically plural but the other is singular then the singular victim is considered to be ONE OF the plural victims; in this case, if at least two victims match the generically plural victim then it will be discarded, (4) if both victims are plural with specific numbers then the smaller number is subtracted from the greater number but the remaining victims still need to be matched, or (5) if both victim objects have the same number then they are an exact match.

Physical Target Objects

Physical targets are represented as structures with the following slots:

PHYS-OBJ: id num type effects

Physical object structures are created in the same manner as victim structures except that it

is not necessary to extract a title for the physical target. Therefore, all appositives that get attached to physical target slot fillers are discarded. Consolidation also uses a set of inference rules to infer semantic features for physical targets. For example, the procedure *infer-organization-office* looks for the following patterns:

office(s) of union(s)
<building> of <organization>

The procedures for merging physical target structures are basically the same as those used for merging victim objects. One exception is that we don't care about the compatibility of effects. In principle, objects with inconsistent effects should never be merged but we didn't pick up physical target effects reliably enough to benefit from them.

Perpetrator Objects

Perpetrator objects are represented as structures with the following slots:

PERPETRATOR: id org word senses confidence new-info

Perpetrators are tricky because it is often difficult to determine whether a noun phrase refers to an individual or an organization. For example, groups of individuals such as "death squads", "commando group", etc. usually belong in the perpetrator-individual slot of a template. However, groups of people that represent an organization, but perhaps not a previously encountered organization, usually belong in the perpetrator-organization slot. Another difficulty is that many noun phrases refer to a specific group of individuals that includes a reference to the organization to which the individuals belong. In this case, the organization must be extracted from the noun phrase to fill the **org** slot but the entire noun phrase should still fill the **id** slot. For example, *FMLN commandos* ⇒ **id**=*FMLN commandos* ; **org**=*FMLN*.

The following procedure is used to extract organization names from slot fillers:
Case 1: If the NP contains a known organization then extract the organization name
Case 2: If the NP contains any word with the semantic feature *ws-organization* then use the NP (and PP beginning with "of") to fill the **org** slot
Case 3: If a PP beginning with "of" follows the NP and contains a word with the semantic feature *ws-organization* then use only the PP to fill the **org** slot

Examples:
Case 1: *FMLN commandos* ⇒ **org** = *FMLN*
Case 2: *government of Cristiani* ⇒ **org** = *government of Cristiani*
Case 3: *members of Cristiani government* ⇒ **org** = *Cristiani government*

The procedure for extracting individual perpetrators is very simple: if the head-noun does not have the semantic feature *ws-organization* then use the entire NP to fill the **id** slot. The **type** slot is filled by extracting word senses from the head noun; this information is used to fill the CATEGORY OF INCIDENT slot in the response template, i.e. military perpetrators ⇒ STATE-SPONSORED VIOLENCE and terrorist perpetrators ⇒ TERRORIST ACT. The confidence slot is filled directly from the concept node but may be updated later during rule-

based merging or via default heuristics (see Section 4.2.4). Finally, the **new-info** slot is filled directly from the concept node when the parser has reason to believe that this is new perpetrator information. When consolidation identifies an old event (more than 2 months prior to the wire date) then it will discard the event *unless* it contains new perpetrator information (e.g. someone was recently arrested for a crime that was committed several months ago).

The procedures for merging perpetrator structures are basically the same as those used for merging victims and physical targets except that we do not handle plurals in the same way. Perpetrators are often mentioned with only vague descriptions (e.g. *two men* or *three guerrillas*) and in the MUC-3 corpus we rarely saw situations where the text gives a general description of a group of perpetrators and later identifies them individually. That is, in general one might assume that the two men were SOME OF the three guerrillas but in practice these situations almost always referred to different sets of people. For these reasons, we decided that trying to handle plurals systematically would cause more trouble than it was worth. So in most cases, we rely on ids, organizations, types, and confidence values to distinguish between different perpetrators. In general, two perpetrators are consistent if they have compatible confidences and organizations and their ids or types match. Note that two perpetrator structures with different ids may be considered compatible if both ids have consistent types; in this case, one of the ids is arbitrarily chosen. For example, ids such as *guerrillas* and *guerrilla column* are compatible since they both have the type *ws-terrorist*. When their perpetrator structures are merged, one of these ids is arbitrarily chosen.

Dates

Dates come in three forms, based on the template encoding guidelines:

- **Single Date:** (D) ⇒ incident happened on a specific day D.
- **End Range:** (- D) ⇒ incident happened before or on day D.
- **Full Range:** (D1 - D2) ⇒ incident happened sometime between days D1 and D2, inclusive.

Deciding whether two dates are consistent is a fairly straightforward problem. In general, two dates are considered to be compatible unless they are contradictory. For example, a single date is compatible with an end range or full range as long as the single date is contained in the range; e.g., (09 JAN 89) is compatible with (- 11 JAN 89) but is not compatible with (- 08 JAN 89). When two dates are judged to be compatible, they are merged into a new date that is the most specific combination of the original dates. For example:

(09 JAN 89) and (- 11 JAN 89) ⇒ (09 JAN 89)
(09 JAN 89 - 12 JAN 89) and (- 11 JAN 89) ⇒ (09 JAN 89 - 11 JAN 89)
(09 JAN 89 - 12 JAN 89) and (11 JAN 89 - 13 JAN 89) ⇒ (11 JAN 89 - 12 JAN 89)

Locations

One of the resources provided to the MUC-3 participants was a database containing information about all of the geographic locations that appear in the corpus. Consolidation uses this database to set up a hash table of known locations. Each entry in the hash table is a linked list of location structures corresponding to a proper name. For example, there is a single entry for *La Paz* that is a linked list of length 4 since there are 3 departments named *La Paz* (in Bolivia, El Salvador, and Honduras) and one city named *La Paz* in Bolivia. Special disambiguation routines are used to determine which instance of a location name is actually being referenced. Each location structure has the following format:

LOCATION: name country ancestor[12] type next

As an example, the entry for La Paz is shown below:

```
#S(LOCATION :name La_Paz :country Honduras :ancestor nil :type department :next
  #S(LOCATION :name La_Paz :country El_Salvador :ancestor nil :type department :next
  #S(LOCATION :name La_Paz :country Bolivia :ancestor nil :type department :next
  #S(LOCATION :name La_Paz :country Bolivia :ancestor nil :type city :next nil))))
```

For each location filler in a concept node, consolidation tries to extract the most specific location structures that are consistent with the filler from the database. First, it looks for location names in the filler. Second, it looks for location type words in the filler, e.g. "city", "department", etc. The location name is used to index the hash table and all location structures for this entry that are consistent with the type information are extracted. For example, given the location filler *"the department of La Paz"* consolidation indexes the hash table by *La Paz* and extracts all location structures that have a department type. If there was no location type in the filler then all of the structures for the name are returned. The resulting linked list is added to the set of locations for the c-structure currently being constructed.

Since locations are attached to concept nodes in a bottom-up fashion, there may be multiple locations attached to a single concept node. In this case, a c-structure may be assigned multiple locations; its location slot will contain a list of linked lists, one for each location name that was attached to the concept node. For example, a concept node may have two attachments, one for *Miraflores district* and a second for *Lima*. In this case, the linked lists for both *Miraflores* and *Lima* will be added to the location slot of the c-structure.

To determine whether two location lists are compatible, we combine the two lists and search for an interpretation that is consistent with all of them. If all the locations cannot be resolved consistently then the two lists are not considered to be compatible. The first step towards resolving these locations is to identify a country common to all the locations. If a country is explicitly contained in the list, then it is chosen. If not, we try to infer a country using the following heuristics:

[12]The ancestor slot is always empty. This slot was originally intended to represent containment relations between locations (e.g. Miraflores is a district IN the city of Lima) which are provided by the database but this was never implemented.

1. If all locations are in the same country, then choose that country.
2. If there is only one location (e.g. La Paz) and one of its structures is a city then choose the country of this city.
3. If there is only one location entry and it has multiple cities where one of these is a default city then choose its country.[13]
4. Else the locations are not compatible.

Once a country is identified, we attempt to resolve the remaining ambiguities. The disambiguation algorithm operates via recursive constraint propagation. First, it compiles a list of all locations that have only one structure (and therefore need no disambiguation). If any two of these locations have the same type, or equivalent types[14], then the two sets of locations are judged to be incompatible. If there are no remaining ambiguities then we are done. Otherwise, we call this algorithm recursively giving it the list of location types that have been resolved successfully as a parameter. These location types shouldn't be considered as possibilities the next time through the procedure.

To illustrate this process, suppose we want to merge two locations: San Miguel and Buenos Aires. The algorithm will try to determine whether these two location names could be consistent with the same place (i.e. whether one could be part of the other). San Miguel has 6 entries in the database: a city, department, jurisdiction, and region in El Salvador, a river in Ecuador, and a city in Argentina. Buenos Aires has two entries in the database: as a city and province in Argentina. First, we try to find a country common to both locations. Argentina is the only country common to both names so all entries in the other countries are ruled out. Second, we see if either of the locations has only a single structure left. In this case, San Miguel has only one structure in Argentina so this structure is selected. However, Buenos Aires is still ambiguous since it has two entries in Argentina so we call the procedure recursively. This recursive call, however, recognizes that we've already found a city so it eliminates the city structure for Buenos Aires from consideration. Now, Buenos Aires only has one remaining structure (the department in Argentina) that is consistent with all the data, so this structure is returned and the locations have been resolved successfully.

In many cases, the locations can be resolved in multiple ways that all lead to consistent interpretations. We take a least commitment approach in this situation; we do not attempt to resolve the ambiguities until the template is actually generated. All of the location structures that have not been ruled out are passed along in the hopes that future information will come along to supply additional constraints and disambiguate them. If the locations are still ambiguous when all of the input has been processed, then consolidation will prefer a city (if a city has not

[13] We maintain a list called *default-cities* to make note of cities with the same name in different countries such that one of the cities is clearly more well-known than the other. For example, there is a city named San Miguel in both El Salvador and Argentina but the city in El Salvador is much more commonly referenced.

[14] We keep a list of location types that are usually synonymous, e.g. city and town.

already been found) or else it will arbitrarily choose the first type that has not already been seen.

4.2.2 Partitioning

One of the most difficult problems faced by consolidation is deciding where one event ends and another begins. If we fail to distinguish between two separate events then an entire template will be missing from the output. Alternatively, if we misjudge the division between incidents, then the information may get distributed incorrectly between multiple templates. And since many of the MUC-3 messages report on multiple incidents, this problem is both pervasive and important.

Fortunately, the text often provides clues to help the reader anticipate a context switch. Textual cues may signal a change of topic, help to distinguish between different events, or otherwise highlight certain classes of incidents. To benefit from these surface cues, consolidation employs a partitioning algorithm to outline a weak organization of the text. This algorithm relies on both textual cues and domain-dependent heuristics.

