# Modelling Discourse Theory

Elaine K. Mahon

M.Sc. 1995

# Modelling Discourse Theory

A Dissertation Presented in Fulfilment of
the Requirement of the M.Sc. Degree

28th of September 1995.

Elaine K. Mahon

School of Computer Applications
Dublin City University

Supervisor: Mr Andrew Way

## Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science in Computer Applications, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Elaine Mahon

**Elaine Mahon, B. Sc.**

**DATE....** 28 / 9/95.

## Acknowledgements

I would like to thank:

My supervisor Andy Way for his advice, encouragement and patience. I'll see you in Kitty O'Sheas for a pint!

Josef van Genabith for guiding me through the boggling world of linguistics. Vielen Dank.

My parents and family for listening, helping and tolerating my highs as well as my lows.

Ian and my fellow postgrads for providing many a social and unsocial outlet.

## Abstract

Anaphora are hidden descriptions found in discourse, which refer to explicitly mentioned entities of the discourse e.g.

*Mary loves tennis, she plays everyday.*

Humans can expand anaphors into fuller descriptions with ease by using intuitive world knowledge, which links the anaphor with a suitable entity of the discourse. In the example above it is obvious to us that *she* may be expanded into, or refer to, the previously mentioned female entity *Mary*. As humans we do not allow *she* to refer to *tennis*, as intuitively we know that the female *she* must refer to another female entity. Modelling anaphora resolution or expansion is a difficult task as so much of what is communicated is implicit in discourse. This thesis investigates the historical development of existing methods of resolving anaphors within discourse, and aims at implementing one such mechanism using a small fragment of English.

# Chapter 1 Introduction

## 1.0 Introduction.

Conversation involves the transfer of knowledge between two or more people. The participants of a discourse - the *speaker* and *hearer*, are assumed to have the same basic knowledge of the world which they draw on to participate in a successful conversation. If they do not have the same world knowledge the conversation fails, as neither *speaker* nor *hearer* has enough background knowledge to reason with, and understand the other person.

*(1). John owns a dog. He feeds him.*

What do the pronouns *He* and *him* refer to in (1)? Humans have no difficulty deciphering what pronouns encode. World knowledge suggests to us that *John*, a human, *feeds* the non-human *dog*. These referring pronouns are called **anaphors** and refer to an entity explicitly mentioned in text e.g. the **anaphor** *him* refers to its **antecedent** ( previously mentioned entity) *dog*. We can also assert that the **anaphor** is **resolved** when *him* refers to *dog* i.e. finds an entity to refer to. Anaphor resolution is a difficult task as so much of what is communicated is implicit in discourse.

(2). *At the supermarket John found the shelf where the milk was.*
*He paid for it at the checkout and left.*

The anaphor *it* has two possible meanings in this discourse: *it* can be *the shelf* or *the milk*. Furthermore *it* doesn't refer to "all the milk" in the shop but just one or two cartons. As humans we would not entertain the thought of *it* referring to *the shelf* or "all the milk", as a supermarket is a place where shelves are not normally sold, and people usually buy cartons of milk not all the milk in the shop. In order to model anaphora resolution i.e. expand the anaphor, world knowledge must be incorporated in to the model in an attempt to model the reasoning behaviour of humans.

Antecedents are not always readily available in discourse as (3) illustrates.

*(3). Mary is engaged to John. They are happy.*

The anaphor *They* does not refer to a single entity but a set of individuals, in this case *Mary* and *John*. The human mind reasons by drawing on intuitive or world knowledge to produce a suitable set of individuals to replace *They*. Modelling even a small natural language system with a limited domain requires a method of anaphor resolution and world knowledge to bring implicit information to the forefront. Our ability to reason in language entails that we are able to recognise how the truth conditions of different sentences are systematically related to each other, crossing sentence boundaries and thereby linking an antecedent and anaphor.

## 1.1 Traditional Modelling Methods

Traditional modelling methods attempt to model the human reasoning process by representing sentences of the discourse in a mathematical format. Historically predicate calculus was used to map words to propositions and predicates, thereby capturing the words and actions of a sentence. Proper nouns and indefinites introduce propositions, with verbs combining these propositions in the guise of predicates. Anaphoric pronouns look for antecedents within a sentence or discourse. When this antecedent is found the meaning of the pronoun in this instance is captured.

*(4).* *John loves Sue, she loves him.*

(5). $\exists x \exists y$ ( John(x) $\wedge$ Sue(y) $\wedge$ loves(x,y) $\wedge$ loves(y,x) ).

Humans can quickly resolve the anaphors of (4), by intuitively linking *she* with the female human *Sue,* and *him* with *John.* Predicate calculus works well within single sentences, capturing the antecedents of anaphors easily as the truth conditions of (5) show. Problems arise however when this method of analysis is used in discourse, as breaking (4) into two sentences forming the mini-discourse of (6) illustrates

*(6).* *John loves Sue. She loves him.*

The discourse is represented by (7) and (8).

(7). $\exists x \exists y$ ( John(x) $\wedge$ Sue(y) $\wedge$ loves(y,x) ).

(8). $\exists w \exists y$ ( woman(w) $\wedge$ man(u) $\wedge$ loves(w,u) ).

The calculus is unable to pass the variables that represent a word in one sentence through the sentence boundaries so that the variable may be used in other sentences. Each sentence is represented in isolation from the other. The anaphoric pronouns of (8) cannot refer to *John* and *Sue* of (7), as variables introduced by predicate calculus producing cannot be shared across the sentence boundaries of (6).

Predicate calculus cannot capture relationships between sentences of discourse, and therefore cannot possibly help the author in resolving anaphor within such a discourse.
This thesis investigates existing methods of resolving anaphors within a discourse, and aims at implementing one such mechanism for a discourse using a small fragment of English. The thesis is constructed as follows:

- **Chapter 2** - With the failure of Predicate calculus this chapter discusses the historical development of approaches to anaphora resolution that led to contemporary thinking, and Discourse Representation Theory (DRT).

- **Chapter 3** - This chapter takes a more detailed look at one approach to anaphora resolution namely Discourse Representation Theory (DRT) as presented by Kamp and Reyle (1993). The author produces an extended DRT theory that is implemented in chapter 4. A wide range of syntactic structures is examined in this chapter, with the construction rules used for such structures in DRT.

- **Chapter 4** - This chapter discusses an implementation of the extended DRT theory. The chapter is divided into two modules, the syntactic and semantic modules. The syntactic module discusses the parsing techniques available and the one chosen by the author. The semantic module analyses negated, conditional, quantified, conjunctive, and disjunctive sentences producing semantic representations.

- **Chapter 5** - This chapter presents conclusions and suggestions for further work with an overview of the contents of each chapter of the thesis.

# Chapter 2 The development of systems to treat anaphora.

## 2.0 Introduction

*Anaphora is the problem of coreference or reference to an entity explicitly mentioned in the text....Interpretation of anaphoric expressions involves uncertainty. The understander arrives at the most probable rather than 'correct' interpretation of anaphor.*

(Van Dijk, 1985).

As Van Dijk states anaphora refer to entities already mentioned in discourse. The meaning of an anaphor is captured by isolating a suitable entity or antecedent in the discourse. In this way, a complete semantic representation of discourse may be presented. The core problem of anaphoric resolution is the linking of anaphor and antecedent; selecting a suitable referent from all the available ones to arrive at the most suitable one. Representing as vast a domain as that of Natural Language requires the construction of a model. A common theme running through all the methods designed to resolve anaphoric resolution is the different treatment given to indefinite NPs. An analysis may regard the NP as a variable, a referring, non-referring or quantifying article. How the indefinite is regarded forms the hub of the many methods developed. Traditional approaches to natural language semantics focus on the sentence only, the narrowness of these approaches being highlighted when an attempt is made to cross sentence boundaries. It becomes obvious that the narrow sense of meaning, quantification and bound variables must be changed to give a new more dynamic approach to anaphor resolution.

Sentences may be ambiguous because:

(a). more than one antecedent may be available.

(b). the scope of quantification may be undetermined.

Disambiguated representation is designed to resolve the problems of (a) and (b) with anaphoric relations marked with numerical subscripts, and with the scope and restrictor of quantifiers clearly marked. The next two sections investigate the historic development of theories to resolve anaphoric resolution, starting with the now standard static approaches.

## 2.1 The Treatment of Indefinite NPs - The Static Approaches

As mentioned in the introduction to this chapter, the treatment of indefinite NPs forms the core element of various anaphoric resolution techniques. The spotlight is on indefinite NPs for the following reasons:

(1). They can range across conjunctions or sequences of sentences, and can set up a referent that can be picked up by subsequent discourse, unlike other quantified NPs.

(2). They behave differently to other NPs when found in conditionals.

(3). An indefinite is capable of anteceding a pronoun across a relative clause, whereas other NPs cannot.

The only conclusion that can be drawn from points (1) to (3) is that the anaphoric and quantificational ability of indefinite NPs differs significantly from other NPs. The treatment of these NPs can be divided into four distinct schools:

(1). those that regard indefinite NPs as existential quantifiers e.g. Russell (1905).
(2). those that regard indefinite NPs as referring articles e.g. Chastain (1975).
(3). those that treat indefinites as quantified NPs e.g. Egli (1979).
(4) those that consider indefinites as variables e.g. Heim (1982).

The intricacies of each technique will be discussed in the next 3 subsections. Truth conditions used to capture meaning, as our ability to reason in natural language entails that we are able to recognise how the truth conditions of different sentences are systematically related to each other. If the relationship between truth conditions of different sentences can be captured a theory of semantics could capture the meaning of a complete discourse.

### 2.1.1 Russell's School of thought - Indefinite NPs do not refer.

Betrand Russell (1905) and followers claim that indefinite NPs have the power of existential quantifiers and do not refer. This causes a problem in that anaphoric pronouns seem to pick up the reference of their antecedents.

(1). *A dog came in.*

The indefinite NP *a dog* in (1) has the meaning of an existential quantifier. It does not refer to a particular dog, agreeing with Russell's claim that indefinites do not refer. In Russellian terms sentence (1) simply implies that the set of dogs that came in is not empty; it does not denote any particular dog any more than *Every dog* in (2) does.

(2). *Every dog came in.*

One argument in favour of Russell claims that if *a dog* refers to anything then it either refers to something new, or the same thing on each occasion where it is uttered, or else it never refers to anything at all.

(3). *A dog came in. It lay under the table.*

Common sense implies that the pronoun *it* in (3) refers to the indefinite NP *a dog*; but how could this be if the indefinite itself does not refer to a particular dog?

*Russell seems to have shown that indefinites do not have a reference, they should not be able to serve as antecedents for anaphoric pronouns but they do.* (Heim 1982).

Russell's advocates attempt to explain this contradiction and have formulated various approaches to show how indefinites can have the power of existential quantifiers, and still act as antecedents for anaphors within a discourse.

Geach (Geach 1962) completely denies that anaphora need to pick up reference and uses truth conditions to represent discourse, looking at the whole discourse and not just sentences in isolation. Sentence (3) can be treated by Geach producing (4).

(4). $\exists$ x ( x is a dog & x came in & x lay down under the table ) ( Heim 1982).

