University of Manitoba: Description of the PIE System Used for MUC-6

*Dekang Lin**
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
lindek@cs.umanitoba.ca

Background

The PIE (Principar-driven Information Extraction) system takes a different approach to the problem of information extraction from the NUBA system that was used in MUC-5. The NUBA system did not have a parser and relies on an abductive reasoner to construct the semantic relationships between domain specific concepts mentioned in a sentence. The PIE system, on the other hand, relies heavily on a principle-based broad-coverage parser, called PRINCIPAR [2, 6, 8], that we have developed over the past three years. Most of the information extracted are directly "read-off" the parser outputs by a subtree pattern-matcher, bypassing the usual step of constructing semantic representations.

In spite of the radical difference between the high-level approaches of our MUC-5 and MUC-6 systems, over 85% of the code of the MUC-5 system was reused. This is largely due to the unified abductive view of different tasks in natural language understanding [7]. PRINCIPAR and the abductive semantic interpreter share the same message passing algorithm for abduction. They differ only in the contents of the messages and constraints on message propagation.

The architecture of the PIE system is shown in Figure 1. The processing is sequential. A text is first broken up into sentences. The lexical analyzer turns a stream of tokens into a lattice of lexical items. The lexical items may then be combined or deleted by lexical rules, which are responsible for recognizing named entities. PRINCIPAR takes the lattice of lexical items and output a dependency tree between the words in a sentence. PRINCIPAR attempts to construct a parse for the full sentence. However, when it fails to do that, it retrieves parse fragments that cover the complete sentence. Information is extracted from the dependency trees by a subtree pattern matcher. The format of NE and CO outputs do not meet the standard of the scoring software because of arbitrary insertion and deletion of white spaces. A separate program is used to resolved the differences.

The PIE system is implemented in about 38k lines of C++, about 33k to 34k lines were written before MUC-6. It contains an interpreter for LISP-like expressions so that all the knowledge structures, such as finite automata for finding sentence boundaries, the lexicon, the lexical rules, the grammar network, and extraction rules, are written in LISP-like expressions.

Performance in MUC-6

Considering the very limited time and human resources with which the PIE system was developed, it did very well in all of the four evaluation tasks, especially in coreference recognition. Table 1 summarizes the

