Detection of Coreferences in Automatic Specifications Analysis

by

S. Shankaranarayanan

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Walling R. Cyre

Dr. James R. Armstrong

Dr. F. G. Gray

January, 1994

Blacksburg, Virginia

Detection of Coreferences in Automatic Specifications Analysis

by

S. Shankaranarayanan

Dr. Walling R. Cyre, Chairman

Electrical Engineering

(ABSTRACT)

Specifications on digital hardware systems typically contain descriptions and requirements expressed in natural language and diagrams of various types. The objective of the research reported here is the automatic detection of common references ("coreferences") to objects in natural language specification statements in order to permit automatic integration of requirements. This thesis describes a prototype system for detecting coreferences. First, the natural language statements are translated into conceptual graphs (semantic nets). Then, these graphs are scanned by a rule-based system to determine whether each concept that is encountered is the definition of a new concept or a reference to a previously defined concept. Tests performed on the system developed indicate a high percentage rate of correct classifications.

Acknowledgments

I would like to thank Dr. Walling Cyre for the guidance and support he provided throughout the duration of the this project. I am particularly grateful for the numerous ideas and suggestions he provided that helped me overcome some of the problems that are unique to natural language processing systems. Also, I would like to thank him for his attention to detail that often made me realize aspects that I had overlooked. I would also like to thank Dr. James Armstrong and Dr. F. G. Gray for serving on my committee. In addition, I thank Semiconductor Research Corporation for the support and funding they provided for this project.

I am grateful to all the students in the lab for all the help and the pleasant company they have provided. I would like to thank my parents for all the sacrifices they have made for my sake. I thank my grandmother for her infinite love and caring, and for all the good values she has helped me learn. Finally, I thank God for bearing with all my shortcomings and for blessing me with courage and perseverance and all the good things in life.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

The process of design of digital systems has made remarkable strides in advancement because of the progress made in the area of synthesis. Synthesis systems usually use some formal scheme of representation like hardware descriptive languages to specify system requirements. In practice, the design of digital systems usually begins with the development of a set of specifications outlining the requirements of the desired system. These specifications are usually composed of block diagrams, timing diagrams, flow-charts and natural language. Natural language is the most popular and the most versatile form of specification though it is the most ambiguous.

The problems associated with natural language are a direct result of the ambiguity of natural language itself. The same statement or statements in English, for example, could be interpreted in several ways. Consequently, the engineer that uses the specifications may interpret the specifications differently from what the specifications author intended, leading to design errors.

Several types of flaws are inherent in natural language specifications [3]. The set of specifications may not be complete enough to produce a correct implementation. Some vital information could be left out by the author, either assuming that the reader would know, or by sheer oversight. Some of the information contained in the specification could be incorrect or inconsistent. Inconsistencies in specifications are either syntactic or semantic. Syntactic inconsistencies have to do with such details as

names and number of inputs. Semantic inconsistencies deal with the interpretation of the behavior of the system being specified. Such problems are difficult to detect and eliminate.

Automated interpretation of specifications could avoid such problems, reduce design cycle times and improve efficiency by eliminating much of the manual interpretation required in the design process.

1.2 The ASPIN System

The work reported here is part of a project called the Automated Specifications Interpreter (ASPIN) being developed to increase efficiency in creating and interpreting requirements of a digital system. The ASPIN system takes various forms of informal specifications as input and creates formal engineering models [5]. Simulation, analysis and further synthesis can then be performed with these formal models. The formal models of the specifications will be expressed in the VHSIC hardware description language (VHDL) [7] which is becoming a standard language for modeling and synthesis in industry.

Figure 1 shows an overview of the ASPIN system. Inputs to the ASPIN system are entered in various forms and are mapped by the system into the conceptual graph knowledge representation [12]. Natural language input in the form of English language statements is represented in conceptual graph form by a parser and a semantic analyzer [5]. The parser creates parse trees that represent sentence structure which are then input to the semantic analyzer for conversion into conceptual

graphs. The semantic analyzer analyzes the trees to create conceptual graphs that represent the information in the natural language specification. These conceptual graphs representing individual sentences of a specification must be integrated by joining them on concepts that refer to the same entity. The integration is performed by the Coreference Detector which analyzes the multiple conceptual graphs and links them at concepts pointing to the same entity. These integrated conceptual graphs are then processed to create VHDL models describing the behavior of the system specified. The graphs are also used to create a graphical representation of the design information.

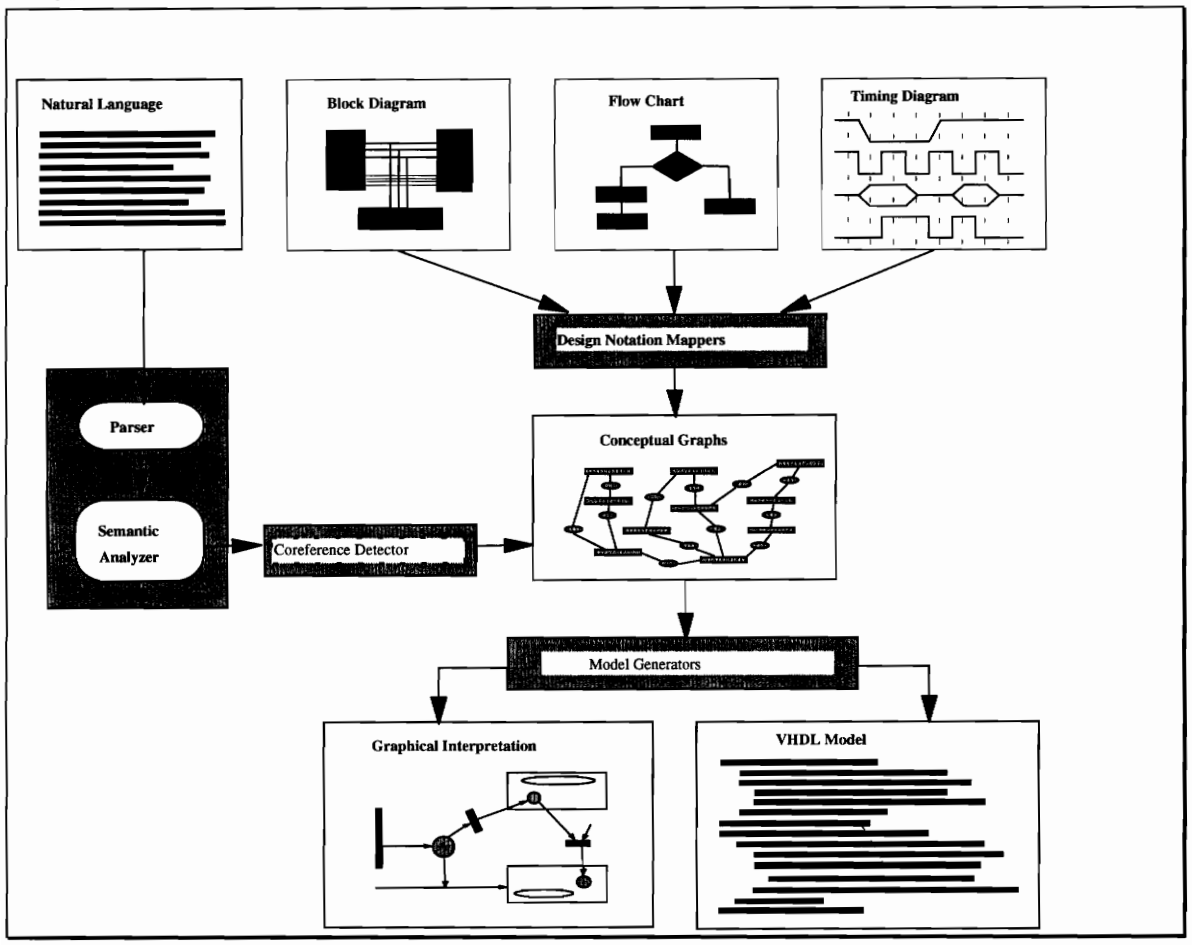

Figure 1. ASPIN System Overview

Figure 2 represents the internal structure of the ASPIN system. The specifications author enters the design specifications and requirements. This information is then processed by the specifications interpreter consisting of a parser, a semantic analyzer and a coreference detector and is converted into conceptual graphs. These conceptual graphs are then processed by the LINKER [6] and the Graphical Model Generator [14]. The Graphical Model Generator creates a graphical representation of the specification which is fed back to the specifications author for validation. The LINKER (not shown in figure) converts the conceptual graphs into a form that can be used by the Modeler's Assistant [11] to create VHDL models. The Modeler's Assistant is an interactive graphical tool for creating VHDL behavioral models of digital systems. The models are created by entering *Process Model Graphs* that represent the structure of the VHDL models. These graphs consist of nodes, which represent processes and arcs which represent the signals used in communication among the processes. With the VHDL models so constructed, the designer can analyze and simulate the models for verification.

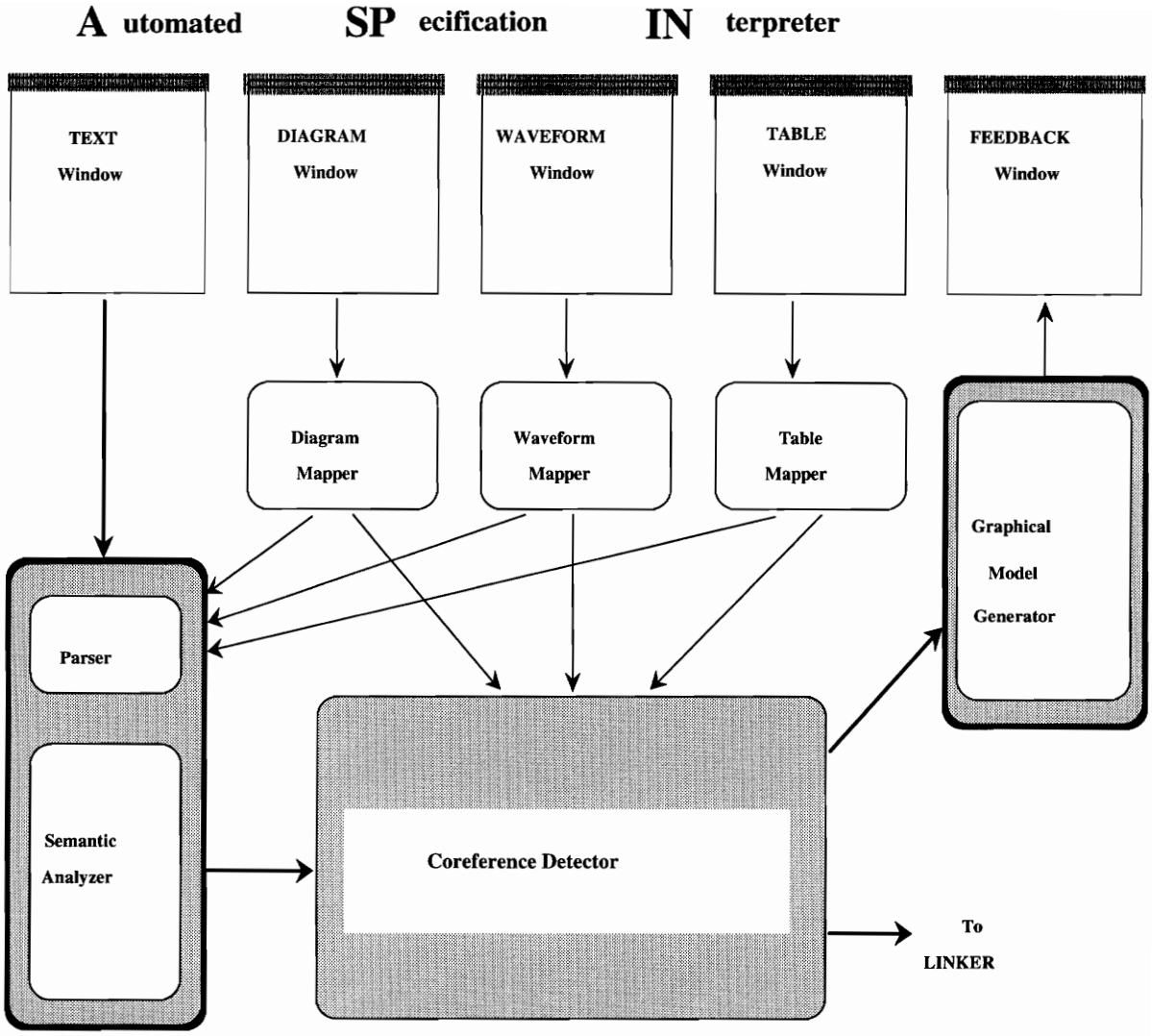

Figure 2. ASPIN System Organization

1.3 Research Approach

The approach of the research reported here is based on the structure of English sentences that describe digital systems. A previously developed parser and semantic analyzer convert natural language statements into conceptual graphs. The parser applies grammar rules and creates parse trees that represent the structure of the

sentence. The semantic analyzer uses these parse trees and builds a conceptual graph by joining canonical graphs of words found in the input sentence. These conceptual graphs serve as input to a coreference detector that makes use of an algorithm described in this thesis to link all concepts that are identical across sentences, or ***coreferences***. The parser and the coreference detector have been developed in C and the semantic analyzer has been developed in Quintus Prolog[1] , all three pieces of software are currently running on a Sun SPARCstation[2] platform.

1.4 Thesis Organization

This thesis is organized into an abstract, 7 chapters, a glossary, bibliography and appendices A through G. Chapter 1 provides the motivation for coreference detection, an introduction to the work reported by this thesis and an overview of the ASPIN system. Chapter 2 discusses related research done in the area of anaphora and graph matching and lays the foundation for the discussion of the rules developed for coreference detection. Chapter 3 provides an overview of conceptual graph theory and how the anaphora problem is mapped to the conceptual graph domain. Chapter 4 details the rules developed for coreference detection and chapter 5 discusses the implementation along with some analysis examples. Chapter 6 presents the results obtained and chapter 7 summarizes the system capabilities, limitations and presents some insight into future enhancements and modifications to the set of rules developed for coreference detection. The glossary contains a list of terms used in this thesis. Appendix A lists the concept hierarchy used by the Coreference Detector. Appendix B

---

[1] Quintus is a trademark of the Quintus Corporation.
[2] SPARCstation is a trademark of Sun Microsystems, Inc.

shows the definitions table generated by the Coreference Detector for a test set of sentences. Appendix C and D contains the interpretation of the results obtained from a test set of sentences (Test Set II). A complete listing of the test set (Set II) of sentences, their conceptual graphs and the integrated conceptual graph produced by the Coreference Detector is included in Appendix E. Appendix F lists the sentences in Test Sets I and III. Appendix G contains a User Manual for the system for coreference detection developed.

2. Related Work

2.1 Introduction

This chapter discusses some of the research done in the area of anaphora detection relevant to this thesis and presents Sowa's definition for coreference. Later, the chapter briefly touches upon the tools developed as part of the ASPIN system for converting natural language sentences into the target representation format for the system, namely, conceptual graphs. Since the coreference detection problem can be mapped into a graph theory problem, projection and mapping of graphs in conventional graph theory is relevant. This chapter also discusses two papers that deal with projection and mapping in graph theory.

2.2 Theory of Anaphora and Coreference

Sentences in natural language are composed of words and phrases. We use the term **expression** to denote a group of words or a phrase that occurs in natural language sentences. Expressions that identify real world entities or actions are called **references**. The entities and actions themselves are the **referents** of such references. References could introduce referents or could point to referents that have already been introduced (anaphoric) or to referents that are introduced later (cataphoric) [4]. Thus, an anaphoric reference has a backward pointer to an antecedent reference whereas a cataphoric reference has a forward pointer to a referent that occurs later. In either case, the two expressions are said to be **coreferences**. For example,

DMA transfers consists of a fetch cycle and a deposit cycle. (2.1)

DMA transfers may be initiated by direct register programming, by external requests on the DRQ pin or by timeouts of Timer 2. (2.2)

In the two example sentences presented above, the first time the reference "DMA transfers" occurs (i.e., in sentence 2.1), it is being introduced and is therefore an introduction. The occurrence of "DMA transfers" in the sentence 2.2, however, is a reference to "DMA transfers" in sentence 2.1, an expression already introduced, and is therefore an anaphoric reference. Anaphoric coreference [4] is said to exist between the two occurrences of "DMA transfer" in sentences 2.1 and 2.2. The following sentence presents an example of a cataphoric coreference.

During the fetch cycle, which is part of a DMA transfer, the byte or word data is accessed according to the source pointer register. (2.3)

This statement can be thought of as a combination of two statements,

During the fetch cycle, the byte or word data is accessed according to the source pointer register. (2.4)

and

A fetch cycle is part of a DMA transfer. (2.5)

Therefore, the first occurrence of "fetch cycle", i.e., the occurrence in sentence 2.4, is actually a reference to the expression "fetch cycle" that is introduced later in sentence

2.5, and is thus a cataphoric reference. Cataphoric coreference [4] is said to exist between the two expressions.

2.2.1 Types of References

References can be broadly classified into three types:

2.2.1.1 Indefinite Reference

Expressions that occur with the indefinite article are called indefinite references [4]. For example,

A fetch cycle is part of a DMA transfer. (2.6)

2.2.1.2 Definite Reference

Expressions which occur with the definite article are called definite references [4]. Example,

The processor reads the data from memory. (2.7)

2.2.1.3 Linguistic Reference

Expressions in which the anaphoric determiner is bound to the earlier mention of the same noun are called linguistic references [4]. In these cases, the reference with the determiner has a backward reference to an antecedent, and therefore, the two references are coreferent. For example,

*As soon as a non-maskable interrupt occurs, **the** interrupt is serviced.* (2.8)

The anaphoric determiner "the" binds the anaphora "interrupt" (second occurrence) to the antecedent "interrupt" (first occurrence) in sentence 2.8. In this sentence, the second occurrence of "interrupt" is a pointer to the first (antecedent) and the two occurrences are thus coreferences. The determiner could be non-binding in certain cases, as can be seen in example sentence 2.9.

*As soon as a non-maskable interrupt occurs, **an** interrupt is serviced.* (2.9)

The determiner of the second occurrence of "interrupt" is "an" in this case, making it non-anaphoric and thus the two occurrences of "interrupt" are not coreferent.

2.2.1.4 Noun Phrase Reference

Expressions in which certain determiners are used to signal that a noun phrase is referentially equal to a previous noun phrase are called noun phrase references [4], and the two references that are referentially equal are coreferent. Examples of such determiners are "the", "this" - "these", "that" - "those".

During the fetch cycle, the byte or word data is accessed according to the source pointer register. (2.10)

During this cycle, the data is read into an internal temporary register. (2.11)

The determiner "this" in sentence 2.11 signals that the concept that it qualifies, "cycle" is referentially equal to "cycle" in sentence 2.10. Coreference between two references can also be emphasized by the use of certain adjectives like "same", "identical", and "very". Examples of such coreferences are given below.

During the fetch cycle, the byte or word data is accessed according to the source pointer register. (2.12)

*In the **same** cycle, the data is read into an internal temporary register.* (2.13)

The adjective "same" qualifying "cycle" in sentence 2.13 signals that "cycle" in sentence 2.13 is coreferent with "cycle" in sentence 2.12. When the adjectives "former" and "latter" precede expressions, reference to one of two previously introduced expressions is intended [4].

The i-486 and the pentium processors are two of the latest processors introduced by Intel. (2.14)

One version of the former functions at a frequency of 66 MHz. (2.15)

There is another interesting basis on which references can be classified. There is, however, considerable similarity between the two bases of classification. This second basis is discussed in the following subsections.

2.2.1.5 Noun Phrases with the Definite Article

Noun phrases with the definite article could be references either to previously introduced referents or to referents that have not been introduced [1]. If the noun phrase refers to a previously introduced referent, it is coreferent with the reference that introduced the referent. If the noun phrase points to a referent that has not been previously introduced, the noun phrase itself provides the introduction. This classification is, in a sense, a combination of the definite reference and the noun phrase reference discussed earlier.

A fetch cycle is part of a DMA transfer. (2.16)

During the fetch cycle, the byte or word data is accessed according to the source pointer register. (2.17)

The noun phrase "fetch cycle" with the definite article in sentence 2.17 is coreferent with the noun phrase "fetch cycle" in sentence 2.16. If the reference "fetch cycle" had not occurred prior to sentence 2.17 (i.e., if sentence 2.16 had not occurred in the system description), "fetch cycle" in sentence 2.17 would point to a referent not previously introduced.

2.2.1.6 Pronouns

Pronouns are references that always point to references that have occurred earlier. Because of this property, pronoun references are always coreferent to references occurring elsewhere in the system description [1]. For example,

During the fetch cycle, the data is read into an internal temporary register. (2.18)

It is written into memory or I/O space during the deposit cycle. (2.19)

2.2.1.7 Proper Names

Proper names that occur in natural language are references to particular objects and thus are references [1]. These references could either be coreferent with another reference or could introduce referents themselves. The first occurrence of the proper name introduces a referent and all subsequent occurrences are coreferent with the first occurrence.

Full input handshake mode is selected when HNDS is one and OIN is zero. (2.20)

Full output handshake mode is selected when HNDS and OIN are ones. (2.21)

2.2.1.8 Temporal References

Expressions that occur with adjectives like former and latter are references to expressions that occur elsewhere in the description [1]. Example,

The i-486 and the pentium processors are two of the latest processors introduced by Intel. (2.22)

One version of the former functions at a frequency of 66 MHz. (2.23)

The discussion of references and coreferences above has been taken from works on natural language understanding. From a mathematical standpoint, another definition

for coreference that is of relevance is Sowa's definition for coreference. Sowa's description of coreference nodes in conceptual graphs in terms of a *line of identity* and *context* is discussed in the following subsection.