Geach proposes to analyse the problematic anaphoric pronouns as bound variables thereby forcing the assignment of a truth condition to the complete discourse. His theory relies on attributing to indefinite NPs a scope that may encompass as much as an entire text. As (4) illustrates, he would analyse (3) as the set of dogs that came in and lay down under the table being not empty. The method works well however in instances where sentences within a discourse do not contradict each other; but fails miserably in cases like sentence (5).

(5). (a). *A man fell over the edge.* (b). *He didn't fall he jumped.*

The analysis assigns a truth condition for the discourse as a whole, so conflicting information as supplied by (5) (a). and (b) leads to an overall false truth condition for the whole discourse. The truth conditions of (5) are represented in (6) which contains the conflicting information implying that x both fell and jumped.

(6). $\exists$ x ( x is a man & x fell over the edge & not(x fell) & x jumped )

Interest in this theory has waned, since it is unable to assign truth conditions to single utterances and predicts false truth conditions for discourses that contain conflicting information.

Another approach is that of Kripke (Kripke 1980), which is based on the teachings of Grice (Grice 1975). He claims that finding a reference for anaphora is not a problem. He asserts that the problem lies in the confusion between the two notions of "reference"; the *Speaker's* and *Semantic* reference, that is there is no dilemma at all. The *Speaker's* referent is that individual which the speaker wishes to talk about at the time of the utterance. The *Semantic* referent concerns whether a referent can be found in the language, and is a matter for the rules of language in which the utterance is being made. In this case it is possible for indefinites not to refer, because an utterance that does not have *Semantic* reference may have *Speaker's* reference.

(7). *A woman taught Jeff in first year.*

Assume sentence (7) is part of a discourse, where the *Speaker's* referent is the specific individual that the speaker has in mind e.g. Mrs Jones. The speaker assumes that both he and the hearer have the same world knowledge, and are thereby able to isolate the intended individual. The *Semantic* referent is isolated by syntactic and semantic rules of language producing a set of probable individuals. The *Speaker's* reference is a mental representation found in the minds of the speaker and hearer, whereas the Semantic reference is a reference isolated by semantic and syntactic processes.

The assumption of different types of referents accounts for contradictions that may occur within discourse as in (5), by simply stating that (5b) is an instance of *Speaker's* reference, it is the speaker's opinion, he has a particular man in mind who jumped off the bridge. Although Kripke can handle the problem of conflicting information, the approach is limited in its domain as sentence (8) illustrates.

(8). *A dog has been rummaging in the garbage can. It has torn open all the bags.* ( Heim 1982 ).

Utterance (8) can be asserted when the speaker has no knowledge of the dog, rather the evidence of the torn bags suggests the presence of a dog, but there is no *Speaker's* referent available as the speaker has no particular dog in mind. The *Semantic* referent must be referred to in order, to find a referent for the indefinite NP. This is the problem Kripke originally set out to solve i.e. whether the indefinite NP was to refer or not. He also takes the presence of the *Speaker's* reference as a sufficient precondition for a subsequent anaphoric pronoun and so the word ordering of the utterance becomes important.

Lewis (Lewis 1979), developed another variation on the Gricean theme and thereby laid the foundations for Kamp's (Kamp 1981) Discourse Representation Theory which will be discussed in Chapter Three. He believes that a well-run conversation contains presupposition, which the speaker and hearer take for granted. He also asserts that these presuppositions are continuously evolving in conversation, with the evolution governed by rules of accommodation. Lewis compares language and baseball implying that like baseball every discourse has a score, which is manipulated by rules of accommodation that cope with the introduction of new information, and highlight the salient elements present in the discourse.

> *If at a time t something is said that requires, if it is to be acceptable, that x be more salient than y; and if, just before t, x is no more salient than y; then....at t, x becomes more salient than y.*
> (Lewis 1979).

He claims that a pronoun may refer to whatever object is maximally salient in the situation of its utterance. Heim explains how an object may be promoted to maximal salience:

> *A necessary condition for an utterance to promote an object X to maximal salience is that the sentence containing the utterance contains either an NP that refers to X, or a singular indefinite NP whose predicate is true of X.* ( Heim 1982).

Lewis's thinking is that:

*Although indefinite descriptions...are not themselves referring expressions, they may raise the salience of particular individuals in such a way as to pave the way for referring expressions.*

(Lewis 1979).

Grice (Grice 1975), Kripke (Kripke 1980) and Lewis (Lewis 1979) do not endorse the view that indefinites should be regarded as variables or quantifiers, but state instead that the contribution of indefinites to sentences should be measured in degrees of salience and differing types of reference. To sum up, their approach to indefinites is that anaphoric pronouns refer to the maximally salient object; whose salience is due to the content of preceding utterances. In Kripke's analysis the *Speaker's* reference refers to a particular individual made salient in the speaker's mind by previous utterances. The downside of these approaches is found in the addition of extra syntactic work, where the grammar will have to assign salience raising potential to expressions as well as a local analysis. These methods are at a disadvantage as they still leave the implementor with a pronoun resolution problem along with the added chore of assigning salience levels to all expressions.

Evans' (Evans 1980) approach attempts to maintain Russell's view. He says there are three types of pronoun: bound variable, pragmatic (those referring by virtue of their referent's salience), and the unique E-type pronoun. He claims that E-type pronouns always have quantified NPs as their antecedents but are not bound by them, and that they may be regarded as descriptions in disguise waiting to be expanded. Anaphoric pronouns are E-type pronouns, and Evans proposes to analyse them as paraphrases of certain definite descriptions present in the discourse. These paraphrases are inserted systematically to replace the pronouns in discourse.

(9). *A dog came in. It lay under the table.*

The indefinite NP *a dog* of (9) is treated as a potential description that may replace a pronoun in utterances to come. Evans asserts that the pronoun *it* is a shorthand description waiting to be expanded. This expansion is achieved by replacing *it* by a definite description formed from information provided by previous discourse utterances. In this case the information is provided by the first sentence of (9). Sentence (10) displays the results of expanding the E-type pronoun using an available NP.

(10). *A dog came in. The dog that came in lay under the table.*

The domain of possible definite descriptions available to an E-type pronoun is small, with the descriptions selected from a range of relevant individuals. In order for this substitution theory to work, Evans makes two assumptions:

Assumption (1). Certain anaphoric pronouns mean the same thing as certain definite descriptions.

Assumption (2). Definite descriptions are to be analysed in a certain way

Assumption (1) is stated so that substitutions can take place, while assumption (2) proves to be very loose and the failing point of the E-type theory.

Definite descriptions that are anaphoric to indefinites probably work more or less the same way as pronouns that are anaphoric to indefinites; once a solution has been found for one problem it will apply to the other.

> *Evan's proposal to paraphrase anaphoric pronouns away in favour of anaphoric definite descriptions is not a solution.* (Heim 1982).

Definite NPs cannot be used to solve the problems of anaphoric pronouns if the theory of definites is left undeveloped, as the same problems apply in both cases. This proposal has little substance as the treatment of definite descriptions is left undeveloped by Evans.

He also asserts that E-type pronouns carry a uniqueness implication. This is widely disputed and is another failing of this form of analysis.

(11). *If John has a donkey, he beats it.*

Evans analyses this sentence as (12):

(12). *If John has a donkey, John beats the donkey that John owns.*

This analysis (12), presupposes the unique condition that *John* has just one *donkey.* The uniqueness condition does not carry easily to other sentences as in (13), where it cannot be distinguished which *cardinal* the pronoun *he* is talking about: it could be either cardinal or both cardinals i.e. each blessing each other, so the uniqueness condition does not hold true.

(13). *If a cardinal meets another cardinal he blesses him* (Kamp 1981)
(14). *The man who gave his paycheque to his wife is wiser than the one who gave it to his mistress* (Karttunen 1969).

Also according to Evans the E-type pronoun *it* of sentence (14) is unique. Assume *his paycheque* is the antecedent of *it*, referring to the paycheque of the first man. The position of the pronoun *it* in the sentence implies that it refers to the paycheque of the second man therefore the pronoun does not have a unique referent. A final illustration of how this uniqueness assertion fails is found in sentence (15).

(15). *If someone is in Rhodes, he is not in Athens.* (Heim 1982)

The word *someone* does not presuppose a unique person but supposes that there is at least one unique person that can be found in *Rhodes*. Evans' method replaces the pronoun *he* with the definite description *the person who is in Rhodes*, (16). This is an improper description as there will be many people who are in *Rhodes*.

(16). *If someone is in Rhodes, the person who is in Rhodes is not in Athens.*

The concept of E-type pronouns works well in some cases, but the uniqueness condition is implausible and the theory is too undeveloped in the form Evans presents. The problem of indefinite NPs and anaphors cannot be solved using definite NPs if the theory behind definite NPs is left undeveloped. We are simply translating the problem from one form to another. The author will now explore the anti-Russell school of thought.

### 2.1.2 The Anti-Russell School - Indefinite NPs can refer.

This school is led by Chastain and Strawson whose claim is not that indefinite NPs *always* refer, but that they *can* refer. They concluded that the extent to which indefinites participate in anaphoric relations simply defies Russell's analysis.

Strawson (Strawson 1952) says that some anaphoric relationships are coreferential, and that indefinites can be used to refer. He states that anaphoric pronouns found in sentences like (3) are instances of plain coreference. Heim further explains Strawsons thinking:

> *..that we must therefore acknowledge that indefinites can, at least sometimes, be used to refer.* (Heim 1982).

Chastain (Chastain 1975) is of the same opinion, and claims that sentences containing indefinites are ambiguous and have two meanings - the referential and the existential reading - where the analysis of the indefinite is ambiguous between the two. He explains that the existential reading of the indefinite does not emerge if something else in the sentence has wider scope. Chastain's referential reading relies on the thinking that:

(17). *There is a mosquito in here*

Sentence (17) may be read in two ways; the referential reading refers to a particular mosquito, and the existential reading refers to a place that is not mosquito-less, but where there is no particular mosquito in mind. He does not claim that indefinites always refer but simply that they can refer. Every utterance produces two possible readings with the referential reading being one of the possible readings.

> *Sentences containing indefinite descriptions, are ambiguous. Sometimes 'A mosquito is in here' and its stylistic variant 'There is a mosquito in here' must be taken as asserting merely that the place is not wholly mosquito-less, but sometimes they involve an intended reference to one particular mosquito.*
> (Chastain 1975).

Chastain relies heavily on the plausibility of the belief that:

> *An anaphoric pronoun with an indefinite antecedent ( where the latter does not bind it ) is coreferential with its antecedent.* (Heim 1982).

This approach is very similar to Kripke's (Kripke 1980) theory of *Speaker's* and *Semantic* referents. Instead of thinking of coreference we should think instead of the *Speaker's* reference and a correlation may be found between the two theories. Due to the similarity of approach, both theories share many of the same problems.

## 2.1.3 Indefinites as Quantified NPs - The Problems of Donkey Sentences.

Up to now in this thesis indefinite NPs have been treated as referring or non-referring NPs. They will now be treated as quantified NPs. This change of treatment is due to the introduction of a new problematic sentence type, the donkey sentence. These sentences are called donkey sentences after work in this area presented by Geach (Geach 1962), in which he cited only sentences involving donkeys. Donkey sentences (already encountered in the section on non-uniqueness and E-type pronouns e.g. (13) and (14)), are sentences that contain an indefinite NP which is inside an if-clause or relative clause, and a pronoun which is outside that clause, but is related anaphorically to the indefinite NP as in (18) and (19). The two types of donkey sentences, the conditional and universal sentence, have the same truth conditions as they basically represent the same instruction i.e. IF A, B - if A is true then B is performed, and Every A,B - for all A then B is performed.