Exploiting Textual Cues

People often use phrasal cues to usher in a change of topic. There are undoubtedly countless phrases that may be used to change topics, but certain phrases and patterns seem be used frequently and regularly in the MUC-3 corpus. Consolidation relies on *event markers* to exploit these textual cues. When CIRCUS recognizes a special phrasal cue or pattern, it generates an *$event-marker* concept node which then gets passed along to consolidation. There are currently 4 different types of event markers used by consolidation: $new-event-markers, $possible-new-event-markers, $separate-event-markers, and $generic-event-markers.

a. New-event-markers

New-event-markers are triggered by phrases or patterns that introduce a new event. The following phrases and patterns will generate a $new-event-marker:

furthermore
meanwhile,
a week later
<num> weeks ago
a few days ago
<num> days ago
likewise,

In addition, the following phrases and patterns will trigger a $new-event-marker only when they occur at the *beginning* of a sentence:

on <date>,
in another action
in another incident
in addition,
in <location>,
in the <location-type> of <location>
finally,
another report

To illustrate how these are useful, the sentences below are examples of texts that will trigger a $new-event-marker:

A car-bomb exploded in front of the PRC embassy, which is in the Lima residential district of San Isidro. **Meanwhile,** *two bombs were thrown at a USSR embassy vehicle that was parked in front of the embassy located in Orrantia district, near San Isidro.*

Castellar is the second mayor that has been murdered in Colombia in the last 3 days. **On 5 January,** *Carlos Julio Torrado, mayor of Abrego in the northeastern department of Santander, was killed apparently by another guerilla column, also belonging to the ELN.*

b. Possible-new-event-markers

Possible-new-event-markers are triggered by phrases that may signal a context switch but only in certain situations. The specific conditions are discussed in Section 4.2.2. The following phrases will produce a $possible-new-event-marker:

also reported
also said

For example, the following sentence will trigger a $possible-new-event-marker:

It was **also reported** *that four Treasury police officers were wounded last night when urban commandos attacked them in Soyapango.*

c. Separate-event-markers

Separate-event-markers indicate that a single sentence contains information about multiple events. The marker is used to separate the sentence into different pieces. Currently, this marker is triggered only by one phrase: **the day before**. For example, the following sentence will generate a $separate-event-marker:

These operations seek to clear up both a crime that has shocked the nation and also the killing of two patrolmen who were murdered in cold blood **the day before** *the general was murdered.*

d. Generic-event-markers

Generic-event-markers are triggered by phrases that suggest a generic event or series of events. This definition of "generic" is with respect to the MUC-3 encoding guidelines that dictate when an incident is too general or otherwise irrelevant to our task, e.g. the incident occurred more than 2 months prior to the wire date.

The following phrases and patterns will trigger a $generic-event-marker:

wave of
series of
in less than 1 month
in <num> months
several months ago
raids
this year
in the last <num> years
let us recall
we remember

E.g.,
The war between the Medellin and Cali drug cartels has already claimed more than 120 deaths and has provoked a **wave of** *bomb attacks in Medellin.*

The Partitioning Algorithm

The partitioning algorithm involves 3 phases: segmentation, shuffling and refinement. The first phase assigns the c-structures to partitions primarily based on sentence divisions, the second phase shuffles the c-structures within a partition, and the third phase further revises the partitions using domain-specific heuristics.

Segmentation

The first phase of partitioning, *segmentation*, groups together c-structures that were triggered by the same sentence[15] with a couple of noteworthy exceptions. The assumption is that concept nodes triggered by the same sentence usually belong to the same family of events (see Section 4.2.3 for an explanation of template families). This assumption isn't always valid, but seems to be a good heuristic; the $separate-event-marker (see Section 4.2.2) was designed to permit exceptions to this rule.

[15]I will sometimes refer to c-structures as if they were concept nodes in situations where the distinction is unimportant. For example, if I claim that a structure was triggered by the word "murder", I mean that the concept node from which the structure was derived was triggered by the word "murder". There is actually a many-to-one mapping from concept nodes to c-structures, but all concept nodes that contributed to a c-structure must have been triggered by the same word.

This phase is also responsible of recognizing and discarding irrelevant c-structures. Task-specific heuristics are used to recognize c-structures that do not refer to specific events or are deemed to be irrelevant with respect to the MUC-3 encoding guidelines. These heuristics were added as an additional filter for CIRCUS to prevent false hits from getting through to consolidation.[16] There are currently 6 heuristics for recognizing irrelevant c-structures:

1. **THREATS-WO-TARGET:** $threat triggered by "threats" with no targets or victims, e.g.
   "... in view of the threats against Argentine democracy"

2. **DEATH-WO-VICTIM:** $murder triggered by "death" with no victims, e.g.
   "... our war is to the death."
   "Meanwhile, the terrorists continue on their path of death and destruction"
   "Death to communism!"
   "Death hangs over Barrancabermeja."

3. **TOO-MANY-VICTIMS:** $murder with $> 10{,}000$[17] victims, e.g.
   "... an internal war that has left over 70,000 dead in 9 years."

4. **NO-PERPETRATOR:** $perpetrator with "no" in actor's predicates, e.g.
   "No organization claimed responsibility for this attack ..."

5. **PLURAL MURDERS:** $murder triggered by "murders", e.g.
   "... to attain justice for the murders of Febe Elizabeth Velasquez, other cooperativists, the ten union leaders, and Msgr Arnulfo Romero."

6. **PLURAL EVENTS:** $attack triggered by plural noun preceded by "several" or "its" OR any event c-structure triggered by a plural noun and preceded by "some", e.g.

   "On Friday, 16 March, the Farabundo Marti National Liberation Front (FMLN) decreed a suspension of its attacks on civilian government officials ..."

   "The Salvadoran people can rest assured that the Armed Forces control everything; there are only some bombings and shootings that will not lead to anything."

The algorithm for segmentation is as follows:

```
For each c-structure

If it is deemed irrelevant then throw it away
```

[16] In theory, these filters could (and perhaps should) have been added to CIRCUS in the form of additional enabling conditions; such conditions would have prevented these concept nodes from getting triggered in the first place. In practice, however, it was sometimes easier to implement these heuristics in consolidation.

[17] This number is arbitrary.

```
Else If it is a $separate-event-marker
     Then start a new group but throw away the marker
     Else If it is a $new-event-marker or a $possible-new-event-marker
          Then start a new group but retain the marker
          Else If it was triggered by a different sentence than the
                  previous c-structure
               Then begin a new group
               Else add it to the current group
```

In the absence of event markers, this procedure will by default partition the c-structures into groups based on the sentences from which they came. However, if event markers are found then they may force multiple partitions to be generated for c-structures from the same sentence.

Shuffling

The second phase of partitioning is called *shuffling*; the c-structures within a group are shuffled in an attempt to put the most important information first. The advantage of shuffling becomes apparent only by understanding how the rule-based merging algorithm works (see Section 4.2.3). Briefly, rule-based merging holds the first c-structure in a partition responsible for committing the entire partition to a template family (see Section 4.2.3). In particular, by placing the c-structure with the most relevant information in the first position of the partition, we are trying to increase the likelihood that the first c-structure will find the best family match for the group.

The algorithm itself is quite simple: non-event structures are placed after the first event structure that follows them in the group[18], e.g.,

($injury $murder $kidnapping $weapon $destruction $bombing)
⇒ ($murder $injury $kidnapping $bombing $weapon $destruction)

The intuition here is that the information most critical for correct template merging is more likely to be found in an event c-structure than in a non-event c-structure. Ideally, all relevant information about an event should be in a single structure but then the problem becomes circular (i.e. how do we know which concept nodes belong in the same structure?). This problem clearly deserves more attention; shuffling is merely a quick and dirty attempt to get around the problem.

One additional side effect of this phase is that c-structures that represent certain types of summary information are removed. These c-structures are temporarily put aside to be used during the *normalization* phase (see Section 4.2.4).

[18]Event structures represent MUC-3 events (e.g. $kidnapping, $bombing, etc.) whereas non-event structures represent effects and other detail information (e.g. $injury, $perpetrator, etc.).

Refinement

The third phase of partitioning involves a second pass through the partitions to further revise the original groups. Heuristics based on c-structure groups are used to make additional assessments about relevancy and partitioning. This procedure discards irrelevant groups of c-structures, reevaluates the relevancy of event markers, and dynamically creates $new-event-markers using domain-specific heuristics that take advantage of indefinite articles, special predicates, etc. The current set of refinement heuristics are listed below. These heuristics are tried in order so the first one that applies will take precedence over the others.

1. **IRRELEVANT-GROUP:** If the partition contains only a $new-event-marker OR it contains a $generic-event-marker OR an event structure that was triggered by a noun phrase that included the predicate "various" then throw away all structures in the partition.
2. **IGNORE-EVENT-MARKER:** If a $new-event-marker is in a sentence with an event triggered by a noun phrase containing a definite article OR a $new-event-marker in a sentence that didn't generate any event structures, $weapon, or $destruction structures OR a $possible-new-event-marker in a sentence that didn't generate any event structures then throw away the marker.
3. **LEGAL-NEW-EVENT-MARKER:** If the partition begins with a $new-event-marker then retain the marker.
4. **LEGAL-POSSIBLE-NEW-EVENT-MARKER:** If the partition begins with a $possible-new-event-marker then replace it with a $new-event-marker.
5. **SEPARATE-INCIDENT:** If the partition contains an event-structure or $weapon=vehicle-bomb that is preceded by the words "another" or "second" OR the partition contains an $attack preceded by "also" or a $bombing triggered by "blown up" and preceded by "also" OR event-structures triggered by an attack noun and preceded by an indefinite article OR the partition contains at least 2 new event types that were not in the previous partition then insert a $new-event-marker at the front of the partition.
6. **DEFAULT:** don't change the partition.

For example, the IRRELEVANT-GROUP heuristic will throw away all c-structures that come from this sentence:
"Morales said that the group has killed several people and has carried out **various bomb attacks** *in Guatemala."*

And the SEPARATE-INCIDENT heuristic will insert a $new-event-marker at the front of the partition for the following sentence:
"Seven private bank branches were **also attacked** *and seriously damaged in Lima."*

4.2.3 Rule-based Merging

The ultimate job of rule-based consolidation is to merge the concept nodes produced by CIRCUS into a set of templates that represent the relevant terrorist incidents in the text. Once the concept nodes have been converted into c-structures and partitioned, the partitions are sent to the rule-based merging module. This module uses a domain-specific rule base to sequentially merge these c-structures into a final set of response templates.

The Rule Base

Consolidation relies on a large domain-specific rule base to decide how to integrate each c-structure into the current context. A rule may merge a c-structure with an existing template, create a new template from the c-structure, or in some cases do nothing (effectively ignore the c-structure). The current rule base contains 139 rules. Each rule is indexed with respect to the type of c-structure to which it applies; hence, there are 31 subsets of rules since there are 31 different types of c-structures.[19]

Most rules are condition-action pairs in which the condition specifies whether a particular c-structure and template are compatible and the action dictates how to merge the c-structure into the template. The firing conditions typically check for at least date, location, and family compatibility (families are discussed in Section 4.2.3) and may or may not check for compatible victims, targets, instruments, etc. There are also default rules and special rules to generate a new template instead of merging the c-structure with an existing template.