*Part of this research was conducted when the author was on leave at MIT AI Lab

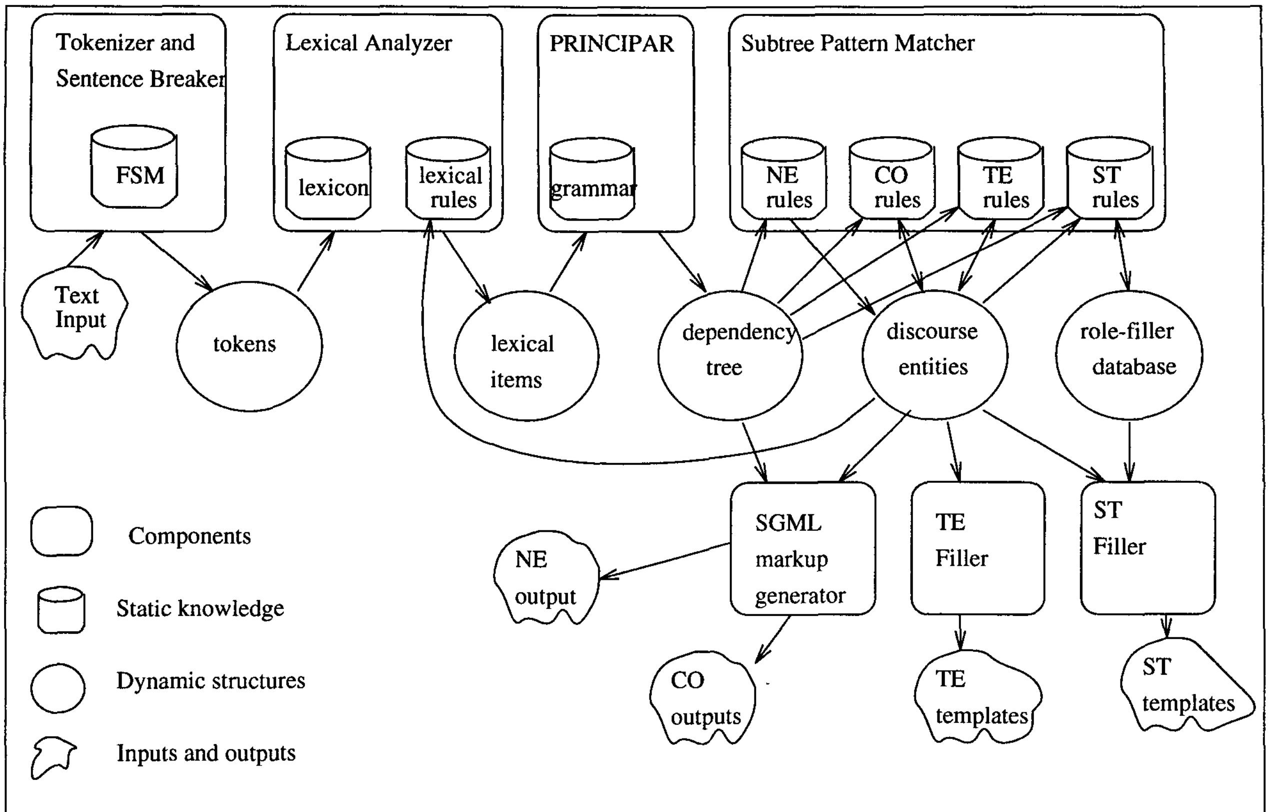


Figure 1: The PIE architecture

performance of PIE system in MUC-6.

	NE	CO	TE	ST
Recall	92	63	71	39
Precision	95	63	78	62
F-measure (P&R)	93.33		74.32	48.14
Time* (sec.)	618	683	2510	

*The time to run the MUC-6 formal tests (30 articles for NE and CO, 100 articles for TE and ST) on a Pentium-90 PC with 24MB memory.

Table 1: The Performance of PIE System in MUC-6

We have also included the timing data in Table 1, even though speed is not an evaluation criterion. A single timing datum is shown for TE and ST tests because they are generated in a single run. The PIE system runs quite fast. It takes from 20 to 25 seconds to process an article on an inexpensive PC. Most of the processing time was spent on parsing. The difference between the time for NE and CO is roughly the time taken by the coreference recognition algorithm.

Lexical Analysis and Named Entity Recognition (NE)

The lexicon we used contains only syntactic information such as parts of speech and subcategorization frames. Some proper noun entries have semantic features to indicate whether they are company names, company name designators, locations, family or given names, *etc.* Sample entries are listed in (1).

```
(1) (name
     (syn (C))
     (syn (Dn.n))
     (syn (Tn))
     )
    (succeed
     (syn (T[n]))
     (syn (Dn.pr.as))
     )
    (Japan
     (syn (PN (sem (+country))))
     (syn (Tn))
     (phrases (Japan Current) (Japan wax) (Japan Automobile Dealers' Association)
              (Japan Federation of Economic Organizations))
     )
```

The entries in the lexicon are organized in a similar way as paper dictionaries. Each entry consists of a head word and a list of usages. The symbols `C, Dn.n, Tn` and `T[n]`, *etc.* are abbreviations for bundles of features. The definitions of these abbreviations are stored in a separate file. Table 2 explains some of them. The use of abbreviations not only results in simpler and more readable lexicon files, but also makes the modifications to the entries much easier. For example, to modify the features of a class of words, one needs only to modify the abbreviation file.

Abbrev	Meaning	Features
C	countable noun	((cat n) (nform norm) +ct)
Tn	transitive verb	((cat v) (agr (N -adv)) (args (((cat n)))))
Dn.n	ditransitive verb	((cat v) (agr (N -adv)) (args (((cat n)) ((cat n)))))
PN	proper noun	((cat n) (nform norm) +pn)

Table 2: Sample definitions of abbreviations

The lexicon contains about 110K root entries. A small number of the entries are manually-coded. Others are extracted from machine-readable versions of Oxford Advanced Learner's Dictionary and Collins English Dictionary (both from the Oxford Text Archive), and public domain proper name lists from Consortium for Lexical Research at New Mexico State University.

Since the lexicon is derived from machine readable dictionaries, it contains many obscure usages of the words. For example, the word "japan" is also listed as a transitive verb[1].

When a word or a phrasal word is found in the lexicon, a lexical item is created for each of its usages. A lexical item consists of the position of the word in the sentence, the root form of the word, and its feature values. After the retrieval of all the words in a sentence, a set of lexical rules are matched against the lexical items. One of the purposes of lexical rules is to recognize relevant semantic entities. For example, (2) is a lexical rule for recognizing references to money.

```
(2) (defrule money2
        (seq (! (word (reg-match [0-9][0-9]*[.]?[0-9]*$))
```

[1] To cover (wood or metal) with a special kind of paint that gives a black shiny surface, as first done in Japan (Longman Dictionary of Contemporary English).

```
        (att-val (sem "AttVec*" (contain sem (+number)))))
     (? (word (in (hundred thousand million billion trillion))))
     (* (word (in ../dict/moneymod.txt)))
     (word (in ../dict/moneyunit.txt)))
(prog
  (delete-smaller (low) (high))
  (assert (low) (high) (attvec syn (PN (sem (+money)))))))
```

A lexical rule consists of a pattern and an action. Both are specified with LISP-like expressions. The pattern in (2) consists of four components. The first component matches a word that is made up of a sequence of digits or a lexical item that has the semantic feature **+number**. The second component matches an empty string or one of {**hundred thousand million billion trillion**}. The third component matches a sequence of zero or more modifying word for money, such as "New," "Taiwan," "Canadian," *etc.* The last component matches a money unit, such as "dollar," "cent," "dong," or "franc."