(18). *If a man owns a donkey, he beats it.*

(19). *Every man who owns a donkey, beats it.*

Sentences (18) and (19) are represented by the truth conditions of (20) where the indefinite NP is universally quantified with a scope that extends beyond that of the NP's clause. The indefinite can therefore bind a pronoun outside its own clause. In (20) the indefinite NP *a donkey*, has now got the semantic meaning of *all donkeys*, due to the universal quantifier that binds it. The question must be raised as to whether these truth conditions are correct or not, as reading the sentence human intuition awards the NP the power of an existential quantifier i.e. refers to a unique donkey.

(20). $\forall x \forall y ( \text{man}(x) \wedge \text{donkey}(y) \wedge \text{owns}(x,y) \rightarrow \text{beats}(x, y) )$

The assumption of quantificational force i.e. indefinite NPs acting as quantifiers, and scope possibilities of donkey pronouns, is necessary to account for this sentence type and anaphoric resolution within these sentences. An explanation is required, however, to account for the exact conditions under which an indefinite may receive this wide scope universal interpretation.

Smaby (Smaby 1979) and Egli (Egli 1979) construct an artificial language which shares 'every' NPs, indefinite NPs, proper names and pronouns as well as many other NPs with natural languages. They specify an algorithm that converts texts to logical language formulae in this artificial language. Both authors consider donkey sentences as a major application of their proposals, but after this their proposals differ.

Egli (1979) claims that indefinites can take scope beyond a matrix sentence and relative clause. He asserts that an equivalence relation exists between sentences and logical forms which is similar to the valid equivalence relation of predicate logic, (21).

(21). If P and Q are formulas and P does not contain the free variable x, then the following formula is logically true: $(\exists xP \rightarrow Q) \leftrightarrow \forall x(P \rightarrow Q)$. (Heim 1982).

Egli parses sentences, applying context free rules of to form syntactic analyses which are then transformed into logical forms using a set of specified translation rules, and relations of predicate logic. He claims there is an algorithm consisting of a full set of translation rules and equivalence relations that converts every sentence into its logical form. The consequences of this claim are as follows:

> (a). An indefinite inside an if-clause is read as a wide-scope universal that may bind a pronoun in the then-clause
>
> (b). An indefinite in the first conjunct of an 'and' conjunction can be read as an existential quantifier, whose scope extends throughout the second conjunct and can bind a pronoun here.
>
> (Heim 1982).

The donkey sentence of (22) will illustrate the workings of this algorithm and how the indefinite NP obtains quantificational force.

(22). *Every farmer who owns a donkey beats it.*

Sentence (22) is partially transformed by the conditional translation rules to (23), introducing a variable and predicate for the *farmer*.

(23). $\forall$ x (( farmer(x) $\wedge$ x owns a donkey ) $\rightarrow$ ( x beats it ))

which may be further converted to (24), by dealing with the indefinite NP *a donkey*.

(24). $\forall x ((\text{farmer}(x) \wedge x \text{ owns } y \wedge \text{donkey}(y)) \rightarrow (x \text{ beats } y))$

Semantic interpretations are not assigned to syntactic representations directly but to the logical forms created by the construal rules. Egli's theory is subject to the same objections as that of Geach as truth conditions are assigned to complete texts only, with weak truth conditions predicted when the theory is extended to plurals. A sentence has weak truth conditions when these conditions do not impart any useful information, and encompass too vast a domain. This all means that Egli's theory does not improve on the theories that have gone before. He also treats universal quantifiers and existential quantifiers as the same quantifier, allowing under certain conditions ( i.e. $(\forall xP \rightarrow Q)$ where x is not free in Q), the universal quantifier to turn into a wide scope existential quantifier, $\exists x(P \rightarrow Q)$. This cannot ever occur, as false semantic representations may result from predicting a symmetry between universal and existential quantifiers as (25) illustrates.

(25). *A man is dead* this is represented by $\exists \mathbf{x} ( \mathbf{man(x)} \rightarrow \mathbf{dead(x)} )$

This representation may be transformed by (21) to: $\forall \mathbf{x} ( \mathbf{man(x)} \rightarrow \mathbf{dead(x)} )$ which falsely implies that all men are dead.

In certain cases Egli fails to account for the readings that indefinites exhibit, and can only cope if ad hoc construal rules are added for each new example.

Smaby's (Smaby 1979) proposal is very similar to that of Egli, but instead of a set of construal rules Smaby relies on one single rule R-Smaby. The rule states that an existential quantifier in the left conjunct of a conjunction must be exported to take scope over the entire conjunction and works for donkey sentences

(R-Smaby). Rewrite $\exists x \; S_1 \wedge ( S_2 \text{.....pronoun....})$

as $\exists x ( S_1 \wedge ( S_2 \text{.....x...........})$

Smaby thereby predicts the correct truth conditions for non-donkey sentences while being able to handle donkey sentences also. Let's take sentence (16) once more, to illustrate this point.

(16). *If someone is in Athens, he is not in Rhodes.*

This sentence holds the implication of (26).

(26). (*If someone is in Athens* $\rightarrow$ *he is not in Rhodes*)

Changing the implication to the logically equivalent negation and conjunction leads to (27)

(27). $(\text{NOT}(\exists x \ (\text{in\_Athens}(x) \land \text{NOT}(\text{NOT}(\text{in\_Rhodes}(he))))))$.

(Heim 1982)

Applying R-Smaby exports the existential quantifier, and replaces the pronoun *he* with x to yield (28).

(28). $\text{NOT}(\exists x (\text{in\_Athens}(x) \land \text{NOT}(\text{NOT}(\text{in\_Rhodes}(x)))))$

(Heim 1982).

moving the NOT into the expression results in the logically equivalent expression of (28)

(29). $\forall x (\text{NOT}(\text{in\_Athens}(x) \land \text{in\_Rhodes}(x)))$.

Comparing this method with that of Egli's, it has the advantage of only exporting original existential quantifiers. Unfortunately it runs into the same problems as Egli's approach i.e. it is unable to account for certain indefinite readings. e.g. sentences (13) and (14) in section 2.1.1.

(13). *If a cardinal meets another cardinal, he blesses him.*

(Kamp 1981)

In (13), R-Smaby does not give any guidelines as to which is performing the action of blessing the other cardinal, therefore since this distinction cannot be made the pronouns cannot be replaced. The analyses proposed by Smaby and Egli cannot account for all options and handle all the facts available, and only work in a limited domain.

Another approach is that of Parsons (Parsons 1978) who agrees with Evans about there being a third semantically distinct reading for pronouns, with this reading treated on a par with definite descriptions. He present a Montague fragment which translates not only NPs with indefinite articles but also other personal pronouns into logical expressions of the form of (30).

(30). $\exists x ( \forall y (.....( y ) \leftrightarrow x = y ) \ \wedge P( x ))$

Parsons' method contains a syntactic transformation that substitutes certain definite descriptions for pronouns, namely those definite descriptions matching a clause somewhere to their left which contains an indefinite. Taking the sentence (31), the pronoun *it* is replaced in (32) by a definite description formed from the clause *John owns a donkey.*

(31). *John owns a donkey and beats it.*

This sentence is paraphrased as (32) with transformation as meaning preserving.

(32). *John owns a donkey and beats the donkey such that John owns it.*

The formula of (30) represents the structure definite NPs and pronouns will be mapped onto, to represent the logical forms of the said NPs. It asserts the presence of a unique x such that for all y, a predicate exists involving y where x and y are equal and there is a further operation on x. Parsons presents a rule fragment that maps NPs onto this formula, and the equality x = y is designed to link the newly produced definite descriptions of the paraphrased utterance (32) with the original unique indefinite NP. This process is illustrated by (31)-(33) where (32), illustrates the definite NP introduction to replace pronouns. The next step involves the introduction of logical expressions, with the template specified in (30) used to represent the new definite NP *the donkey such that John owns it*, with *beat(j,u)* filling the predicate position of P(x).

(33). $\exists u(\text{own}(j,u) \wedge \text{donkey}(u)) \wedge \underbrace{\exists u(\forall v((\text{own}(j,v) \wedge \text{donkey}(v)) \leftrightarrow u = v) \ \wedge \text{beat}(j,u))}_{\textit{From formula (30).}}$

This strategy works well for donkey sentences, resolving the anaphoric pronoun easily. The theory is, however, not fully developed, only covering some options. The logical form of (33) implies that there is at most one donkey that John owns and beats i.e. a unique donkey. Parsons analysis appears to agree fully with that developed by Evans, even down to the uniqueness condition. The E-type strategy which treats pronouns in donkey sentences as disguised descriptions assumes the referent of the object denoted by the pronoun is unique. There is however, no systematic uniqueness requirement in donkey sentences as sentences (13) and (14) illustrated in section 2.1.1. A more recent approach by Parsons (Parsons 1990) relegates this uniqueness condition to one of context, claiming that conditionals and when-clauses involve a quantification over eventualities or situations.

(34). *John sings.*

Sentence (34) may be interpreted as the occasion when there exists a person called John who sings, represented as (35), where $\exists_O$ signifies the occasion where *John sings*.

(35). $\exists_O$ ( sings (j,o)) where John is j and o is the occasion.

Parsons claiming that pronouns are descriptive according to certain minimal situations. Yet this still does not handle the problems posed in sentences like (13), as for any minimal situation S in which a cardinal$_i$ meets another cardinal$_k$ the cardinal in that situation blesses the cardinal in that situation, but which cardinal performs the blessing? The approach does not work as it is not clear which cardinal is being referred to at any one time. The author concludes that donkey sentences are not satisfactorily interpreted by treating the crucial pronoun as E-type and its indefinite antecedent as an existential quantifier, as Parsons proposes. This conclusion is drawn from the uniqueness condition specified by Evans to which donkey sentences do not adhere, and the failure of an undeveloped definite NP theory to solve the problems of indefinite NPs.

### 2.1.4 Indefinite NPs as variables - The application to donkey sentences.

Continuing with the static analyses of indefinite NPs and anaphors, the author now considers the treatment of indefinites by Kamp (Kamp 1981) and Heim (Heim 1982). They claim that indefinites do not have quantificational force of their own, and state that indefinites show their true nature in donkey sentences, that they have no force of quantification, and should be treated like variables which get bound by whatever quantifier is there to bind them. A further claim is that indefinite NPs never contribute anything to the sentence in which they occur, other than this variable interpretation. Whenever an indefinite NP seems to be acting like a quantifier, it is really something else in the sentence that is causing this effect.

*Their assumption is that a discourse-level existential quantifier binds variables contributed by indefinite NPs which do not become bound in some other way.* (Rooth 1986)

(36). *Every man who owns a donkey beats it.*

In section 2.1.3 the indefinite *a donkey* was treated like a quantifying NP gaining the power to bind variables beyond the NP clause i.e. (37).

(37). $\forall$ x,y ( man(x) $\wedge$ donkey(y) $\wedge$ own(x,y) $\rightarrow$ beat(x,y))

In this section (36) is treated differently by Heim and Kamp who claim that the indefinite acts like a variable which in this case is bound by the universal quantifier *Every*. The thinking behind the theories of Heim and Kamp is extremely similar although they use different analogies to model their theories, which are discussed later in this chapter. In practice the ideas of Heim and Kamp lead to the following logical forms or disambiguated representations of (38) and (41).