2.3 Sowa's Definition for Coreference

Sowa considers the anaphora problem in terms of links among coreferences and defines coreference in mathematical terms as follows:

A *line of identity* [12] is defined to be an undirected graph g whose nodes are concepts and whose arcs are pairs of concepts, called *coreference links* [12] such that

- No concept may belong to more than one identity
  - A concept a in g is said to *dominate* another concept b if there is a path $(a_1, a_2, a_3, \ldots, a_n)$ in g where $a=a_1$, $b=a_n$, and for each i, either a_i and a_{i+1} both occur in the same context or the context of a_i dominates the context of a_{i+1}
- Two concepts a and b are *coreferent* if either a dominates b or b dominates a and states that lines of identity show anaphoric references.

Next, a brief discussion of the theory behind some of the tools that are part of the ASPIN System is presented in section 2.4. A discussion of two papers that are relevant to this research is presented in section 2.5.

2.4 The Parser and the Semantic Analyzer

The parser used to construct trees representing sentence structure is a bottom-up, chart parser that uses a phase-structured grammar. The grammar was constructed manually through a study of various VLSI product descriptions [2] . The parser uses a dictionary that contains the lexical categories for over 1200 words.

Every word in the vocabulary of the sublanguage has one or more syntactic and one or more semantic types [5]. While a syntactic type represents the lexical category of the word (used in building parse trees), a semantic type represents a meaning the word can have in English interpretation, and is represented by a conceptual graph called the canonical graph. Parse trees representing input sentences are converted into conceptual graphs by the Semantic Analyzer, which makes use of a library of semantic rules and over 250 canonical graphs [5]. The parse tree's nodes correspond to grammatical constructs, for each of which there is a rule in the Semantic Analyzer. Processing of a parse tree begins at the root of the tree and proceeds along each sub-tree of the root node, until a leaf node is reached. Then, a canonical graph is either retrieved from the library or is constructed. This process is repeated for every subtree of the parent node. Finally, when all subtrees have been processed, the graphs that have been created are linked together if possible. The result is a conceptual graph for the sentence represented by the parse tree. Figure 3 shows the parse tree generated by the parser for the sentence ***the machine can perform a read or write asynchronously if the rdwrite signal is high .***

```
s(19,
c(   s1,[
   c(  ss2,[
      c(   n1,[
         c( np12,[
            t(det    ,the),
            c( head1,[
               t(noun   ,machine)])])]),
      c( pred4,[
         c( avs3,[
            t(mod    ,can),
            t(vinf   ,perform)]),
         c(   n6,[
            c(  nc1,[
               c( np12,[
                  t(det    ,a),
                  c( head1,[
                     t(noun   ,read)])]),
               t(conj   ,or),
               c( np20,[
                  c( head1,[
                     t(noun   ,write)])])])]),
         c(   d2,[
            t(adv    ,asynchronously)]),
         c(   d4,[
            t(scon   ,if),
            c(  ss2,[
               c(   n1,[
                  c( np12,[
                     t(det    ,the),
                     c( head4,[
                        t(id     ,rdwrite),
                        c( head1,[
                           t(noun   ,signal)])])])]),
               c(pred14,[
                  c( evs1,[
                     t(be     ,is)]),
                  c( adjs1,[
                     t(adj    ,high)])])])])]), t(.,.)])).
```

Figure 3. Parse tree generated for the sentence ***the machine can perform a read or write asynchronously if the rdwrite signal is high .***

Figure 4 below shows the conceptual graph representation (conceptual graphs are discussed in detail in chapter 3) generated by the Semantic Analyzer for the same sentence.

```
[ 1 : perform : perform ]
    ->( gindx : 1 )
    ->( mod : can )
    ->( agnt : agnt ) -> [ 2 ]
    ->( obj : obj ) -> [ 3 ]
    ->( man : adv ) -> [ 4 ]
    ->( cond : if ) -> [ 5 ]
[ 2 : machine : machine ]
    ->( gindx : 1 )
    ->( det : the )
[ 3 : move : or ]
    ->( or : null ) -> [ 6 ]
    ->( or : null ) -> [ 7 ]
[ 6 : read : read ]
    ->( gindx : 1 )
    ->( det : a )
[ 7 : write : write ]
    ->( gindx : 1 )
[ 4 : act_struct : asynchronously ]
[ 5 : is : is ]
    ->( gindx : 2 )
    ->( agnt : agnt ) -> [ 8 ]
[ 8 : signal : signal ]
    ->( gindx : 1 )
    ->( name : null ) -> [ 9 ]
    ->( det : the )
    ->( val : adj ) -> [ 10 ]
[ 9 : id : rdwrite ]
[ 10 : constant : high ]
    ->( gindx : 1 )
```

Figure 4. Conceptual graph generated for the sentence ***the machine can perform a read or write asynchronously if the rdwrite signal is high .***

The next section discusses the two papers in Conceptual Graph Matching that are of interest to this research.

2.5 Conceptual Graph Matching

Graph matching is a critical problem for researchers in a wide variety of application areas. In the area of natural language understanding, (for example, given two conceptual graphs representing two sentences, two words that point to the same referent can be identified by determining the extent to which the two concepts representing the two words share the same conceptual structure). In the following two subsections, two algorithms (developed by Myaeng and Lopez [10], and Mugnier and Chien [9] respectively) for conceptual graph matching and their relevance to the work reported by this thesis are discussed.

2.5.1 Myaeng and Lopez's Algorithm for Matching Conceptual Graphs

Myaeng and Lopez, in their paper titled "A Flexible Algorithm for Matching Conceptual Graphs", present an algorithm for matching Sowa's conceptual graphs based on the notion of "association graphs" [10]. The underlying principle is that the degree of similarity between two conceptual graphs can be determined from the extent to which the two share a common conceptual structure. This algorithm, the authors claim, is flexible and general enough to accommodate context sensitive matching heuristics and therefore is useful in a variety of matching schemes and applications.

The algorithm consists of two parts: the "basic" part that finds maximal common subgraphs for the two graphs being matched ignoring the information included in the CG's and the "application dependent" part that takes into account such information.

The basic part produces all the possible maximal subgraphs in two steps. The first step is to construct what is known as an association graph. Let G and G' be the two graphs being matched. The association graph shows all the possible correspondences between any two concept nodes, one from G and the other from G' as well as those between relations nodes and arcs respectively. The second step is to take the resulting association graph and link association pairs to construct the set of possible maximal subgraphs that exist in both of the two graphs being matched.

Thus the steps in the matching algorithm are

1. construct association graphs from the graphs being matched
2. identify and expand local connections around each pair of matchable nodes
3. eliminate redundant connections and construct the final set of maximal subgraphs

The second part of the algorithm is to incorporate all the semantic information existing in the two graphs being matched using the maximal join operation defined in conceptual graph theory.

This algorithm gives an insight into what conceptual graph matching is all about. Projecting one conceptual graph onto another or joining two conceptual graphs at similar or identical nodes requires some knowledge of the degree of similarity between

the two conceptual graphs. If the degree of similarity is high, then we can estimate that the number of identical nodes and therefore the number of "joins" between the two graphs would be high. This information could be utilized in determining the "confidence level" of a particular join.

Since this algorithm produces the set of all maximal subgraphs, it is in a sense a super set of the solution that this thesis is aiming at. This algorithm would produce several projections of one conceptual graph onto another whereas the problem of coreference detection at hand requires the best projection or joins between conceptual graphs. The rules developed in this thesis are also different from the approaches presented by Myaeng and Lopez in their paper insofar as the extent to which the searches are conducted to match conceptual graphs. While the method described by Myaeng and Lopez goes right to the very bottom of every tree in the conceptual graph from the beginning of the concept node where matching is being done, the rules developed for coreference detection as part of this thesis stop at one level below the concept which is being matched, while tracing paths right till the end above the concept in the conceptual graph.

2.5.2 Mugnier and Chien's Algorithms for Projection and Mapping

Mugnier and Chien, in a paper titled "Polynomial Algorithms for Projection and Mapping", present algorithms for two operations on conceptual graphs namely, projection and mapping. A projection from one conceptual graph to another is essentially a join of the two conceptual graphs at identical nodes, similar to the

problem dealt with in this thesis. Let us look at the algorithms outlined by Mugnier and Chien and their relevance to the topic of this thesis.

2.5.2.1 Projections and Mappings

Projections

Given two conceptual graphs G and G', a projection P from G to G' is an ordered pair of mappings from (R(G), C(G)) to (R(G'),C(G')) where R(G) and R(G') are the sets of all relation nodes in G and G' respectively, and C(G) and C(G') are the sets of all concept nodes in G and G' respectively such that

1. for all edge rc of G with label i, P(r)P(c) is an edge of G' with label i
2. for all r belonging to R(G), type(P(r)) = type)r); for all c belonging to C(G), type (P(c))<=type(c)

Mappings

Given two graphs G1 and G2, find a correspondence of a certain type between two maximal subgraphs of G1 and G2, say G1' and G2'. this correspondence, which is a Matching is specified by several parameters such as "compatibility criteria" between the c-vertex labels, the kind of correspondence between the vertex sets of the two maximal subgraphs and the type of maximality of the subgraphs G and G'.

Projections from a Tree to a Graph

Let T be a tree and G be a graph.

Given any R-vertex r, and c neighbor of r, define Pr[i] as the set of the numbers on the edges between r and c, i.e., the weights of the edges.

A **projection** is defined as follows:

P is a projection from T to G with P(a) = c if it satisfies

1. label(a) >= label(c)
2. for all r successor of a, r' = P(r) is a neighbor of c such that
   (i) type(r) = type(r')
   (ii) Pr(a) $\subseteq$ Pr(c) and
   (iii)for all ai successor of r, v = P(ai) is the neighbor of r' such that
   Pr[ai] $\subseteq$ Pr'[v] and
   P(Ti), the restriction of P to the subtree Ti induced by ai is a projection from Ti to G

The algorithm for coreference detection developed in this thesis (discussed in chapters 4 and 5) is basically the projection of a reference (represented by a tree) onto another reference (represented by another tree). The principles involved in the rules developed for coreference detection and the algorithm to implement it is essentially similar to the algorithm described by Mugnier and Chien in their paper. The coreference detection rules developed as part of this thesis also use the semantics of the English Language in coreference detection.

3. Background

This chapter discusses the theory behind the research described in this thesis and lays the foundation for the discussion of the rules for coreference detection developed. First, a discussion of conceptual graph theory, operations on conceptual graphs, generalization and specialization of conceptual graphs is presented. Later, there is a brief discussion on coreferences and their occurrences in system specifications. Finally, the last section briefly talks about how the problem of anaphora detection in natural language is mapped by this thesis to the conceptual graph domain.

3.1 Conceptual Graph Theory

3.1.1 Conceptual Graphs

Knowledge expressed by natural language statements can be represented in a form of semantic networks called conceptual graphs [12]. A conceptual graph is a finite, bipartite, connected graph consisting of a set of concept nodes, a set of relation nodes and directed arcs joining the concept and relation nodes. Each concept and relation node is labeled with a type marker and a referent marker. The referent marker is a name or identifier that points to the referent of the concept. Concepts without specific referents are called generic concepts. There is at least one concept node in every conceptual graph and every relation node is connected to at least one concept node.

The set of concept type markers form a partially ordered lattice called the type hierarchy. The universal type is at the top of the lattice and the absurd type is at the

bottom of the type hierarchy. In other words, the universal type is a supertype of every concept type and the absurd type is a subtype of every concept type. This is illustrated in Figure 5.

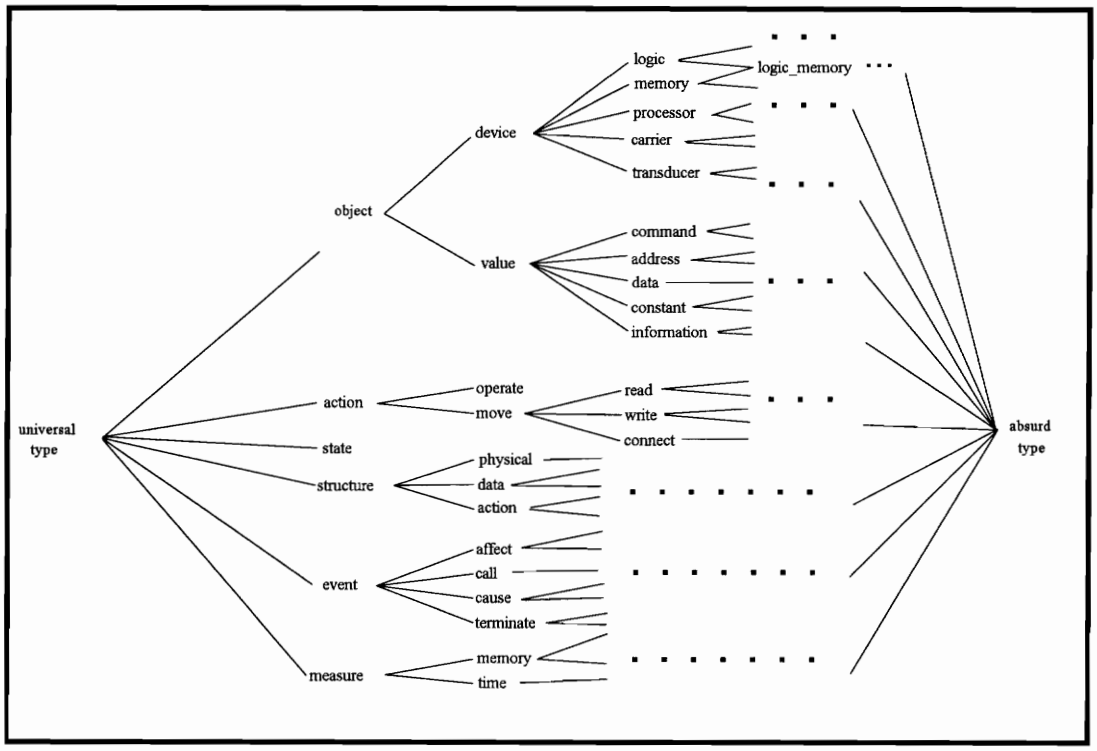

Figure 5. Partial Conceptual Type Hierarchy [5]

There are two notations for representing conceptual graphs, the graphical notation and the linear notation. In the graphical notation, concepts are drawn as boxes and relations are drawn as ellipses. The arcs linking the concepts and relations are drawn as arrows. Figure 6 shows the graphical representation of the conceptual graph of the statement "the processor reads the data from memory."

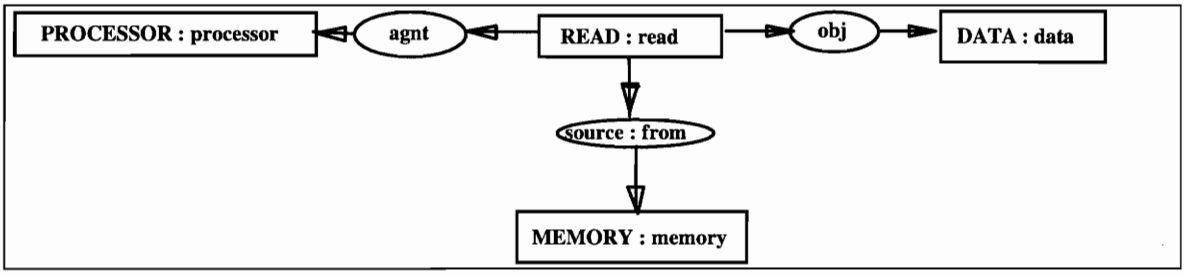

Figure 6. Graphical representation of ***the processor reads the data from memory***.

In the linear notation, concept nodes are denoted by three fields separated by colons and enclosed by square brackets, [<id> : type : referent marker]. The first field (the id) is an arbitrary concept number assigned uniquely to every concept in the conceptual graph. This field serves as the identifier of the concept in the conceptual graph. The second field represents the type to which the concept belongs. The third field represents the referent marker which identifies the conceptual graph nodes with the individuals or abstractions they represent. The referent field can contain several different types of markers. An asterisk "*" in the referent field of a concept is called a generic marker and represents an unspecified individual of the given type. An individual marker is a proper name (string of characters) or represented by a pound sign "#" followed by some identification number. Conceptual relation nodes, on the other hand, have only the type and referent fields within parentheses, (type : referent). In either case, the referent marker may be omitted. Each concept node is followed by a list of conceptual relations that relate it with other concept nodes.

A graph representing the statement "the processor reads the data from memory" is represented linearly as shown in Example 1:

```
[ 1 : read : * ]
        ->( agnt : agnt ) -> [ 2 ]
        ->( obj : obj ) -> [ 3 ]
        ->( src : from ) -> [ 4 ]
[ 2 : processor : # ]
        ->( det : the )
[ 3 : data : # ]
        ->( det : the )
[ 4 : memory : * ]
```

Example 1. Conceptual graph for ***the processor reads the data from memory***.

3.1.2 The Canonical Basis

A conceptual graph is a combination of concept nodes and relation nodes where every conceptual relation is linked to at least one concept. However, among all such combinations, only a limited number make sense. These conceptual graphs that are meaningful in the domain of interest are called **canonical graphs.** A set of canonical graphs from which all meaningful graphs in the domain of interest can be formed by the canonical formation rules is known as the canonical basis. A requirement is represented by a canonical graph derived from the canonical basis by the application of canonical formation operations. Thus, for every concept, all meaningful structures associated with it are represented as canonical graphs. For example, the verb "load" has the following canonical graph:

```
[ LOAD : load ]
->(agnt) ->[PROCESSOR]
->(obj) -> [VALUE]
->(dest) -> [MEMORY]
```

Example 2. Canonical Graph for ***load***

This canonical graph contains three relations by which other concepts can be attached to "load". If a concept is to be attached to "load" using the "agent" relation, it has to be of type PROCESSOR. Similarly, if a concept is to be attached using the "object" relation, it must be of type VALUE. Canonical graphs in the basis used here are trees with a root concept with all the other concepts as leaves of the tree.

Since English words are often used in several senses, more often than not, it is necessary for a word to have multiple canonical graphs. Another canonical graph for the verb load could be

```
[ LOAD : load ]
->(agnt) ->[COMMAND]
->(obj) -> [VALUE]
->(dest) -> [MEMORY]
```

Example 3. Another canonical graph for ***load***

These two canonical graphs for the verb "load" shown in Example 2 and Example 3 respectively could be used in representing the following sentences "the CPU loads the data into the register" and "the load instruction loads the data into the register."

3.1.3 Operations on Conceptual Graphs

Integration of requirements represented as conceptual graphs requires a set of operations which may be applied to conceptual graphs to form new conceptual graphs. The four operations defined on conceptual graphs are copy, restrict, join and simplify [12].

The **copy** operation reproduces one conceptual graph as a new conceptual graph. A copy of a canonical graph is also canonical.

The **restrict** operation specializes a concept by replacing its type marker with the type marker of a subtype, or by replacing a generic referent marker with an individual referent marker. For instance, [2 : processor : processor] could be [2 : processor : i80486] in Example 1 above, restricting the scope of the concept "processor" in the original graph only to mean the i80486.

The **join** operation unifies two conceptual graphs on matching concepts. Two conceptual graphs may be joined by overlaying one graph on the other so that the identical concepts merge into a single concept. For example, consider the conceptual graph of the sentence "the processor writes the data to memory" shown in Example 4.

```
[ 1 : write : * ]
        ->( agnt : agnt ) -> [ 2 ]
        ->( obj : obj ) -> [ 3 ]
        ->( dest : to ) -> [ 4 ]
[ 2 : processor : # ]
        ->( det : the )
[ 3 : data : # ]
        ->( det : the )
[ 4 : memory : * ]
```

Example 4. Conceptual graph for ***the processor writes the data to memory.***

This conceptual graph joined with the conceptual graph of the sentence "the processor reads the data from memory" shown in Example 1 is represented as follows:

```
[ 1 : read : * ]
        ->( agnt : agnt ) -> [ 3 ]
        ->( obj : obj ) -> [ 4 ]
        ->( src : from ) -> [ 5 ]
[ 2 : write : * ]
        ->( agnt : agnt ) -> [ 3 ]
        ->( obj : obj ) -> [ 4 ]
        ->( loc : to ) -> [ 5 ]
[ 3 : processor : # ]
        ->( det : the )
[ 4 : data : # ]
        ->( det : the )
[ 5 : memory : * ]
```

Example 5. The conceptual graph resulting from the join performed on Examples 1 and 4.

The **simplify** operation deletes any *duplicate* relations that sometimes exist when conceptual graphs are joined. Relations are considered duplicate if they are linked to the same concepts in the same order. Since only duplicate information is removed, the simplify operation does not result in any loss of information.

The four operations discussed above are known as canonical formation operations because they can be used to form new canonical graphs from a set of canonical graphs with the stipulation that restriction operations obey the type hierarchy and the restricted referents conform to their concept or relation types.

3.1.4 Generalization and Specialization of Conceptual Graphs

The operations on conceptual graphs discussed above are *specialization* operations. The restrict operation specializes a concept by replacing the type marker with the type marker of a subtype or by replacing a generic referent marker with an individual referent marker. The join operation specializes a graph by adding conditions and

attributes to one of its concepts from another graph. The copy and simplify operations do not specialize a graph but they do not generalize the graph either.

Generalization proceeds in the reverse order of specialization. Specialization does not always preserve the truth, but generalization does. For example

Some ports in the 68HC11 are 8 bits wide. (3.7)

Port A in the 68HC11 is 8 bits wide. (3.8)

Statement 3.8 which is a specialization of statement 3.7 provides information only about one port in the 68HC11 whereas statement 3.7 provides information about ports in the 68HC11 in general. Specializations may not always preserve the truth.