If all the four components in the pattern in (2) are matched by a sequence of lexical items or words, the action part of the rule will be executed:

- Lexical items that span on a subsequence of the match will be deleted.
- A lexical item will be created that spans on the sequence of words and contains the semantic feature value +money.

The lexical rules are also used to recognize names of organizations, persons, locations, time expressions, and dates. The rule (3) detects sequences of capitalized words that are not recognized as any of the above categories, it then calls the function fund-prev-occ to find whether or not this sequence of words is a substring of a previously recognized proper noun. If it is, the sequence words will have the same semantic features as the previously recognized entity.

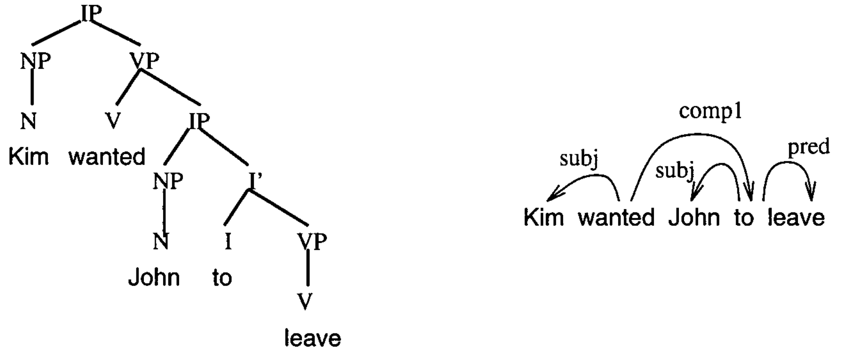
```
(3)   (defrule cap-word-sequence
        (seq (* (seq (word (and (reg-match [A-Z]) (not (in ../dict/begwords.txt))))
                     (? (word (str= -)))))
             (word (and (reg-match [A-Z]) (not (in ../dict/begwords.txt))))
             (! (word (reg-match [^A-Z]))
                ($)))
        (when
          (not (exist-lexitem
               (int -1) ($ 2 (high))
               (pred
                 (and (att (defined (sem)))
                      (att (value sem "AttVec*"
                             (defined-one-of sem
                               (money time date corpname percent location city
                                province person country))))))))
          (find-prev-occ (low) ($ 2 (high)))
          (isolate (low) ($ 2 (high)))))
```

Principle-based Parsing

PRINCIPAR is a principle-based parser [6, 8]. It has reasonablely broad coverage for the English language. An evaluation of PRINCIPAR with the SUSANNE corpus [10] shows that it is able to correctly identify dependency relationships for about 75% of the words [5]. PRINCIPAR is also very efficient. Sentences with 50-60 words can usually be parsed within 10 seconds on a Pentium-90 PC. The longest sentences in the SUSANNE corpus, consisting of 215 words, was parsed in 56 seconds.

Given an input sentence, such as (4), PRINCIPAR returns either a constituency tree (Figure 2a) or a dependency tree (Figure 2b).

(4) Kim wanted John to leave



a. a constituency tree

b. a dependency tree

Figure 2: Constituency Tree vs. Dependency Tree

In a dependency tree [9], every word in the sentence is a modifier of exactly one other word (called its head or modifiee), except the head word of the sentence. The dependency tree in Figure 2b is output as (5).

(5)
```
(
  (Kim      =      N        <      wanted subj)
  (wanted   want   V:IP     *             )
  (John     =      N        <      to     subj)
  (to       =      I        >      wanted compl)
  (leave    =      V:[NP]   >      to     pred)
)
```

Each row in (5) is a tuple that corresponds to a word in the input sentence. A tuple has the format:

`(word root cat position modifiee relationship)`

where

`word` is a word in the sentence;

`root` is the root form of `word`; if root is "=", then `word` is in root form;

`cat` is the lexical category or subcategory of `word`; V:IP is the subcategory of verbs that take an IP as the complement; V:[NP] is the subcategory of verbs that take an optional NP as a complement;

`modifiee` is the word that `word` modifies;

`position` indicates the position of `modifiee` relative to `word`. It can take one of the following values: {<, >, <<, >>, <<<, ..., *}, where < (or >) means that the `modifiee` of `word` is the first occurrence to the left (or right) of `word`; << (or >>) means `modifiee` is the second occurrence to the left (or right) of `word`. If `position` is '*', then the word is the head of the sentence;

`relationship` is the type of the dependency relationship between `modifiee` and `word`, such as `subj` (subject), `adjn` (adjunct), `compl` (first complement), `spec` (specifier), *etc.*

Subtree Pattern Matching

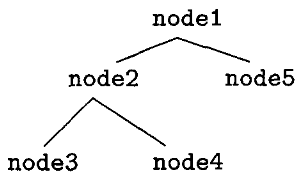
Once a dependency tree is constructed by the parser, the subtree pattern matcher searches the tree for partially specified subtrees and acts on the matches. Each pattern can be matched against nodes in a tree in either top-down or bottom-up order.