(38). *A man walked in. He was wearing a hat* (Chierchia 1994).

Rules operate on the constituents of sentence (38) to change this sentence into a disambiguated representation. The first step in this disambiguation is to associate any indefinites with free variables. The remaining parts of the sentence are shown in their first order format.

(39). man(x) $\wedge$ walked_in(x) $\wedge$ hat(y) $\wedge$ wear(x,y).

It is assumed that the pronoun *he* is associated with the same index as the indefinite *a man*. A rule of existential closure is applied to assign existential force to indefinites that are not in the scope of a quantifier. As can be seen from (40) indefinite NPs can bind across sentence boundaries.

(40). $\exists$ x y ( man(x) $\wedge$ walked_in(x) $\wedge$ hat(y) $\wedge$ wear(x,y) )
(Chierchia 1994).

A more complicated sentence structure is the quantified sentence. These sentences are tripartite structures with a quantifier, restrictor and scope, which may be divided up accordingly as sentence (41) illustrates.

(41). *Every ( man who owns a donkey)(beats it with a stick)*

$$\text{Every } \underbrace{(\text{ man who owns a donkey})}_{\text{RESTRICTOR}}\underbrace{(\text{beats it with a stick})}_{\text{SCOPE}}$$

(Chierchia 1994).

Processing begins with the introduction of first order predicates and free variables for all predicates of (41) resulting in (42).

(42). $\text{Every ( man(x)} \wedge \text{donkey(y)} \wedge \text{own(x,y) ) (stick(z)} \wedge \text{beat\_with(x,y,z) )}$ .

(Chierchia 1994)

The quantifier *Every* must be dealt with next. Quantifiers are unselective and bind all variables in the restrictive clause, therefore the quantifier *Every* binds x and y. The variables the quantifier binds are copied into the quantifier in (43).

(43). $\forall \text{ x,y } \underbrace{(\text{ man(x)} \wedge \text{donkey(y)} \wedge \text{own(x,y) })}_{\textit{Restrictor}}\underbrace{(\text{ stick(z)} \wedge \text{beat\_with(x,y,z)})}_{\textit{Scope of universal quantifier}}$

The indefinite NP of the scope condition is not quantified so the rule of existential closure is applied to give it existential power (44). This rule is applied to all non-quantified indefinite NPs. The logical form of (44) yields the correct truth conditions for sentence (40).

(44). $\forall \text{ x,y } ( \underbrace{(\text{ man(x)} \wedge \text{donkey(y)} \wedge \text{own(x,y) })}_{\textit{Restrictor}} \rightarrow \underbrace{\exists \text{ z ( stick(z)} \wedge \text{beat\_with(x,y,z) ) )}}_{\textit{Scope of universal quantifier}}$

(Chierchia 1994)

The last two examples illustrate how indefinites may bind across stretches of discourse, with disambiguated representations formed by applying rules of transformation or construal. What exactly do these rules look like ? We will now look more closely at the construal rules employed by Heim. She associates syntactically analysed expressions of English with certain disambiguated representations or logical forms using rules of construal. Heim makes basic assumptions that every NP carries a referential index, all non-pronominal NPs are adjoined to the S node, and introduces rules of construal to deal with donkey sentences. Some of the basic rules of construal are listed below.

(1).NP Indexing
Assign every NP a referent index.

(2).NP Prefixing
In the parse tree adjoin every non-pronominal NP to S, leaving behind a coindexed empty NP.

(3).Quantifiers
Attach every quantifier as a leftmost immediate constituent of S.

(4).Existential closure
Insert ∃ in the nuclear scope of the quantifier and if a NP is non-quantifying insert a ∃.

(5).Conditionals
An if-clause is positioned between the quantifier it restricts and the rest of the sentence.

(6).Quantifier Indexing
*Copy the index of each indefinite NP onto the nearest quantifier that shares the same branching node. A quantifier binds all and only those variables whose referential indices match one of the quantifier indices.*

( Heim 1982).

Indefinites are interpreted as formulae containing free variables whose quantifying force is determined by the closest quantifier that does not dominate the indefinite, but is dominated by a node that dominates the indefinite. Indefinite NPs must always carry a new referential index with the added stipulation that an indefinite cannot carry the index of an NP that is situated directly to its left in the same sentence.

(45). *Every man arrived.*

Assuming that a simple syntactic analysis exists for sentence (45), the sentence will be analysed using Heim's system of construal rules.

- The first step in this analysis assigns a referential index to each NP, this step can be seen in the parse tree Fig 2.1.

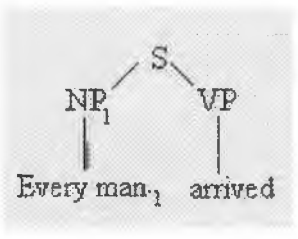


**Fig 2.1. Developing the logical form**

- Non-pronominal NPs are assigned to an S node leaving behind a coindexed empty NP, in this case $e_1$. The node $e_1$ represents an empty NP which was introduced by the removal of the NP, *Every man* Fig 2.2.

- Finally the quantifier construal rule is applied, which attaches every quantifier as a leftmost immediate constituent of the S node. Fig 2.3 shows the parse tree, with the linear representation found in (46).

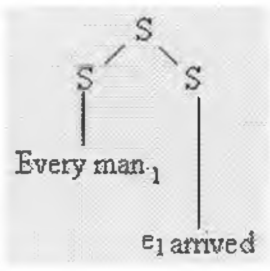


**Fig 2.2. A partial logical form.**

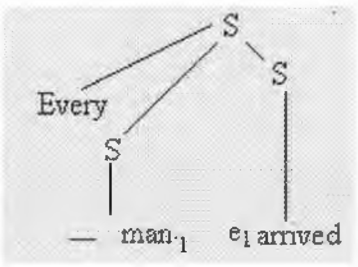


**Fig 2.3 The completed disambiguated logical form**

(46). $[_S$Every [man$_1$ $[_S$ e$_1$ arrived ]]] (Heim 1982).

The analysis produces the representation of sentence (45), in Fig 2.3 and (46), which may be interpreted as every variable assignment that satisfies *man*$_1$ also satisfies the scope condition *e*$_1$ *arrived.* There is, however, no direct way for a pronoun to impose its referential index onto a quantifier.

*The only way for a pronoun to get bound is indirectly by virtue of its being anaphoric to and thus coindexed with another NP.*

(Heim 1982).

Heim has thus presented a static standard method of capturing the semantics of discourse by direct interpretation of logical forms, which are formed by construal rules.

## 2.2 The Treatment of Indefinite NPs -The Dynamic Approaches

Another way of presenting things, while still staying with the concept of indefinites as variables, simplifies the construal component by using a different notion of meaning and presents a more dynamic view of the semantics.

*This dynamic aspect is taken as a cue for deviating from the safe (and static) path of standard logic....these theories more or less try to combine static semantics with an analysis of dynamic aspects of language use*

(Cooper et al 1995)

In dynamic semantic theories new sentences in discourse are interpreted in the context of what has gone before, and in this way new information is built into an existing structure. The core philosophy behind these theories is that *meaning is change* (Cooper et al 1995). Heim and Kamp introduce metaphors to deal with this more dynamic form of semantics, all of which hinge on the concept of context, and how the context or state of mind of the interpreter is changed by every new utterance of the discourse. Heim uses the metaphor of the file, with indefinites introducing new cards to the files. Kamp uses a Discourse Representation Structure, a type of flowchart to represent the disambiguated semantic meanings. Each of the theories relies on a small body of construal rules (as a discussion of

Heims static system illustrated in the last section), with the emphasis moved from the construal component to a now bigger semantic component.

### 2.2.1 Heim and File Change Semantics

Heim works with the metaphor of File Change Semantics, where information is stored in cards of files. The metaphor provides a simple picture of how information is built up in discourse. A small body of construal rules are used to produce a disambiguated representation. Files are introduced as an extra level of analysis between language and the world, encoding information from non-linguistic and linguistic sources. Processing a discourse involves adding information to the file which contains numbered cards. The standard condition is that indefinite NPs are novel and add new cards to the file, whereas definite NPs are familiar and update existing cards. In model-theoretic terms a picture of what information is available to act as antecedents is built up and stored in the domain of the file Dom(f), along with the numbers on cards of the file. The domain of a file keeps track of the variables being used by the file, as well as relationships between these variables. The content of discourse i.e. the predicates and formulae encountered in the file as well as the propositional content of the discourse is stored in the satisfaction set Sat(f). Every new utterance updates the contents of the file, changing the domain and satisfaction sets. A file can be represented as an ordered pair < Dom(f), Sat(f) >, with the semantic value of the discourse represented in terms of the context or file change potential of the file. Files represent contexts with any change in the file represented by the file or context change potential of that file. This is a function from file to file that represents the change new information brings to the existing file, to produce a new updated file i.e. *existing file f + new information = updated file f '.*

(47). *A man loves a woman*

Sentence (47) introduces two new indefinites to the file f, each of which introduces a new card in the file, card(1) and card(2). The cards contain the following information:

card(1). {man(1), 1 loves 2}, card(2). {woman(2), 1 loves 2}.

The domain and satisfaction sets of this file are

Dom(f) = {1, 2}, Sat(f) = { man(1), woman(2), 1 loves 2 } with the file f represented by the ordered pair

$$< \{ 1, 2 \}, \qquad \{ \text{man}(1), \text{woman}(2). \ 1 \text{ loves } 2 \} >$$

Heim's file change semantics is based on the teachings of Lewis (Lewis 1979) who views a new utterance as a change of context, which brings existing or new information to the forefront. Every time new information is added the file must be revised and sequences that do not agree with the new context must be removed. The satisfaction set gets more specific with each new bit of information, as (48) illustrates.

(48). *A television was stolen. No, George borrowed it yesterday.*

The mini discourse of (48), asserts in sentence 1 that the television was stolen, which is later refuted in sentence 2 with the new information that George actually borrowed it. The original information - the fact that the television was stolen is now defunct - and must be removed from the file sequence. This mirrors the way in which we as humans update our information stores.

Files are interpreted in certain contexts, so the truth condition of a file is defined by the context change potential of that file which is updated as new information is added. When this new information is added to the file, sequences present in the file that do not agree with the new context are removed and the domain and satisfaction sets are updated accordingly.

File generation starts with the interpretation of the semantic content of the discourse. New information introduces new cards or updates existing cards, and whichever action is chosen a bridge of cross-reference must be built between the new card or updated card and the pre-existing file, resulting in a hypertext-like system with links from file to file instead of node to node. This is carried out by accommodation rules which can be applied at any point during the evaluation of a complex logical form. These rules look after the addition of anaphoric pronouns and definite NPs to the file.

The outcome of File Change Semantics (FCS) is the simplification of the construal component but at the cost of an arguably more complex semantics. An improvement provided by FCS is found in how it handles definite NPs. Heim rejects Russell's analysis of definite NPs as quantifying NPs, saying that a definite description which does not contain any bound variables has a unique referent. Definites are therefore treated like free variables whose value is fixed by the context of the sentence. She also assumes that the content of the description is presupposed. The condition used to add definite NPs to files must now be updated to handle two types of definite: the definite NP having a unique referent and that NP that introduces new information to the file, i.e. an unfamiliar definite. For the unfamiliar definite NP the rule of accommodation introduces a new file card for the NP, and accommodates the new card with the existing state of affairs in the file. Attempts are made first of all to accommodate globally, and failing this local accommodation is attempted.