Generalizations, however, preserve the truth. For example, if we take statement 3.8 and generalize it to produce statement 3.7, the information expressed by statement 3.7 covers what is expressed by statement 3.8, and thus preserves the truth expressed in statement 3.8.

Generalizations do not, however, preserve selectional constraints. For example

The IBM personal computer has an Intel processor. (3.9)

A personal computer has an Intel processor. (3.10)

Statement 3.10, which is a generalization of statement 3.9, does not preserve the constraint "IBM" for the concept "personal computer". This leads to possible loss of

information in generalizations. In the ASPIN system, the parse trees for natural language sentences generated by the parser are analyzed by the Semantic Analyzer to produce conceptual graphs. The Semantic Analyzer takes appropriate canonical graphs from its database and checks the parse trees to see if any generalizations of the canonical graphs are possible. This is repeated for every node in the parse tree and "joins" are performed to construct a conceptual graph.

3.2 Coreferences

Sentences in natural language are composed of words and phrases. Recall that we use the term **expression** to denote a group of words or a phrase that occurs in natural language sentences. Expressions that identify real world entities or actions are called **references**. The entities and actions themselves are the **referents** of such references. References could introduce referents or could point to referents that have already been introduced (anaphoric) or to referents that are introduced later (cataphoric) [4]. Thus, an anaphoric reference has a backward pointer to an antecedent reference whereas a cataphoric reference has a forward pointer to a referent that occurs later. In either case, the two expressions are said to be **coreferences**.

3.2.1 Coreferences in System Specifications

Often in system specifications, the same referent may be referred to several times. These coreferences can occur in several ways:

1. With the same expression, for example,

Before any DMA request can be generated, the internal bus must be granted to the DMA Unit. (3.11)

A certain amount of time is required for the processor to grant this internal bus to the DMA Unit. (3.12)

In the above example, the reference ***internal bus*** in sentence 3.11 is referred to again in sentence 3.12, with the same expression ***internal bus.*** In the conceptual graph representation, both references to the referent "bus" would have the same referent markers "bus" in the respective conceptual graphs.

2. With synonyms, as in

The data is read into an internal temporary register which is not accessible by the CPU. (3.13)

During the deposit cycle, the processor writes the data to memory or I/O space at the address in the destination pointer register. (3.14)

In this example, the reference ***CPU*** in sentence 3.13 is referred to in sentence 3.14 with the word ***processor*** which could be thought of as a synonym for ***CPU*** in the given context.

3. With less specific or more general terms, as in

The MC68HC11A8 is an 8-bit microcontroller unit. with highly sophisticated, on-chip peripheral capabilities. (3.15)

The HCMOS technology used on this device combines smaller size and higher speeds with the low power and high noise immunity of CMOS. (3.16)

In the examples above, the reference ***microcontroller unit*** in sentence 3.15 is referred to in sentence 3.16 with the term ***device*** which is a more general term for ***microcontroller unit*** in this context.

Such multiple references to the same referent are said to be coreferences. This thesis considers only coreferences of concepts having the same referent markers (Case 1). Coreference detection is further restricted to object concepts across sentence boundaries. For purposes of coreference detection, a set of terms have been defined and a set of rules have been developed. The following section presents a discussion of mapping the anaphora problem in natural language understanding to conceptual graphs. Terms that have been defined for this purpose are discussed. Rules developed for coreference detection are discussed in chapter 4.

3.3 Mapping the Anaphora Problem into Conceptual Graphs

Conceptual graphs, as discussed in section 3.1, are a form of semantic networks used in representing information expressed by English language statements. The concept nodes in a conceptual graph represent the various actions and objects concerned with the information and the relation nodes represent the interconnection among the various concept nodes. In the linear notation used by the ASPIN system, conceptual graphs are trees with the references represented by subtrees. The sentence *the data is read into an internal temporary register* would be represented in conceptual graph form as shown in Example 6.

```
[ 1 : read : * ]
        ->( obj : obj ) -> [ 2 ]
        ->( dest : into ) -> [ 3 ]
[ 2 : data : # ]
        ->( det : the )
[ 3 : register : register )
        ->( attr : adj ) -> [ 4 ]
        ->( attr : adj ) -> [ 5 ]
[ 4 : characteristic : internal ]
[ 5 : characteristic : temporary ]
```

Example 6. Conceptual graph for ***the data is read into an internal temporary register.***

The reference "register" is represented in the conceptual graph as the subtree shown in Example 7.

```
[ 3 : register : register )
        ->( attr : adj ) -> [ 4 ]
        ->( attr : adj ) -> [ 5 ]
[ 4 : characteristic : internal ]
[ 5 : characteristic : temporary ]
```

Example 7. The reference tree for the reference ***internal temporary register.***

The remainder of this chapter is devoted to the discussion of mapping the anaphora problem in natural language understanding to the problem of detecting coreferences in the conceptual graph domain. For this purpose, a set of terms have been defined. These terms are discussed in the following subsections.

3.3.1 Definitions and Coreferences

Definition of a referent (entity or action) is said to occur when a reference occurs for the first time in the system specification under consideration. The reference in this

case is said to be a definitional reference or simply a definition. When a reference recurs, it is said to be a non-definitional or recurring reference or ***coreference.*** For example, in the sentences "the system has two modes of operation, the high speed and the low speed modes" and "by default however, the system operates in high speed mode", the reference "system" is a definitional reference in the first sentence and is a coreference in the second.

3.3.2 Head of the Reference

References in the conceptual graph domain are represented by a concept with all its pendant relations. We call this concept the ***head*** of the reference. All the pendant relations of the head act as modifiers of the head of the reference.

3.3.3 Restrictive Relations

Concepts often have their generality restricted by their relationships with other concepts. Such relations that concepts may have are called ***restrictive relations***. Consider the example, "the floppy drive of machine A in diagram I". Here, the generality of the concept "drive" is restricted by its three restrictive relations, one each with the concepts "floppy", "machine A" and "diagram I" as depicted in Figure 7.

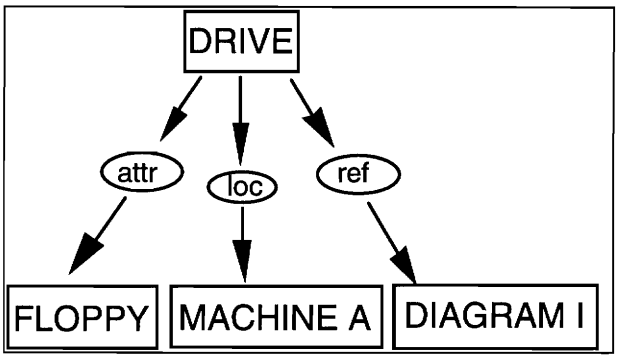

Figure 7. Restrictive Relations

These restrictive relations provide information about the concept that can be used to determine coreference. The algorithm for coreference detection developed is centered around restrictive relations that concepts have. These restrictive relations indicate the scope of the concept, its restrictions and specializations. Restrictive relations could be in the form of adjectives modifying the meaning of the concept itself or could be in the form of location information about the concept, for example, "the data in memory". Currently, for the purposes of coreference detection, eight relation types have been identified as restrictive relations, namely, "attr" for attribute (adj, adv), "dest" for destination, "name" for the name of the object, "purp" for purpose, "loc" for location, "type", "quant" for quantifier, and "size" for size of the quantity the object concept may represent. Table 1 shows all the relation types used by the Coreference Detector as restrictive relations. Table 2 shows the concept types used by the Coreference Detector.

Table 1: All the Relation Types used by the Coreference Detector [5]

Type	Abbr.	Definition	Example
attribute	attr	*a characteristic of the object*	[GATE] -> (attr) -> [HCMOS]
destination	dest	*the destination object of an action*	[WRITE] -> (dest) -> [MEMORY]
location	loc	*location of a stored value*	[VALUE] -> (loc) -> [MEMORY]
name	name	*the name of an object*	[SIGNAL] -> (name) -> [ID : STRB]
purpose	purp	*the purpose of a device or action*	[USE] -> (purp : for) -> [RESET]
quantity	quant	*the number of objects*	[REGISTERS] -> (quant : 8)
size	size	*the size of an object*	[DATA] -> (size) -> [MEASURE : 8-bit]
type	type	*a description of the type of object*	[GATE] -> (type) -> [NAND]

Table 2: Some of the Concept Types used by the Coreference Detector [5]

Concept Types	Subtype Examples
DEVICE	circuit, unit, component, chip
PROCESSOR	computer, microprocessor, controller
LOGIC	adder, alu, gate, decoder
MEMORY	ram, flip-flop, eprom, latch
LOGIC_MEMORY	counter, timer, shift_register
CARRIER	bus, line, wire, pin
TRANSDUCER	sensor, detector, keyboard
DATA	operand, sum, result, multiplier
CONSTANT	0, 1, low, high
INFORMATION	flag, signal, pulse
MEMORY_MEASURE	bit, byte, word, line
TIME_MEASURE	period, cycle, sequence, second

3.3.4 Coherence

The coherence of a concept is a chain from the head concept of the sentence to the head of the reference. The coherence of the head of a reference is also used in the algorithm developed for coreference detection. Coherences provide information about belonging, as in "the floppy drive of machine A" where the concept "floppy" has a coherence with the concept "drive". Restrictive relations, on the other hand provide information about the

specificity or the specializations of the concept. In the conceptual graph notation, there is an arc between any two concepts that are related. Thus, arcs originating from a concept node in a conceptual graph provide information about the concept's relation with other concepts.

Coherences for every definition are stored in a linked list in the implementation of the coreference detector (the implementation of the coreference detector is discussed in chapter 5). The last node in the coherence chain is the concept for which the coherence is stored, the node before that is a concept that has a forward relation with the concept node that represents the definition and so on. For example, if there is a forward relation from concept A to concept B and one from concept B to concept C, then, the coherence for concept C would have B as the first element of the coherence linked list and A as the second element. This coherence is used to determine similarity in cases where little or no information about the reference is available in the form of restrictive relations, quantifiers and determiners.

In the sentence "these bits can select destination synchronized requests on the DRQ pin", whose conceptual graph is shown below, the concept representing the head of the reference "pin" has a directed path (coherence) beginning at the concept "select" terminating at its node, as shown in Figure 8.

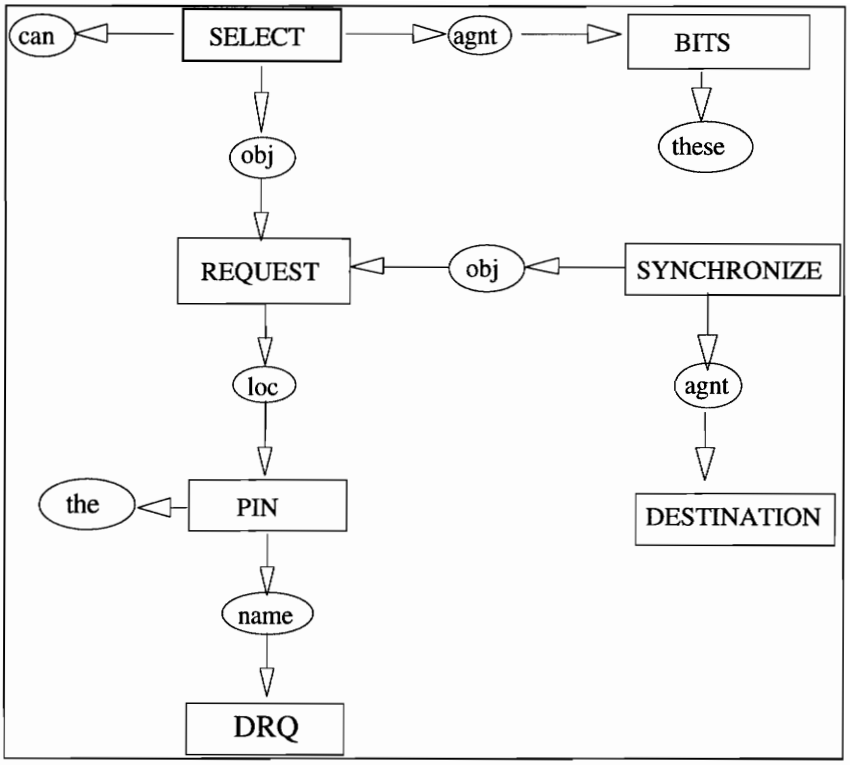

Figure 8. Coherences

This directed path which represents the coherence is stored in a linked list in the data structures that are used in the implementation of the coreference detector.

The discussion above lays the foundation for presenting the set of rules developed for coreference detection. The set of rules developed for coreference detection and the implementation are discussed in the following two chapters.

4. Rules for Coreference Detection

The ASPIN system creates conceptual graphs, one for each input sentence in the system description, that represent information contained in the sentences. These individual conceptual graphs need to be integrated and fed to the Linker and the Model Generator for generating VHDL models. This integration is accomplished by linking all identical concepts across sentences (coreferences), for which purpose, coreferences need to be identified. This chapter discusses a set of rules that has been developed for the purpose of coreference detection of object concepts. This set of rules for coreference detection has been implemented by a Coreference Detector. The implementation and results are discussed in subsequent chapters.

4.1 Coreference Detection

Recall that the input to the ASPIN system is converted into conceptual graphs by the parser and the semantic analyzer. This set of conceptual graphs that represent a system specification is scanned and a definitions table is built in which are stored definitional references, their restrictive relations and coherences. Making use of the rules developed for coreference detection, coreferent concepts are identified and linked together, forming a single large conceptual graph representing the system specified.

4.2 Rules for Coreference Detection

The following coreference detection rules have been developed for deciding if a reference is a definition.

4.2.1 Definition by Introduction

If a reference introduces a referent (as defined in chapter 3), i.e., the head of the reference has not occurred previously, the reference is considered to be a definition. In other words, any reference that points to a referent that has not been *defined* before, i.e., references with head concepts that are distinct from those of references already introduced, are definitional references.

4.2.2 Definition by Implication

If references occur with head concepts having an indefinite article preceding them, regardless of whether other references with the same head concept (same referent marker for the head concept) have occurred before, they are definitions themselves. In such cases, the indefinite articles preceding these head concepts imply the introduction of the references. Thus, if any reference occurs with an indefinite article preceding its head concept, the reference is a definition. For example,

<u>An</u> address register is used as a stack pointer in the Motorola 68000 microprocessor.

(4.1)

In this example, the indefinite article *an* before the head of the reference *register* implies that this occurrence of address register is distinct from all other prior mentions (if any) of address registers in the system description. Therefore, this occurrence of address register is considered to be a definition.

4.2.3 Definition by Modification

If head concepts of references that have occurred elsewhere in the description occur with distinguishing adjectives or qualifiers like "another", "different", and "new", then the references are considered to be definitions. These qualifiers have the effect of modifying the referent pointed to by the reference that occurred earlier, thereby making this occurrence of the reference a definition itself. Further, any reference with a head concept that has any of these qualifiers, independent of other properties it may have, can be classified as a definition. For example,

An address register is used as a stack pointer in the Motorola 68000 microprocessor. (4.2)

<u>Another</u> address register is used for . . . . (4.3)

In this example, though the reference "address register" in sentence 4.3 has occurred earlier in sentence 4.2, the modifier "another" implies that this address register is different from the address register referred to in sentence 4.2. If the head concept of a reference is qualified by an adjective such as "another", "different", or "new", regardless of other considerations, it can be concluded that the reference is a definition. There are exceptions to this rule, however. When heads of references are qualified by adjectives that are time

and place relaters, these references actually point to a referent that has already been defined [4]. Some examples of such time and place relaters are *earlier*, *former*, *preceding*, *prior* and *previous* . For example,

Address registers and data registers are the general purpose registers on the MC68000. (4.4)

While some of the <u>former</u> can be used as the stack pointer, the <u>latter</u> cannot. (4.5)

The time relaters "former" and "latter" in sentence 4.5 point to the references "address registers" and "data registers" respectively in sentence 4.4.

4.2.4 Definition by *Dissimilarity*

If a reference does not fall into any of the three categories discussed above, it is a definitional reference only if the referent it points to is not ***similar*** to any of the referents pointed to by other references that have occurred before. Note that two references that have the same English language expressions may actually point to two different referents. For example, in the sentence

The data stored in memory is replaced with the data in the register. (4.6)

the first occurrence of the reference "data" and the second occurrence of "data" in sentence 4.6 point to two different referents, despite having the same English language expression.

Rules for detecting ***similarity*** of referents pointed to by references are discussed below.

4.3 The Similarity Detection Process

The coreference detection process maintains a list of concepts that represent definitional references in a ***definitions table***. This table is built as the process scans through the set of conceptual graphs representing the system description and is similar to the definition table used by a translator. For every definition, the type marker for its head concept, the determiner (if any), graph number, concept number, restrictive relations of the head concept of the reference (if any), quantifiers (if any), a list of the concepts of similar references (determined by the process), and all coherences for the head concept are stored.

Coherences can at times provide valuable information about a reference that may help in deciding its coreference with another reference. When concepts representing references do not have quantifiers, determiners or restrictive relations or when no decision can be made based on the quantifiers, determiners and restrictive relations, and if both concepts (the one representing the current reference and the other representing the definitional reference) have coherences, decisions about coreference can sometimes be made based on their coherences. Example 8 illustrates a reference that has no information in the form of determiners, quantifiers and restrictive relations, but has a coherence that is helpful in determining coreference.

```
[ 1 : write : increment ]
        -> ( obj  ) ->[ 2 ]
[ 2 : data  :  data ]
[ 3 : write : store ]
        ->( obj ) -> [ 2 ]
        ->( loc : in ) -> [ 4 ]
[ 4 : memory : * ]
```

Example 8 Conceptual Graph for ***"data stored in memory is incremented."***

In Example 8, which is the conceptual graph for the sentence "data stored in memory is incremented", of particular interest is the concept corresponding to the reference ***data.*** It has no restrictive relations, no quantifiers and no determiners attached to it. Consider the occurrence of ***data*** in another sentence "data read from the register is incremented", whose conceptual graph is represented in Example 9 below:

```
[ 1 : increment : increment ]
        ->( obj ) -> [ 2 ]
[ 2 : data : data ]
[ 3 : read : read ]
        ->( obj ) -> [ 2 ]
        ->( from ) -> [ 4 ]
[ 4 : register : register ]
        ->( det : the )
```

Example 9. Conceptual graph for ***"data read from the register is incremented."***

When the occurrences of "data" in the two examples are compared, because neither reference has restrictive relations, quantifiers or determiners, no conclusion can be made about their coreference from this information alone. However, it is apparent to the reader that the two occurrences point to two different pieces of data. It is here that coherences help. In Example 8, the concept corresponding to the reference "data" has a directed path

write -> data

representing the coherence for the concept "data". In Example 9, the concept for the reference "data" has a directed path

read -> data

representing the coherence for the concept "data". From this information, since the coherence of the second reference does not match the coherence of the first, it can be concluded that the two are not coreferences. A more detailed discussion of the matching of coherences is provided in Rule 7 of the similarity detection process.

When a reference in a description being scanned in the conceptual graph under consideration, or the ***current reference***, has the same English language expression as that of a reference in the definition table (in the conceptual graph representation, the two will have the same referent markers), the restrictive relations and the coherences of the current reference are compared with those of the definitional reference. Coreference by ***similarity*** of two references, one the current reference and the other a definitional reference with which the current reference is being compared, is decided by a process of elimination based on the following:

1. If the head concept corresponding to the current reference has a different type marker from that of the definition, the two references are not coreferent. Though two references have head concepts with similar referent markers (recall that comparison of two references is done only if their heads have identical referent markers), if their concept types are different, then the two are not coreferences.

2. If the head of the current reference has more restrictive relations than that of the definition, then the two are not coreferent. For example,

The floppy drive of machine A is accessed. (4.7)

The floppy drive of machine A in diagram I is written to. (4.8)

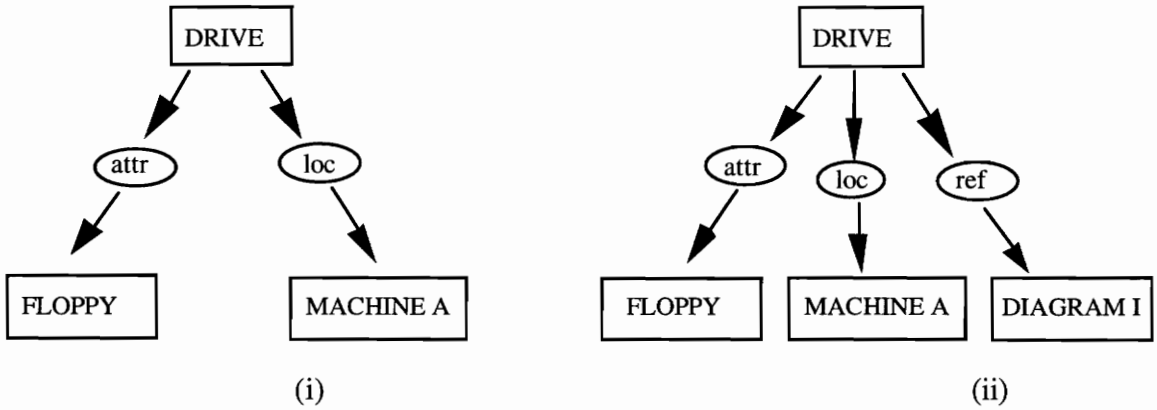

Figure 9. Head of a reference with more restrictive relations (ii) than that of another (i)

As illustrated in Figure 9, the head concept "drive" of the reference "floppy drive" in 4.7 has two restrictive relations, one each with the concepts "floppy" and "machine A" respectively. The head concept "drive" of the reference "floppy drive" in 4.8, however, has three restrictive relations, one each with the concepts "floppy", "machine A" and "diagram I". Assuming that sentence 4.7 occurs before sentence 4.8 in the system specification, it can be inferred that the reference "drive" in sentence 4.8 is more specialized than that in sentence 4.7 and hence the two references are not coreferent. Definitions are more likely to contain all the information about a reference than are coreferences. Therefore, definitions tend have all the restrictive relations that a referent actually possesses and thus candidate for coreference which has more restrictive relations (and is thus more

specialized) than the definitional reference with which it is being compared is not coreferent with that definition.