Rules in the subtree pattern matcher are in the following form, where CONDITION and ACTION are LISP-like expressions.

```
RULE     ::=  (down PATTERN ACTION)                      ;; a top-down rule
              (up PATTERN ACTION)                        ;; a bottom-up rule
PATTERN  ::=  (node CONDITION)                           ;; matches a single node
              (tree CONDITION (PATTERN ... PATTERN))     ;; matches a subtree
```

Example: The pattern for matching the subtree in (6a) has the format in (6b):

(6) a.



b.

```
(tree <condition-on-node1>
      ((tree <condition-on-node2>
             ((node <condition-on-node3>))
             ((node <condition-on-node4>)))
       (node <condition-on-node5)))
```

Example: The pattern in (7) matches a node in the dependency tree whose surface string is capitalized and is not one of the words listed in "../dict/prons.txt".

```
(7) (up (node (node2word (and (reg-match [A-Z])
                              (not (in ../dict/prons.txt)))))
      (when (not (comp (muc-state) (str= HL)))
        (bind-pn-to-same (match))))
```

If the sentence being processed is not inside <HL> and </HL> (i.e., a part of the headline), the action part of (7) passes the node to a function, called bind-pn-to-same, which searches the previous text for the same proper noun and asserts that the two occurrences are equivalent.

Coreference Recognition (CO)

PIE maintains a set of discourse entities and constraints between them. Each discourse entity is an equivalence class of noun phrases. The discourse entities are constructed sequentially by a list of equality and inequality assertions. Equality assertions may cause equivalence classes to be merged. Inequality assertions are recorded as constraints. Each assertion is assigned a weight. The higher the weight, the higher the likelihood that the assertion is true. The assertions are accepted in the decreasing order of their weights. Assertions that contradict previous assertions are ignored.

Example: Suppose there are 6 elements: a, b, c, d, e, and f. The following table shows equivalent classes and constraints after each assertion:

Assertion	Weight	Classes	Constraints	Comments
initial state		[a] [b] [c] [d] [e] [f]		
equiv(a, b)	100	[a b] [c] [d] [e] [f]		
diff(a, c)	50	[a b] [c] [d] [e] [f]	a≠c	
equiv(d, f)	40	[a b] [c] [d f] [e]	a≠c	
equiv(c, f)	40	[a b] [c d f] [e]	a≠c	
equiv(b, d)	30	[a b] [c d f] [e]	a≠c	contradicts a≠c, ignored
diff(c, d)	30	[a b] [c d f] [e]	a≠c	c and d already in same class, ignored
diff(e, d)	20	[a b] [c d f] [e]	a≠c e≠d	
equiv(c, e)	10	[a b] [c d f] [e]	a≠c e≠d	contradict e≠d, ignored

This representation of discourse entities does not determine or restrict how the equivalence and difference relationships are proposed. Its function is to integrate different (sometimes conflicting) assertions that come from different sources.

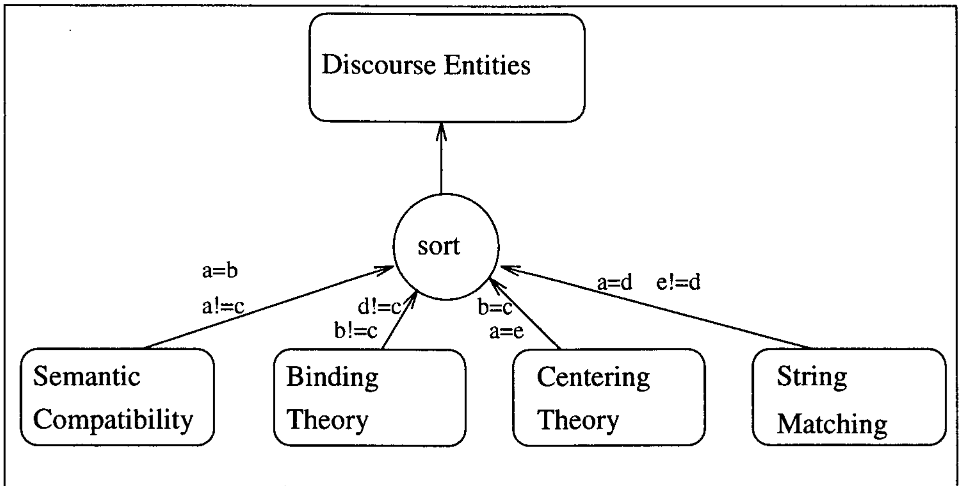


Figure 3: An Architecture for Coreference Recognition

In PIE, the coreference assertions are proposed by the following knowledge sources:

- Assumptions about proper names
  - Identical proper nouns in a text refer to the same entity.
  - A pair of proper nouns in which neither is a substring of the other refer to different entities.
- Binding Theory constraints
  - Reflexive pronouns must refer to a c-commanding noun phrase in the same clause or the same noun phrase.
  - A (non-reflexive) pronoun cannot refer to a c-commanding noun phrase in the same clause or the same noun phrase.
  - Other noun phrases (R-expressions) cannot refer to a c-commanding noun phrase in the same sentence.