The author's preliminary investigation of dynamic semantics illustrates the basic premise of *meaning as change*. The notion of context is captured by the chosen structures (e.g. files for Heim) that are constantly updated with the addition of newly processed utterances. Truth is captured using a variant of model-theoretic semantics, in the form of satisfaction sets and domains in the case of File Change Semantics. The next section presents another dynamic theory, namely Discourse Representation Theory developed by Kamp (Kamp 1981).

### 2.2.2. Kamp and Discourse Representation Theory

Kamp defines a formal language that corresponds to the logical forms the author has presented already, thereby cutting straight to the problem of resolving anaphors. This language is designed to allow anaphoric relations to be expressed directly, without the complexities of relative clauses, agreement and general problems of natural language taking centre stage. The language copes with areas including quantification, negation and implication and is directly and simply rewritten in a Discourse Representation Structure (DRS).

A DRS is a type of flowchart or diagram that is built while the discourse is processed, with the final DRS representing the semantic content of the discourse as a whole. Within the flowchart the DRS is made up of a set of discourse referents or variables and a set of conditions. DRSs may be nested inside one another to represent complex conditions. It is possible to provide a set of construal rules that yield logical forms having essentially the same semantics as DRSs, as they have a close affinity with certain formulae of symbolic logic, more specifically those of first order predicate calculus. Each DRS can be regarded as a formula of predicate logic in disguise, with a function mapping from logic to DRS.

The Discourse Representation Theory (DRT) itself processes a text T consisting of sentences $S_1 \ldots S_n$ and determines a semantic representation, a DRS. As the discourse is processed each new sentence updates the existing DRS producing a new output DRS which captures the context of the discourse processed so far. The process advocated by Kamp involves the construction of the DRS, with truth defined as an embedding of the completed DRS in a model. DRT differs from other forms of model-theoretic semantics in that it offers a theory of truth conditions and interpretation.

Embedding the DRS in a model to capture truth conditions involves mapping from the discourse referents or variables of the DRS to the individuals in the model. In more formal terms a model M for a vocabulary V is the set $<U_M, Name_M, Pred_M>$, where $U_M$ is the universe of model M, $Name_M$ is a function from the set of individual constants of V to $U_M$, and $Pred_M$ maps from the set of predicate constants of V into suitable objects associated with the universe of M. A DRS is correct with respect to a model M if it is possible to embed the universe U of the DRS truthfully into $U_M$ i.e. a function f operates on $U_K$ and verifies all the conditions in $U_M$.

The formal language created by Kamp is not essential for the development of a DRS as surface structures are directly mapped into DRS. Kamp provides a set of construction rules or trigger rules that readily convert text to DRS format, without having to refer to the formal language. Accessibility within the DRS determines when a pronoun can be anaphorically linked to an indefinite antecedent. The discussion of DRT continues in Chapter Three which presents an overview of the surface structure of DRT, discussing the treatment of definites, indefinites, quantification and negation within the DRT framework. Chapter Four then implements a DRT module for a small fragment of English.

# Chapter 3 An overview of Discourse Representation Theory.

## 3.0 Introduction

Discourse Representation Theory (DRT) developed by Hans Kamp (Kamp 1981), works on a discourse processing it sentence by sentence building on what has gone before to produce a semantic representation, the Discourse Representation Structure (DRS). It is a departure from traditional methods which treat sentences in isolation giving each sentence a separate semantic representation, and so fail to find antecedents for anaphoric pronouns within a discourse. In DRT anaphora are analysed not as a relationship between pronouns and other NPs, but as one between pronouns and the discourse referents that are already present in the semantic representation under construction. The hypothesis is that anaphoric pronouns select their referents from certain sets of antecedently available entities or discourse referents. By representing the whole discourse within the same DRS, all suitable referents for anaphoric pronouns are available to act as antecedents, and are easily accessible from within the structure. The concepts of suitability and accessibility are central to the thinking behind DRT and will be discussed in this chapter.

This theory is an attempt to bring two schools of thought together: the logicians who see meaning as truth conditions and the philosophers who regard meaning as that which the user grasps when he understands the words he hears. Sentences of the discourse are analysed one by one adding to the existing DRS. The first stage of the DRT cycle involves the syntactic analysis of a sentence of the discourse, followed by the application of construction rules to the said syntactic representation extending the existing DRS. DRT is not tied to a theory of English or syntax and so the third stage defines truth as the embedding of the newly formed or extended DRS in a model of the world.

In this chapter the basic mechanisms, construction rules and antecedent prediction methods of DRT will be discussed.

## 3.1 The basic building blocks of Discourse Representation Theory

As mentioned above the basic unit of DRT is the Discourse Representation Structure (DRS). A DRS is made up of two parts < U, Con >, where U is the set of referents or variables presently in use and Con is the set of conditions of the DRS. The DRS itself is represented by a rectangular box that is divided in two with the top half reserved for U while the bottom is the set Con. DRSs are called $K_i$ where $i$ is given the value from 1 upwards with U and Con of a DRS $K_j$ called $Uk_i$ and $Conk_i$

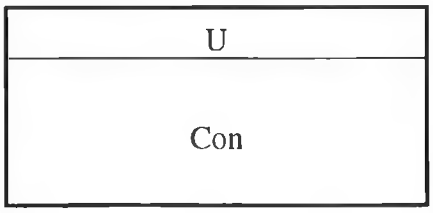


**Fig 3.1 The template for a simple DRS.**

### 3.1.1 An overview of basic construction rules of DRT.

Construction rules search for trigger words and categories in the parse trees of sentences. Firing a trigger may involve the introduction of a new discourse referent x to U accompanied by a condition in Con, the condition being generally of the form **word(x)**. Construction rules exist for many categories, including proper nouns and indefinites.

The following example highlights the proper noun construction rule, and illustrates the development of a simple DRS. Taking the small discourse of (1), processing begins with the first sentence *John owns a book.*

*(1).* *John owns a book. He reads it.*

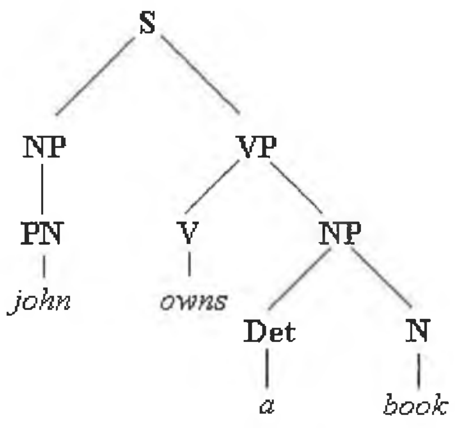


**Fig 3.2 The parse tree of *John owns a book.***

The parse tree is processed category by category from left to right. The proper noun category fires a construction rule which introduces a new referent *x* in U and the condition *John(x)* in Con. The word *John* is then replaced by *x* in the parse tree changing the sentence to *x owns a book.*

The verb *owns* follows. There are no construction rules for verbs and *owns* is quickly skipped over leading to the indefinite, *a book*. The indefinite construction rule is a simple one, again introducing a new discourse referent *y* to $Uk_i$ and condition *book(y)* to $Conk_i$.

Indefinite and definite NPs are treated differently in DRT: indefinite NPs introduce new information to the DRS, while definite NPs are considered familiar and generally refer to something explicitly mentioned before in the discourse. Definite NPs will be discussed more thoroughly in section 3.1.3.

Continuing the example, the indefinite is replaced in the sentence by the referent *y* to give the new sentence *x owns y*. The tour of the parse tree is complete and the sentence has been fully processed, the discourse analysis continues with the formation of a parse for the next sentence, extending the existing DRT $K_1$.

| x, y |
|---|
| John(x) |
| x owns a book |
| book(y) |
| x owns y |

**Fig 3.3 The completed DRS of *John owns a book.***

Processing of discourse (1), continues with the second sentence (2).

*(2). He reads it.*

This sentence contains anaphoric pronouns, whose antecedents must be found from the set U of available, suitable referents. The construction rule for anaphoric pronouns introduces a new discourse referent *w* to $Uk_i$ with the anaphoric pronoun replaced in the sentence by *w*. The partner condition $w = x$ is added to $Conk_i$, where *x* is a **suitable** referent already existing in $Uk_i$.

The idea of **suitability** introduces a weak point in Kamp's theory. DRT gives precise conditions for anaphoric linking, but does not give a way to choose the appropriate referent when more than one is available. All that is said is that the chosen antecedent of an anaphoric pronoun must be suitable. A suitable referent, therefore, can be thought of as a referent with the same gender and number as the anaphor, and can be found in the set of available referents of the DRS. In a simple

DRS $K_i$, the set $Uk_i$ is the set of available referents. There may of course be more than one suitable referent, in such a case each suitable referent must be bound to the anaphor producing the appropriate DRS.

A single discourse may produce two or three DRSs, depending on the number of suitable referents available and accessible from an anaphor at any one time. This case is illustrated by sentences *(12)* and *(13) of chapter 2. (12)* is ambiguous as one cannot decide which *cardinal* is performing and which is receiving the blessing or are they both mutually blessing each other? Each possibility must be represented by a complete DRS, resulting in three different DRSs for *(12)*

*(12). If a cardinal meets another cardinal he blesses him.*

DRT relies on the addition of external theories to aid in the choice of a suitable antecedent. Extra information must be encoded e.g. gender and number agreement between anaphor and antecedent. This may be performed in the syntax module, by adding more information to our grammar or semantically by adding extra information.

> *The strategies used in selecting the referent of anaphoric pronouns are notoriously complex; they usually employ background assumptions about the real world, grammatical clues, such as the requirement of number and gender agreement between the anaphor and its antecedent, and the order in which the potential referents were introduced by the preceding discourse.*
>
> (Kamp 1981).

*(3). John saw Luke dead. He cried.*

The thinking behind suitability implies that *He* can only be *John,* as dead people like *Luke* cannot cry. **Suitability** is really another name for the encoding or attachment of world knowledge to DRT. Kamp assumes this mechanism exists, and will work hand and hand with the mechanics of DRT.

Processing of sentence (2) continues as the sentence is further reduced to *w reads it*, the verb *reads* is left untouched but the anaphoric neuter pronoun *it* fires the personal pronoun construction rule. This rule also deals with expletive pronouns, but there is a separate reflexive pronoun construction rule. The author will not discuss this rule in this thesis, dwelling instead on the treatment of personal pronouns.

| x, y, w, u |
|---|
| John(x) |
| x owns a book |
| book(y) |
| x owns y |
| He reads it |
| w reads it |
| w = x |
| w reads u |
| u = y |

**Fig 3.4 The completed DRS of *John owns a book. He reads it.***

Finally the pronoun construction rule introduces a referent **u** to $Uk_1$ to represent *it*, with the changed sentence *w **reads** u*, and the condition $u$ = *??* where *??* is a suitable referent. The search continues as before, there is now a choice from *w, x* or *y* in $Uk_1$ (*u* itself is obviously excluded). As reflexive pronouns are omitted, the referent *w* represents a non reflexive anaphoric pronoun and is therefore excluded from consideration leaving the set {x,y} with *x* (male) and *y* (neuter), i.e. *y* is a suitable referent for the neuter anaphoric pronoun. The condition $u = y$ is added to $Conk_1$. The sentence *w **reads** u*, is irreducible so processing stops. The completed DRS is shown in Fig 3.4.