As an example, consider a simple rule *merge-kidnapping-kidnapping* to merge a $kidnapping c-structure with a KIDNAPPING template:

```
IF victims are compatible and
   dates are compatible and
   locations are compatible and
   families are compatible
THEN
   merge $kidnapping victims, dates, locations, and perpetrators
   into the KIDNAPPING template
```

More complicated rules may involve fairly sophisticated inferences, e.g. the rule *merge-car-bombing-murders* merges a $bombing c-structure with a MURDER template under very specific conditions:

```
IF  the $bombing contains a single target and
    that target is a transport vehicle and
```

[19]There are 24 basic types plus we used 7 types of threat structures that are distinguished via their subtype slots.

```
    there are MURDER templates on the stack with no instrument and
    dates are compatible and
    locations are compatible and
    families are compatible
THEN
    merge these MURDER templates into a single MURDER template
    set the MURDER instrument to be the same as the $bombing instrument
    create a new BOMBING template from the MURDER template and the $bombing
```

In effect, this rule says: If person(s) were murdered but no instrument was specified AND a vehicle was bombed AND the dates and locations of the murder(s) and bombing are compatible THEN infer that the people were in the vehicle and were killed by the bomb. This is an example of a very domain-specific inference that is applied implicitly via the rule base.

Rule application works in the following manner. Each c-structure is considered for merging in the order in which it is received from the partitioning module. As templates are created, they are pushed onto a context stack. Given a c-structure to be merged, the rules are applied to each template on the stack, in turn, until one template is found to be compatible with the c-structure or until the stack is exhausted. If a compatible template is found then the rule fires, merges the c-structure into that template, and moves the template to the top of the stack.[20] If no compatible template is found then default rules decide whether a new template should be created from the c-structure. At most one rule will fire per c-structure. If no rules fire then the c-structure is effectively ignored. The rule base is completely flat and there is no chaining of multiple rules.

To illustrate this process, consider the following context stack (top to bottom):

BOMBING
KIDNAPPING1: date = (05 JAN 89)
KIDNAPPING2: date = (01 JAN 89 - 03 JAN 89)

Suppose consolidation receives a $kidnapping c-structure with date = (02 JAN 89). First, each rule is applied, in turn, to the pair <$kidnapping, BOMBING>.[21] Let's assume that there are no rules to merge $kidnapping c-structures with BOMBING templates. Then we apply the rule base to the pair <$kidnapping, KIDNAPPING1>. The previous rule *merge-kidnapping-kidnapping* will be tested but will not fire because the dates (02 JAN 89) and (05 JAN 89) are not compatible. If no other rules are applicable, then the rule base will be applied to the pair <$kidnapping, KIDNAPPING2>. This time, *merge-kidnapping-kidnapping* is applicable because the dates (02 JAN 89) and (02 JAN 89 - 03 JAN 89) are compatible. The

[20] Hence, this is not strictly a stack since templates are not always removed from top. However the templates are checked for compatibility from top to bottom.

[21] Note that this algorithm is sensitive to the ordering of the rules in the rule base; the first rule that fires will be applied. In practice, this does not cause much difficulty because there are only a few rules (usually only one or two) for each type of c-structure/template pair.

$kidnapping will be merged into KIDNAPPING2 and this template will be moved to the top of the stack, yielding:

KIDNAPPING2: date = (02 JAN 89 - 03 JAN 89)
BOMBING
KIDNAPPING1: date = (05 JAN 89)

If no rule had been applicable to this pair then a default rule, assuming one exists, would have been invoked to create a new KIDNAPPING template from the $kidnapping c-structure which would then have been pushed onto the top of the stack.

Families

Rule-based merging also involves maintaining families of templates. Texts often contain information about multiple events that were perpetrated by the same people on the same day and in the same location. For example, if a commando group goes on a rampage killing people, bombing vehicles, and burning farms then 3 response templates should be generated (murder, bombing, and arson) with the same perpetrators, dates, and locations. We call this a *family* of events.

To keep these events together, each template is tagged with a family id number. This is where the partitions and $new-event-markers come into play. All c-structures within a partition are forced to merge with templates in the same family. For example, if the first c-structure in a partition is merged into a template in family #2, then the remaining c-structures in that partition must also be merged into templates in family #2. Therefore the first c-structure in a partition effectively commits the entire group to a particular family. Alternatively, if a $new-event-marker is the first c-structure in the partition then this tells consolidation to start a new family for this partition. A $new-event-marker will trigger a special rule in the rule base (sort of a meta-rule) that makes all of the previous templates inaccessible for rule-based merging during the processing of this partition. So the next c-structure in the partition must start a new family (and hence a new template) which then forces the remaining c-structures in the partition to commit to this new family as well.

4.2.4 Normalization

The last stage of consolidation is *normalization*. This is a clean-up phase that integrates summary information, normalizes families, resolves pronouns, adds default slot fillers, and discards irrelevant templates. Each aspect of this phase is discussed separately in the next few sections.

Summary Information

Summary information is difficult for our system because it is not easily handled by our current architecture. For our purposes, we define summary information as information that applies to multiple events. For example, the following sentence refers to injuries and targets that apply to several distinct incidents:

S1: *"At least 18 casualties, among both soldiers and civilians, resulted from several guerrilla attacks on the president's residence, the central electoral council, and other military installations in San Salvador and several departments throughout the country."*

It is not always easy to identify summary information since the interpretation of a sentence may depend on text that follows it, for example:

"Two U.S. citizens have been killed in the last few hours ..."

This sentence actually refers to two separate incidents, but this is not clear until later in the text. However, one could easily imagine how this sentence might be followed by different text that would imply that both people were killed during a single incident. Hence, it is a non-trivial problem just to identify the sentences that represent summary information.

Similarly, some sentences clearly refer to multiple events but the distinction between the events may not be fine-grained enough (w.r.t. the encoding guidelines) to warrant separate templates. For example, consider the following sentence:

"Three bombs went off simultaneously in Tegucigalpa, one at the Central Park, another at the state-run University of Honduras, and another at a training school for teachers."

The word "simultaneously" clearly indicates that these were separate incidents, but since all 3 bombings took place on the same day and in the same city, the distinction between these incidents cannot be represented in the templates. In this situation, the encoding guidelines mandate that all of the events should be represented in a single template even though are actually separate incidents. This is a good example of how the encoding guidelines (and the target representation) can affect the rules of the game; the human interpretation of the text is not always the interpretation that should be reflected in the output.

Fortunately, many of the texts that contain summary information follow a pattern wherein the first few sentences contain summary information and subsequent paragraphs discuss each incident separately in greater detail. The frequency of this pattern, combined with the problems mentioned above and the difficulty of dealing with summary information in our current architecture, persuaded us not to focus our energies on this problem. In most cases, we don't even try to identify sentences that contain summary information; they are treated as any other sentence and the concept nodes that they generate are merged into templates as though they apply to a single event. This often means that information that should be distributed among many events will only be applied to one of them. For example, a concept node representing 2 attacks in Bogota will be converted into a single $attack c-structure. If these two attacks

are further detailed in subsequent paragraphs (which is often the case) then one of them will probably merge with the attack template generated from the $attack c-structure and the second incident may or may not merge with it depending upon whether their fillers are compatible. If it does not merge with the first attack, then a second attack template will be generated which will likely be missing the location, Bogota. This is probably the most common problem resulting from summary information.

Alternatively, summary information for multiple events may end up in a single template instead of being distributed between several templates. For example, a concept node representing 18 injuries that belong to 2 attacks will be converted into a single $injury c-structure that contains all 18 injuries. If the text later specifies that 10 of the injuries correspond to one attack and 9 to a second attack, then the template generated from the $injury will be incorrect. One way that we tried to deal with this problem was via segmentation heuristics (see Section 4.2.2). Some of these heuristics identify concept nodes that are likely to contain summary information, especially in the situation where the summary information precedes a more detailed account of the individual events. For example, in S1 the concept node triggered by *several guerrilla attacks* would be discarded by the segmentation phase in accordance with the **PLURAL EVENTS** heuristic. Since these events are discussed individually later in the text, consolidation will receive more specific concept nodes for each of them eventually. These heuristics, therefore, can prevent summary information from getting in the way during the rule-based merging process.

There is one type of summary information that we do handle in a systematic way: information about similar incidents that took place in multiple locations. For example:

"A few days ago, the Chilean-U.S. cultural institutes in Vina Del Mar, Talca, and Temuco were the target of bomb attacks."

This single sentence should generate 3 bombing templates because each bombing took place in a different location. These sentences are easy to identify (by looking for a conjunction of locations) and usually contain most (if not all) of the relevant information about the events. The partitioning phase is responsible for removing these concept nodes from the rule-based merging process (see Section 4.2.2); in the meantime, they are stored in a global variable *summary-info* until the normalization phase is ready for them.

The normalization phase uses these c-structures to make sure that all of these incidents have a corresponding response template. More specifically, each of these c-structures contains multiple locations. If a c-structure has N locations in it, then it is separated into N c-structures with all of the same information but each with a different location. For each of these new c-structures, we first check to see whether rule-based merging has already created a similar template.[22] If so, then the c-structure is discarded and no additional templates are created. If not, then a new template is created from the c-structure. This procedure ensures that all of the events covered by this summary information are represented in the final set of response templates.

[22]Special functions are used to search for a template with compatible types, victims, targets, dates, and locations.

Normalizing Families

As described in Section 4.2.3, templates that correspond to events that were all part of the same incident are tagged with the same family id number. Template families are meant to capture the situation where a terrorist perpetrates multiple acts in a single crime spree. In the most trivial case, a victim may be killed as the result of another action (e.g., bombing) so we must generate two templates, one for the main action (the bombing) and another for the murder. In principle, a family of events should have the same perpetrator, dates, and locations. The normalization phase of consolidation makes sure that templates in the same family share this information.

In general, for each family of templates, consolidation merges all the perpetrators, dates and locations contained in the templates and then re-distributes the combined information to each of the templates. In this way, information that may have been distributed throughout a family of templates will be collected and shared among them. There are exceptions for certain cases in which this simple type of information sharing is not sufficient; for example, if a kidnapping and murder are in the same family and the murder took place on January 9 then this does NOT imply that the kidnapping took place on January 9. However it does imply that the kidnapping happened on or before January 9. Some special cases like these are recognized and treated accordingly, but in general information is shared uniformly across templates.

Pronoun Resolution

Occasionally, concept node slots will contain pronouns. When these slots are incorporated into templates, a pronoun resolution algorithm can be used to find the referents for the pronouns.

The algorithm is rather simple. It takes as input a pronoun p, a concept node slot c, and a sentence s. In essence, the algorithm searches backwords from p until it finds a suitable referent. In more detail, the search proceeds as follows.

1. Parse s, yielding an ordered list of noun phrases np.
2. Search np for the noun phrase n* in which p first occurs.
3. For each noun phrase n that appears before n* in np:
4. If n satisfies the slot constraints for c
   and n is compatible with p
   and n is not a pronoun itself
5. Then return n as a resolution for p.
6. Else continue.
7. If no resolution is found in s, then attempt to find a resolution in the two sentences preceeding s.

A noun phrase satisfies the slot constraints for a concept node slot if the word senses for the noun phrase match the word senses making up the slot constraints. A noun phrase is compatible with a pronoun if they agree in number and in type (e.g., both most refer to a single person).