  In Government-Binding theory [3], the c-command relationship is defined as follows: a node α c-commands another node β if (a) α does not dominate β and (b) the parent of α dominates β. Since we use dependency structures instead of constituency structures, the c-command relationship is defined as follows: a word α c-commands another word β if (a) α or its modifiee dominates β; and (b) α precedes β.
- The WordNet [1]

  For each pair of noun phrases (NP1, NP2), where NP2 precedes NP1 in the text, a manually constructed decision tree computes a weight, using the classification of the heads of NP1 and NP2 in the WordNet [1], as well as their semantic and syntactic features. If the weight is negative, NP1 and NP2 are incompatible, "`diff(NP1, NP2)`" is asserted. If the weight is positive, the pair (NP1, NP2) is a potential instance of coreference relationship. The positive weight is then adjusted according to the proximity of (NP!, NP2), the grammatical role of NP2.
- Heuristics that are inspired by the centering theory [4, 11].

  If NP1 in the pair (NP1, NP2) is a pronoun, bonus weights b_i's are added to the weight of pair according to the following rules:

b_1 if (a) NP2 is a pronoun; and (b) both NP1 and NP2 are in the current sentence.

b_2 if (a) NP2 is a pronoun; and (b) NP2 is in the previous sentence.

b_3 if (a) NP2 is a pronoun; and (b) both NP1 and NP2 are at the same type of grammatical role (subject, object of verb, object of preposition, *etc.*).

b_4 if NP2 is a subject.

b_5 if NP2 is an object of a verb.

b_6 if NP2 c-commands NP1. The closer NP2 is to NP1, the larger the bonus weight.

Example: Consider the following discourse example from [11]:

(8) a. Susan_1 drives a Ferrari_2.

b. She_3 drives too fast.

c. Lyn_4 races her_5 on weekends_6.

d. She_7 often beats her_8.

e. She_9 wins a lot_{10} of trophies_{11}.

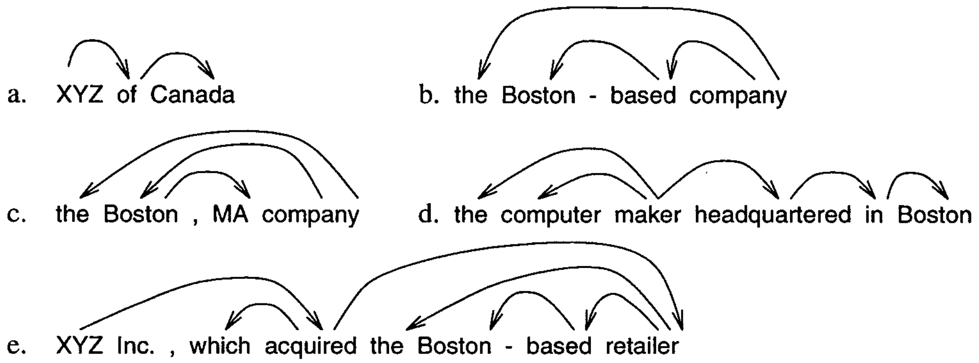
After each sentence is processed, the potential coreference relationships are shown in Table 3. We assume that the values of w_i's and b_i's are defined such that

$$w_1 > w_2 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6$$
$$w_2 > w_3 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6$$
$$b_1 > b_2 + b_3 + b_4 + b_5 + b_6$$
$$b_2 > b_3 + b_4 + b_5 + b_6$$
$$\ldots.$$

Template Elements (TE)

A template element frame for an organization or a person consists of all the relevant information, such as locale and alias, about the organization or the person mentioned in the text. After identifying the coreference relationships, the PIE unifies these properties associated with each instance of a domain entity and fills out the templates. The properties associated with each instance of a domain entity are extracted by the subtree pattern matcher. For example, the locale of an organization is identified by finding the closest (indirect) dependent of the organization that has the semantic feature `+location`, unless there is an intervening organization node. This single rule covers many kinds of expressions for specifying the locale of an organization, such as (9a), (9b), (9c), and (9d). The constraint that the organization and its locale must not be separated by another organization prevents "Boston" to be assigned as the locale of XYZ in (9e), since "retailer" is also a kind of organization.