### 3.1.2 Definite NPs and construction rules.

In section 3.1.2, the indefinite, proper noun and anaphoric pronoun construction rules, introducing new referents and giving guidelines for the introduction of new conditions to Con were discussed. A construction rule is also triggered by a definite NP introducing a new referent and a corresponding condition. The author is ignoring generic definite NPs e.g. *the Romans*, *the King of France*, in this thesis.

(4). *The man works in Dublin.* (5). *A man works in Dublin.*

Sentence (5) introduces a new man into conversation, so a new referent and corresponding condition are added to the DRS. In discourse the speaker assumes that the listener knows what man is being talked about in (4), because this man will have been mentioned or named already.

When the definite NP *the man* is used, or any definite, the listener should have enough information to link the definite with something or somebody already mentioned; if no link can be formed the discourse fails and does not make sense. These facts must be represented in the condition introduced by the definite NP to Con.

The assumption of DRT is that indefinite NPs are treated as reference establishing terms, while definite NPs are familiar entities.

**Dealing with singular definite NPs.**

Definite descriptions can be used in different ways. The first way isolates a unique individual in discourse as in (6), where *the man* acts like the pronoun - *he*, isolating the individual *a man*. The description *the man* serves to pick either, whichever *man* was last mentioned or the most salient *man* (Lewis 1989) present in the discourse (see chapter 2 section 2.1.1).

*(6).* *A man and a woman went to the cinema. The man bought two tickets.*

*(7).* *A car is parked over there. The door is open.*

In (7), *The door* is understood to refer to *the door* of the *car* mentioned previously. The description *the door* is not coreferential with that of a *car* but by bridging refers to a part of the car.

Processing definite NPs being a complex affair, Kamp advocates a construction rule for definite NPs that introduces a new referent $x$ to $Uk_i$, a condition definite_NP(x) e.g. *the woman(x)* to $Conk_i$, and replaces the definite NP in the sentence by x. This treats the condition *definite(x)* as irreducible.

At this stage, Kamp asserts that if possible a condition $x = y$ must be introduced to $Conk_i$ where y is the **most salient or suitable** referent that the definite NP refers to as in (8). This condition links the definite NP with an existing referent in the text, treating it as a familiar element or an expanded pronoun.

*(8).* *John bought a cake. He put the cake away.*

Assume the construction of the parse tree. Construction starts in the first sentence with proper noun *John* and indefinite *cake*, which introduce simple new conditions *John*(x), *cake*(y) and referents *x* and *y*. The second sentence contains an anaphoric pronoun and definite NP, the male pronoun introduces a new referent *u* and a condition *u* = *other male suitable referent* i.e. *x*. The definite NP introduces a new referent of *w* with a condition *the cake(w)*. Kamp advocates that processing for definites should stop here, but if a suitable referent *?* is available for *w* add the condition *w* = *?*, thereby isolating the particular individual the definite refers to. In this way the particular individual or thing, the definite stands for can be isolated. The referent w is neuter and so is *y*, giving the condition *w* = *y*.

| x, y, w,u |
|---|
| John bought a cake |
| John(x) |
| x bought a cake |
| cake(y) |
| x bought y |
| |
| He put the cake away |
| u put the cake away |
| u = x |
| the cake(w) |
| u put w away |
| w = y |

**Fig 3.5 The DRS of *John bought a cake. He put the cake away.***

In Fig 3.5 the conditions *the cake(w)* and *w* = *y* are added to the DRS to represent the definite NP *the cake*, the second condition attempts to find an antecedent for w if at all possible. This is found readily in *y* using the notions of accessibility and suitability.

## 3.2 Sub-DRSs and Accessibility.

The sentences processed so far have all been represented by simple DRSs. As sentences get more complex so do the DRSs, and nesting occurs as DRSs can be embedded in each other. Complex sentences like conditionals, negation and universally quantified sentences all introduce sub-DRSs. Each structure has a specific sub-DRS template, introduced by construction rules. The first DRS generated is the principal DRS and all other DRSs are subordinate to this. DRSs are called $K_i$ and are numbered upwards, with i incremented by 1 each time.

### 3.2.1 Negation.

Negated sentences introduce a sub DRS. The parse tree of such a sentence contains categories **NP AUX not VP**. The negation construction rule is fired when **AUX not** is encountered within the parse tree introducing a sub DRS.

*(9). David does not own a porsche*

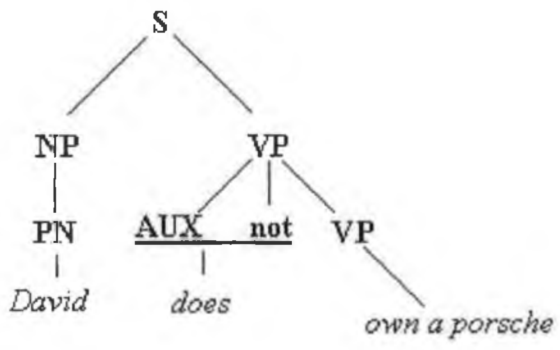


**Fig 3.6 A parse tree of a negated sentence.**

The representation of a negated sentence should involve the positive sentence it negates. That is, looking at (9) the positive sentence *David owns a porsche* will be involved in the DRS construction. Construction of a DRS for (9), starts at the proper noun node introducing a new referent *x* and condition *David(x)* with the sentence changed to *x does not own a porsche.*

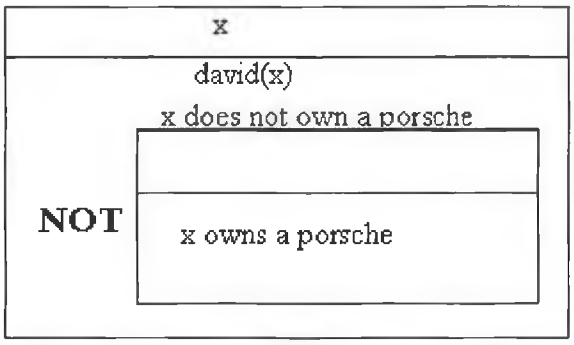


**Fig 3.7 The negated DRS template.**

Moving on in the parse tree, **Aux not NP** triggers the negated construction rule introducing a sub DRS of the form of Fig 3.7. The sub DRS for a negated sentence has a NOT outside the DRS and the positive sentence *x owns a porsche* ready to be processed inside the DRS Processing continues now as normal within the sub DRS $K_2$. The verb *owns* is ignored, moving processing to the indefinite *a porsche* that introduces a new referent y to $Uk_2$ and condition *porsche(y)* to $Conk_2$, leaving the changed irreducible sentence *x owns y*.

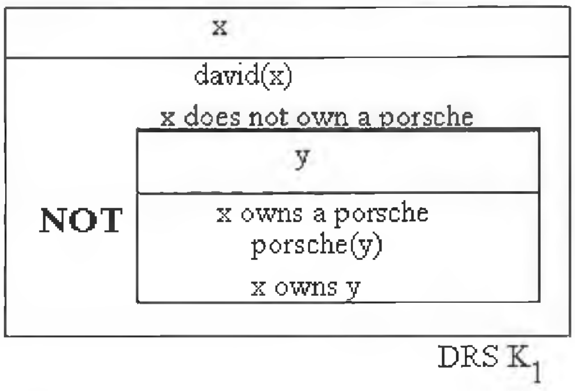


**Fig 3.8 The completed DRS for *David does not own a porsche.***

### 3.2.2 Referent Accessibility.

Simple DRSs have only one set of accessible referents available, the set $Uk_i$ of the DRS $K_i$. With the introduction of sub-DRSs more referents are available to act as antecedents for anaphoric pronouns. The pairing of anaphor and antecedent must be controlled. This control is achieved in DRT using accessibility. Accessibility governs the selection of referents, and is controlled by a subordination relation which is defined below in three parts:

(1). A DRS A is **subordinate** to another DRS B if A is nested in B.

This definition needs extension to deal with negated DRSs and continuous nesting of DRSs.

(2). A DRS A is **immediately subordinate** or has access to the referents of any DRS B, iff B represents a negation condition nested in $Con_A$

An example of immediate subordination is found in Fig 3.9 where the negated DRS $K_4$ is nested in $K_3$. This part of the subordination relation governs which referents are accessible from negated sub-DRSs.
The author has established the subordination relation between neighbouring DRSs but these relationships need to be passed up to other higher DRSs; this is achieved by introducing transitivity to the relation.

(3). If a DRS A is subordinate to or nested in DRS B, and C is subordinate to A, then C is subordinate to B

Fig 3.9 illustrates transitivity where $K_3$ is subordinate to $K_2$ and $K_4$ is immediately subordinate to $K_3$ implying that $K_4$ is subordinate to $K_2$.

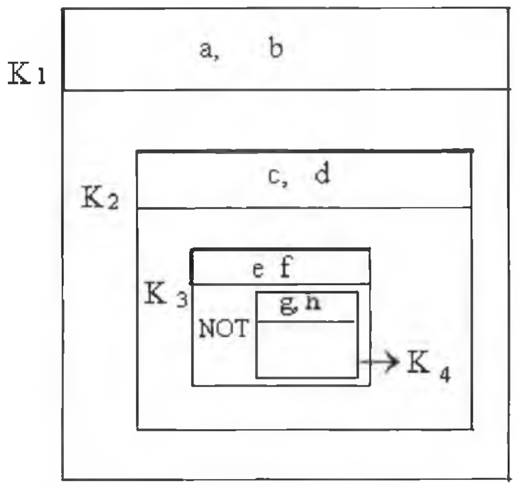


**Fig 3.9 An illustration of DRS embedding.**

The subordination relation between DRSs is expressed using <, if A is subordinate to B it is written as A < B. Establishing the subordination relation starts with the DRS with the greatest index i.e. the most nested sub-DRS, in this case $K_4$. The fact that $K_4$ is subordinate to $K_3$ and $K_2$, was established above, and is written as $K_4 < K_3 < K_2$. The DRS $K_2$ is nested in the principal DRS $K_1$, $K_2 < K_1$, transitivity extends the relation of the DRSs of Fig 3.9 to give $K_4 < K_3 < K_2 < K_1$. In this case the subordination relation included all nested DRSs in the same relation, this is not always the case. As sub-DRSs become more complex many different subordination relations can result for the same DRS system. This will be illustrated in section 3.3.3

Finally we will look at the referent sets formed by the subordination relation that are accessible from $K_{1,2,3}$ and $K_4$ of Fig 3.9. The principal DRS $K_1$ has access to the referents of $Uk_1$, so the accessible set of suitable referents for $K_1$ is {a,b}. $K_2$ has access to {c,d} of $Uk_2$ and the referents of $Uk_1$ (a,b) the only DRS above $K_2$ as $K_2 < K_1$. This gives an accessible set of {a,b,c,d}. DRS $K_3$ has access to {e,f} of $Uk_3$ and {a,b,c,d}, the referents of DRSs above $K_3$ as $K_3 < K_2 < K_1$ giving the accessible set {a,b,c,d,e,f}. $K_4$ has access to (g,h) and all the referents of DRSs above it $K_4 < K_3 < K_2 < K_1$, that is {a,b,c,d,e,f}, giving the accessible set is {a,b,c,d,e,f,g,h}.