Adding Defaults

Before we output the final response templates, consolidation makes a final attempt to fill slots that did not get filled during rule-based merging. Domain-specific and task-specific heuristics are called upon to add default fillers to slots that would otherwise be left empty.

For each response template, the following set of heuristics are applied:

1. **Single Perp Org Heuristic:** During parsing, CIRCUS puts the names of all known terrorist organizations that appear in the text into a global variable *org*. If the parser finds only a single terrorist organization mentioned in the entire text, then this organization is used as a default perpetrator for any response templates that don't already have a perpetrator organization.

2. **Location Default:** If a template does not have a filler in its location slot, then a default location is added according to the following procedure: If the wire-city[23] is from one of the relevant countries[24] then the country of the wire city is used to fill the location slot. Otherwise, if any of the other templates contains a location filler then its country is used to fill the location slot (the first one found is used). Otherwise, we use El Salvador as a default location.[25]

3. **No Damage Default:** If the template is an attempted event and there are physical targets and there are no known effects on these targets then fill in NO DAMAGE as the default effect.

4. **Clandestine Single Org Heuristic:** If the header of the text is marked as CLANDESTINE (instead of a wire city) and a template has a single perpetrator organization with no confidence value then its confidence value is assigned to be CLAIMED OR ADMITTED.

In addition, there are some basic defaults that are added when no other slot filler is available. In particular, the date defaults to an end-range containing the wire date[26], e.g. if the wire date = 09 JAN 89 then the default will be (- 09 JAN 89). Human target types will default to CIVILIAN, physical target types default to OTHER, and perpetrator confidence defaults to REPORTED AS FACT.

---

[23]The city listed in the header of the text.

[24]The nine relevant nations, with respect to the encoding guidelines, are: Argentina Bolivia Chile Colombia Ecuador El Salvador Guatemala Honduras Peru

[25]El Salvador was chosen as a default because it was the most frequent target of terrorist attacks in the development corpus.

[26]The date that appears in the header of the text

Discarding Irrelevant Templates

Finally, consolidation throws away templates that do not qualify as relevant incidents with respect to the MUC-3 encoding guidelines. The following situations will cause a template to be deemed irrelevant:

1. **CLASH:** If the template was generated by a $clash concept node and all victims and targets are irrelevant. In addition, a CLASH template with no victim or targets will be discarded.
2. **IRRELEVANT TARGETS:** If the template contains only irrelevant victims and targets, i.e. if they are all terrorists, military, or law-enforcement and do not have a foreign nationality.
3. **OLD EVENT:** If the template contains an old date, i.e. more than 2 months previous to the wire date.
4. **IRRELEVANT LOCATION:** If the template's location does not belong to one of the nine relevant countries AND no targets or victims have nationalities from one of the 9 relevant countries.[27]

4.3 Case-based Consolidation

There are two phases to case-based consolidation. The first phase starts with a set of stories and their associated key templates, and produces a case base. The second phase starts with a case base and a story, and produces a set of templates for the story. The entire case-based consolidation process is illustrated in Figure 4.1. The case extractor produces a case base from stories and their templates; retrieval and the template generator together produce a set of templates from a story and the case base.

Each case in the case base associates a set of concept nodes with a template of a particular type. Each case is generated from a key template, its associated story, and the concept nodes created when parsing the story. Cases are used to determine whether or not a set of concept nodes all contribute to the same template. The concept nodes are transformed into a probe to the case base. If retrieval succeeds for the probe, it is assumed that all the concept nodes contribute to a single template; otherwise, it is assumed that they contribute to more than one template.

Retrieval for a story results in a set of cases, representing a set of hypothesized templates. The cases are used to generate new actual templates, each containing slot information obtained from the concept nodes associated with its generating case.

[27] An incident is considered to be relevant even if it occurred in an irrelevant country as long as it was perpetrated against a victim or target from one of the relevant countries.

Sections 4.3.1, 4.3.2, and 4.3.3 describe case extraction, retrieval, and template generation in greater detail.

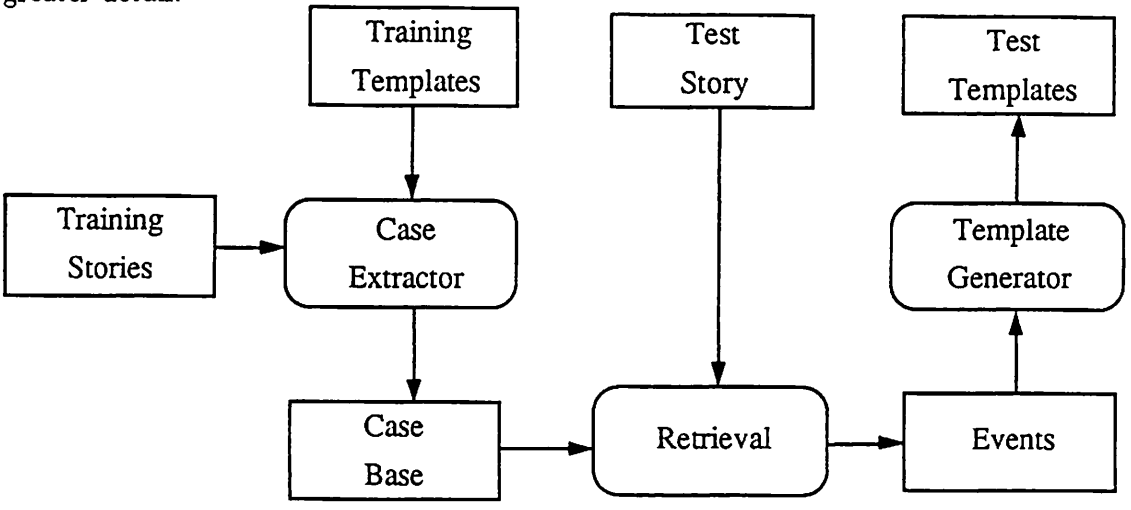

Figure 4.1: Case base generation and use. Rectangles represent data, boxes with rounded corners represent operations.

4.3.1 Extracting Cases from Texts

Generating a case base involves examining a set of stories and their associated templates, and extracting cases from each. The case extraction process is performed for each template of each story used to generate the case base. Figure 4.2 illustrates the template generation process for a single template. Input to the process is the template and its corresponding story; output is a case. Unique cases resulting from this process are added to the case base.

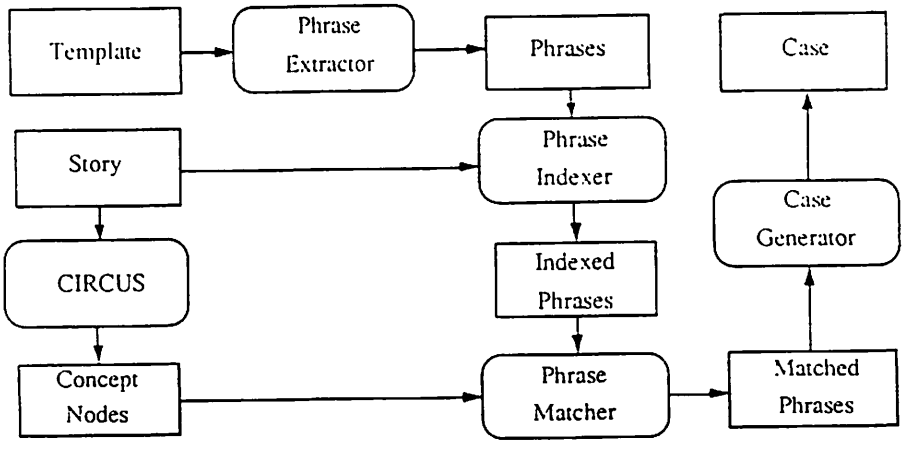

Figure 4.2: The case extraction process for a single template. Rectangles represent data, boxes with rounded corners represent operations.

There are five subprocesses to case extraction:

1. The *phrase extractor* examines template slots and extracts slot fillers.
2. The *phrase indexer* examines the phrases output by the phrase extractor, and indexes them by the sentence in which they first occur in the story.
3. CIRCUS parses the story and produces a set of concept nodes.
4. The *phrase matcher* examines the output of the phrase indexer, and further annotates it by supplying the slots of the concept nodes whose values match the phrases.
5. The *case generator* examines the output of the phrase matcher, and puts the case in its final form.

To see how these subprocesses contribute to extracting a case, consider the following four-sentence story:

PANAMA'S ARCHBISHOP MARCOS MCGRATH, PANAMA'S EPISCOPAL CONFERENCE PRESIDENT JOSE LUIS LACUNZA, AND PANAMANIAN ABDIEL ADAMES, PRESIDENT OF THE CENTRAL AMERICAN HIGHER UNIVERSITY COUNCIL (CSUCA), HAVE CONDEMNED THE ASSASSINATION OF SIX JESUITS IN EL SALVADOR AT DAWN TODAY. MCGRATH, WHO DESCRIBED THE MULTIPLE ASSASSINATION AS "MONSTROUS," POINTED OUT THAT "IT IS EVERYBODY'S DUTY, ESPECIALLY OF THAT COUNTRY'S GOVERNMENT, TO IDENTIFY, ARREST, AND TRY THE CULPRITS AND THOSE WHO ORDERED THE ACTION." THIS "MUST NOT ONLY BE A JUST TRIAL OF THEM, BUT ALSO A MEANS TO ELOQUENTLY EXPRESS THE REJECTION OF THE ENTIRE SOCIETY, WHICH CALLS ITSELF CHRISTIAN, AGAINST SUCH A CRUEL AND UNJUST CRIME THAT REVEALS ABSOLUTE CONTEMPT FOR HUMAN LIFE," A COMMUNIQUE SIGNED BY ARCHBISHOP MCGRATH STATES. ACCORDING TO SOURCES FROM EL SALVADOR'S JESUIT CURIA, THE SPANISH JESUIT IGNACIO ELLACURIA, RECTOR OF EL SALVADOR'S CENTRAL AMERICAN UNIVERSITY (UCA), FIVE OTHER PRIESTS, AND TWO SALVADORAN WOMEN WERE MURDERED AT DAWN TODAY BY MEN WEARING MILITARY UNIFORMS.