3. If one or more of the restrictive relations of the head of the current reference does not have an equivalent in the set of restrictive relations of the head of the definition, the two are not coreferent.

The floppy drive of <u>*machine A*</u> *in diagram I is accessed.* (4.9)

The floppy drive of <u>*machine B*</u> *in diagram I is enabled.* (4.10)

Figure 10 illustrates this idea.

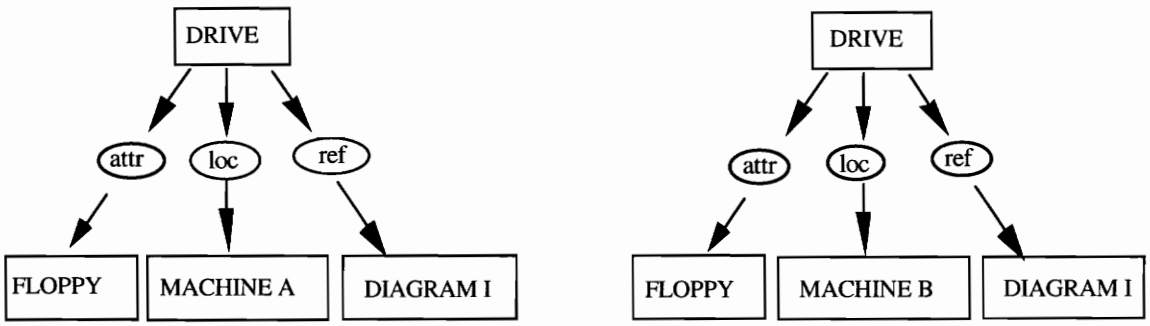

Figure 10. Heads of references with different restrictive relations.

In sentence 4.10, it is clear that the referent pointed to by the reference "floppy drive" is physically located in machine B, whereas the referent pointed to by "floppy drive" in sentence 4.9 is physically located in machine A, and thus the two references are not coreferent.

4. If either the definitional reference or the current reference is qualified by quantifiers and

   if the quantifiers are different, then the two references are not coreferent, for example,

   All registers should be cleared before being used. (4.11)

   Some registers are cleared when the system starts up. (4.12)

   Further, if the quantifier is "some" in one reference, then, regardless of the quantifier that the other reference has, the two references are not coreferent. In fact, if the current reference has the quantifier "some", it is considered to be a definitional reference itself.

5. If the current reference is qualified by relative pronouns ("this", "that", "these", and "those"), then it is coreferent with the most recent definitional reference (having the same referent marker), as illustrated in the example below:

   Figure 1 shows a number of registers and control bits mentioned in the previous section. (4.13)

   These control bits are used to enable other on-chip peripheral subsystems. (4.14)

It is clear that "control bits" in sentence 4.14 is a reference to "control bits" in sentence 4.13. (These two examples were taken from the M68HC11 Reference Manual [8])

6. If the head of the current reference has the definite article (the) as its determiner, and has no restrictive relations or coherences, the current reference is coreferent with the

most recent definitional reference having the same English Language expression (the same referent marker in the conceptual graph representation) in the system description. If there is no reference having the same English Language expression, then the current reference is a definition itself.

7. If the current reference has no determiner, and inconsistencies or contradictions are detected in the coherences of the head of the current reference and that of the definition, then the two are not coreferent. Comparable or peer concept nodes (nodes at the same distance from the terminal node in the coherence of the concepts being compared) are compared as follows: if the concept of one of the peer nodes is not a subtype of that of the other in the concept hierarchy, then we assume that there is an inconsistency in the coherences of the heads of the two references.

8. If the head of the current reference's referent marker is a generic referent marker (a generic referent marker is a "*" in the notation used by the ASPIN system), that reference is not similar to another reference with the generic referent marker, and, in fact, is assumed to be a definitional reference.

If in this process of similarity detection, it is determined that the current reference is not coreferent with a definitional reference, the comparison is resumed with the next definition in the definitions table that has the same referent marker. If the current reference has been compared with all entries in the definitions table and if no coreference is detected, it is concluded that the current reference is a definitional reference. If, however, it is determined that the current reference is coreferent with some definition, then the graph number and the concept number of the head of the current reference is added to the

definitional reference's list of coreferences, the comparison process is abandoned, and the process is continued with the next reference in the conceptual graph input set.

This algorithm, as stated in the introduction, has been implemented and has been proven to be effective in detecting coreferences. Chapter 5 discusses the implementation in detail. Chapter 6 looks at some examples and analyzes the results for some test sets. Chapter 7 draws conclusions on the limitations of the algorithm and briefly discusses developments and enhancements that are feasible.

5. The Implementation

The rules for coreference detection discussed in the previous chapter have been implemented by a program called the Coreference Detector. The Coreference Detector uses a definitions table in which all relevant information about definitions are stored. This chapter discusses the structure of the definitions table and provides an overview of the functioning of the Coreference Detector. The first section deals with the structure of the definitions table. The second section details the scanning and storing of references, one at a time, from the input set of conceptual graphs. The next section discusses the implementation of the rules for coreference detection and addition of references to the definitions table. The last section discusses the generation of the integrated conceptual graph.

5.1 The Definitions Table

A definitions table is used in the Coreference Detector (CD) to store definitional references and information associated with them that is extracted from the input conceptual graphs. This table is similar to the symbol table used in translators. The entries in the definitions table are arranged in reverse alphabetical order of the referent markers. Table 3 shows the fields in the definitions table that contain information about the definitional reference:

Table 3. The Definitions Table

name	referent marker of the head of the definitional reference
type	type marker for the head of the reference
no_of_relations	number of restrictive relations of the head
graph_no	sequential number of the conceptual graph to which the reference belongs in the input set
concpt_no	unique identification number associated with the head concept of the reference
similar_to	a list containing the head concept identifier and the graph number of every reference determined by the CD to be coreferent with the definitional reference this entry represents
det	determiner of the head of the reference this entry represents
quant	quantifier for the head of the reference this entry represents
rel	a list that stores the referent and type markers of all restrictive relations of the reference this entry represents
related_to	a list that stores the coherences for this reference

5.2 The Coreference Detector

The set of conceptual graphs that represent individual sentences in a system description are stored in a file that is scanned by the Coreference Detector. During this scanning process, the CD builds the definitions table and compares every new reference that is a subtype of the "object" concept type with references in the definitions table that have the same referent markers. If there are multiple entries in the definitions table that have the same referent markers, comparison proceeds with

the most recent entry (since this definitional reference has the highest probability of being referred to again), followed by the next most recent entry and so on. If it is concluded that the extracted reference and some reference in the definitions table are coreferences, the similar_to field of the definitional reference is updated and the process continues with the next concept in the input set. If the extracted reference is not "similar" to any reference in the definitions table, it is added as a new definitional reference to the definitions table itself. The functioning of the Coreference Detector is described in the following subsection in greater detail.

5.2.1 The Functioning of the Coreference Detector

5.2.1.1 The Input

The CD extracts concepts one at a time from the input set of conceptual graphs. The unique identifier label associated with every concept is stored in the concept number field by the CD. Conceptual graph boundaries are indicated by a period in the input set. The CD initializes the graph number to be 1 when the analysis commences and increments this value every time a graph boundary is sensed.

Along with every concept, relevant information associated with the concept is extracted and stored in a temporary structure whose format resembles the format of one element of the definitions table. The relevant information helps in determining coreference and is entered into the definitions table along with references that are determined by the Coreference Detector to be definitions. The restrictive relations of every concept are stored in the temporary structure as follows: each relation that the

concept has in the conceptual graph is examined. If the relation is one of "attr" for attribute (adj, adv), "dest" for destination, "name" for the name of the object, "purp" for purpose, "loc" for location, "type", "quant" for quantifier, or "size", then, the identifier of the concept to which the reference is related is stored. Then, the CD searches for the identifier in the entire conceptual graph, locates the concept, picks up its referent and type markers and stores them in the restrictive relations field of the temporary structure. All the relations that the concept has are examined in the same fashion and restrictive relations are stored. The directed paths terminating at the concept node of the reference that represent coherences are also detected and stored as follows: the current concept is stored as the leaf node of the directed path (tree). Concepts that are related to the current concept are detected and stored as the father nodes of these leaf nodes in separate trees. Concepts that have relations with the concepts of these father nodes are stored one level higher in the trees. In this manner, all coherences are detected and stored.

```
[ 1 : read : * ]
        ->( agnt : agnt ) -> [ 2 ]
        ->( obj : obj ) -> [ 3 ]
        ->( src : from ) -> [ 4 ]
[ 2 : processor : # ]
        ->( det : the )
[ 3 : data : # ]
        ->( det : the )
[ 4 : memory : * ]
```

Example 10. Conceptual Graph for ***the processor reads the data from memory.***

Example 10 represents the conceptual graph for the sentence "the processor reads the data from memory". During the input, when the "read" concept with the referent marker "*" is extracted, all its associated relations,

->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(src : from) -> [4]

are also extracted and information about the concept is stored in the temporary structure. The concept with identifier 3 ("data") in the conceptual graph has a coherence with the concept with identifier 1, and thus concept 1 will be stored in concept 3's coherence list. Note that though coreference detection is performed only for concepts that are subtypes of the "object" concept type, the Coreference Detector fetches every concept and its associated information and stores it during the input phase, regardless of the type to which the concept belongs. Concepts that are not subtypes of the "object" concept type, however, are discarded during the analysis phase which is the topic of discussion in the next subsection.

5.2.1.2 The Analysis

Once the input phase is completed, the CD begins the analysis phase. This phase performs the task of identifying definitions and coreferences. Certain checks are made on the extracted references and they are compared with definitions in the definitions table based on the rules for coreference detection discussed in Chapter 4. If the result of a check indicates that a reference is a definition, the reference is added to the definition table. If the result of a comparison indicates that the extracted reference is coreferent with the definition with which it is being compared, information about the extracted reference is added to the similar_to field in the definition's entry in the definition table. If the result of a check indicates that the extracted reference is not coreferent with the definition with which it is being compared, the extracted reference

is compared with the next definition in the definitions table that has the same referent marker. This process is continued either until all definitions with the same referent marker in the definitions table are exhausted or a coreference is detected. If all definitions with the same referent marker are exhausted, the CD decides that the extracted reference is a definition and adds it to the definition table.

The first check that is performed in this phase is to see if the extracted concept is a subtype of the "object" concept type. For this purpose, the type of the concept is looked up in the "isa" concept hierarchy (discussed in Chapter 3). With reference to example 5.1, the referent marker "*" is of type "read", and hence "read" is looked up the concept hierarchy to determine if it is a subtype of the "object" concept type. The hierarchy has an entry for "read" as follows:

read isa move.

Next, "move" is looked up the concept hierarchy and the following entry for move is located:

move isa action.

Next action is looked up to yield the following entry:

action isa universal.

This hierarchy lookup indicates that "read" is a subtype of "move" which is a subtype of "action". In turn, "action" is a subtype of the "universal" concept type. The CD therefore concludes that "read" is not a subtype of the "object" concept type, discards the concept (since the CD handles only concepts of type "object") and its associated information, and continues with coreference detection, extracting the next concept from the input set.

```
[ 1 : action : access ]
    ->( obj : obj ) -> [ 2 ]
    ->( during : during ) -> [ 3 ]
    ->( condition : according ) -> [ 4 ]
[ 2 : data : # ]
    ->( type : adj ) -> [ 5 ]
    ->( det : the )
[ 3 : cycle : cycle ]
    ->( attr : adj ) -> [ 6 ]
    ->( det : the )
[ 4 : register : register ]
    ->( attr : adj ) -> [ 7 ]
    ->( det : the )
[ 5 : mem_measure : byte ]
[ 6 : move : fetch ]
[ 7 : device : pointer ]
    ->( attr : adj ) -> [ 8 ]
[ 8 : characteristic : source ]
```

Example 11. Conceptual graph for *during the fetch cycle, the byte data is accessed according to the source pointer register.*

When an extracted concept is determined to be a subtype of the "object" concept type, the definitions table is examined to see if any concepts with the same referent markers are present. If none is present, the CD concludes that the current reference (the reference extracted from the input set of conceptual graphs and stored in the temporary structure) is a definitional reference by introduction (as discussed in chapter 4) and updates the definitions table with the reference and its associated information. The new reference is inserted into the table in the reverse alphabetical order of the reference marker. This makes all references with the same referent markers cluster together in the definitions table, with the most recent occurrence of the reference being the first element of the cluster. With reference to Example 11, assuming that this conceptual graph is the first in the input set, references with

referent markers "#", "register", "cycle", "byte" and "pointer" will be inserted into the definition table. For example, the entry for "#" (data) will be as shown in Table 4 below.

Table 4. A Definition Table Entry

name	#
type	data
no_of_relations	1
graph_no	1
concpt_no	2
similar_to	nil
det	the
quant	nil
rel	mem_measure, byte
related_to	read

Recall that the "rel" field has two elements, the type of the reference and the referent marker of the reference. The entire definitions table built by the Coreference Detector for a test set of 27 conceptual graphs is shown in Appendix B.

If there are definitional references with the same referent markers, however, the CD initiates a comparison of the extracted reference with the most recently occurred definitional reference. The first check that the CD does in the process of comparison is to see if the extracted reference is a generic reference (the referent marker "*" represents a generic reference). Every generic reference is considered unique and therefore a definition (rule 8) and hence is added to the definitions table. If the

reference is non-generic, the type markers of the two references are compared. If the two type markers are different, the CD decides that the two references are not coreferent (rule 1). This rule is particularly helpful in certain type classifications in the type hierarchy used by the ASPIN system. For example, the referent "output" could be one of three possible concept types, viz., "output_1", "output_2", or "output_3", where

output_1 isa information.
output_2 isa write.
output_3 isa carrier.

Examples 12 and 13 illustrate this difference

[1 : is : is]
->(gindx : 2)
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
[2 : result : result]
->(gindx : 1)
->(det : the)
[3 : signal : signal]
->(gindx : 1)
->(val : adj) -> [4] ->(det : a)
->(loc : on) -> [5]
[4 : constant : high]
->(gindx : 1)
[5 : ***output_3*** : output]
->(gindx : 6)
->(det : the)

Example 12. Conceptual graph for *the result is a high signal on the output.*

[1 : disable : disable]
->(gindx : 1)
->(obj : obj) -> [2]
->(agnt : by) -> [3]
[2 : ***output_2*** : output]
->(gindx : 2)
->(det : the)
->(agnt : of) -> [4]
[4 : clock : clock]
->(gindx : 1)
->(det : the)
->(loc : on) -> [5]
[5 : id : t1]
[3 : reset : reset]
->(gindx : 1)
->(obj : obj) -> [6]
[6 : processor : processor]
->(gindx : 1)
->(det : the)

Example 13. Conceptual Graph for *the output of the clock on t1 is disabled by reset of the processor .*

The reference "output" in Example 12 is of type "output_3" (a carrier), and is of type "output_2" (a write) in Example 13. The CD, by matching the type markers of the two references being compared, decides that the two references are not coreferences.

The CD next looks at the determiner of the head of the extracted reference. If the determiner is the indefinite article ("a" or "an"), the CD decides that the reference is a definitional reference (definition by implication) and adds the reference to the definitions table. If no determiner is present or if the determiner is "the", the CD proceeds to compare the quantifiers (for example, "all", "some", "many", and cardinals) of the heads of the two references. If the quantifiers do not match, then the CD decides that the two are not coreferences. If the quantifier for the head of the extracted reference is "some", the CD decides that the extracted reference is a definition and adds it to the definitions table (rule 4). The CD then looks at the determiner of the head of the extracted reference. If the determiner is "these" or "this", the CD decides that the extracted reference and the definitional reference with which it is being compared are coreferences (rule 5), updates the similar_to field of the definitional reference and continues with the next reference in the input set.

If the determiner of the head of the extracted reference is not one of "a", "an", "this" or "these", the comparison proceeds to the examination of the restrictive relations of the heads of the two references. If the head of the extracted reference has more restrictive relations than the definitional reference (as indicated by the no_of_relations fields for the two references), the CD concludes that the two references are not coreferent (rule 2). This, as explained in Chapter 4, is because, in this case, the extracted reference becomes more specialized than the definitional reference. The

comparison of the extracted reference then proceeds with the next definitional reference having the same referent marker, if any, and this process continues until all definitional references with the same referent markers are exhausted or a coreference is detected. If all definitional references that have the same referent markers as the extracted reference are exhausted, the CD concludes that the extracted reference is a definition and adds it to the definitions table. If a coreference is detected, however, the CD updates the "similar_to" list of the definitional reference and retrieves the next reference from the input set. If the heads of both references have the same number of restrictive relations, then, the CD checks to see if every restrictive relation of the head of the extracted reference has an equivalent in the set of restrictive relations of the head of the definition (rule 3). If any restrictive relation of the head of the extracted reference has no equivalent in the set of restrictive relations of the head of the definitional reference, the CD concludes that the two references are not coreferent and proceeds to compare the extracted reference with other definitional references with the same referent markers. The CD also watches for any adjectives like "another", "different" and "new" and if any of these are present as restrictive relations (recall that adjectives are stored as restrictive relations), the CD concludes that the extracted reference is unique (definition by modification) and adds it to the definition table.

If the head of the extracted reference has no restrictive relations and no coherences, but has "the" as the determiner, the CD concludes that the extracted reference is coreferent with the most recent definitional reference with the same referent marker (rule 6). Since comparison of the extracted reference is performed with definitional references (with the same referent markers) beginning with the most recent

definitional reference, the CD concludes that the extracted reference is coreferent with the first definitional reference with which it is compared.

If, at this point, the CD has not been able to conclude that the two references being compared are coreferent or not, and if the two references belong to the same conceptual graph, it decides that the two references are coreferent and updates the "similar_to" list of the definitional reference and retrieves the next reference from the input set of conceptual graphs. If the two references are from different conceptual graphs, however, the comparison process is continued.

The last step in the comparison process is to compare the coherences of the heads of the two references. Recall that the coherences of the head of every reference are stored as directed paths in separate linked lists. The CD takes entries in the coherences linked list of the head of the extracted reference one by one and checks if they are subtypes or supertypes of corresponding entries in the definitional reference's coherences list. If the coherences list of the extracted reference is longer than the corresponding list of the definitional reference, or if an entry in the list of the extracted reference is not a subtype or a supertype of the corresponding entry in the definition's coherences list, then the CD concludes that the two references are not coreferent (rule 7). This is performed for all the coherences lists that the extracted reference may have.

5.2.1.3 The Output

Once an extracted concept is analyzed, irrespective of whether it is of type "object", it is immediately written into an output file, with all its relations intact as in the input set, and with its identifier (which indicates the concept number in the graph) modified to reflect both its graph number and the concept number. For example, if the input set had a concept in graph number 3 in the form

[2 : data : #]

where 2 is the identifier of the concept, the identifier is modified and written to the output file in the form

[3_2 : data : #]

where 3_2 represents the graph number and the concept number concatenated by an underscore ("_") and this becomes the unique identifier for the concept in the integrated conceptual graph written to the output file.

If the CD has determined that the extracted reference (the reference that is being written into the output file) is coreferent with a reference in an earlier conceptual graph in the input set, say a reference with identifier "3" in graph number 1, the CD adds a ***same as*** relation from the extracted reference to the earlier, coreferent reference as follows:

[3_2 : data : #]

.