### 3.2.3 Conditionals.

Conditional claims are made by a pair of sentences the first of which begins with some such phrase as *suppose, assume* or *if* e.g. *if Kevin owns a book, (then) he reads it.* These sentences are of the basic format **If/Suppose/Assume A (then) B**, and are represented by two DRSs, one for A and the other for the consequence B.

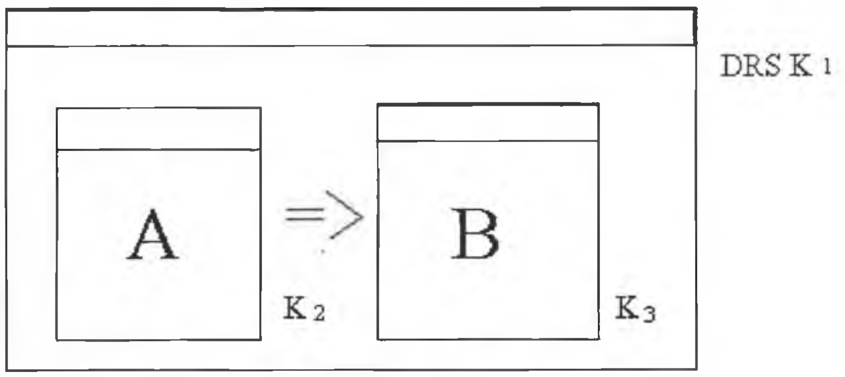


**Fig 3.10 The DRS template for conditional sentences.**

In Fig 3.10, The standard subordination relation for conditional DRSs is $K_3 < K_2 < K_1$ implying that, K3 has access to its own referents and the referents of $K_1$ and $K_2$, with DRS $K_2$ having access to its own referents and those of $K_1$ the principal DRS.

*(10). Mary is a teacher. If a student causes trouble, then she suspends her.*

Processing of discourse (10) starts with sentence 1, *Mary is a teacher.* This sentence contains a proper noun and indefinite which introduce new referents *x* and *y* and conditions *Mary(x)* and *teacher(y).* The existing sentence is changed to *x is y*, which suggests that the individuals represented by x and y coincide.

The second sentence, a conditional, is processed next. The *If* causes the introduction of the conditional sub-DRS template with *a student causes trouble* considered as part A and *she suspends her* as part B

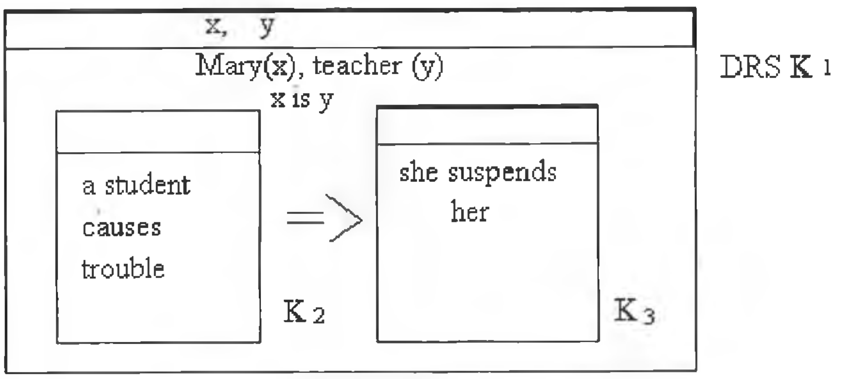


**Fig 3.11 A partially processed DRS of sentence (10).**

Processing continues within the sub-DRS, the parse tree is toured and any new conditions or referents for part A are put in K $_2$. Part A contains an indefinite NP, that introduces a new referent *u* and condition *student(u)* to K $_2$, the sentence now becomes *u causes trouble* which is irreducible. Part B, *she suspends her* contains two anaphoric pronouns.

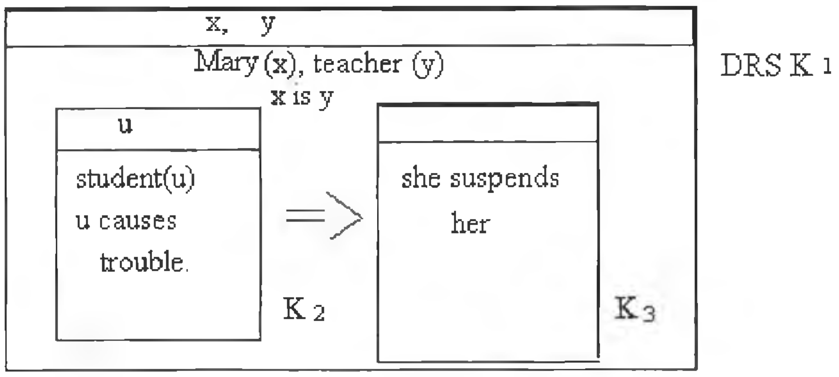


**Fig 3.12 Developing the DRS of sentence (10).**

The first pronoun of part B introduces a new referent v, changing the sentence to *v suspends her* with a new condition *v = ?*, where ? is a suitable discourse referent already existing in $K_3$ or a DRS above $K_3$. At present, *v* must find a suitable referent from the set of $K_3$, $K_2$ and $K_1$; that is from the set {*u,x,y*}. The discourse referents *x* and *y* are equal, so the set may be reduced to {*u,x*}. The referent u is male or female representing the indefinite *a student, x* is a female referent representing the person *Mary* who is *a teacher* and *v* represents the female pronoun *she*. Suitability requires that *v* must have a female antecedent and uses world knowledge to isolate this referent. The referent *x* is a teacher, world knowledge implies that teachers are in positions of power and may suspend students, where u is such a student. The principles of accessibility and suitability have isolated *x* as the antecedent for *v*, with the condition *v = x*, Fig 3.13

The second female pronoun introduces referent w and condition *w = ?*, with an accessible suitable referent list {*x,y,u,v*}. This list may be reduced to {*x,u*} as *y* and *v* equal *x*, the syntax implies that *w* cannot equal *v* as part B of (10) would be reflexive, and as *x* is *v*, the set is further reduced to *u* which must be the suitable referent. Even more evidence that *w* = *u* is provided by world knowledge which asserts that students are suspended by teachers therefore *w* must equal *u* as *u* is being suspended, Fig 3.13. If there is more than one suitable referent available at any one time, a DRS must be generated and fully processed taking each member of the suitable referent list one at a time. A single sentence can therefore result in more that one DRS.

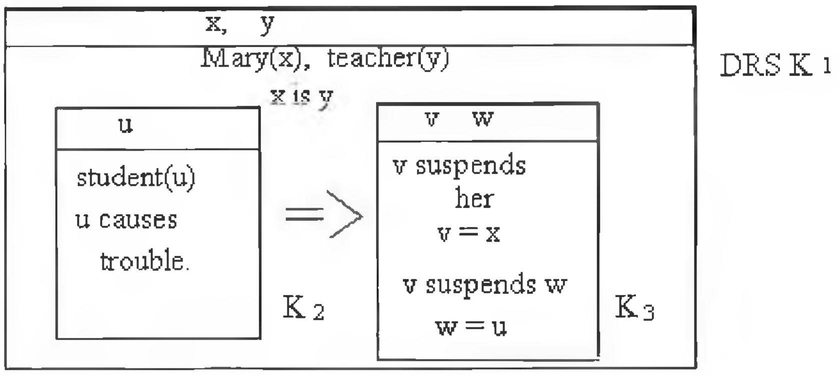


**Fig 3.13 The complete DRS**

Let's take another example, (11).

*(11). Mary is a teacher. If she does not know a student then she is a bad teacher.*

The first sentence of discourse (11) is processed as normal, placing referents and conditions in DRS $K_1$. The second sentence introduces the conditional DRS template, $K_2$ => $K_3$, with the sentence divided accordingly between the two sub-DRSs. At present $K_3 < K_2 < K_1$ is the only existing subordination relation. Processing *she does not know a student* of $K_2$ introduces a new referent *p* for *she* with the condition *p = a suitable accessible referent*. The subordination relation stated above provides the referent set *(m,t)*. The condition *m is t* states that these referents are equal so it is irrelevant which one is chosen as the suitable referent, Fig 3.14.

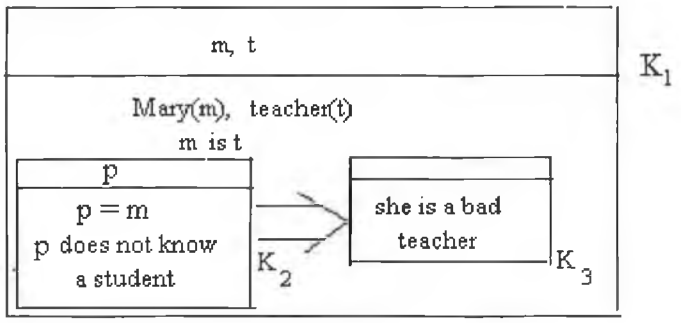


**Fig 3.14 The partial DRS.**

Next a sub-DRS $K_4$ is introduced to $K_2$ as negation is encountered in (11) in the form of *does not know a student*, Fig 3.15. Processing within $K_4$ introduces a new referent and condition for *a student.* The negated sub-DRS $K_4$ is immediately subordinate to $K_2$ and in turn subordinate to K1, introducing another subordination relation $K_4 < K_2 < K_1$. No link can be made between $K_3$ and $K_4$, as they are not involved in the same subordination relation. This means that the referent s is not accessible from $K_3$ the pronoun *she* of $K_3$ cannot equal s. There are now two subordination relation governing referent set formation, $K_3 < K_2 < K_1$ and $K_4 < K_2 < K_1$.

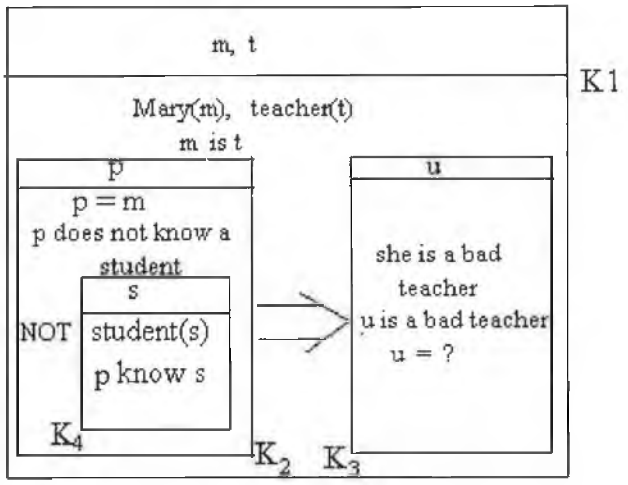


**Fig 3.15 Processing the DRS.**

The DRS $K_3$ contains the sentence *she is a bad teacher.* Processing normally starts by trying to find a suitable referent for the pronoun *she* using the referent set presented by the accessibility rules. This time the author is going to ignore the rules of suitability and accessibility by assuming that any referent can form a link with $K_3$. A referent u is introduced to represent the female pronoun *she*, Fig 3.15. Without the constraints of suitability and accessibility the referent u may equal any other referent i.e. *s, p m* or *t*. Either one of the referents *m* or *t* can be omitted from the set as they are exactly the same, and *p* is also equal to *m*, this leaves the referent set $\{s,m\}$. Since no rules of suitability are to be used, we cannot distinguish between referents, therefore two DRSs must be produced one with $u = s$ as a condition and another with the condition $u = m$. The final indefinite *a bad teacher* introduces a new referent and condition into each of the two DRSs.