Associated with the story is the template:

0. MESSAGE ID	DEV-MUC3-0050 (SITE)
1. TEMPLATE ID	1
2. DATE OF INCIDENT	16 NOV 89
3. TYPE OF INCIDENT	MURDER
4. CATEGORY OF INCIDENT	-
5. PERPETRATOR: ID OF INDIV(S)	"MEN WEARING MILITARY UNIFORMS" / "MEN"
6. PERPETRATOR: ID OF ORG(S)	-
7. PERPETRATOR: CONFIDENCE	-
8. PHYSICAL TARGET: ID(S)	*
9. PHYSICAL TARGET: TOTAL NUM	*
10. PHYSICAL TARGET: TYPE(S)	*
11. HUMAN TARGET: ID(S)	"JESUITS"
"	IGNACIO ELLACURIA" ("JESUIT" / "SPANISH JESUIT" / "RECTOR OF EL SALVADOR'S CENTRAL AMERICAN UNIVERSITY" / "RECTOR OF EL SALVADOR'S CENTRAL AMERICAN UNIVERSITY (UCA)" / "RECTOR")
	"PRIESTS"
	"WOMEN"
12. HUMAN TARGET: TOTAL NUM	8
13. HUMAN TARGET: TYPE(S)	CIVILIAN: "WOMEN"
	CIVILIAN: "PRIESTS"
	CIVILIAN: "IGNACIO ELLACURIA"
	CIVILIAN: "JESUITS"
14. TARGET: FOREIGN NATION(S)	SPAIN: "IGNACIO ELLACURIA"
15. INSTRUMENT: TYPE(S)	-
16. LOCATION OF INCIDENT	EL SALVADOR
17. EFFECT ON PHYSICAL TARGET(S)	*
18. EFFECT ON HUMAN TARGET(S)	*

The template is handed to the phrase extractor, and the following list of phrases results.

```
(HUMAN JESUITS
      IGNACIO ELLACURIA
      JESUIT
      SPANISH JESUIT
       RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
       UNIVERSITY
       RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
       UNIVERSITY (UCA)
      RECTOR
      PRIESTS
      WOMEN)
 (PHYS)
 (PERP-ORG)
 (PERP-IND MEN WEARING MILITARY UNIFORMS
       MEN)
 (TYPE MURDER)
```

Phrases are only extracted for slots 3 (incident type), 5 (perpetrator individuals), 6 (perpetrator organizations), 8 (physical targets), and 11 (human targets). In the extracted list of phrases, slot 3 is denoted by TYPE, slot 5 by PERP-IND, slot 6 by PERP-IND, slot 8 by PHYS, and slot 11 by HUMAN. The list of phrases shows that slots 6 and 8 were empty, slot 3 had one filler, slot 5 two fillers, and slot 11 had 9 fillers. Phrase extraction does not distinguish between conjunctions and disjunctions of phrases, nor between proper names and titles.

The phrase indexer searches the story for occurrences of each of the extracted phrases, and results in the following list of indexed phrases.

```
(MURDER
 (0 HUMAN JESUITS)
 (3 HUMAN IGNACIO ELLACURIA)
 (3 HUMAN JESUIT)
 (3 HUMAN SPANISH JESUIT)
 (3 HUMAN RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
    UNIVERSITY)
 (3 HUMAN RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
   UNIVERSITY (UCA))
 (3 HUMAN RECTOR)
 (3 HUMAN PRIESTS)
 (3 HUMAN WOMEN)
 (3 PERP-IND MEN)
 (3 PERP-IND MEN WEARING MILITARY UNIFORMS))
```

This list shows that the phrase "JESUITS" appeared in the first sentence, while all other phrases appeared in the fourth sentence. The type slot in the original list of phrases serves as a label for the indexed phrases.

Running CIRCUS on the story results in one concept node for the first sentence, and one for the fourth sentence.[28] The concept node for the first sentence is:

```
***    TYPE = MURDER
***    VICTIM = WS-CLERGY
***    noun group = (JESUITS)
***    predicates = (&&6)
***    REL-LINK (LOC2  (EL_SALVADOR)))
***    REL-LINK (TIME  (NOV_16_89)))
```

[28]The fourth sentence actually produced three concept nodes, but the second two were redundant.

The concept node for the fourth sentence is:

```
***     TYPE = MURDER
***     ACTOR = WS-HUMAN
***     noun group = (MEN)
***     VICTIM = WS-PROPER-NAME
***      noun group = ((JESUIT IGNACIO ELLACURIA >APP RECTOR
                          OF EL @SALVADOR@S
                          CENTRAL_AMERICAN_UNIVERSITY)
                      (PRIESTS) (WOMEN))
***     predicates = ((SPANISH) (&&5 OTHER)
                      (&&2 SALVADORAN))
***     determiners = ((THE) NIL NIL)
***     REL-LINK (TIME  (NOV_16_89)))
```

The phrase matcher annotates indexed phrases with concept node slots, noting the degree of match. For this example, the phrase matcher returns the following:

```
(MURDER
 (0 VICTIM JESUITS) : CORRECT
 (3 VICTIM IGNACIO ELLACURIA) : PARTIAL
 (3 VICTIM JESUIT) : PARTIAL
 (3 VICTIM SPANISH JESUIT) : PARTIAL
 (3 VICTIM RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
   UNIVERSITY) : PARTIAL
 (3 VICTIM RECTOR OF EL SALVADOR'S CENTRAL AMERICAN
   UNIVERSITY (UCA)) : PARTIAL
 (3 VICTIM RECTOR) : PARTIAL
 (3 VICTIM PRIESTS) : CORRECT
 (3 VICTIM WOMEN) : CORRECT
 (3 PERP MEN) : CORRECT
 (3 PERP MEN WEARING MILITARY UNIFORMS) : PARTIAL)
```

Each of the indexed phrases at least partially matched a concept node slot. Note that the slot names ("PERP" and "VICTIM") now denote concept node slots rather than template slots. The type (MURDER) is carried over from the indexed phrases.

Finally, a case is extracted from the matched phrases. A case has two parts: a type and a set of sentence patterns. A case's type is the type of the template used to generate it. Each sentence pattern has two parts: the relative position of a sentence in the story from which the case was generated (denoted by an integer), and a list of concept node slots that appeared in that sentence. In this example, the following case results.

```
(MURDER
  (3 PERP
     VICTIM)
  (0 VICTIM))
```

This case describes a murder with a victim concept node slot appearing in one sentence, and perpetrator and victim slots appearing three sentences later.

4.3.2 Case Retrieval

Figure 4.3 illustrates the process of retrieving cases from the case base. There are four subprocesses to retrieval.

1. CIRCUS is run on a story, and produces a set of concept nodes.
2. The CN *grouper* determines a subset of concept nodes to be used for retrieval.
3. The *probe generator* takes as input a set of concept nodes, and returns a retrieval probe.
4. The *case matcher* attempts to find a case that matches a retrieval probe. If such a case is found, it is returned and instantiated as an event.

It is important to note that the CN grouper, probe generator, and case matcher will in general be run multiple times for each story (i.e., for each set of concept nodes that CIRCUS generates). In the current implementation, these three processes are run at least once for every concept node for a given process. The precise number of iterations is determined by the following algorithm.

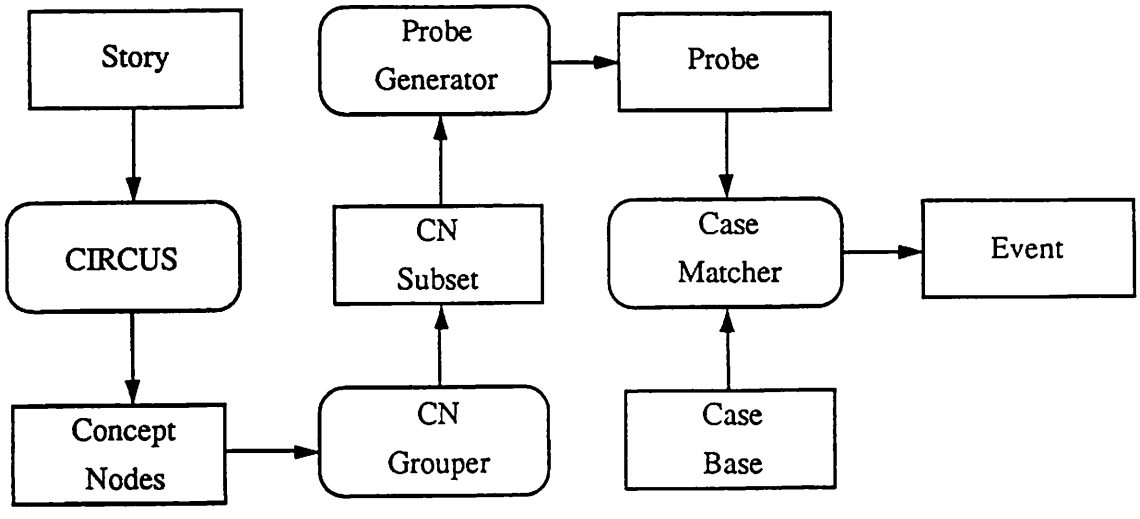

Figure 4.3: The retrieval process for a single story. Rectangles represent data, boxes with rounded corners represent operations.

1. FOR EACH CN c returned by CIRCUS:
2. Determine the type t of c.
3. Find the last successful retrieval r containing CNs of type t.
4. Group c with the CNs of r, yielding the group g.
5. Generate a probe p for g.
6. Attempt to find a case e that matches p.
7. If successful, then
8. create a new event by storing e with g. Discard event r.
9. else
10. Generate a probe p^1 for c.
11. Attempt to find a case e^1 that matches p^1.
12. If successful, create a new event by storing e^1 with c.

In this algorithm, the work of the CN grouper is distributed among lines 2, 3, and 4.

The probe used to retrieve a case is the set of sentence patterns that results from a set of concept nodes, labeled by the type of all of the concept nodes. A sentence pattern for a retrieval probe is much like a sentence pattern for a case. All non-empty slots for concept nodes triggered in the same sentence are grouped into one pattern. There are as many patterns as there are sentences in which concept nodes have been triggered.

A case matches a retrieval probe if it is of a compatible type and its sentence patterns subsume the sentence patterns in the probe. If no such case exists for a given probe, then retrieval fails for that probe.

Maximal probes are constructed by grouping CIRCUS output into maximal clusters that yield successful probes. In this way, we attempt to identify large groups of concept nodes that all contribute to the same template.

As an example of the retrieval process, consider the following story.

> THE BRAZILIAN EMBASSY IN COLOMBIA HAS CONFIRMED THE RELEASE OF REDE GLOBO JOURNALIST CARLOS MARCELO WHO WAS KIDNAPPED BY COLOMBIAN ARMY OF NATIONAL LIBERATION GUERRILLAS. MARCELO WAS WRITING AN ARTICLE ON THE KIDNAPPING OF THE THREE BRASPETRO [PETROBRAS INTERNATIONAL, INC.] ENGINEERS WHEN HE WAS ABDUCTED BY THE GUERRILLAS. THE GUERRILLAS SENT A COMMUNIQUE THROUGH THIS JOURNALIST SAYING THAT THEY WILL ONLY RELEASE THE ENGINEERS WHEN COLOMBIA DECIDES TO NATIONALIZE THE PETROLEUM SECTOR.

CIRCUS returns three interesting concept nodes for this story. The first, from the first sentence, is shown below.

```
***     TYPE = KIDNAPPING

***     ACTOR = WS-TERRORIST
***       noun group = (ARMY_OF_NATIONAL_LIBERATION
                            GUERRILLAS)
***     predicates = (COLOMBIAN)
***     VICTIM = WS-PROPER-NAME
***     noun group = (REDE GLOBO JOURNALIST CARLOS MARCELO)
```

The second concept node is triggered in the second sentence.

```
***          TYPE = KIDNAPPING

***          VICTIM = WS-ENERGY
***          noun group = (BRASPETRO)
***          predicates = (&&3)
***          determiners = (THE)
```

The third concept node, like the second, is triggered in the second sentence.

```
***          TYPE = KIDNAPPING

***          ACTOR = WS-TERRORIST
***          noun group = (GUERRILLAS)
***          determiners = (THE)
***          VICTIM = WS-HUMAN
***          noun group = (HE)
```

Figure 4.4 summarizes the retrieval process for the story's three concept nodes. Each rectangle in the figure represents a retrieval attempt. For each attempt, the set of probes at the time of retrieval is shown, with each probe pointing to the case retrieved for it (if any). There are four retrieval attempts, three of which are successful.