-> (same as) -> [1_3]

This is illustrated in Examples 14 and 15

```
[ 1 : action : consist ]
    ->( obj : obj ) -> [ 2 ]
    ->( agnt : agnt ) -> [ 3 ]
[ 2 : cycle : and ]
    ->( and : null ) -> [ 4 ]
    ->( and : null ) -> [ 5 ]
[ 3 : action : transfer ]
    ->( attr : adj ) -> [ 6 ]
    ->( quant : all )
[ 4 : cycle : cycle ]
    ->( attr : adj ) -> [ 7 ]
    ->( det : a )
[ 5 : cycle : cycle ]
    ->( attr : adj ) -> [ 8 ]
    ->( det : a )
[ 6 : id : DMA ]
[ 7 : action : fetch ]
[ 8 : action : deposit ]
```

Example 14a Conceptual Graph for ***all DMA transfers consist of a fetch cycle and a deposit cycle***

```
[ 1 : action : access ]
    ->( obj : obj ) -> [ 2 ]
    ->( during : during ) -> [ 3 ]
    ->( condition : according )->[4 ]
[ 2 : data : data ]
    ->( type : adj ) -> [ 5 ]
    ->( det : the )
[ 3 : cycle : cycle ]
    ->( attr : adj ) -> [ 6 ]
    ->( det : the )
[ 4 : register : register ]
    ->( attr : adj ) -> [ 7 ]
    ->( det : the )
[ 5 : mem_measure : byte ]
[ 6 : action : fetch ]
[ 7 : device : pointer ]
    ->( attr : adj ) -> [ 8 ]
[ 8 : characteristic : source ]
```

Example 14b Conceptual Graph for ***during the fetch cycle, the byte data is accessed according to the source pointer register.***

```
[ 1_1 : action : consist ]
    ->( obj : obj ) -> [ 1_2 ]
    ->( agnt : agnt ) -> [ 1_3 ]
[ 1_2 : cycle : and ]
    ->( and : null ) -> [ 1_4 ]
    ->( and : null ) -> [ 1_5 ]
[ 1_3 : action : transfer ]
    ->( attr : adj ) -> [ 1_6 ]
    ->( quant : all )
[ 1_4 : cycle : cycle ]
    ->( attr : adj ) -> [ 1_7 ]
    ->( det : a )
[ 1_5 : cycle : cycle ]
    ->( attr : adj ) -> [ 1_8 ]
    ->( det : a )
[ 1_6 : id : DMA ]
[ 1_7 : action : fetch ]
[ 1_8 : action : deposit ]
```

```
[ 2_1 : action : access ]
    ->( obj : obj ) -> [ 2_2 ]
    ->( during : during ) -> [ 2_3 ]
    ->( condition : according ) -> [ 2_4
]
[ 2_2 : data : data ]
    ->( type : adj ) -> [ 2_5 ]
    ->( det : the )
[ 2_3 : cycle : cycle ]
    ->( attr : adj ) -> [ 2_6 ]
    ->( det : the )
    ->( same as ) -> [ 1_4 ]
[ 2_4 : register : register ]
    ->( attr : adj ) -> [ 2_7 ]
    ->( det : the )
[ 2_5 : mem_measure :byte ]
[ 2_6 : action : fetch ]
[ 2_7 : device : pointer ]
    ->( attr : adj ) -> [ 2_8 ]
[ 2_8 : characteristic : source ]
```

Example 15 The integrated graph for the two graphs shown in examples 14a and 14b

Examples 14a and 14b are part of a test set of conceptual graphs that were input to the Coreference Detector and example 15 is a part of the integrated graph that the Coreference Detector created. Example 14a is the conceptual graph for the sentence ***all DMA transfers consist of a fetch cycle and a deposit cycle***. Example 14b is the conceptual graph for the sentence ***during the fetch cycle, the byte data is accessed according to the source pointer register.*** Note that after analysis, the CD concludes that "(fetch) cycle" in conceptual graph two is a coreference with "(fetch) cycle" in conceptual graph one.

A set of conceptual graphs representing individual sentences in a system description are fed into the Coreference Detector and the output is a large conceptual graph,

linked at coreferent nodes, that represents the entire system description. This output can then be fed into the VHDL Linker for the creation of VHDL models representing the system specifications that are input. The implementation has been tested on three test sets of conceptual graphs representing over 100 sentences. The results obtained are presented in tabular form along with a discussion in the next chapter.

6. Results

A test version of the coreference detection program has been developed and implemented in the 'C' programming language on Sun Workstations. The Coreference Detector has been tested with three sets of conceptual graphs and the results obtained indicate a high rate of correct classifications. All the errors are incorrect classifications of coreferences as definitions. In no cases were definitions misclassified as coreferences. This chapter discusses the results obtained by the implementation of the Coreference Detector.

6.1 Experimental Results

The Coreference Detector was tested with three sets of conceptual graphs, the first representing a set of 20 sentences, the second representing 27 sentences and the third representing 77 sentences. The first and second sets were conceptual graph representations of sentences taken in order from an Intel Reference Manual and the M68HC11 Reference Manual [8] respectively. The third set contained conceptual graphs of sentences selected from several product descriptions of a single-chip computer. The results of the tests performed on the first and second sets are shown in tabular form in Tables 5 and 6. Since the sentences in the third set (taken from the product description of a single-chip computer) had relatively few coreferences, the results from this set have not been tabulated.

In test runs with these three sets of conceptual graphs, it was observed that the Coreference Detector classified all concept definitions and 85% of all coreferences

correctly. The system errs favorably in that 15% of the coreferences were incorrectly interpreted as definitions, but no definitions were classified as coreferences. Recall that the conceptual graphs from the input set are linked by the Coreference Detector at all coreferent nodes and are integrated into a single conceptual graph. When a coreferent node is improperly classified as a definition, a join that should be performed on the node is missed. On the other hand, when a definition is incorrectly classified as a coreference, two nodes that are not coreferent have a join incorrectly performed on them. Clearly, errors of the first type are better those of the second because an integrated conceptual graph with no incorrect joins (but having less joins than the maximum number of joins possible) is better than a conceptual graph resulting from improper joins performed on two or more conceptual graphs.

6.1.1 Test Set I

In the test run with the set of conceptual graphs of 20 sentences taken from the Intel 80186 Manual (Set I) which had 39 object concept definitions and 24 object coreferences, it was observed that all concept definitions and 22 out the 24 references were correctly detected. The two coreferences that went undetected were incorrectly classified as definitions. The following is a discussion of the two incorrect classifications. The two relevant sentences for the first incorrect classification were "all DMA transfers consist of a *fetch cycle* and a *deposit cycle*" and "*these two bus cycles* cannot be separated by a bus HOLD, a refresh cycle, or any other condition except reset". This is a case of coreference with different expressions, a case not handled by the Coreference Detector. For the second incorrect classification, the two relevant sentences were "this register is not accessible by the *CPU*" and "DMA

transfer rates depend on *processor* bus width and the DMA control unit operating mode", which was again a case of coreference with different expressions not handled by the Coreference Detector.

Table 5 below illustrates the results obtained from this input set (Set I) of conceptual graphs of 20 sentences. Each row corresponds to an object reference in a sentence. A "D" entry in the table indicates that the definition in that sentence has been correctly detected by the Coreference Detector. An "R" indicates that a coreference has been correctly detected by the Coreference Detector. An "X" indicates that a coreference has been incorrectly classified as a definition. There are no definitions wrongly classified as coreferences.

Table 5. Results from Test Set I

Concept		Sentence Number																			
NAME	TYPE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DMA	id	D						R			R	R	R	R	R			R		R	
cycle	cycle	D	R																		
cycle	cycle	D				R															
byte	mem_measure		D																		
pointer	device		D																		
data	data		D	R			R														
register	device			D	R																
CPU	processor				D																
memory	memory					D															
pointer	device					D															
HOLD	id						D														
bus	bus						D														
cycle	cycle						X														
cycle	cycle						D														
bus	bus							D													
processor	processor							X			R										
unit	device							D				R									
logic	logic										D										
bus	bus											D									
HOLD	id												D								
priority	value												D								
HOLD	id													D							
bus	bus													D							
cycle	cycle													D				R			
time	time													D							
clock	device														D			R			
latency	time														D						
DRQ	id															D	R	R			
clock	device															D					
signal	signal															D	R				
bus	bus																D				
logic	logic																D				
time	time																D				
80186	id																		D		
DACK	id																		D		R
processor	processor																		D	R	
signal	signal																		D		
device	device																			D	
PCS	id																				D
line	carrier																				D
signal	signal																				D

Table 6. Results from Test Set II

Concept		Sentence Number																										
NAME	TYPE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
HNDS	id	D													R													
OIN	id	D													R													
output	output_1		D																									
signal	signal		D																									
strobe B	strobe_1		D	R					R		R	R	R	R			R		R	R		R						
system	system		D		R				R								R			R	R							
HC11	id			D																								
data	data				D																							
port C	port				D				R							R	R	R			R		R	R				
port B	port				D																							
input	input					D	R					R									R							
strobe A	strobe_1					D	R	R				R									R	R						
data	data						D																					
device	device						X																					
input	input						D																					
port D	port						D																					
PORTCL	id							D		R	R		R					R										
data	data							D		R																		
register	register							D			R		R					R										
data	data								D																			
strobe F	strobe_1									D																		
clock	device													D														
cycle	cycle													D							R							
data	data															D		R										
system	system															D												
output	output_3																D											
data	data																D				R							
signal	signal																X			R								
pin	carrier																	D					R	R				
output	output_1																						D					
DDRC	id																							D				
bit	mem_measure																							D				
STAF	id																								D	R	R	
I/O	id																								D			
element	element																								D			
subsystem	system																								D			
PH2	id																									D		
bit	mem_measure																									D	R	
clock	device																									D		
STAI	id																											D

6.1.2 Test Set II

Table 6 above illustrates the results obtained from the second input set (Set II) of conceptual graphs of 27 sentences taken from the M68HC11 Reference Manual [8]. In this input set, there were 38 definitions of type object, all of which were correctly detected, and 51 coreferences of which all but 2 were correctly detected by the Coreference Detector. The two coreferences that were not detected were incorrectly classified as definitions. Table 6 shows the definitions and coreferences that were classified by the Coreference Detector. The table shows 40 definitions because two of the coreferences were incorrectly classified by the Coreference Detector as definitions.

The relevant sentences for the three incorrect classifications were: "in this mode, strobe B output is the ready signal to *the external system*" and "when a ready condition is recognized, *the external device* places data on the port D inputs, and then pulses the strobe A input", a case of coreference with synonyms (a case not handled by the Coreference Detector), and "in the pulsed mode, *strobe B* is asserted when the PORTCL register is read" and "*the strobe B output signal* indicates that port C data is ready for the external system", again a case of coreference with dissimilar expressions (not handled by the Coreference Detector). However, the reference "strobe B" which is part of the reference "the strobe B output signal" in the first of these two sentences and the reference "strobe B" in the second sentence are classified as coreferences by the Coreference Detector. This capability is discussed in detail in section 7.1.

6.1.3 Test Set III

In another test run with the set of conceptual graphs of a set of 77 sentences selected from several product descriptions of a single-chip computer, it was observed that the Coreference Detector detected all object concept definitions and approximately 85% of all coreferences correctly. 15% of the coreferences were incorrectly interpreted as definitions.

6.2 Pattern of Coreferences

Figure 11 shows the pattern of coreferences in the first six sentences in Test Set I. The first column is the section heading and the remaining six columns represent the six sentences. The nodes represent the references and links between references represent coreferences. Figure 12 shows the pattern of distances (in sentences) between definitions and coreferences in Test Sets I and II. From Figure 12, it can be seen that 11 coreferences occur exactly one sentence after their definitions, 8 coreferences occur two sentences away from their definitions and so on. Figures 13 and 14 represent the frequency of coreferences for definitions in Test Sets I and II respectively. Thus, from Figure 13, it can be seen that 9 definitions have just one coreference each, 3 definitions have 2 coreferences each, and one definition has 8 coreferences. This gives an idea of the pattern in which coreferences occur in system descriptions.

From Figure 12 and from the results in tables 5 and 6, it can be observed that the first coreference could occur even 10 to 15 sentences away from the sentence in which the

definition occurs. But, the probability of coreference for a definition decreases as we move farther away from the sentence in which the definition occurs in the system description.

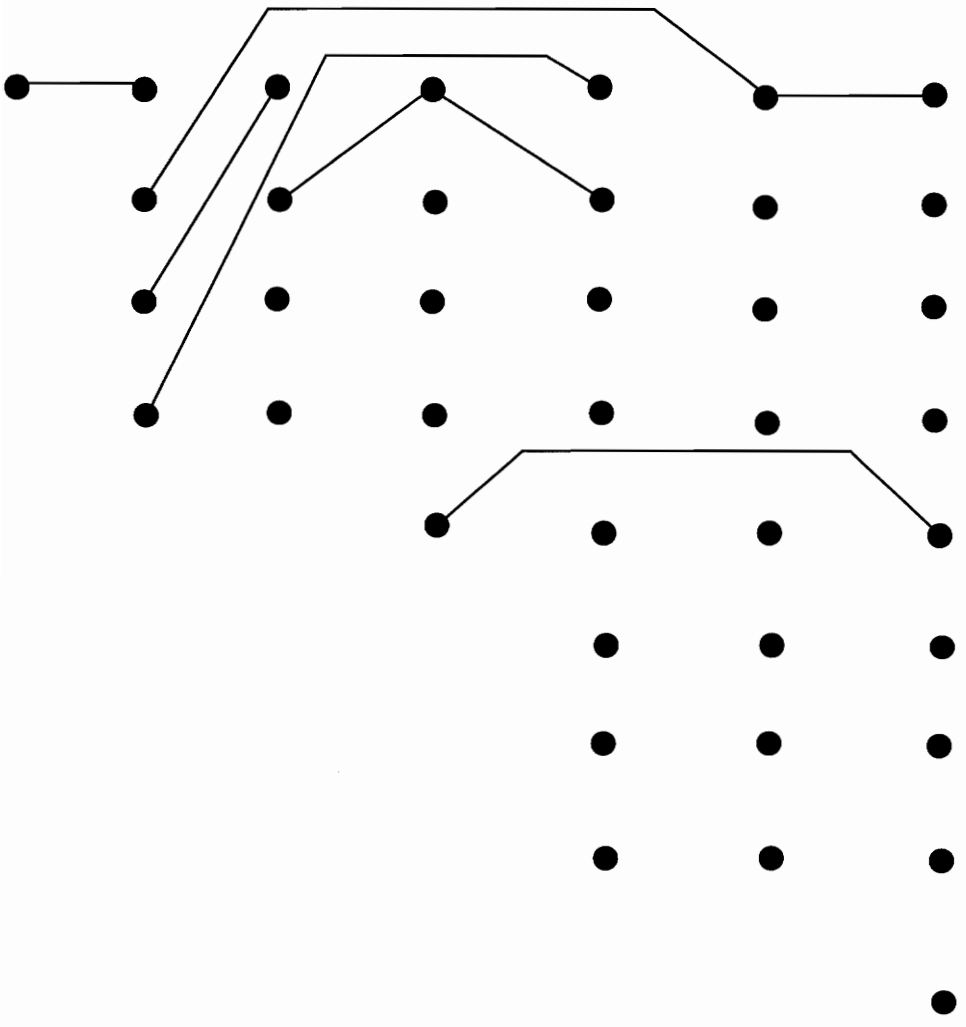

Figure 11. Pattern of Coreferences in Sentences from Test Set I

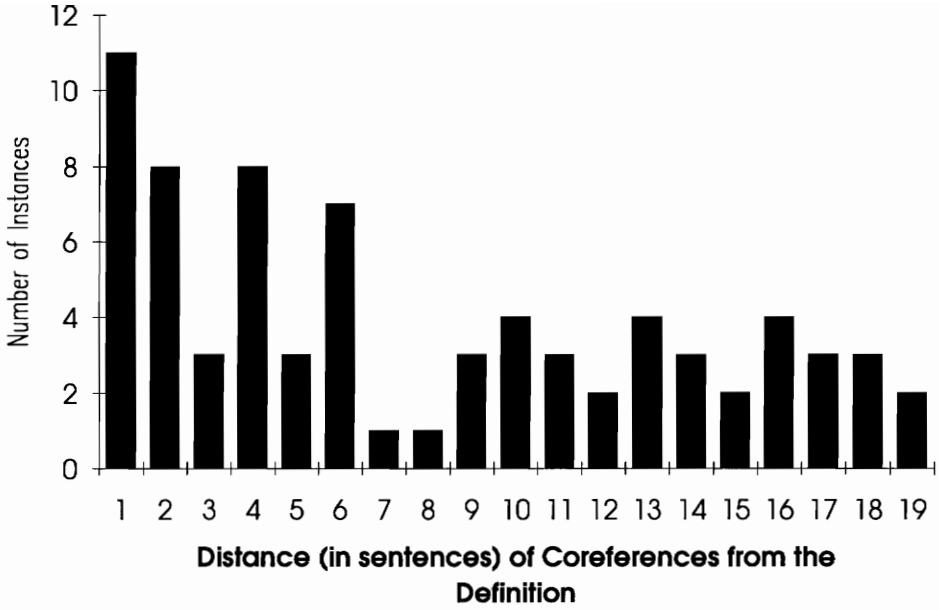

Figure 12. Distance distribution between definitions and coreferences

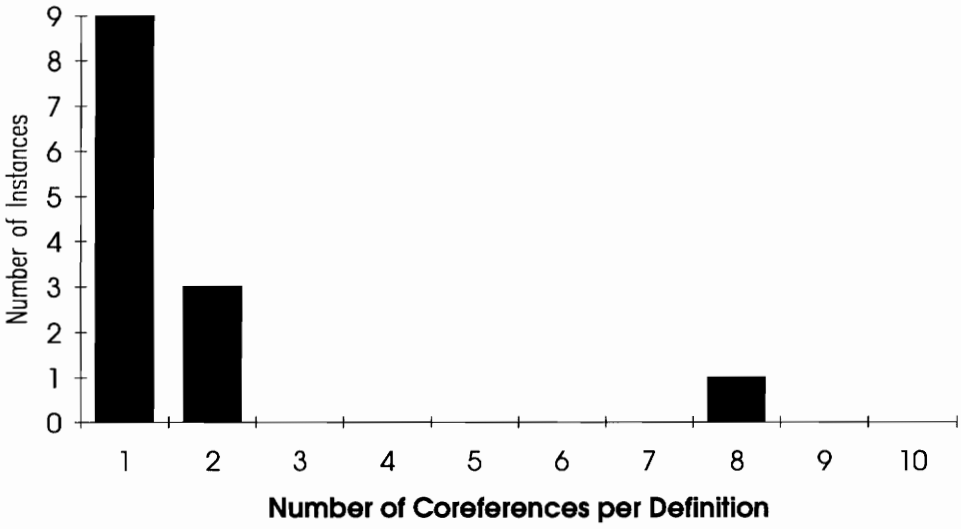

Figure 13. Frequency of coreferences for definitions in Test Set I

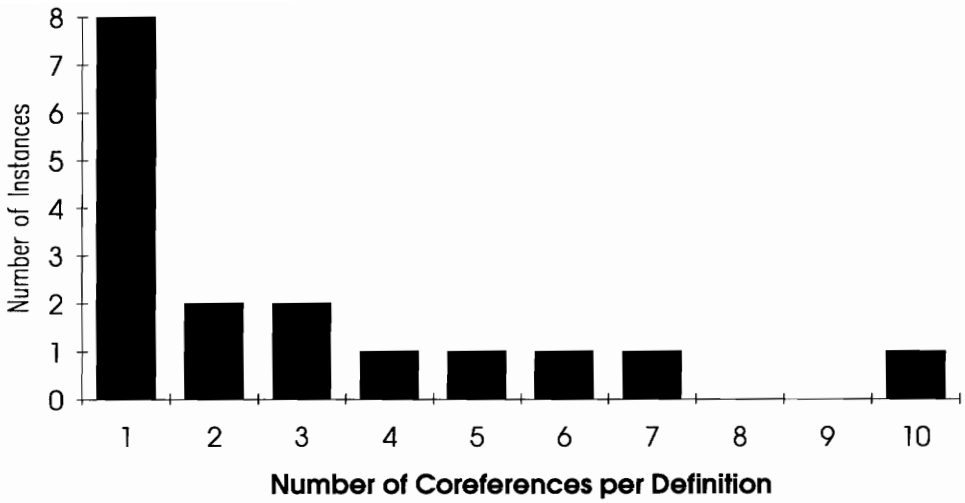

Figure 14. Frequency of coreferences for definitions in Test Set II

Chapter 7 provides a discussion on the limitations of the Coreference Detector and the rules of coreference detection. It also provides an insight into expanding the rules to coreference detection of synonyms and action concepts. A complete listing of the conceptual graphs of sentences in test set II, and the integrated conceptual graph for generated by the Coreference Detector for this set is attached to Appendix E.

7. Conclusions and Future Work

A prototype system has been developed to detect coreference of "object" type concepts in natural language specifications. The results indicate that the detection rules are quite reliable, and when failure occurs it tends to be conservative and does not classify definitions as coreferences. Because English is highly ambiguous, and coreference detection is a major problem in linguistics, resolution of questionable cases will need to rely on user interaction. Further studies are needed to decide when the interaction should occur. This chapter summarizes the capabilities and limitations of the system and suggest directions for future work.

7.1 System Capabilities

The Coreference Detector is able to handle a large number of coreferences that occur with the same expression in system specifications. In test runs with these three sets of conceptual graphs, it was observed that the Coreference Detector correctly classified all concept definitions and 85% of all coreferences. The system errs favorably in that 15% of the coreferences were incorrectly interpreted as definitions, but no definitions were classified as coreferences. The Coreference Detector currently handles all concepts that are classified as a subtype of the "object" concept type in the type hierarchy. Eight relation types have been identified as restrictive relations in the process of coreference detection. Multiword expressions representing references may have parts of them representing other references. For example, "the strobe B output signal", contains the references "strobe B", "output" and "the strobe B output signal".

The rules developed for coreference detection and the implementation detect coreference of these references as well. Table 7 shows this capability.

The four columns in the table represent the reference that is classified as a definition in the first column, references that are coreferent with the definition in the second column, and the graph number and the concept number for the heads of the references in the third and fourth columns. The head of each reference is shown in bold. The table shows that the reference "strobe B" in graph number 16 (shown in italics) which is part of the reference "the strobe B output signal" has been marked as a coreference to "strobe B" in graph number 2. A more detailed analysis of the results is presented in tabular form in Appendices C and D.

7.2 System Limitations

The Coreference Detector does not handle coreferences of object concepts that occur with different expressions. For example, in the sentences "in the full input handshake mode, strobe B is the ready signal to the *external system*" and "when a ready condition is recognized, the *external device* places data on the port D inputs and then pulses the strobe A input", the reference "external device" in the second sentence is coreferent with "external system" in the first, but occurs with a different expression. Coreferences of this type are not handled by the Coreference Detector. Further, the Coreference Detector does not handle coreference detection of action concepts.