(9)


a. XYZ of Canada
b. the Boston - based company
c. the Boston , MA company
d. the computer maker headquartered in Boston
e. XYZ Inc. , which acquired the Boston - based retailer

(a)	Susan$_1$ drives a Ferrari$_2$.		
s_1	Susan$_1\neq$Ferrari$_2$	w_1	
Action:	s_1 is asserted		
(b)	She$_3$ drives too fast.		
s_2	She$_3$=Susan$_1$	$w_3 + b_4 + b_3$	
Action:	s_2 is asserted		
(c)	Lyn$_3$ races her$_5$ on weekends$_6$.		
s_3	Lyn$_3\neq$Susan$_1$	w_1	
s_4	Lyn$_3\neq$Ferrari$_2$	w_1	
s_5	Lyn$_3\neq$her$_5$	w_2	"her$_5$" cannot refer to a c-commanding element in the same clause
s_6	Lyn$_3\neq$weekends$_5$	w_2	"weekends" cannot refer to a c-commanding element
s_7	her$_5$=She$_3$	$w_3 + b_2 + b_4$	
s_8	her$_5$=Susan$_3$	$w_3 + b_4$	
Action:	s_3, s_4, s_5, s_6, and s_7 are asserted. s_8 is ignored because of s_7.		
(d)	She$_7$ often beats her$_8$.		
s_9	She$_7\neq$her$_8$	w_2	"her$_8$" cannot refer to a c-commanding element in the same clause
s_{10}	She$_7$=her$_8$	$w_3 + b_1$	It is possible to have immediate contradictions, since s_9 and s_{10} are made by different rules.
s_{11}	She$_7$=Lyn$_3$	$w_3 + b_4 + b_3$	
s_{12}	She$_7$=her$_5$	$w_3 + b_2 + b_5$	
s_{13}	She$_7$=Susan$_1$	$w_3 + b_4$	
s_{14}	her$_8$=Lyn$_3$	$w_3 + b_4$	
s_{15}	her$_8$=her$_5$	$w_3 + b_2 + b_3 + b_5$	
s_{16}	her$_8$=Susan$_1$	$w_3 + b_4$	
Order:	s_9, s_{10}, s_{15}, s_{12}, s_{11}, s_{13}, s_{14}, s_{16}		
Action:	s_9 is accepted. s_{10} is ignored because of s_9. s_{15} is accepted. s_{12} is ignored because of s_9 and s_{15}. s_{11} is accepted. s_{13}, s_{14} and s_{16} are ignored.		
(e)	She$_9$ wins a lot$_{10}$ of trophies$_{11}$.		
s_{17}	She$_9$=She$_7$	$w_3 + b_2 + b_3 + b_5$	
s_{18}	She$_9$=her$_8$	$w_3 + b_2 + b_4$	
s_{19}	She$_9$=Lyn$_3$	$w_3 + b_4$	
s_{20}	She$_9$=her$_5$	w_3	
s_{21}	She$_9$=Susan$_1$	$w_3 + b_4$	
Order:	s_{17}, s_{18}, s_{21}, s_{19}, s_{20}.		
Action:	s_{17} is accepted. All other assertions are ignored.		

Table 3: The coreference recognition process for (8)

Scenario Templates (ST)

The ST module is executed after all the sentences in the text have been parsed, all the coreference relationships has been asserted and all the template elements have been created. The scenario templates are generated in two steps: First, a post-holder database is created. Then ST templates are filled according to the contents in the database.

A record in the post-holder database is made up of the following fields:

(10) post: e.g., CEO, president
holder: a pointer to a discourse entity representing a person
org: a pointer to a discourse entity representing an organization
status: one of IN, OUT, UNK

The records in the database are created by three rules:

`static:` recognizes a static situation in which a person holds a post in an organization.

`single:` recognizes succession events in which one person is explicitly mentioned.

`double:` recognizes succession events in which two persons are explicitly mentioned.

The rule `static` is triggered by a person. The pattern matcher searches the dependency tree rooted at the person and find a post or posts and an organization, skipping all the nodes that are also persons. The status field in the records created by the rule `static` is usually UNK, unless the post has modifiers such as "former" or "retired." For example, the `static` rule creates the records in (12) if any of the expressions in (11) is encountered.

(11) a. XYZ president and CEO John Smith ...
b. XYZ's president and CEO John Smith ...
c. John Smith, who has been XYZ's president and CEO since 1990, ...
d. John Smith, president and CEO of XYZ Inc. ...

(12) post: CEO holder: John Smith org: XYZ Inc. status: UNK
post: president holder: John Smith org: XYZ Inc. status: UNK

If the pronoun "he" is determined by the coreference module to refer to "John Smith," then (13) will also cause the two records in (12) to be created.

(13) he has been XYX's president and CEO since 1990

The rule `single` is triggered by any of words in (14).[2]

(14) "take over" assume appoint promote name become "act as" "serve as" resume "force out" sack resign "step down" retire fire "lay off" remove dismiss leave oust appointment dismissal ouster resignation retirement removal

The person involved in the succession event is assumed to be a dependent of the trigger word or the parent node of the trigger word in the dependency tree. The post is a dependent of the trigger word. The organization is either a dependent or a c-commanding NP of the trigger word. The status is determined by the trigger word. If the post or organization is not found in the dependency tree, then the person's position in the post-holder database is used.