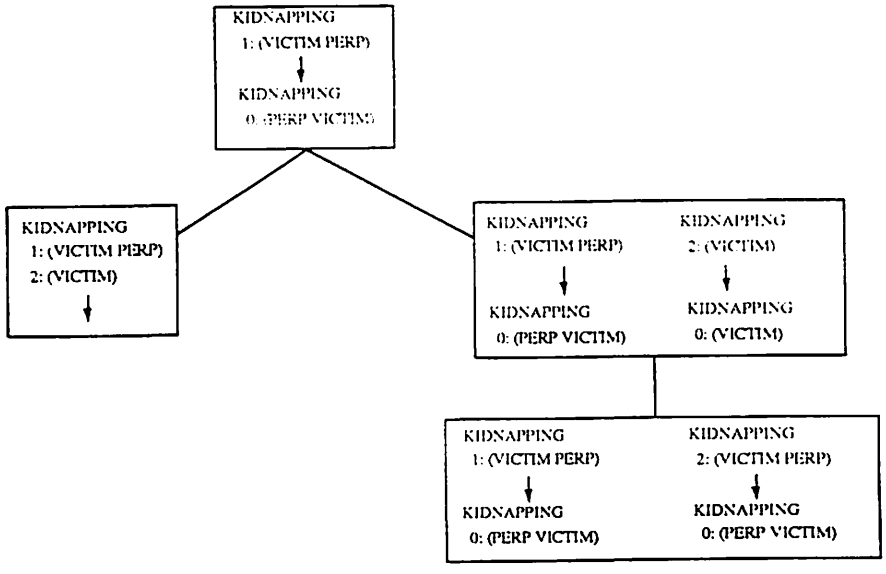

Figure 4.4: Probes and retrieved cases for each retrieval attempt in the example.

The first round of grouping, probe generation, and matching is done on the story's first concept node. Grouping produces the set containing that node. Probe generation produces the probe:

```
KIDNAPPING
  1: (VICTIM PERP)
```

A match is found between the probe and a case in the case base:

```
KIDNAPPING
  2: (VICTIM)
```

Next, the second concept node is grouped with the first. The following probe is produced:

```
KIDNAPPING
  1: (VICTIM PERP)
  2: (VICTIM)
```

There is no case that matches this probe. A new probe, derived from the second concept node by itself, is produced:

```
KIDNAPPING
  2: (VICTIM)
```

Matching produces the following case, and associates the second concept node with it.

```
KIDNAPPING
  0: (VICTIM)
```

Finally, the third concept node is grouped with the second. Together, they generate the probe:

```
KIDNAPPING
  2: (VICTIM PERP)
```

This retrieves the case:

```
KIDNAPPING
  0: (PERP VICTIM)
```

The event associating the second concept node with a case is discarded; both the second and third nodes are now associated with the newly retrieved case. The final set of cases is:

```
KIDNAPPING
  0: (PERP VICTIM)
KIDNAPPING
  0: (PERP VICTIM)
```

There are two maximal probes, one associated with each case. The first is made up of just the first concept node, while the second is made up of the second and third nodes.

4.3.3 Using Cases

Retrieval for the concept nodes in a story produces a set of events, each consisting of a type and a set of concept nodes. An event's concept nodes make up a maximal probe for a case retrieval; that case's type is the type of the event. It is assumed that each such event corresponds to a template, with the event's type corresponding to the template's type and the concept nodes contributing to the template's slot fills.

The process of extracting a template from an event is shown in figure 4.5. The event is handed to an *event analyzer*, which separates the event into a type and a set of concept nodes. These are then given as input to a *template builder*, which produces a template of the appropriate type and with the appropriate slots filled in.

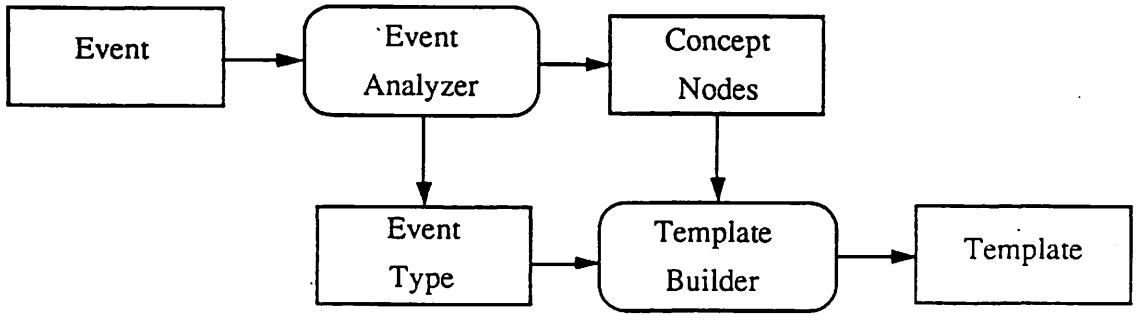

Figure 4.5: The template extraction process for a single template. Rectangles represent data, boxes with rounded corners represent operations.

For MUC-3, case-based consolidation worked in conjunction with rule- based consolidation. Rule-based consolidation was run first, and the case base was used to generate templates that the rules might have missed. The event analyzer thus served a secondary purpose of filtering out events for which the rule base had already generated templates.

To illustrate the template extraction process, let us return to the example in section 4.3.2. In this example, CIRCUS generated three kidnapping concept nodes, and retrieval hypothesized two kidnapping events from these concept nodes. Each of these events is handed to the event analyzer.

For this example, the rule base had already generated a single kidnapping template; thus, the event analyzer discards the first kidnapping event it is handed. The second event is divided into a KIDNAPPING type and a set of concept nodes and handed to the event builder, and the following template is created:

0. MESSAGE ID	TST2-MUC3-0100
1. TEMPLATE ID	2
2. DATE OF INCIDENT	- 26 MAY 89
3. TYPE OF INCIDENT	KIDNAPPING
4. CATEGORY OF INCIDENT	TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S)	"GUERRILLAS"
6. PERPETRATOR: ID OF ORG(S)	"ARMY OF NATIONAL LIBERATION"
7. PERPETRATOR: CONFIDENCE	REPORTED AS FACT: "ARMY OF NATIONAL LIBERATION"
8. PHYSICAL TARGET: ID(S)	*
9. PHYSICAL TARGET: TOTAL NUM	*
10. PHYSICAL TARGET: TYPE(S)	*
11. HUMAN TARGET: ID(S)	"HE"
12. HUMAN TARGET: TOTAL NUM	1
13. HUMAN TARGET: TYPE(S)	CIVILIAN: "HE"
14. TARGET: FOREIGN NATION(S)	-
15. INSTRUMENT: TYPE(S)	*
16. LOCATION OF INCIDENT	EL SALVADOR
17. EFFECT ON PHYSICAL TARGET(S)	*
18. EFFECT ON HUMAN TARGET(S)	-

The version of the event builder used here merges all of the concept nodes together and applies a default rule to the result.

The fourth sentence actually produced three concept nodes, but the second two were redundant.

4.4 Future Work

There are several problems with our system that suggest directions for future research. In general, consolidation does not have access to information that was not picked up by a concept node. For instance, secondary information about victims (titles, types, full names, etc.) is often provided by text that surrounds the sentence(s) that actually describe the event. Our system currently has no facility for getting this information. Along different lines, although our rule

base served us well in MUC-3, we suspect that we could do better by organizing events in a more intelligent manner. For example, our system might merge a bombing of target X with a different but compatible bombing near the top of the stack despite the fact that a bombing template for target X is already on the stack but has gotten buried underneath a different bombing. An intelligent memory organization would allow us to find the best match more naturally. In addition, it should allow us to handle forms of summary information that could not be handled easily by our current architecture.

We are also excited about the prospects for case-based consolidation. Our CBR module was developed very late in the project so there are still many avenues to explore. One possible approach is to integrate rule-based and case-based consolidation. By starting with only a small set of rules, the cases could be used to augment the rule base and expand its coverage. We would also like to experiment with different retrieval algorithms, case representations, and case bases. We find CBR to be particularly appealing because it holds great promise for improving portability and scale-up problems; since cases can be acquired automatically, a successful case-based solution could be transferred quickly and easily to new domains.

Chapter 5

Bibliography

1. Lehnert, W.G. "Symbolic/Subsymbolic Sentence Analysis: Exploiting the Best of Two Worlds", Technical Report No. 88-99, Department of Computer and Information Science, University of Massachusetts. 1988. Also available in *Advances in Connectionist and Neural Computation Theory*, Vol. I. (ed: J. Pollack and J. Barnden). pp. 135-164. Ablex Publishing, Norwood, New Jersey. 1991.

2. Lehnert, W., Cardie, C. and Riloff, E., "Analyzing Research Papers Using Citation Sentences," in *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Boston MA. pp. 511-518. 1990.

3. Wermter, S., "Integration of Semantic and Syntactic Constraints for Structural Noun Phrase Disambiguation", in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* pp. 1486-1491. 1989.

4. Wermter, S., "Learning Semantic Relationships in Compound Nouns with Connectionist Networks", in *Proceedings of the Eleventh Annual Conference on Cognitive Science.* pp. 964-971. 1989.

5. Wermter, S. and Lehnert, W.G., "A Hybrid Symbolic/Connectionist Model for Noun Phrase Understanding," in *Connection Science.* Vol. 1, No. 3. pp. 255-272. 1989.

6. Riesbeck, C., "Micro ELI", in *Inside Computer Understanding*, (eds: R. Schank and C. Riesbeck), pp. 354-372. Lawrence Erlbaum Associates. 1981.

7. Cardie, C. and Lehnert, W., "A Cognitively Plausible Approach to Understanding Complex Syntax," in *Proceedings of the Ninth National Conference on Artificial Intelligence.* Anaheim, CA. 1991.