Table 7. System Capabilities

Definition	Coreferences	Graph Number	Concept Number
strobe B		2	5
	strobe B	3	3
	strobe B	8	5
	strobe B	10	2
	strobe B	11	2
	strobe B	12	2
	strobe B	13	2
	strobe B	16	6
	strobe B	18	4
	strobe B	19	6
	strobe B	21	6
the strobe B output **signal**		16	3
	this strobe B **signal**	19	3
the ready **signal**		2	3
set		28	9
the PORTCL **register**		7	4
	the PORTCL **register**	10	6
	the PORTCL **register**	12	6
	the PORTCL **register**	17	6
ready		6	8
port D		6	9
port B		4	7
port C		4	4
	port C	8	8
	port C	15	2
	port C	16	8
	port C	17	7
	port C	20	10
	port C	22	4
	port C	23	4
port C pins		17	4
	port C pins	22	2
	port C pins	23	2
one		28	7
	one	14	9
	one	1	9
port C **data**		16	4
	data	17	2
	data from port C	20	9

7.3 Directions for Future Work

The problem of handling coreference detection of head concepts of references having different referent markers (coreference with different expressions) has to be handled differently. When the head of a coreference has a different referent marker from that of the definition it is coreferent with, typically it is either a synonym of the definition or is a more generic referent marker for the definition. To take care of the synonym problem, one possible solution is to maintain a list of synonyms. This problem can be taken care of in the dictionary of the parser. Coreference by a more generic referent marker can be handled with the help of the type hierarchy. Every candidate for coreference of type "object" should be compared with every definition that is a super type of the candidate.

The problem of coreference detection of action references is far more complex. Some rules that could be used for coreference detection of action references by elimination are

1. Two action references are different if the objects of the actions are not the same.
2. Two action references are different if the agent or the instrument of one is different from that of the other. However, in some instances, the instrument of one action may be a part of that of the other. In these cases, the conclusion that the two action references are necessarily different cannot be made as evident in the two sentences below:

   The *ALU* increments the counter.

   The *CPU* increments the counter.

3. Two action references are different if they have different or contradictory attributes.

   e. g. The counter is incremented *twice*.

   The counter is incremented *thrice*.

   The action reference "increment" is qualified by different attributes in the two sentences and thus the two instances of "count" in the two sentences are different.

4. If one action occurs in one tense and another in a different tense, the two action references could be different.

   e.g. The program counter *is incremented* to point to the next instruction in the instruction queue.

   When the next instruction is fetched and executed, the program counter *will be incremented.*

   However, some combinations of tenses for actions could indicate coreference.

   e.g. The program counter *is incremented* to point to the next instruction in the instruction queue.

   Once the program counter *has been incremented*, the next instruction is fetched from the instruction queue and is executed.

5. If the two references have different modifiers, the two actions are different. In particular, any action that is modified by a modal like "can", "could", and "may" that indicate possibility is generic and unique and is therefore a definition.

   e. g. The counter *can* be incremented.

   The counter will be incremented.

If reliable coreference detection of object as well as action references can be developed, it will be a significant step forward in automated analysis of specifications, and this thesis presents one step toward achieving that objective.

Bibliography

[1] Arendse Berth, "Treatment of Anaphoric Problems in Referentially Opaque Contexts", International Scientific Symposium on Natural Language and Logic, Hamburg, Germany, May 9-11, 1989.

[2] Walling Cyre, "Design of a Restricted Sublanguage", Proc. Theoretical Approaches to Natural Language Understanding Workshop, Halifax, Novia Scotia, May 28 - 30, 1985.

[3] J. J. Granacki, Jr., Understanding Digital Systems Specifications Written in Natural Language, Technical Report CRI-87-02, Computer Research Institute, USC, Los Angeles, CA, December 1986.

[4] Sidney Greenbaum et al., A Grammar of Contemporary English, Longman Group Limited, London, 1973.

[5] Rob Greenwood, Semantic Analysis for System Level Design Automation, Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA 1992.

[6] Alex Honcharik, J. Armstrong and W. R. Cyre, "Mapping Conceptual Graphs to VHDL Descriptions, VHDL International User's Forum, Washington, DC, October 18-21, 1992.

[7] IEEE, IEEE Standard VHDL Language Reference Manual, IEEE, NY, 1988.

[8] MOTOROLA, M68HC11 Reference Manual, PRENTICE HALL, Englewood Cliffs, N. J. 07632, 1989.

[9] M. L. Mugnier and M. Chien, "Polynomial Algorithms for Projection and Matching", Proc. 7th Annual Workshop on Conceptual Graphs, Las Cruces, NM, pp. 49-58, July 1992.

[10] Sung H. Myaeng and Aurelio Lopez-Lopez, "A Flexible Algorithm for Matching Conceptual Graphs", Proc. 6th Annual Workshop on Conceptual Graphs, Binghamton, NY, pp 135-152, July 1991.

[11] B. Singh, J. Wicks, P. Wright and J. R. Armstrong, "The Modeler's Assistant: A CAD Tool for Behavioral Model Development", CHDL 93, Toronto, Canada, April 1993.

[12] J. F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, MA, 1984.

[13] J. F. Sowa, "Semantic Networks", Encyclopedia of Artificial Intelligence, S. C. Shapiro, ed. , John Wiley & Sons, New York, 1011-1024, 1987.

[14] Aniruddha Thakar and Walling Cyre, Visual Feedback for Validation of Informal Specifications, accepted for publication at the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Durham, January 1994.

[15] T. Winograd, Understanding Natural Language, Academic Press, Inc. , NY, 1972.

Glossary

referent a real world action or abstraction that is referred to in a system description

reference phrase or clause in a sentence that points to a referent; in the current context it is a conceptual tree with the root node of the conceptual tree as its **head**

coreference a reference that identifies a previously defined referent

definitional reference a reference that introduces a new referent

concept the representation of a reference in a conceptual graph, consisting of a type marker and a referent marker

referent marker the referent marker is the name or identifier of the referent that the concept represents

type marker the type to which the referent represented by the concept belongs

coherence the chain of concepts and relations from the head of the sentence tree to the head of the reference tree

Appendix A. The Concept Type Hierarchy

'flip-flop' isa memory.
'op-code' isa command.
0 isa constant.
1 isa constant.
accept isa read.
access isa read.
accomplish isa action.
accumulate isa action.
accumulator isa memory.
acknowledge isa write.
act isa action.
act_struct isa structure.
action isa universal.
activate isa action.
add isa operate.
addend isa data.
adder isa logic.
address isa action.
address isa value.
affect isa event.
affect isa event.
allow isa action.
also isa characteristic.
alter isa affect.
alu isa logic.
and isa operate.
apply isa write.
are isa be.
argument isa data.
array isa data_struct.
arrive isa action.
assembler isa command.
assert isa apply.
assert isa write.
attempt isa action.
augend isa data.
average isa operate.
be isa state.
begin isa cause.
bit isa mem_measure.
branch isa call.
breakpoint isa address.
buffer isa memory.
bus isa carrier.
byte isa mem_measure.
cable isa carrier.
cache isa memory.
cache_1 isa memory.
cache_2 isa action.
call isa event.
carrier isa device.
cause isa event.
change isa affect.
characteristic isa universal.
chip isa device.
circuit isa device.
circuitry isa device.
clear isa write.
clock isa device.
code isa command.
code isa information.
coincide isa state.
command isa value.
compare isa operate.
complete isa terminate.
component isa device.
compute isa operate.
computer isa processor.
connect isa move.
constant isa value.
contain isa physic_struct.
content isa information.
continue isa state.
control isa affect.
controller isa processor.
convert isa operate.
count isa data.
counter isa logic_memory.
cpu isa processor.
cycle isa act_struct.
data isa value.
data_struct isa structure.
deassert isa apply.
decode isa operate.
decoder isa logic.
decrement isa operate.
delete isa write.
destroy isa affect.
detect isa action.
detector isa transducer.
determine isa action.
device isa object.
difference isa data.
direct isa affect.
disable isa affect.
disk isa memory.
display isa transducer.
divide isa operate.
dividend isa data.
divisor isa data.
drive isa write.
edge isa event.
eeprom isa memory.
element isa device.
emit isa action.
enable isa affect.
enter isa call.
eprom isa memory.
equal isa state.
event isa universal.
execute isa operate.
fetch isa read.
fill isa action.
flag isa information.
follow isa act_struct.
force isa action.
function isa action.
gate isa logic.
generate isa action.
halt isa terminate.
handle isa action.
have isa physic_struct.
hold isa physic_struct.
human isa universal.
i80486 isa processor.
id isa object.
improve isa affect.

increment isa operate.
index isa address.
indicate isa state.
indicator isa transducer.
information isa value.
inhibit isa affect.
initialize isa write.
initiate isa cause.
input_1 isa read.
input_2 isa carrier.
input_3 isa information.
instruction isa command.
interpret isa action.
interrupt isa call.
is isa state.
isa(co-processor,processor).
isa(mode,state).
issue isa action.
jump isa call.
keyboard isa transducer.
label isa address.
latch_1 isa read.
latch_2 isa memory.
level isa information.
line isa carrier.
link isa carrier.
load isa write.
locate isa action.
location isa address.
logic isa device.
logic_memory isa logic.
logic_memory isa memory.
m6811 isa processor.
machine isa processor.
macro isa command.
make isa cause.
manipulate isa action.
measure isa universal.
mem_measure isa measure.
memory isa device.
message isa information.
mhz isa measure.
microcomputer isa processor.
microcontroller isa processor.
microprocessor isa processor.
modify isa affect.
move isa action.
multiplexer isa logic.
multiplicand isa data.
multiplier isa data.
name isa universal.
next isa act_struct.
object isa universal.
occupy isa action.
occur isa event.
operand isa data.
operate isa action.
order isa structure.
output_1 isa information.
output_2 isa write.
output_3 isa carrier.
overflow isa state.
parameter isa data.
pass isa write.
perform isa operate.
period isa time.
peripheral isa device.
physic_struct isa structure.
pin isa carrier.
point isa action.
pointer isa address.
poll isa read.
port isa memory.
pos_edge isa edge.
procedure isa command.
processor isa device.
produce isa action.
product isa data.
program isa command.
prom isa memory.
pulse_1 isa information.
pulse_2 isa action.
push isa write.
quotient isa data.
ram isa memory.
read isa move.
receive isa operate.
record isa write.
register isa memory.
release isa operate.
remain isa state.
remainder isa data.
remove isa operate.
represent isa action.
request isa call.
require isa action.
reset isa write.
result isa data.
resume isa cause.
return isa call.
rise isa event.
rom isa memory.
rotate isa operate.
routine isa command.
run isa action.
save isa write.
selector isa logic.
send isa move.
sensor isa transducer.
service isa action.
set isa write.
shift isa operate.
signal isa information.
software isa command.
specify isa action.
stack_1 isa data_struct.
stack_2 isa operate.
startup isa cause.
state isa universal.
step isa action.
stop isa terminate.
storage isa memory.
store isa write.
strobe_1 isa information.
strobe_2 isa action.
structure isa universal.
subprogram isa command.
subroutine isa command.
subtrahend isa data.
sum isa data.
switch isa logic.
system isa device.
take isa read.
tcycle isa act_struct.

terminal isa transducer.
terminate isa event.
test isa operate.
time isa measure.
timer isa logic_memory.
total isa measure.
transducer isa device.
transfer isa move.
translate isa operate.
transmit isa write.
unit isa device.
use isa action.
value isa object.
variable isa data.
vector isa mem_measure.
wire isa carrier.
write isa move.

Appendix B shows the definitions table generated by the Coreference Detector for test set II.

Appendix B. The Definitions Table for Test Set II

HEAD	TYPE	D	Q	N	RELATIONS		COHERENCES			GRAPH NO. CONCEPT NO. PAIR OF COREFERENCES																						
zero	constant			0			is	is	select	1	11																					
system	system		some	1		external	output	use		15	6																					
system	system	the		1	external		signal	is		2	7	4	2	8	3	16	5	19	2	20	2											
subsystem	system	the		1	I/O		element	is		24	5																					
strobe	strobe_1			1	F		assert	cause		9	6																					
strobe	strobe_1			1	A		input	is		5	4	6	11	7	5	11	9	20	7	21	5											
strobe	strobe_1			1	B		output	is		2	5	3	3	8	5	10	2	11	2	12	2	13	2	16	6	18	4	19	6	21	6	
signal	signal	the		2	strobe	output	indicate			16	3	19	3																			
signal	signal	the		2	ready	system	is			2	3																					
set	constant			0			is	generate		28	9																					
register	register	the		1	PORTCL		latch			7	4	10	6	12	6	17	6															
ready	value			0			condition	place		6	8																					
port	port			1	D		input	place		6	9																					
port	port			1	B		assert	strobe		4	7																					
port	port			1	C		strobe			4	4	8	8	15	2	16	8	17	7	20	10	22	4	23	4							
pin	carrier			1	port		output			17	4	22	2	23	2																	
output	output_1			1	three_state		act			22	3																					
output	output_1			1	strobe		is			2	2	16	7																			
one	constant			0			is	is	select	1	9	14	9	28	7																	
input	input_2	the		1	port		place			6	5																					
input	input_2	the		1	strobe		is			5	2	6	10	11	7	20	3															
hardware	device			0			interrupt	request	generate	28	10																					
element	element	a		1	key		is			24	3																					
device	device	the		1	external		place	pulse	place	6	2																					
data	data			1	port		is	indicate		16	4	17	2	20	9																	
data	data			0			output	use		15	5																					
data	data		any	1	new		strobe	inhibit		8	7																					
data	data			0			latch			7	3	9	4																			
data	data			0			place			6	3																					
data	data			0			strobe			4	3																					
cycle	cycle		two	0			remains			13	4																					
clock	clock	the		2	internal	PH2	synchronize			25	3																					
clock	clock	the		1	E		cycle	remains		13	5																					
bit	bit	the		1	STAF		synchronize			25	2	26	3	28	8																	
bit	bit			1	DDRC		clear			23	3																					
STAI	id	the		0			determine			27	2	28	6																			
STAF	id	the		0			is			24	2	25	4	26	6	28	11															
PORTCL	id			0			register	latch		7	7	9	5	10	7	12	7	17	9													
PH2	id			0			clock	synchronize		25	6																					
OIN	id			0			is	is	select	1	10	14	10																			
I/O	id			1	handshake		subsystem	element	is	24	6																					
HNDS	id			0			is	is	select	1	8	14	8																			
HC11	id	the		0			assert	is	assert	3	2																					
DDRC	id			0			bit	clear		23	5																					

LEGEND

D	the determiner for the Head
Q	the quantifier for the Head
N	the number of restrictive relations of the Head

Appendix C lists sentences and references of type object in the sentences. The graph number and the concept number of the head of the references, the head of the reference itself, its classification by the Coreference Detector (D: definition, R: Coreference, X: incorrect classification of a coreference as a definition) are given. If the classification is an R (indicating that the reference is coreferent with some definition), the graph number and the concept number of the head of the definition with which the reference is coreferent is also given.

Appendix C. References and Coreferences in Test Set II

Graph No.	Concept No.	Sentences and References	Head of Reference	D/R/X	Coreferences	
					Graph No.	Concept No.
		Full input handshake mode is selected when HNDS is one and OIN is zero.				
1	8	HNDS	HNDS	D		
1	9	one	one	D		
1	10	OIN	OIN	D		
1	11	zero	zero	D		
		In this mode, strobe B output is the ready signal to the external system.				
2	2	strobe B output	output	D		
2	3	the ready signal	signal	D		
2	5	strobe B	strobe	D		
2	7	the external system	system	D		
		The HC11 asserts strobe B when it is ready.				
3	2	the HC11	HC11	D		
3	3	strobe B	strobe	R	2	5
		The external system should not strobe data into port C until port B is asserted.				
4	2	the external system	system	R	2	7
4	3	data	data	D		
4	4	port C	port	D		
4	7	port B	port	D		
		The strobe A input is edge-sensitive.				
5	2	strobe A input	input	D		
5	4	strobe A	strobe	D		
		When a ready condition is recognized, the external device places data on the port D inputs, and then pulses the strobe A input.				
6	2	the external device	device	X		
6	3	data	data	D		
6	5	the port D inputs	input	D		
6	8	ready	ready	D		
6	9	port D	port	D		
6	10	strobe A input	input	R	5	2
6	11	strobe A	strobe	R	5	4
		The active edge on strobe A latches data into the PORTCL register.				
7	3	data	data	D		
7	4	the PORTCL register	register	D		
7	5	strobe A	strobe	R	5	4
7	7	PORTCL	PORTCL	D		
		Deassertion of strobe B inhibits the external system from strobing any new data into Port C.				
8	3	the external system	system	R	2	7
8	5	strobe B	strobe	R	2	5
8	7	any new data	data	D		
8	8	port C	port	R	4	4
		Reading the latched data from PORTCL causes strobe F to be asserted.				
9	4	the latched data	data	R	7	3
9	5	PORTCL	PORTCL	R	7	7
9	6	strobe F	strobe	D		
		In the interlocked mode, strobe B is asserted when the PORTCL register is read.				

Graph No.	Concept No.	Sentences and References	Head of Reference	D/R/X	Coreferences	
					Graph No.	Concept No.
10	2	strobe B	strobe	R	2	5
10	6	the PORTCL register	register	R	7	4
10	7	PORTCL	PORTCL	R	7	7
		In the interlocked mode, strobe B is deasserted when an active edge is detected at the strobe A input.				
11	2	strobe B	strobe	R	2	5
11	7	the strobe A input	input	R	5	2
11	9	strobe A	strobe	R	5	4
		In the pulsed mode, strobe B is asserted when the PORTCL register is read.				
12	2	strobe B	strobe	R	2	5
12	6	the PORTCL register	register	R	7	4
12	7	register	PORTCL	R	7	7
		Strobe B only remains asserted for two cycles of the E clock.				
13	2	strobe B	strobe	R	2	5
13	4	two cycles	cycle	D		
13	5	the E clock	clock	D		
		Full-output handshake mode is selected when HNDS and OIN are ones				
14	8	HNDS	HNDS	R	1	8
14	9	ones	one	R	1	9
14	10	OIN	OIN	R	1	10
		In this mode, Port C is used to output data to some external system.				
15	2	port C	port	R	4	4
15	5	data	data	D		
15	6	some external system	system	D		
		The strobe B output signal indicates that port C data is ready for the external system.				
16	3	the strobe B output signal	signal	D		
16	4	port C data	data	D		
16	5	the external system	system	R	2	7
16	6	strobe B	strobe	R	2	5
16	7	strobe B output	output	R	2	2
16	8	port C	port	R	4	4
		In an output-handshake transaction, data is output to Port C pins by writing to the PORTCL register.				
17	2	data	data	R	16	4
17	4	port C pins	pin	D		
17	6	the PORTCL register	register	R	7	4
17	7	port C	port	R	4	4
17	9	PORTCL	PORTCL	R	7	7
		This action causes strobe B to be asserted.				
18	4	strobe B	strobe	R	2	5
		The external system recognizes this strobe B signal as a ready indication.				
19	2	the external system	system	R	2	7
19	3	this strobe B signal	signal	X	16	3
19	6	strobe B	strobe	R	2	5

Graph No.	Concept No.	Sentences and References	Head of Reference	D/R/X	Coreferences	
					Graph No.	Concept No.
		After accepting the data from port C, the external system pulses the strobe A input to acknowledge the receipt of data.				
20	2	the external system	system	R	2	7
20	3	strobe A input	input	R	5	2
20	7	strobe A	strobe	R	5	4
20	9	the data from port C	data	R	16	4
20	10	port C	port	R	4	4
		The active edge on strobe A causes strobe B to be deasserted.				
21	5	strobe A	strobe	R	5	4
21	6	strobe B	strobe	R	2	5
		Port C pins act as three state outputs.				
22	2	port C pins	pin	R	17	4
22	3	three state outputs	output	D		
22	4	port C	port	R	4	4
		Port C pins have their corresponding DDRC bits cleared.				
23	2	port C pins	pin	R	17	4
23	3	DDRC bits	bit	D		
23	4	port C	port	R	4	4
23	5	DDRC	DDRC	D		
		The STAF is a key element of the handshake I/O subsystem.				
24	2	STAF	STAF	D		
24	3	a key element	element	D		
24	5	the handshake I/O subsystem	subsystem	D		
24	6	handshake I/O	I/O	D		
		The STAF bit is synchronized to the internal PH2 clock.				
25	2	the STAF bit	bit	D		
25	3	the internal PH2 clock	clock	D		
25	4	STAF	STAF	R	24	2
25	6	PH2	PH2	D		
		The STAF bit is cleared by a two step, automatic clearing sequence.				
26	3	the STAF bit	bit	R	25	2
26	6	STAF	STAF	R	24	2
		The STAI determines whether STAF will cause interrupts.				
27	2	the STAI	STAI	D		
27	4	STAF	STAF	R	24	2
		When STAI is one, a hardware interrupt request is generated whenever the STAF bit is set.				
28	6	STAI	STAI	R	27	2
28	7	one	one	R	1	9
28	8	the STAF bit	bit	R	25	2
28	9	set	set	D		
28	10	hardware	hardware	D		
28	11	STAF	STAF	R	24	2

Appendix D lists definitions followed by all its coreferences. The graph number and the concept number of the head of the definition as well as those of its coreferences (if any) are provided.