If the event is an IN-event, the post-holder database is searched. If the person holds multiple positions in different organizations, or incompatible jobs[3] in the same organization, we assume that the person will give up those positions, and corresponding records of OUT-events will be created.

Example:

[2] The list (14) is what we used in our MUC-6 system. "Quit" and "hire" are obvious omissions.

[3] we assume two jobs are incompatible if a word appears in both job titles

(15) ABC announced that John Smith, president and CEO of XYZ, is named its next chairman.

The word "named" triggers the `single` rule. "John Smith," "its next chairman," and "ABC" are then recognized as the holder, post, and organization of an IN-event. The record (16) is created:

(16) post: chairman holder: John Smith org: ABC status: IN

Assuming that the `static` rule had created the records in (12), the IN-event triggered by "named" would also cause records in (17) to be inserted into the database.

(17) post: CEO holder: John Smith org: XYZ Inc. status: OUT
post: president holder: John Smith org: XYZ Inc. status: OUT

The rule `double` is triggered by the following words:

(18) succeed accede replace follow succession replacement

If the trigger is `-passive`, the IN person is the subject of the trigger or c-commands the trigger. If the trigger is `+passive`, the IN person is the object of "by". The OUT person is similarly defined.

Once the post-filler database has been created by the above three rules, the ST templates are filled by examining the records in the database. For each distinct pairs of "post" and "org" fields, if there exists a record with status=IN or status=OUT, a succession frame is filled.

Analysis of the Walkthrough Article

NE

PIE had 90% recall and 93% precision for the walkthrough article in the NE test. The errors included the following:

- Five occurrences of "Coke" were not recognized as entities.
- "Hollywood" was not recognized as a location.
- Two mistakes were made in the following sentence:

  Dooner, who recently lost 60 pounds over three-and-a-half months,

  The lexical rules classified "60 pounds" as MONEY and "-half" as a DATE.
- "New York Times" and Coca-Cola within "Coca-Cola Classic" are mistakenly recognized as companies.
- The words "John Dooner Will Succeed James" were grouped together as a single person name.
- The company "Fallon McElligott" is incorrectly recognized as a person.

CO

The coreference scores for the walkthrough article were:

Recall: 101/132 = 0.77
Precision: 101/128 = 0.79

These are much higher than the scores for the test corpus. The reason, we believe, is that the majority of the coreference relationships in the article involved proper names and personal pronouns. The coreference algorithm is very good at matching proper names and determining the antecedents of personal pronouns.

TE

PIE got 73% recall and 77% precision for the walkthrough article. There were 15 errors: 5 incorrect, 6 missing, and 4 spurious, many of which were due to failures of the NE module:

- Incorrectly treating "New York Times" as an organization caused 2 TE errors.
- Incorrectly treating "Fallon McElligott" as a person, instead of a company caused 3 TE errors.
- Incorrectly treating "John Dooner Will Succeed James" as a name caused 1 TE error.
- Failing to recognize "Coke" as a company name was partly responsible for 2 TE errors.

The errors in coreference recognition caused 3 incorrect descriptors in TE.

PIE failed to infer the location of "Coke" from the phrase "from Coke headquarters in Atlanta" because "Atlanta" is not a dependent of "Coke."

ST

The ST recall and precision for the article are 63% and 72% respectively. Most of the correct fillings can be attributed to its successful analysis of (19) and (20).

(19) One of the many differences between Robert L. James, chairman and chief executive officer of McCann-Erickson, and John J. Dooner Jr., the agency's president and chief operating officer, is quite telling:

(20) He will be succeeded by Mr. Dooner, 45.

The `static` rule created 4 entries in the post-holder database according to (19):

(21)

post: chief executive officer	org: McCann - Erickson	holder: Robert L. James	status: UNK
post: chairman	org: McCann - Erickson	holder: Robert L. James	status: UNK
post: chief operating officer	org:	holder: John J. Dooner Jr.	status: UNK
post: president	org:	holder: John J. Dooner Jr.	status: UNK

The `double` rule is triggered by "succeed" in (20). The coreference module correctly determined that "He" in (20) refers to "Dooner" and generated two succession events for the chairman and CEO positions of McCann-Erickson.

A spurious OUT event (Dooner out as president) was generated because of (22).

(22) There are no immediate plans to replace Mr. Dooner as president

The omission of "hire" in the trigger word list is responsible for a missing IN-event implied by (23):

(23) In addition, Peter Kim was hired from WPP Group's J. Walter Thompson last September as vice chairman, chief strategy officer, world-wide.