Chapter 6

Appendices

Appendix A: CIRCUS Output for DEV-MUC3-0099

PARSE WILL RECEIVE:
(POLICE HAVE REPORTED THAT TERRORISTS ON OCT_25_89 >CO TONIGHT BOMBED THE EMBASSIES OF THE PRC AND THE SOVIET_UNION >PE)

Parse completed. The final representation =

```
==========================================================
***     TYPE = BOMBING
***     ACTOR = WS-TERRORIST
***     noun group = (TERRORISTS)
***     TARGET = WS-DIPLOMAT-OFFICE-OR-RESIDENCE
***     noun group = (EMBASSIES)
***     determiners = (THE)
***     REL-LINK (TIME  (OCT_25_89)))
==========================================================
```

PARSE WILL RECEIVE:
(THE BOMBS CAUSED DAMAGE BUT NO INJURIES >PE)

Parse completed. The final representation =

```
==========================================================
***     TYPE = WEAPON
***     INSTR = BOMB
==========================================================
***     TYPE = INJURY
***     MODE = NEG
==========================================================
```

PARSE WILL RECEIVE:
(A CAR_BOMB EXPLODED IN_FRONT_OF THE PRC EMBASSY >CO IN THE LIMA RESIDENTIAL DISTRICT OF SAN_ISIDRO >PE)

Parse completed. The final representation =

```
==========================================================
***     TYPE = WEAPON
***     INSTR = CAR_BOMB
***     REL-LINK (TARGET  (PRC EMBASSY)))
***     REL-LINK (LOC2  (LIMA DISTRICT)))
```

```
*** TYPE = BOMBING
*** INSTR =
  >>>          TYPE = WEAPON
  >>>          INSTR = CAR_BOMB
  >>>          REL-LINK (TARGET OBJECT (WS-DIPLOMAT-OFFICE-OR-RESIDENCE))
  >>>          REL-LINK (LOC2 OBJECT (WS-GENERIC-LOC))
*** noun group = (CAR_BOMB)
*** determiners = (A)
*** REL-LINK (TARGET (PRC EMBASSY)))
*** REL-LINK (LOC2 (LIMA DISTRICT)))
```

PARSE WILL RECEIVE:
(MEANWHILE >CO &&2 BOMBS WERE THROWN AT A USSR EMBASSY VEHICLE THAT WAS PARKED IN_FRONT_OF THE EMBASSY LOCATED IN ORRANTIA DISTRICT >CO NEAR SAN_ISIDRO >PE)

Parse completed. The final representation =

```
*** TYPE = MARKER
*** SUBTYPE = NEW-EVENT
```

```
*** TYPE = WEAPON
*** INSTR = BOMB
```

```
*** TYPE = BOMBING
*** INSTR =
  >>>          TYPE = WEAPON
  >>>          INSTR = BOMB
*** noun group = (BOMBS)
*** predicates = (&&2)
*** TARGET = WS-TRANSPORT-VEHICLE
*** noun group = (USSR EMBASSY VEHICLE)
*** determiners = (A)
```

PARSE WILL RECEIVE:
(POLICE SAID THE ATTACKS WERE CARRIED_OUT ALMOST SIMULTANEOUSLY AND THAT THE BOMBS BROKE WINDOWS AND DESTROYED THE &&2 VEHICLES >PE)

Parse completed. The final representation =

```
*** TYPE = ATTACK
```

```
*** TYPE = WEAPON
*** INSTR = BOMB
```

```
*** TYPE = DESTRUCTION
*** EFFECT = SOME_DAMAGE
*** INSTR =
  >>>          TYPE = WEAPON
  >>>          INSTR = BOMB
*** noun group = (BOMBS)
*** determiners = (THE)
*** OBJECT = WS-PHYS-TARGET
*** noun group = (WINDOWS)
```

```
*** TYPE = DESTRUCTION
*** EFFECT = DESTROYED
*** INSTR =
  >>>        TYPE = WEAPON
  >>>        INSTR = BOMB
*** noun group = (BOMBS)
*** determiners = (THE)
*** OBJECT = WS-TRANSPORT-VEHICLE
*** noun group = (VEHICLES)
*** predicates = (&&2)
*** determiners = (THE)
```

PARSE WILL RECEIVE:
(NO_ONE HAS CLAIMED RESPONSIBILITY FOR THE ATTACKS SO_FAR >PE)

Parse completed. The final representation =

```
*** TYPE = ATTACK
```

PARSE WILL RECEIVE:
(POLICE SOURCES >CO HOWEVER >CO HAVE SAID THE ATTACKS COULD HAVE BEEN CARRIED_OUT BY THE MAOIST SHINING_PATH GROUP OR THE GUEVARIST TUPAC_AMARU_REVOLUTIONARY_MOVEMENT GROUP >PE)

Parse completed. The final representation =

```
*** TYPE = PERPETRATOR
*** CONFIDENCE = POSSIBLE
*** NEW-INFO = T
*** PERPETRATOR = WS-TERRORIST
*** noun group = ((SHINING_PATH GROUP)
    (TUPAC_AMARU_REVOLUTIONARY_MOVEMENT GROUP))
*** predicates = ((MAOIST) (GUEVARIST))
*** determiners = ((THE) (THE))
```

PARSE WILL RECEIVE:
(THE SOURCES ALSO SAID THAT THE SHINING_PATH HAS ATTACKED SOVIET INTERESTS IN PERU ON -DEC_31_80 >CO IN THE PAST >CO >PE)

Parse completed. The final representation =

```
*** TYPE = MARKER
*** SUBTYPE = POSSIBLE-NEW-EVENT
```

```
*** TYPE = ATTACK
*** ACTOR = WS-ORGANIZATION
*** noun group = (SHINING_PATH)
*** determiners = (THE)
*** TARGET = WS-ENTITY
*** noun group = (INTERESTS)
*** predicates = (SOVIET)
```

```
***     VICTIM = WS-ENTITY
***     noun group = (INTERESTS)
***     predicates = (SOVIET)
***     REL-LINK (LOC2  (PERU)))
***     REL-LINK (TIME  (-DEC_31_80)))
```

PARSE WILL RECEIVE:
(ON JUL_01_89-JUL_31_89 >CO THE SHINING_PATH BOMBED A BUS CARRYING NEARLY &&50 SOVIET MARINES INTO THE PORT OF EL_CALLAO >PE)

Parse completed. The final representation

```
***     TYPE = MARKER
***     SUBTYPE = NEW-EVENT
```

```
***     TYPE = BOMBING
***     ACTOR = WS-ORGANIZATION
***     noun group = (SHINING_PATH)
***     determiners = (THE)
***     TARGET = WS-TRANSPORT-VEHICLE
***     noun group = (BUS)
***     determiners = (A)
***     REL-LINK (TIME  (JUL_01_89-JUL_31_89)))
```

PARSE WILL RECEIVE:
(FIFTEEN SOVIET MARINES WERE WOUNDED >PE)

Parse completed. The final representation =

```
***     TYPE = INJURY
***     VICTIM = WS-ACTIVE-MILITARY
***     noun group = (FIFTEEN MARINES)
***     predicates = (SOVIET)
```

PARSE WILL RECEIVE:
(ON -DEC_31_80 >CO SOME &&3 YEARS AGO >CO &&2 MARINES DIED FOLLOWING A SHINING_PATH BOMBING OF A MARKET USED BY SOVIET MARINES >PE)

Parse completed. The final representation =

```
***     TYPE = MARKER
***     SUBTYPE = NEW-EVENT
***     REL-LINK (TIME  (-DEC_31_80)))
```

```
***     TYPE = MURDER
***     VICTIM = WS-ACTIVE-MILITARY
***     noun group = (MARINES)
***     predicates = (&&2)
***     REL-LINK (TIME  (-DEC_31_80)))
```

PARSE WILL RECEIVE:
(IN ANOTHER INCIDENT ON -DEC_31_80 >CO &&3 YEARS AGO >CO >CO A SHINING_PATH MILITANT WAS KILLED BY SOVIET EMBASSY GUARDS INSIDE THE EMBASSY COMPOUND >PE)

Parse completed. The final representation =

```
*** TYPE = MARKER
*** SUBTYPE = NEW-EVENT
*** REL-LINK (TIME (-DEC_31_80)))
```

```
*** TYPE = MURDER
*** ACTOR = WS-DIPLOMAT
*** noun group = (EMBASSY GUARDS)
*** predicates = (SOVIET)
*** VICTIM = WS-TERRORIST
*** noun group = (SHINING_PATH MILITANT)
*** determiners = (A)
*** REL-LINK (TIME (-DEC_31_80)))
```

PARSE WILL RECEIVE:
(THE TERRORIST WAS CARRYING DYNAMITE >PE)

Parse completed. The final representation =

```
*** TYPE = WEAPON
*** INSTR = DYNAMITE
```

PARSE WILL RECEIVE:
(THE ATTACKS ON OCT_25_89 >CO TODAY COME AFTER SHINING_PATH ATTACKS DURING WHICH LEAST &&10 BUSES WERE BURNED THROUGHOUT LIMA ON OCT_24_89 >CO >PE)

Parse completed. The final representation =

```
*** TYPE = ATTACK
*** VICTIM = WS-TIME
*** noun group = (OCT_25_89)
*** TARGET = WS-TIME
*** noun group = (OCT_25_89)
*** LOC1 = WS-TIME
*** noun group = (OCT_25_89)
*** REL-LINK (TIME (OCT_25_89)))
```

```
*** TYPE = ATTACK
*** REL-LINK (TIME (OCT_25_89)))
```

```
*** TYPE = ARSON
*** EFFECT = SOME_DAMAGE
*** OBJECT = WS-TRANSPORT-VEHICLE
*** noun group = (BUSES)
*** predicates = (&&10)
*** REL-LINK (LOC2 (LIMA)))
*** REL-LINK (TIME (OCT_24_89)))
```

Appendix B: Rule-Based Consolidation Output for DEV-MUC3-0099

0. MESSAGE ID	DEV-MUC3-0099
1. TEMPLATE ID	1
2. DATE OF INCIDENT	25 OCT 89
3. TYPE OF INCIDENT	BOMBING
4. CATEGORY OF INCIDENT	TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S)	"TERRORISTS"
6. PERPETRATOR: ID OF ORG(S)	-
7. PERPETRATOR: CONFIDENCE	-
8. PHYSICAL TARGET: ID(S)	"EMBASSIES OF THE PRC AND THE SOVIET UNION" "PRC EMBASSY"
9. PHYSICAL TARGET: TOTAL NUM	PLURAL
10. PHYSICAL TARGET: TYPE(S)	DIPLOMAT OFFICE OR RESIDENCE: "EMBASSIES OF THE PRC AND THE SOVIET UNION" DIPLOMAT OFFICE OR RESIDENCE: "PRC EMBASSY"
11. HUMAN TARGET: ID(S)	-
12. HUMAN TARGET: TOTAL NUM	-
13. HUMAN TARGET: TYPE(S)	-
14. TARGET: FOREIGN NATION(S)	PEOPLES REP OF CHINA: "EMBASSIES OF THE PRC AND THE SOVIET UNION" PEOPLES REP OF CHINA: "PRC EMBASSY"
15. INSTRUMENT: TYPE(S)	*
16. LOCATION OF INCIDENT	PERU: LIMA (CITY): SAN ISIDRO (NEIGHBORHOOD)
17. EFFECT ON PHYSICAL TARGET(S)	-
18. EFFECT ON HUMAN TARGET(S)	NO INJURY OR DEATH: "-"