Appendix D. Definitions and their Coreferences in Test Set II

Definition	Coreferences	Graph Number	Concept Number
zero		1	11
some external system		15	6
the external system		2	7
	the external system	4	2
	the external system	8	3
	the external system	16	5
	the external system	19	2
	the external system	20	2
the handshake I/O subsystem		24	5
strobe F		9	6
strobe A		5	4
	strobe A	6	11
	strobe A	7	5
	strobe A	11	9
	strobe A	20	7
	strobe A	21	5
strobe B		2	5
	strobe B	3	3
	strobe B	8	5
	strobe B	10	2
	strobe B	11	2
	strobe B	12	2
	strobe B	13	2
	strobe B	16	6
	strobe B	18	4
	strobe B	19	6
	strobe B	21	6
the strobe B output signal		16	3
	this strobe B signal	19	3
the ready signal		2	3
set		28	9
the PORTCL register		7	4
	the PORTCL register	10	6
	the PORTCL register	12	6
	the PORTCL register	17	6
ready		6	8
port D		6	9
port B		4	7
port C		4	4
	port C	8	8
	port C	15	2
	port C	16	8
	port C	17	7
	port C	20	10
	port C	22	4
	port C	23	4

Definition	Coreferences	Graph Number	Concept Number
port C pins		17	4
	port C pins	22	2
	port C pins	23	2
three state outputs		22	3
strobe B output		2	2
	strobe B output	16	7
one		28	7
	one	14	9
	one	1	9
the port D inputs		6	5
the strobe A input		5	2
	the strobe A input	6	10
	the strobe A input	11	7
	the strobe A input	20	3
hardware		28	10
element		24	3
device		6	2
port C data		16	4
	data	17	2
	data from port C	20	9
data		15	5
any new data		8	7
data		7	3
	latched data	9	4
data		6	3
data		4	3
cycle		13	4
the internal PH2 clock		25	3
the E clock		13	5
the STAF bit		25	2
	the STAF bit	26	3
	the STAF bit	28	8
DDRC bits		23	3
STAI		27	2
	STAI	28	6
STAF		24	2
	STAF	25	4
	STAF	26	6
	STAF	28	11
PORTCL		7	7
	PORTCL	9	5
	PORTCL	10	7
	PORTCL	12	7
	PORTCL	17	9
PH2		25	6
OIN		1	10
	OIN	14	10
I/O		24	6
HNDS		1	8
	HNDS	14	8
HC11		3	2
DDRC		23	5

Appendix E contains a complete listing of the sentences in test set II, their conceptual graphs and the integrated conceptual graph produced by the Coreference Detector. The "INPUT SET" is the set of graphs input to the Coreference Detector. Each sentence in the "INPUT SET" is enclosed within single quotes and is followed by its conceptual graph followed by a period (conceptual graph delimiter). The integrated graph generated by the Coreference Detector is in the following format: all concept identifiers are of the type "A_B" where "A" is the graph number to which the concept belongs in the input set, and "B" is the concept identifier of the concept in the input set. Further, all coreferent references have an extra relation "(same as) -> [X_Y]" appended to their head concepts, where "X_Y" is the concept identifier of the head of the reference they are coreferent with.

Appendix E. Conceptual Graphs in Test Set II and the Integrated Conceptual Graph

THE INPUT SET

'#1 Full input handshake mode is selected when HNDS is one and OIN is zero.'
[1 : action : select]
->(obj : obj) -> [2]
->(cond : when) -> [3]
[2 : mode : mode]
->(attr : adj) -> [4]
->(attr : adj) -> [5]
[3 : and : is]
->(and : null) -> [6]
->(and : null) -> [7]
[4 : characteristic : full_input]
[5 : characteristic : handshake]
[6 : and : is]
->(var : null) -> [8]
->(val : null) -> [9]
[7 : and : is]
->(var : null) -> [10]
->(val : null) -> [11]
[8 : id : HNDS]
[9 : constant : one]
[10 : id : OIN]
[11 : constant : zero]
.
'#2 In this mode, strobe B output is the ready signal to the external system.'
[1 : is : is]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(cond : in) -> [4]
[2 : output_1 : output]
->(attr : adj) -> [5]
[3 : signal : signal]
->(attr : adj) -> [6]
->(dest : to) -> [7]
->(det : the)
[4 : mode : mode]
->(det : this)
[5 : strobe_1 : strobe]
->(name : B)
[6 : characteristic : ready]
[7 : system : system]
->(attr : adj) -> [8]
->(det : the)
[8 : characteristic : external]
.
'#3The HC11 asserts strobe B when it is ready.'
[1 : action : assert]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(cond : when) -> [4]
[2 : id : HC11]
->(det : the)
[3 : strobe_1 : strobe]
->(name : B)
[4 : is : is]
->(attr : adv) -> [5]
->(obj : obj) -> [2]
[5 : state : ready]
.
'#4 The external system should not strobe data into port C until port B is asserted.'
[1 : strobe_2 : strobe]
->(neg : not)
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(dest : into) -> [4]
->(cond : until) -> [5]
[2 : system : system]
->(det : the)
->(attr : adj) -> [6]
[3 : data : data]
[4 : port : port]
->(name : C)
[5 : action : assert]
->(obj : obj) -> [7]
[6 : characteristic : external]
[7 : port : port]
->(name : B)
.
'#5The strobe A input is edge-sensitive.'
[1 : is : is]
->(agnt : agnt) -> [2]
->(attr : adv) -> [3]
[2 : input_2 : input]
->(det : the)
->(attr : adj) -> [4]
[3 : characteristic : edge_sensitive]
[4 : strobe_1 : strobe]
->(name : A)
.
'#6 When a ready condition is recognized, the external device places data on the port D inputs, and then pulses the strobe A input.'
[1 : action : place]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(cond : when) -> [4]
->(loc : on) -> [5]
->(then : then) -> [6]
[2 : device : device]
->(det : the)
->(attr : adj) -> [7]
[3 : data : data]
[4 : state : condition]
->(attr : adj) -> [8]
->(det : a)
[5 : input_2 : input]
->(attr : adj) -> [9]
->(det : the)
[6 : action : pulse]
->(agnt : agnt) -> [2]
->(obj : obj) -> [10]
[7 : characteristic : external]
[8 : value : ready]
[9 : port : port]

->(name : D)
[10 : input_2 : input]
->(attr : adj) -> [11]
->(det : the)
[11 : strobe_1 : strobe]
->(name : A)
.
'#7 The active edge on strobe A latches data into the PORTCL register.'
[1 : action : latch]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(dest : into) -> [4]
[2 : event : edge]
->(det : the)
->(loc : on) -> [5]
->(attr : adj) -> [6]
[3 : data : data]
[4 : register : register]
->(attr : adj) -> [7]
->(det : the)
[5 : strobe_1 : strobe]
->(name : A)
[6 : characteristic : active]
[7 : id : PORTCL]
.
'#8 Deassertion of strobe B inhibits the external system from strobing any new data into Port C.
[1 : action : inhibit]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
->(from : from) -> [4]
[2 : action : deassertion]
->(obj : obj) -> [5]
[3 : system : system]
->(attr : adj) -> [6]
->(det : the)
[4 : action : strobe]
->(obj : obj) -> [7]
->(dest : into) -> [8]
->(det : the)
[5 : strobe_1 : strobe]
->(name : B)
[6 : characteristic : external]
[7 : data : data]
->(attr : adj) -> [9]
->(quant : any)
[8 : port : port]
->(name : C)
[9 : characteristic : new]
.
'#9 Reading the latched data from PORTCL causes strobe F to be asserted.
[1 : action : cause]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
[2 : read : read]
->(obj : obj) -> [4]
->(loc : from) -> [5]
[3 : action : assert]
->(obj : obj) -> [6]
[4 : data : data]
->(det : the)
[5 : id : PORTCL]
[6 : strobe_1 : strobe]
->(name : F)
[7 : action : latch]
->(obj : data)
.
'#10 In the interlocked mode, strobe B is asserted when the PORTCL register is read.'
[1 : action : assert]
->(obj : obj) -> [2]
->(cond : in) -> [3]
->(cond : when) -> [4]
[2 : strobe_1 : strobe]
->(name : B)
[3 : mode : mode]
->(det : the)
->(attr : adj) -> [5]
[4 : read : read]
->(obj : obj) -> [6]
[5 : characteristic : interlocked]
[6 : register : register]
->(attr : adj) -> [7]
->(det : the)
[7 : id : PORTCL]
.
'#11 In the interlocked mode, strobe B is deasserted when an active edge is detected at the strobe A input.'
[1 : action : deassert]
->(obj : obj) -> [2]
->(cond : in) -> [3]
->(cond : when) -> [4]
[2 : strobe_1 : strobe]
->(name : B)
[3 : mode : mode]
->(det : the)
->(attr : adj) -> [5]
[4 : action : detect]
->(obj : obj) -> [6]
->(loc : at) -> [7]
[5 : characteristic : interlocked]
[6 : event : edge]
->(attr : adj) -> [8]
->(det : an)
[7 : input_2 : input]
->(det : the)
->(attr : adj) -> [9]
[8 : characteristic : active]
[9 : strobe_1 : strobe]
->(name : A)
.
'#12 In the pulsed mode, strobe B is asserted when the PORTCL register is read.'
[1 : action : assert]
->(obj : obj) -> [2]
->(cond : in) -> [3]
->(cond : when) -> [4]
[2 : strobe_1 : strobe]
->(name : B)
[3 : mode : mode]
->(det : the)
->(attr : adj) -> [5]
[4 : read : read]
->(obj : obj) -> [6]
[5 : characteristic : pulsed]
[6 : register : register]
->(attr : adj) -> [7]
->(det : the)
[7 : id : PORTCL]
.

```
'#13 Strobe B only remains asserted for two cycles of the E
clock.'
[ 1 : action : remains ]
    ->( obj : obj ) -> [ 2 ]
    ->( attr : adv ) -> [ 3 ]
    ->( mod : only )
    ->( for : for ) -> [ 4 ]
[ 2 : strobe_1 : strobe ]
    ->( name : B )
[ 3 : action : assert ]
[ 4 : cycle : cycle ]
    ->( quant : two )
    ->( of : of ) -> [ 5 ]
[ 5 : clock : clock ]
    ->( det : the )
    ->( name : E )
.
'#14 Full-output handshake mode is selected when HNDS
and OIN are ones'
[ 1 : action : select ]
    ->( obj : obj ) -> [ 2 ]
    ->( cond : when ) -> [ 3 ]
[ 2 : mode : mode ]
    ->( attr : adj ) -> [ 4 ]
    ->( attr : adj ) -> [ 5 ]
[ 3 : and : is ]
    ->( and : null ) -> [ 6 ]
    ->( and : null ) -> [ 7 ]
[ 4 : characteristic : full_output ]
[ 5 : characteristic : handshake ]
[ 6 : be : null ]
    ->( var : null ) -> [ 8 ]
    ->( val : null ) -> [ 9 ]
[ 7 : be : null ]
    ->( var : null ) -> [ 10 ]
    ->( val : null ) -> [ 9 ]
[ 8 : id : HNDS ]
[ 9 : constant : one ]
[ 10 : id : OIN ]
.
'#15 In this mode, Port C is used to output data to some
external system.'
[ 1 : action : use ]
    ->( obj : obj ) -> [ 2 ]
    ->( purp : to ) -> [ 3 ]
    ->( cond : when ) -> [ 4 ]
[ 2 : port : port ]
    ->( name : C )
[ 3 : output_2 : output ]
    ->( obj : obj ) -> [ 5 ]
    ->( dest : to ) -> [ 6 ]
[ 4 : mode : mode ]
    ->( det : this )
[ 5 : data : data ]
[ 6 : system : system ]
    ->( quant : some )
    ->( attr : adj ) -> [ 7 ]
[ 7 : characteristic : external ]
.
'#16 The strobe B output signal indicates that port C data
is ready for the external system.'
[ 1 : action : indicate ]
    ->( obj : obj ) -> [ 2 ]
    ->( agnt : agnt ) -> [ 3 ]
[ 2 : is : is ]
    ->( obj : obj ) -> [ 4 ]
    ->( purp : for ) -> [ 5 ]
[ 3 : signal : signal ]
    ->( attr : adj ) -> [ 6 ]
    ->( attr : adj ) -> [ 7 ]
    ->( det : the )
[ 4 : data : data ]
    ->( attr : adj ) -> [ 8 ]
[ 5 : system : system ]
    ->( det : the )
    ->( attr : adj ) -> [ 9 ]
[ 6 : strobe_1 : strobe ]
    ->( name : B )
[ 7 : output_1 : output ]
[ 8 : port : port ]
    ->( name : C )
[ 9 : characteristic : external ]
.
'#17 In an output-handshake transaction, data is output to
Port C pins by writing to the PORTCL register.'
[ 1 : output_2 : output ]
    ->( obj : obj ) -> [ 2 ]
    ->( agnt : by ) -> [ 3 ]
    ->( dest : to ) -> [ 4 ]
    ->( cond : in ) -> [ 5 ]
[ 2 : data : data ]
[ 3 : write : write ]
    ->( dest : to ) -> [ 6 ]
[ 4 : carrier : pin ]
    ->( attr : adj ) -> [ 7 ]
[ 5 : event : transaction ]
    ->( det : an )
    ->( attr : adj ) -> [ 8 ]
[ 6 : register : register ]
    ->( det : the )
    ->( attr : adj ) -> [ 9 ]
[ 7 : port : port ]
    ->( name : C )
[ 8 : characteristic : output-handshake ]
[ 9 : id : PORTCL ]
.
'#18 This action causes strobe B to be asserted.'
[ 1 : action : cause ]
    ->( agnt : agnt ) -> [ 2 ]
    ->( obj : obj ) -> [ 3 ]
[ 2 : action : action ]
    ->( det : this )
[ 3 : assert : assert ]
    ->( obj : obj ) -> [ 4 ]
[ 4 : strobe_1 : strobe ]
    ->( name : B )
.
'#19 The external system recognizes this strobe B signal as
a ready indication.'
[ 1 : action : recognize ]
    ->( agnt : agnt ) -> [ 2 ]
    ->( obj : obj ) -> [ 3 ]
    ->( purp : as ) -> [ 4 ]
[ 2 : system : system ]
    ->( det : the )
    ->( attr : adj ) -> [ 5 ]
[ 3 : signal : signal ]
    ->( det : this )
    ->( attr : adj ) -> [ 6 ]
[ 4 : action : indication ]
    ->( det : a )
    ->( attr : adv ) -> [ 7 ]
[ 5 : characteristic : external ]
[ 6 : strobe_1 : strobe ]
```

->(name : B)
[7 : characteristic : ready]
.
'#20 After accepting the data from port C, the external system pulses the strobe A input to acknowledge the receipt of data.'
[1 : action : pulse]
->(inst : inst) -> [2]
->(obj : obj) -> [3]
->(purp : to) -> [4]
->(cond : after) -> [5]
[2 : system : system]
->(det : the)
->(attr : adj) -> [6]
[3 : input_2 : input]
->(attr : adj) -> [7]
->(det : the)
[4 : action : acknowledge]
->(obj : obj) -> [8]
[5 : action : accept]
->(obj : obj) -> [9]
->(from : from) -> [10]
[6 : characteristic : external]
[7 : strobe_1 : strobe]
->(name : A)
[8 : action : receipt]
->(obj : of) -> [9]
[9 : data : data]
->(det : the)
[10 : port : port]
->(name : C)
.
'#21 The active edge on strobe A causes strobe B to be deasserted.'
[1 : action : cause]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
[2 : event : edge]
->(det : the)
->(attr : adj) -> [4]
->(loc : on) -> [5]
[3 : action : deassert]
->(obj : obj) -> [6]
[4 : characteristic : active]
[5 : strobe_1 : strobe]
->(name : A)
[6 : strobe_1 : strobe]
->(name : B)
.
'#22 Port C pins act as three state outputs.'
[1 : action : act]
->(agnt : agnt) -> [2]
->(obj : as) -> [3]
[2 : carrier : pin]
->(attr : adj) -> [4]
[3 : output_1 : output]
->(attr : adj) -> [5]
[4 : port : port]
->(name : C)
[5 : characteristic : three_state]
.
'#23 Port C pins have their corresponding DDRC bits cleared.'
[1 : clear : clear]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
[2 : carrier : pin]
->(attr : adj) -> [4]
[3 : bit : bit]
->(attr : adj) -> [5]
->(mod : corresponding)
[4 : port : port]
->(name : C)
[5 : id : DDRC]
.
'#24 The STAF is a key element of the handshake I/O subsystem.'
[1 : is : is]
->(agnt : agnt) -> [2]
->(obj : obj) -> [3]
[2 : id : STAF]
->(det : the)
[3 : element : element]
->(det : a)
->(attr : adj) -> [4]
->(of : of) -> [5]
[4 : characteristic : key]
[5 : system : subsystem]
->(det : the)
->(attr : adj) -> [6]
[6 : id : I/O]
->(attr : adj) -> [7]
[7 : action : handshake]
.
'#25 The STAF bit is synchronized to the internal PH2 clock.'
[1 : action : synchronize]
->(obj : obj) -> [2]
->(to : to) -> [3]
[2 : bit : bit]
->(det : the)
->(attr : adj) -> [4]
[3 : clock : clock]
->(det : the)
->(attr : adj) -> [5]
->(attr : adj) -> [6]
[4 : id : STAF]
[5 : characteristic : internal]
[6 : id : PH2]
.
'#26 The STAF bit is cleared by a two step, automatic clearing sequence.'
[1 : action : clear]
->(agnt : by) -> [2]
->(obj : obj) -> [3]
[2 : event : sequence]
->(det : a)
->(attr : adj) -> [4]
->(attr : adj) -> [5]
[3 : bit : bit]
->(det : the)
->(attr : adj) -> [6]
[4 : characteristic : automatic]
[5 : characteristic : clearing]
[6 : id : STAF]
.
'#27 The STAI determines whether STAF will cause interrupts.'
[1 : action : determine]
->(agnt : agnt) -> [2]
->(obj : whether) -> [3]
[2 : id : STAI]
->(det : the)
[3 : action : cause]

->(obj : obj) -> [4]
->(mod : will)
[4 : interrupt : interrupt]
.