All `VACANCY_REASON` slots are filled with the default value `REASSIGNMENT`. For the walkthrough article, all three fillings happen to be incorrect.

Development Process

The NE module is trained on the 30 dry-run test articles and 50 articles (94*) out of the 100 formal training articles. The CO module is trained on the 30 dry-run test articles. The TE and ST modules are trained on less than 25 of the 100 formal training articles, though they are tested with the complete set during development.

The greatest limiting factor in the development of PIE is the amount of time and human resources available to examine training articles to create domain specific vocabulary and dependency patterns, and to inspect the differences between the answer keys and the system responses to fine-tune the system.

The development of PIE system started in July 1995, when MUC-6 was announced. The development prior to August 27, 1995 was mostly on general tools that can also be used in non-MUC applications. We estimate that 30 person-days were spent on the following components/tasks.

- A sentence boundary identifier.
- A subtree pattern matcher which works on any tree structure (not just dependency trees).
- A sequentially constructed equivalence classes.
- Interface classes to the WordNet.
- A SGML markup generator that takes a set of tuples `(low, high, type, contents)` and generates SGML markups surrounding the words with indices from `low` to `high`. The generator guarantees that overlapping markups are nested.
- Additional lexical rules for recognizing entity names.
- Tuning up PRINCIPAR.

The development of MUC-specific components started on August 28. Table 4 summarizes the process. The first training tests for the CO, TE and ST module were conducted as soon as the system was able to process all the testing articles without crashing. The last training tests was conducted just before running the formal test. The formal testing results for NE, CO and TE are quite close to the training test results. The F-measure for ST in formal testing is significantly lower than that of training tests because the recall dropped from about 50% to 39%. This was not totally unexpected, since the implementation of the ST module did not begin until September 29th. It did not reach a relatively stable state by the time of the formal testing on October 6th.

	Person-	First Training Test		Last Training Test		Formal Test	
Category	Days	Date	P&R	Date	P&R	Date	P&R
NE	10	1-Sep	88	6-Oct	96	6-Oct	93
CO*	14	14-Sep	(49,47)	6-Oct	(65,58)	6-Oct	(63,63)
TE	6	25-Sep	67	6-Oct	78	6-Oct	74
ST	5	2-Oct	45	6-Oct	53	6-Oct	48

*The first number is precision, the second is recall

Table 4: Summary of the MUC-6 development process

Conclusions

We feel that we have achieved the following objectives in MUC-6.

- We developed and tested a domain independent algorithm for coreference recognition that turned out to work quite well.
- We showed that broad-coverage and efficiency can be achieved by principle-based parsing.
- We demonstrated that a broad-coverage parser can be very useful in information extraction. Once the dependency relationships of a sentence are established, a small number rules can be used to extract information that can be expressed in a large number of variations.

Acknowledgment

We wish to thank the Oxford Text Archive, Princeton University, and New Mexico State University for making valuable data resources publicly available. The author is a member of the Institute for Robotics and Intelligent Systems (IRIS) and wishes to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council (NSERC), and the participation of PRECARN Associates Inc. This research has also been partially supported by NSERC Research Grant OGP121338.

References

[1] R. Beckwith, C Fellbaum, D. Gross, and G. Miller. WordNet: a lexical database organized on psycholinguistic principles. In Uri Zernik, editor, *Lexical Acquisition: Using On-Line Resources to Build a Lexicon.* Lawrence Erlbaum, Hillsdale, NJ, 1991.

[2] Bonnie J. Dorr, Dekang Lin, Jye-hoon Lee, and Sunki Suh. Efficient parsing for Korean and English: A parameterized message-passing approach. *Computational Linguistics*, 21(2):255–264, June 1995.

[3] Liliane Haegeman. *Introduction to Government and Binding Theory.* Basil Blackwell Ltd., 1991.

[4] Barbara j. Grosz, Aravind K. Joshi, and Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–226, June 1995.

[5] Dekang Lin. Evaluation of PRINCIPAR with the SUSANNE corpus. (draft).

[6] Dekang Lin. Principle-based parsing without overgeneration. In *Proceedings of ACL–93*, pages 112–120, Columbus, Ohio, 1993.

[7] Dekang Lin. University of Manitoba: Description of the NUBA System as Used for MUC-5. In *Proceedings of the Fifth Message Understanding Conferenc*, pages 263–276, 1993.

[8] Dekang Lin. Principar—an efficient, broad-coverage, principle-based parser. In *Proceedings of COLING–94*, pages 482–488. Kyoto, Japan, 1994.

[9] Igor A. Mel'čuk. *Dependency syntax: theory and practice.* State University of New York Press, Albany, 1987.

[10] Geoffrey R. Sampson. *English for the Computer.* Oxford University Press, 1995.

[11] Linda Z. Suri and Kathleen McCoy. RAFT/RAPR and centering: A comparision and discussion of problems related to processing complex sentences. *Computational Linguistics*, 20(2):289–300, June 1995.