0. MESSAGE ID	DEV-MUC3-0099
1. TEMPLATE ID	2
2. DATE OF INCIDENT	25 OCT 89
3. TYPE OF INCIDENT	BOMBING
4. CATEGORY OF INCIDENT	TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S)	"MAOIST SHINING PATH GROUP" "GUEVARIST TUPAC AMARU REVOLUTIONARY MOVEMENT GROUP"
6. PERPETRATOR: ID OF ORG(S)	"SHINING PATH" "TUPAC AMARU REVOLUTIONARY MOVEMENT"
7. PERPETRATOR: CONFIDENCE	POSSIBLE: "SHINING PATH" POSSIBLE: "TUPAC AMARU REVOLUTIONARY MOVEMENT"
8. PHYSICAL TARGET: ID(S)	"WINDOWS" "VEHICLES" "USSR EMBASSY VEHICLE"
9. PHYSICAL TARGET: TOTAL NUM	PLURAL
10. PHYSICAL TARGET: TYPE(S)	OTHER: "WINDOWS" TRANSPORT VEHICLE: "VEHICLES" TRANSPORT VEHICLE: "USSR EMBASSY VEHICLE"
11. HUMAN TARGET: ID(S)	-
12. HUMAN TARGET: TOTAL NUM	-
13. HUMAN TARGET: TYPE(S)	-
14. TARGET: FOREIGN NATION(S)	USSR: "USSR EMBASSY VEHICLE"
15. INSTRUMENT: TYPE(S)	*
16. LOCATION OF INCIDENT	PERU
17. EFFECT ON PHYSICAL TARGET(S)	SOME DAMAGE: "WINDOWS" DESTROYED: "VEHICLES"
18. EFFECT ON HUMAN TARGET(S)	-

0. MESSAGE ID	DEV-MUC3-0099
1. TEMPLATE ID	3
2. DATE OF INCIDENT	24 OCT 89
3. TYPE OF INCIDENT	ARSON
4. CATEGORY OF INCIDENT	TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S)	"GUEVARIST TUPAC AMARU REVOLUTIONARY MOVEMENT GROUP" "MAOIST SHINING PATH GROUP"
6. PERPETRATOR: ID OF ORG(S)	"TUPAC AMARU REVOLUTIONARY MOVEMENT" "SHINING PATH"
7. PERPETRATOR: CONFIDENCE	POSSIBLE: "TUPAC AMARU REVOLUTIONARY MOVEMENT" POSSIBLE: "SHINING PATH"
8. PHYSICAL TARGET: ID(S)	"BUSES"
9. PHYSICAL TARGET: TOTAL NUM	10
10. PHYSICAL TARGET: TYPE(S)	TRANSPORT VEHICLE: "BUSES"
11. HUMAN TARGET: ID(S)	-
12. HUMAN TARGET: TOTAL NUM	-
13. HUMAN TARGET: TYPE(S)	-
14. TARGET: FOREIGN NATION(S)	-
15. INSTRUMENT: TYPE(S)	*
16. LOCATION OF INCIDENT	PERU: LIMA (CITY)
17. EFFECT ON PHYSICAL TARGET(S)	SOME DAMAGE: "BUSES"
18. EFFECT ON HUMAN TARGET(S)	-

NIL
> (dribble)

Appendix C: Trace of Case-Based Consolidation for DEV-MUC3-0099

;;; The Template Type CB hypothesizes a bombing.
template type: BOMBING: (PERP TARGET TYPE)
from case: BOMBING: (TYPE PERP TARGET)

;;; The Template Pattern CB finds a matching pattern.
template pattern: ($BOMBING-3$)
BOMBING
1: (TARGET PERP)
retrieved template case:
BOMBING
0: (PERP TARGET)
3: (TARGET)

;;; Several more bombings are encountered, but none finds a new template pattern.
template type: BOMBING: (PERP TARGET TYPE)
from case: BOMBING: (TYPE PERP TARGET)

template type: BOMBING: (PERP TYPE)
from case: BOMBING: (TYPE PERP TARGET)

template type: BOMBING: (TARGET TYPE)
from case: BOMBING: (TYPE PERP TARGET)

template type: BOMBING: (TARGET TYPE)
from case: BOMBING: (TYPE PERP TARGET)

template type: BOMBING: (PERP TYPE)
from case: BOMBING: (TYPE PERP TARGET)

;;; The Template Type CB hypothesizes an attack.
template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP TYPE)
from case: ATTACK: (TYPE PERP)

;;; An attack CN is defined to be compatible with a bombing pattern.
;;; A new bombing pattern is established.
template pattern: ($ATTACK-6A$)
ATTACK
8: (PERP)

retrieved template case:
KIDNAPPING
 0: (PERP)
 3: (PERP)

template type: ATTACK: (PERP TARGET VICTIM TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

;;; The bombing pattern now contains 2 attack CNs.
template pattern: ($ATTACK-6$ $ATTACK-6A$)
ATTACK
 8: (VICTIM TARGET PERP)
retrieved template case:
BOMBING
 0: (PERP TARGET VICTIM)

template type: BOMBING: (PERP TARGET TYPE)
from case: BOMBING: (TYPE PERP TARGET)

;;; A third bombing pattern is established.
template pattern: ($BOMBING-3$)
BOMBING
 9: (TARGET PERP)
retrieved template case:
BOMBING
 0: (PERP TARGET)
 3: (TARGET)

template type: ATTACK: (VICTIM TYPE)
from case: ATTACK: (TYPE VICTIM)

;;; An injury CN is defined to be compatible with an attack pattern.
;;; An attack pattern is established.
template pattern: ($INJURY-PASS-1A$)
INJURY
 10: (VICTIM)
retrieved template case:
ATTACK
 0: (VICTIM)

template type: ATTACK: (VICTIM TYPE)
from case: ATTACK: (TYPE VICTIM)

;;; The attack pattern now contains 2 CNs.
template pattern: ($INJURY-PASS-1$ $INJURY-PASS-1A$)
INJURY
 10: (VICTIM)
retrieved template case:
ATTACK
 0: (VICTIM)

;;; A murder type is hypothesized.
template type: MURDER: (VICTIM TYPE)
from case: MURDER: (TYPE PERP VICTIM)

```
;;; A murder pattern is established.
template pattern: ($DIE$)
MURDER
  11: (VICTIM)
retrieved template case:
MURDER
  0: (VICTIM)
  4: (PERP)

template type: MURDER: (PERP VICTIM TYPE)
from case: MURDER: (TYPE PERP VICTIM)

;;; The murder pattern now contains 2 CNs.
template pattern: ($KILL-PASS-1A$ $DIE$)
MURDER
  11: (VICTIM)
  12: (VICTIM PERP)
retrieved template case:
MURDER
  0: (PERP VICTIM)
  1: (PERP VICTIM)

template type: MURDER: (PERP VICTIM TYPE)
from case: MURDER: (TYPE PERP VICTIM)

;;; The murder pattern now contains 3 CNs.
template pattern: ($KILL-PASS-1$ $KILL-PASS-1A$ $DIE$)
MURDER
  11: (VICTIM)
  12: (VICTIM PERP)
retrieved template case:
MURDER
  0: (PERP VICTIM)
  1: (PERP VICTIM)

template type: MURDER: (VICTIM TYPE)
from case: MURDER: (TYPE PERP VICTIM)

;;; The murder pattern now contains 4 CNs.
template pattern: ($KILL-PASS-2$ $KILL-PASS-1$ $KILL-PASS-1A$ $DIE$)
MURDER
  11: (VICTIM)
  12: (PERP VICTIM)
retrieved template case:
MURDER
  0: (PERP VICTIM)
  1: (PERP VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)
```

```
template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

template type: ATTACK: (PERP VICTIM TARGET TYPE)
from case: ATTACK: (TYPE PERP TARGET VICTIM)

;;; An arson type is hypothesized.
template type: ARSON: (PERP TYPE)
from case: ARSON: (TYPE PERP)

;;; The final outcome.
========== types =========
BOMBING: (PERP TARGET TYPE)
ATTACK: (PERP VICTIM TARGET TYPE)
MURDER: (VICTIM TYPE)
ARSON: (PERP TYPE)
========== patterns =========
BOMBING
  0: (PERP TARGET)
  3: (TARGET)
   $BOMBING-3$
BOMBING
  0: (PERP TARGET VICTIM)
    $ATTACK-6$ $ATTACK-6A$
BOMBING
  0: (PERP TARGET)
  3: (TARGET)
   $BOMBING-3$
ATTACK
  0: (VICTIM)
   $INJURY-PASS-1$ $INJURY-PASS-1A$
MURDER
  0: (PERP VICTIM)
  1: (PERP VICTIM)
   $KILL-PASS-2$ $KILL-PASS-1$ $KILL-PASS-1A$ $DIE$

;;; Consolidation structures are created for one of the bombings,
;;; the attack, and the murder. Structures are not created for
;;; two of the bombings and the arson because there already exist
;;; two bombing templates and one arson template.
Attempting to create consolidation structure for BOMBING type...
New consolidation structure from template case:
BOMBING
  0: (PERP TARGET)
  3: (TARGET)
   $BOMBING-3$

Attempting to create consolidation structure for ATTACK type...
New consolidation structure from template case:
ATTACK
  0: (VICTIM)
   $INJURY-PASS-1$ $INJURY-PASS-1A$
```

;;; Consolidation decides that the 4 murder CNs given it should actually
;;; create 2 c-structures.
Attempting to create consolidation structure for MURDER type...
New consolidation structure from template case:
MURDER
 0: (PERP VICTIM)
 1: (PERP VICTIM)
  $KILL-PASS-2$ $KILL-PASS-1$ $KILL-PASS-1A$ DIE
New consolidation structure from template case:
MURDER
 0: (PERP VICTIM)
 1: (PERP VICTIM)
  $KILL-PASS-2$ $KILL-PASS-1$ $KILL-PASS-1A$ DIE

Attempting to create consolidation structure for ARSON type...

;;; Consolidation deletes the two murder templates because of
;;; irrelevant victims, and deletes the bombing template because
;;; it has an old date.
Deleting frame MURDER because all irrelevant victims and targets
Deleting frame MURDER because all irrelevant victims and targets
Deleting frame BOMBING because it has an old date (JUL_01_89 - JUL_31_89)!

;;; The attack sneaks through, however, yielding one spurious template.

0. MESSAGE ID DEV-MUC3-0099 (SITE)
1. TEMPLATE ID 4
2. DATE OF INCIDENT - 25 OCT 89
3. TYPE OF INCIDENT ATTACK
4. CATEGORY OF INCIDENT TERRORIST ACT
5. PERPETRATOR: ID OF INDIV(S) "SOVIET EMBASSY GUARDS"
6. PERPETRATOR: ID OF ORG(S) "SHINING PATH"
7. PERPETRATOR: CONFIDENCE REPORTED AS FACT: "SHINING PATH"
8. PHYSICAL TARGET: ID(S) -
9. PHYSICAL TARGET: TOTAL NUM -
10. PHYSICAL TARGET: TYPE(S) -
11. HUMAN TARGET: ID(S) "FIFTEEN SOVIET MARINES"
12. HUMAN TARGET: TOTAL NUM PLURAL
13. HUMAN TARGET: TYPE(S) ACTIVE MILITARY: "FIFTEEN SOVIET MARINES"
14. TARGET: FOREIGN NATION(S) USSR: "FIFTEEN SOVIET MARINES"
15. INSTRUMENT: TYPE(S) -
16. LOCATION OF INCIDENT PERU
17. EFFECT ON PHYSICAL TARGET(S) -
18. EFFECT ON HUMAN TARGET(S) INJURY: "FIFTEEN SOVIET MARINES"