THE INTEGRATED CONCEPTUAL GRAPH

```
[ 1_1 : action : select ]
    ->( obj : obj ) -> [ 1_2 ]
    ->( cond : when ) -> [ 1_3 ]
[ 1_2 : mode : mode ]
    ->( attr : adj ) -> [ 1_4 ]
    ->( attr : adj ) -> [ 1_5 ]
[ 1_3 : and : is ]
    ->( and : null ) -> [ 1_6 ]
    ->( and : null ) -> [ 1_7 ]
[ 1_4 : characteristic : full_input ]
[ 1_5 : characteristic : handshake ]
[ 1_6 : and : is ]
    ->( var : null ) -> [ 1_8 ]
    ->( val : null ) -> [ 1_9 ]
[ 1_7 : and  : is ]
    ->( var : null ) -> [ 1_10 ]
    ->( val : null ) -> [ 1_11 ]
[ 1_8 : id : HNDS ]
[ 1_9 : constant : one ]
[ 1_10 : id : OIN ]
[ 1_11 : constant : zero ]

[ 2_1 : is : is ]
    ->( agnt : agnt ) -> [ 2_2 ]
    ->( obj : obj ) -> [ 2_3 ]
    ->( cond : in ) -> [ 2_4 ]
[ 2_2 : output_1 : output ]
    ->( attr : adj ) -> [ 2_5 ]
[ 2_3 : signal : signal ]
    ->( attr : adj ) -> [ 2_6 ]
    ->( dest : to ) -> [ 2_7 ]
    ->( det : the )
[ 2_4 : mode : mode ]
    ->( det : this )
[ 2_5 : strobe_1 : strobe ]
    ->( name : B )
[ 2_6 : characteristic : ready ]
[ 2_7 : system : system ]
    ->( attr : adj ) -> [ 2_8 ]
    ->( det : the )
[ 2_8 : characteristic : external ]

[ 3_1 : action : assert ]
    ->( agnt : agnt ) -> [ 3_2 ]
    ->( obj : obj ) -> [ 3_3 ]
    ->( cond : when ) -> [ 3_4 ]
[ 3_2 : id : HC11 ]
    ->( det : the )
[ 3_3 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 3_4 : is : is ]
    ->( attr : adv ) -> [ 3_5 ]
    ->( obj : obj ) -> [ 3_2 ]
[ 3_5 : state : ready ]

[ 4_1 : strobe_2 : strobe ]
    ->( neg : not )
    ->( agnt : agnt ) -> [ 4_2 ]
    ->( obj : obj ) -> [ 4_3 ]
    ->( dest : into ) -> [ 4_4 ]
    ->( cond : until ) -> [ 4_5 ]
[ 4_2 : system : system ]
    ->( det : the )
    ->( attr : adj ) -> [ 4_6 ]
    ->( same as ) -> [ 2_7 ]
[ 4_3 : data : data ]
[ 4_4 : port : port ]
    ->( name : C )
[ 4_5 : action : assert ]
    ->( obj : obj ) -> [ 4_7 ]
[ 4_6 : characteristic : external ]
[ 4_7 : port : port ]
    ->( name : B )

[ 5_1 : is : is ]
    ->( agnt : agnt ) -> [ 5_2 ]
    ->( attr : adv ) -> [ 5_3 ]
[ 5_2 : input_2 : input ]
    ->( det : the )
    ->( attr : adj ) -> [ 5_4 ]
[ 5_3 : characteristic : edge_sensitive ]
[ 5_4 : strobe_1 : strobe ]
    ->( name : A )

[ 6_1 : action : place ]
    ->( agnt : agnt ) -> [ 6_2 ]
    ->( obj : obj ) -> [ 6_3 ]
    ->( cond : when ) -> [ 6_4 ]
    ->( loc : on ) -> [ 6_5 ]
    ->( then : then ) -> [ 6_6 ]
[ 6_2 : device : device ]
    ->( det : the )
    ->( attr : adj ) -> [ 6_7 ]
[ 6_3 : data : data ]
[ 6_4 : state : condition ]
    ->( attr : adj ) -> [ 6_8 ]
    ->( det : a )
[ 6_5 : input_2 : input ]
    ->( attr : adj ) -> [ 6_9 ]
    ->( det : the )
[ 6_6 : action : pulse ]
    ->( agnt : agnt ) -> [ 6_2 ]
    ->( obj : obj ) -> [ 6_10 ]
[ 6_7 : characteristic : external ]
[ 6_8 : value : ready ]
[ 6_9 : port : port ]
    ->( name : D )
[ 6_10 : input_2 : input ]
    ->( attr : adj ) -> [ 6_11 ]
    ->( det : the )
    ->( same as ) -> [ 5_2 ]
[ 6_11 : strobe_1 : strobe ]
    ->( name : A )
    ->( same as ) -> [ 5_4 ]

[ 7_1 : action : latch ]
    ->( agnt : agnt ) -> [ 7_2 ]
    ->( obj : obj ) -> [ 7_3 ]
    ->( dest : into ) -> [ 7_4 ]
[ 7_2 : event : edge ]
    ->( det : the )
    ->( loc : on ) -> [ 7_5 ]
    ->( attr : adj ) -> [ 7_6 ]
[ 7_3 : data : data ]
[ 7_4 : register : register ]
```

```
    ->( attr : adj ) -> [ 7_7 ]
    ->( det : the )
[ 7_5 : strobe_1 : strobe ]
    ->( name : A )
    ->( same as ) -> [ 5_4 ]
[ 7_6 : characteristic : active ]
[ 7_7 : id : PORTCL ]

[ 8_1 : action : inhibit ]
    ->( agnt : agnt ) -> [ 8_2 ]
    ->( obj : obj ) -> [ 8_3 ]
    ->( from : from ) -> [ 8_4 ]
[ 8_2 : action : deassertion ]
    ->( obj : obj ) -> [ 8_5 ]
[ 8_3 : system : system ]
    ->( attr : adj ) -> [ 8_6 ]
    ->( det : the )
    ->( same as ) -> [ 2_7 ]
[ 8_4 : action : strobe ]
    ->( obj : obj ) -> [ 8_7 ]
    ->( dest : into ) -> [ 8_8 ]
    ->( det : the )
[ 8_5 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 8_6 : characteristic : external ]
[ 8_7 : data : data ]
    ->( attr : adj ) -> [ 8_9 ]
    ->( quant : any )
[ 8_8 : port : port ]
    ->( name : C )
    ->( same as ) -> [ 4_4 ]
[ 8_9 : characteristic : new ]

[ 9_1 : action : cause ]
    ->( agnt : agnt ) -> [ 9_2 ]
    ->( obj : obj ) -> [ 9_3 ]
[ 9_2 : read : read ]
    ->( obj : obj ) -> [ 9_4 ]
    ->( loc : from ) -> [ 9_5 ]
[ 9_3 : action : assert ]
    ->( obj : obj ) -> [ 9_6 ]
[ 9_4 : data : data ]
    ->( det : the )
    ->( same as ) -> [ 7_3 ]
[ 9_5 : id : PORTCL ]
    ->( same as ) -> [ 7_7 ]
[ 9_6 : strobe_1 : strobe ]
    ->( name : F )
[ 9_7 : action : latch ]
    ->( obj : data )

[ 10_1 : action : assert ]
    ->( obj : obj ) -> [ 10_2 ]
    ->( cond : in ) -> [ 10_3 ]
    ->( cond : when ) -> [ 10_4 ]
[ 10_2 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 10_3 : mode : mode ]
    ->( det : the )
    ->( attr : adj ) -> [ 10_5 ]
[ 10_4 : read : read ]
    ->( obj : obj ) -> [ 10_6 ]
[ 10_5 : characteristic : interlocked ]
[ 10_6 : register : register ]
    ->( attr : adj ) -> [ 10_7 ]
    ->( det : the )
    ->( same as ) -> [ 7_4 ]
[ 10_7 : id : PORTCL ]
    ->( same as ) -> [ 7_7 ]

[ 11_1 : action : deassert ]
    ->( obj : obj ) -> [ 11_2 ]
    ->( cond : in ) -> [ 11_3 ]
    ->( cond : when ) -> [ 11_4 ]
[ 11_2 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 11_3 : mode : mode ]
    ->( det : the )
    ->( attr : adj ) -> [ 11_5 ]
[ 11_4 : action : detect ]
    ->( obj : obj ) -> [ 11_6 ]
    ->( loc : at ) -> [ 11_7 ]
[ 11_5 : characteristic : interlocked ]
[ 11_6 : event : edge ]
    ->( attr : adj ) -> [ 11_8 ]
    ->( det : an )
[ 11_7 : input_2 : input ]
    ->( det : the )
    ->( attr : adj ) -> [ 11_9 ]
    ->( same as ) -> [ 5_2 ]
[ 11_8 : characteristic : active ]
[ 11_9 : strobe_1 : strobe ]
    ->( name : A )
    ->( same as ) -> [ 5_4 ]

[ 12_1 : action : assert ]
    ->( obj : obj ) -> [ 12_2 ]
    ->( cond : in ) -> [ 12_3 ]
    ->( cond : when ) -> [ 12_4 ]
[ 12_2 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 12_3 : mode : mode ]
    ->( det : the )
    ->( attr : adj ) -> [ 12_5 ]
[ 12_4 : read : read ]
    ->( obj : obj ) -> [ 12_6 ]
[ 12_5 : characteristic : pulsed ]
[ 12_6 : register : register ]
    ->( attr : adj ) -> [ 12_7 ]
    ->( det : the )
    ->( same as ) -> [ 7_4 ]
[ 12_7 : id : PORTCL ]
    ->( same as ) -> [ 7_7 ]

[ 13_1 : action : remains ]
    ->( obj : obj ) -> [ 13_2 ]
    ->( attr : adv ) -> [ 13_3 ]
    ->( mod : only )
    ->( for : for ) -> [ 13_4 ]
[ 13_2 : strobe_1 : strobe ]
    ->( name : B )
    ->( same as ) -> [ 2_5 ]
[ 13_3 : action : assert ]
[ 13_4 : cycle : cycle ]
    ->( quant : two )
    ->( of : of ) -> [ 13_5 ]
[ 13_5 : clock : clock ]
    ->( det : the )
    ->( name : E )
```

[14_1 : action : select]
->(obj : obj) -> [14_2]
->(cond : when) -> [14_3]
[14_2 : mode : mode]
->(attr : adj) -> [14_4]
->(attr : adj) -> [14_5]
[14_3 : and : is]
->(and : null) -> [14_6]
->(and : null) -> [14_7]
[14_4 : characteristic : full_output]
[14_5 : characteristic : handshake]
[14_6 : be : null]
->(var : null) -> [14_8]
->(val : null) -> [14_9]
[14_7 : be : null]
->(var : null) -> [14_10]
->(val : null) -> [14_9]
[14_8 : id : HNDS]
->(same as) -> [1_8]
[14_9 : constant : one]
->(same as) -> [1_9]
[14_10 : id : OIN]
->(same as) -> [1_10]

[15_1 : action : use]
->(obj : obj) -> [15_2]
->(purp : to) -> [15_3]
->(cond : when) -> [15_4]
[15_2 : port : port]
->(name : C)
->(same as) -> [4_4]
[15_3 : output_2 : output]
->(obj : obj) -> [15_5]
->(dest : to) -> [15_6]
[15_4 : mode : mode]
->(det : this)
[15_5 : data : data]
[15_6 : system : system]
->(quant : some)
->(attr : adj) -> [15_7]
[15_7 : characteristic : external]

[16_1 : action : indicate]
->(obj : obj) -> [16_2]
->(agnt : agnt) -> [16_3]
[16_2 : is : is]
->(obj : obj) -> [16_4]
->(purp : for) -> [16_5]
[16_3 : signal : signal]
->(attr : adj) -> [16_6]
->(attr : adj) -> [16_7]
->(det : the)
[16_4 : data : data]
->(attr : adj) -> [16_8]
[16_5 : system : system]
->(det : the)
->(attr : adj) -> [16_9]
->(same as) -> [2_7]
[16_6 : strobe_1 : strobe]
->(name : B)
->(same as) -> [2_5]
[16_7 : output_1 : output]
->(same as) -> [2_2]
[16_8 : port : port]
->(name : C)
->(same as) -> [4_4]
[16_9 : characteristic : external]

[17_1 : output_2 : output]
->(obj : obj) -> [17_2]
->(agnt : by) -> [17_3]
->(dest : to) -> [17_4]
->(cond : in) -> [17_5]
[17_2 : data : data]
->(same as) -> [16_4]
[17_3 : write : write]
->(dest : to) -> [17_6]
[17_4 : carrier : pin]
->(attr : adj) -> [17_7]
[17_5 : event : transaction]
->(det : an)
->(attr : adj) -> [17_8]
[17_6 : register : register]
->(det : the)
->(attr : adj) -> [17_9]
->(same as) -> [7_4]
[17_7 : port : port]
->(name : C)
->(same as) -> [4_4]
[17_8 : characteristic : output-handshake]
[17_9 : id : PORTCL]
->(same as) -> [7_7]

[18_1 : action : cause]
->(agnt : agnt) -> [18_2]
->(obj : obj) -> [18_3]
[18_2 : action : action]
->(det : this)
[18_3 : assert : assert]
->(obj : obj) -> [18_4]
[18_4 : strobe_1 : strobe]
->(name : B)
->(same as) -> [2_5]

[19_1 : action : recognize]
->(agnt : agnt) -> [19_2]
->(obj : obj) -> [19_3]
->(purp : as) -> [19_4]
[19_2 : system : system]
->(det : the)
->(attr : adj) -> [19_5]
->(same as) -> [2_7]
[19_3 : signal : signal]
->(det : this)
->(attr : adj) -> [19_6]
->(same as) -> [16_3]
[19_4 : action : indication]
->(det : a)
->(attr : adv) -> [19_7]
[19_5 : characteristic : external]
[19_6 : strobe_1 : strobe]
->(name : B)
->(same as) -> [2_5]
[19_7 : characteristic : ready]

[20_1 : action : pulse]
->(inst : inst) -> [20_2]
->(obj : obj) -> [20_3]
->(purp : to) -> [20_4]
->(cond : after) -> [20_5]
[20_2 : system : system]
->(det : the)
->(attr : adj) -> [20_6]
->(same as) -> [2_7]
[20_3 : input_2 : input]

->(attr : adj) -> [20_7]
->(det : the)
->(same as) -> [5_2]
[20_4 : action : acknowledge]
->(obj : obj) -> [20_8]
[20_5 : action : accept]
->(obj : obj) -> [20_9]
->(from : from) -> [20_10]
[20_6 : characteristic : external]
[20_7 : strobe_1 : strobe]
->(name : A)
->(same as) -> [5_4]
[20_8 : action : receipt]
->(obj : of) -> [20_9]
[20_9 : data : data]
->(det : the)
->(same as) -> [16_4]
[20_10 : port : port]
->(name : C)
->(same as) -> [4_4]

[21_1 : action : cause]
->(agnt : agnt) -> [21_2]
->(obj : obj) -> [21_3]
[21_2 : event : edge]
->(det : the)
->(attr : adj) -> [21_4]
->(loc : on) -> [21_5]
[21_3 : action : deassert]
->(obj : obj) -> [21_6]
[21_4 : characteristic : active]
[21_5 : strobe_1 : strobe]
->(name : A)
->(same as) -> [5_4]
[21_6 : strobe_1 : strobe]
->(name : B)
->(same as) -> [2_5]

[22_1 : action : act]
->(agnt : agnt) -> [22_2]
->(obj : as) -> [22_3]
[22_2 : carrier : pin]
->(attr : adj) -> [22_4]
->(same as) -> [17_4]
[22_3 : output_1 : output]
->(attr : adj) -> [22_5]
[22_4 : port : port]
->(name : C)
->(same as) -> [4_4]
[22_5 : characteristic : three_state]

[23_1 : clear : clear]
->(agnt : agnt) -> [23_2]
->(obj : obj) -> [23_3]
[23_2 : carrier : pin]
->(attr : adj) -> [23_4]
->(same as) -> [17_4]
[23_3 : bit : bit]
->(attr : adj) -> [23_5]
->(mod : corresponding)
[23_4 : port : port]
->(name : C)
->(same as) -> [4_4]
[23_5 : id : DDRC]

[24_1 : is : is]
->(agnt : agnt) -> [24_2]
->(obj : obj) -> [24_3]
[24_2 : id : STAF]
->(det : the)
[24_3 : element : element]
->(det : a)
->(attr : adj) -> [24_4]
->(of : of) -> [24_5]
[24_4 : characteristic : key]
[24_5 : system : subsystem]
->(det : the)
->(attr : adj) -> [24_6]
[24_6 : id : I/O]
->(attr : adj) -> [24_7]
[24_7 : action : handshake]

[25_1 : action : synchronize]
->(obj : obj) -> [25_2]
->(to : to) -> [25_3]
[25_2 : bit : bit]
->(det : the)
->(attr : adj) -> [25_4]
[25_3 : clock : clock]
->(det : the)
->(attr : adj) -> [25_5]
->(attr : adj) -> [25_6]
[25_4 : id : STAF]
->(same as) -> [24_2]
[25_5 : characteristic : internal]
[25_6 : id : PH2]

[26_1 : action : clear]
->(agnt : by) -> [26_2]
->(obj : obj) -> [26_3]
[26_2 : event : sequence]
->(det : a)
->(attr : adj) -> [26_4]
->(attr : adj) -> [26_5]
[26_3 : bit : bit]
->(det : the)
->(attr : adj) -> [26_6]
->(same as) -> [25_2]
[26_4 : characteristic : automatic]
[26_5 : characteristic : clearing]
[26_6 : id : STAF]
->(same as) -> [24_2]

[27_1 : action : determine]
->(agnt : agnt) -> [27_2]
->(obj : whether) -> [27_3]
[27_2 : id : STAI]
->(det : the)
[27_3 : action : cause]
->(obj : obj) -> [27_4]
->(mod : will)
[27_4 : interrupt : interrupt]

Appendix F. Sentences in Test Sets I and III

SENTENCES IN TEST SET I

1 'All DMA transfers consist of a fetch cycle and a deposit cycle'
2 'During the fetch cycle, the byte data is accessed according to the source pointer register.'
3 'The data is read into an internal temporary register.'
4 'This register is not accessible by the CPU.'
5 'During the deposit cycle, the data is written to memory at the address in the destination pointer register.'
6 'These two bus cycles cannot be separated by a bus HOLD, a refresh cycle, or any other condition except reset.'
7 'DMA transfer rates depend on processor bus width and the DMA control unit operating mode'
8 'Continuous transfer operation is referred to as unsynchronized mode.'
9 'Continuous transfer operation is selected by synchronization.'
10 'The processor synchronizes external requests to the CPU clock before presenting them to the DMA logic.'
11 'Before any DMA request can be generated, the internal bus must be granted to the DMA unit.'
12 'External HOLD and refresh always have priority over DMA transfers.'
13 'The latency time of an internal DMA cycle will suffer during an external bus HOLD.'
14 'The minimum DMA latency is four clocks.'
15 'The DRQ signal is sampled on every falling clock edge.'
16 'The DRQ signal takes this amount of time to propagate through the bus control logic.'
17 'If the DRQ is sampled active, the DMA cycle can be executed four clocks later.'
18 '80186 family of processors do not generate any explicit DACK signal.'
19 'The processors perform a read or write directly to the DMA requesting device.'
20 'A DACK signal can be generated by decoding an address or by using a PCS line.'

SENTENCES IN TEST SET III

1 'a memory write stores the data . '
2 'the counter can increment or decrement . '
3 'the computer can also operate in a high-speed mode . '
4 'the system will not operate in 16-bit mode . '
5 'the system may not correctly operate at 15 mhz . '
6 'the chip has an on-chip , 8k cache . '
7 'resetting stops the execution of the instruction in memory . '
8 'resetting before the system stops deletes the startup program . '
9 'caching the disk in memory improves performance . '
10 'the interrupts to be handled are serviced in order . '
11 'the data to be stored in memory is incremented . '
12 'the processor is not a 16-bit device . '
13 'the memory chip can not be a 16-bit device . '
14 'the gate may also be a 3-input device . '
15 'the system memory can be easily accessed . '
16 'resetting and initializing clear the memory . '
17 'resetting , initializing and loading clear the memory . '
18 'resetting , initializing and loading clear the memory . '
19 'all the registers are cleared when the system resets . '
20 'all the internal registers are cleared . '
21 'all internal registers are cleared when the system resets . '
22 'all registers are cleared when the system resets . '
23 'the first 2 low input signals cause initialization . '
24 'the first 8 bytes of data are stored in memory . '
25 'the first low input signal causes initialization . '
26 'the first byte of data is stored . '
27 'the 2 low input signals initialize asynchronously if strb is low . '
28 'the 8 bytes are stored asynchronously if strb is low when the data arrives . '
29 '1 priority interrupt is serviced before the next cycle . '
30 'reset is completed before the cycle begins . '
31 'data to and from the memory is cached . '
32 'the data is on the bus . '
33 'the result is a high signal on the output . '
34 'the counter is asynchronously reset . '
35 'the buffer is not asynchronously cleared . '
36 'the machine can perform a read or write asynchronously if the rdwrite signal is high . '
37 'the buffer is not asynchronously cleared . '

38 'the counter can not be asynchronously reset . '
39 'before the next instruction is executed , the program counter is read and is incremented . '
40 'before the data is read , the internal register is cleared . '
41 'the system reset is completed before the cycle begins . '
42 'the data to be incremented by the counter is read . '
43 'the counter to increment the data is loaded when ld is high . '
44 'the data to increment is read from the bus . '
45 'the counter is to increment the data . '
46 'the counter to increment the data before conversion is loaded when ld is high . '
47 'the data to increment by the counter is read from the bus . '
48 'the last step is to increment the data before a reset . '
49 'the last step is to increment before a reset . '
50 'the counter cleared asynchronously resets . '

Appendix G. User Manual

1. Input to the Coreference Detector

The set of sentences on which coreference detection is desired is input to the system in the form of conceptual graphs, one for each input sentence. The conceptual graphs need to be stored in a file, one after the other, separated by periods, and this file will serve as input to the Coreference Detector.

2. Format of the Input

The format of the conceptual graphs in the input set should resemble the format used by the ASPIN system. All sentences must be enclosed in single quotation marks. All blank lines are ignored. All conceptual graphs should be terminated by a period. A sample conceptual graph format is given below.

```
'#1 All DMA transfers consist of a fetch cycle and a deposit cycle'
[ 1 : action : consist ]
    ->( obj : obj ) -> [ 2 ]
    ->( agnt : agnt ) -> [ 3 ]
[ 2 : cycle : and ]
    ->( and : null ) -> [ 4 ]
    ->( and : null ) -> [ 5 ]
[ 3 : action : transfer ]
    ->( attr : adj ) -> [ 6 ]
    ->( quant : all )
[ 4 : cycle : cycle ]
    ->( attr : adj ) -> [ 7 ]
    ->( det : a )
[ 5 : cycle : cycle ]
    ->( attr : adj ) -> [ 8 ]
    ->( det : a )
[ 6 : id : DMA ]
[ 7 : action : fetch ]
[ 8 : action : deposit ]
.
```

3. Invoking the Coreference Detector

To invoke the Coreference Detector, go the the appropriate directory and type

co_det <input file name>

where <input file name> is the name of the file in which the input set of conceptual graphs is stored. For example, if the input set of conceptual graphs is stored in a file called "test.inp", to perform coreference detection on this set of conceptual graphs, type

co_det test.inp

at the Unix command prompt.

4. Output of the Coreference Detector

The input set of conceptual graphs is analyzed, coreferences across the graphs are detected and links are created at coreferent nodes, and the linked conceptual graph is written into an output file called "int_graph". The input sentences and the linked conceptual graph are written to a file called "results.out". Also output is a file called "cd" containing the definitions table with the name (referent marker for the head), type, graph number, concept number, restrictive relations, coherences, determiner and quantifier for every definition in the input set.

5. System Defaults

The Coreference Detector uses a number of default maximum values for the different parameters. Some of these default maximum values are given below:

Maximum length of sentences	300 characters
Maximum number of relations classified as "restrictive relations"	9
Maximum length of the referent markers and type markers of the restrictive relations and coherences for the head of a reference	25 characters
Maximum length of determiners	7 characters
Maximum length of quantifiers	20
Maximum number of restrictive relations/ coherence nodes in a coherence a head can have	10
Maximum number of conceptual graphs in the input set	100

These defaults can be modified by changing the appropriate "#defines" in the 'C' implementation. Some of the "#defines" in the implementation are shown below.

```
#define MAX_TYPES 9        /* number of types of relations currently classified as
"restrictive                                relations"      */
#define REL_LEN 25                 /* expected length of relation types and names
        */
#define DET_LEN 7
#define QUANT_LEN 20     /* expected length of quantifiers */
#define NO_OF_RELS 10    /* max no of relations and the length of coherences
expected for a                                  concept        */
#define END_CG "."                 /* marker that denotes the end of a conceptual
graph  */
#define MAX_NUM_SENTENCES 100 /* Maximum number of conceptual graphs in
the input set   */
#define MAX_SENTENCE_LEN 300          /* Maximum number of conceptual
graphs (sentences) in                                        the input set   */
```

Vita

S. Shankaranarayanan was born in Madras, India on December 14, 1968. After completing his high school education at St. Mary's Higher Secondary School, Madras, in June 1986, he began undergraduate study in Electrical Engineering at Birla Institute of Technology & Science, Pilani, India. After completing his B.E. degree in Electrical Engineering, he decided to continue his education at Virginia Polytechnic Institute and State University and pursue a M.S. degree in Electrical Engineering. At Virginia Tech he concentrated his studies in the digital design and VLSI design areas of Computer Engineering. His career interests include Computer Architecture, VLSI/Digital Design and VHDL Modeling.

S. Shankaranarayanan