We will now examine each of these equality conditions, starting with $u = s$. Semantically it is implausible to let $u = s$ as the reading is counter-intuitive, (11). This condition would not have been admitted by the subordination relation as there is no link between DRSs $K_3$ and $K_4$, where *s* is declared. Reintroducing accessibility and suitability, the subordination relation produces the set *{p,m,t}* but both *p* and *t* equal to *m*, so the set may be reduced to *{m}*. This results in the condition *u = m* which generates a single DRS, as in the final analysis only one referent was suitable.

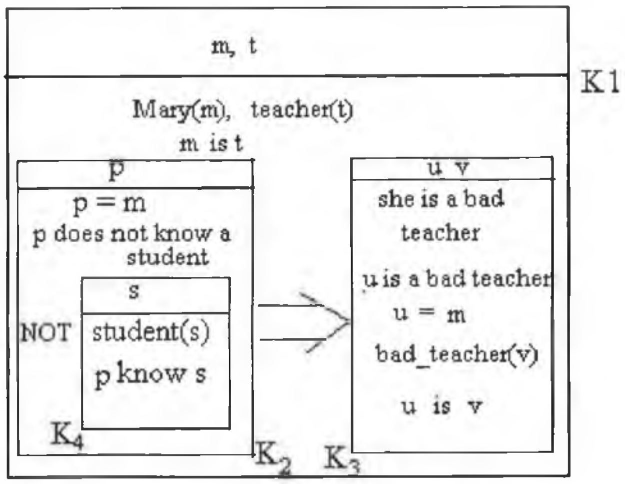


**Fig 3.16 The completed DRS.**

## 3.2.4. Proper Nouns and Sub DRSs.

With the introduction of sub-DRSs processing has become more complicated. Referents may not always be accessible from a particular place in a sub DRS. In certain cases these referents must be made available.

*(12). If a woman loves him, then Pedro courts her.*

This is a conditional sentence which introduces two sub DRSs linked by implication. Part A contains an indefinite NP which is replaced by a referent *w* and condition *woman(w)*. There is also a male pronoun in part A, that introduces a referent x and a condition *x = a suitable referent.* DRS $K_2$ has access to its own referents and those of $K_1$, giving the set (w), but w cannot act as an antecedent for x as they have different genders. At present there is no antecedent available for *x*.

Part B *Pedro courts her* contains a proper noun that is replaced by referent *p* and condition *pedro(p)*. The sentence becomes *p courts her*. The female pronoun introduces a referent *h*, and may choose an antecedent from the referents of $K_3$, $K_2$ or $K_1$ i.e. *{p,w,x}* where *p* is male, *w* is female and *x* is male. The only suitable referent is *w*, this gives the condition *h = w*.

The referent x is a male referent and needs a male antecedent. There is a male referent *p* in $K_3$ which cannot be accessed by $K_2$. The antecedent exists but cannot be accessed from the present position, Fig 3.17

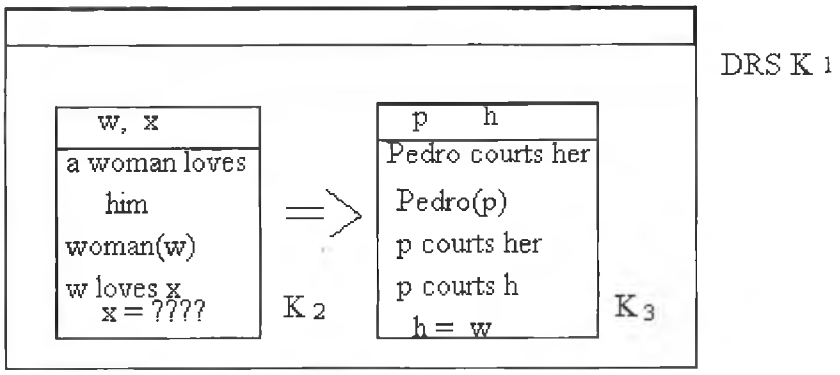


**Fig 3.17 The DRS, with no antecedent for referent x.**

This occurs in many discourses and sentences, as (13) and (14) illustrate.

*(13) George does not like a woman. She loves him.*

The discourse referent for *a woman,* is nested in the negated DRS and cannot act as an antecedent for *She*. Kamp and Reyle ( Kamp & Reyle 1993) assert that in some cases referents and declarations for indefinite NPs should be placed in the principal DRS. Indeed if this was so, a referent would be readily available for the pronoun *She*. The author does not agree with this, if this idea was followed to its conclusion there would be no referents or declarations of any sort remaining in any sub-DRSs. Every declaration would be eventually moved into the principal DRS to cover some unusual sentence structure that could occur.

*(14) If she loves him, Alice will marry Felix.*

Another example is that of (14), where the referents of *she* and *him,* cannot access *Alice* and *Felix* in the consequent DRS. This problem can be solved by putting all conditions and referents of proper nouns, wherever they occur, into the principal DRS so they can be readily accessed from any sub DRS. Kamp and Reyle (1993) assert that in similar situations indefinite NPs can be promoted to the principal DRS. The author does not agree with this ad hoc solution, as what stops us declaring everything in the principal DRS, thereby eroding the whole DRT hierarchical structure? Such a treatment of proper nouns is, however, justified as unlike indefinite NPs, proper nouns are really global variables that refer to their bearer, and will be referred to many times in a discourse, and as such should be placed in the principal DRS.

Applying this solution to the DRS of sentence (12) yields Fig 3.18, with an antecedent readily available for x. When the proper noun *Pedro* is encountered, the referent *p* and condition *pedro(p)* are introduced to the principal DRS. The sentence *Pedro loves her* is changed to *p loves her* in the sub DRS $K_3$ with processing continuing as normal.

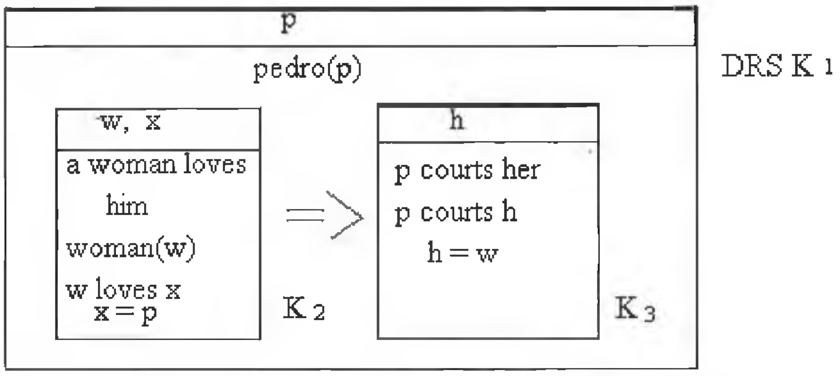


**Fig 3.18 Moving the referents and conditions of proper nouns into the principal DRS**

### 3.2.5. Quantified sentences.

Quantifiers like *most, every, some* and *few* introduce a new sub-DRS template called the duplex condition. This is the condition that will be investigated here, and developed for all quantified sentences. It consists of two set defining expressions in the form of sub DRSs connected by a quantifier and embedded in the main DRS.

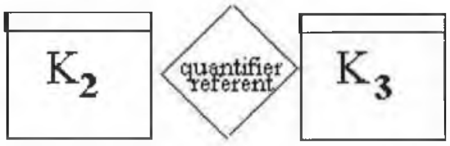
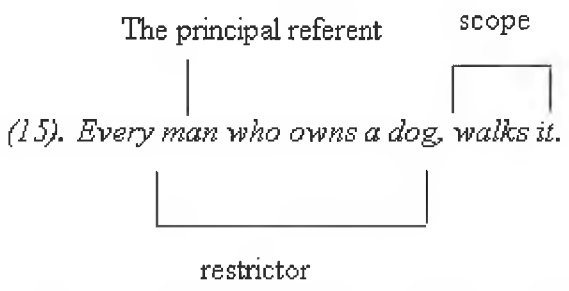


**Fig 3.19 The duplex condition template.**

The DRS $K_2$ contains the restrictor of the quantified sentence, with the scope in $K_1$. The quantifier of the template is the quantifier that is encountered in the sentence, with the referent being the principal discourse referent of the sentence, i.e. the referent that occurs after the quantifier.



In (15), introduction of the duplex condition is triggered by the quantifier *Every.* The restrictor is put in $K_2$ with the scope in $K_3$. The main referent is passed into the scope to form the second sentence *man walks it*, with the referent and condition for *man* are introduced in DRS K1.

The subordination relation is $K_3 < K_2$ implying that the DRS $K_3$ has access to the referents of $K_2$ and the principal DRS the duplex condition is embedded in. DRS $K_2$ has access to its own referents and those of the principal DRS. The partial DRS for sentence (15) is represented in Fig 3.20.

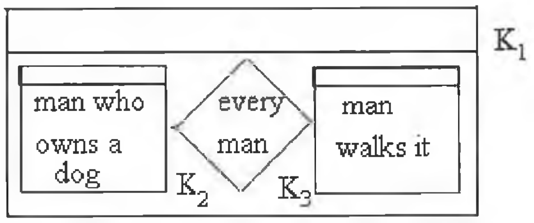


**Fig 3.20 Breaking up the quantified sentence.**

The restrictor is tackled first introducing a referent and condition in $K_2$ for the indefinite *man,* this referent replaces *man* everywhere it occurs in the duplex condition. The indefinite *a dog* introduces a new referent y and condition dog(y). The sentence of $K_2$ is irreducible so processing moves to $K_3$, with the pronoun *it* treated as normal.

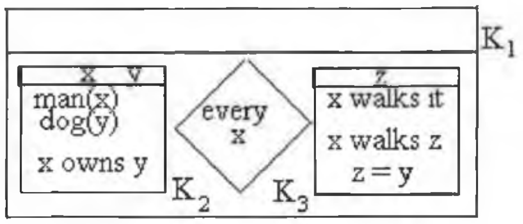


**Fig 3.21 The completed DRS.**

Proper nouns occurring in a quantified sentence are declared in the principal DRS, take for example the sentence (16). The proper noun is declared in the principal DRS. This apparently leaves an empty DRS, Fig 3.22. such a structure is not satisfiable or derivable from the implementation.

*(16). Every John I know is rich.*

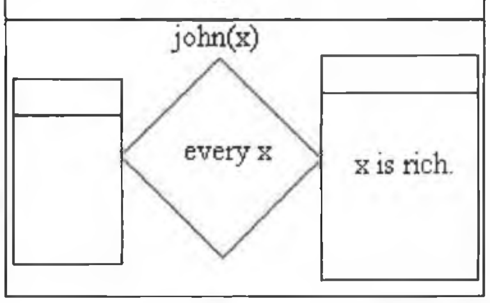


**Fig 3.22 Proper nouns and duplex conditions, (16).**

## 3.2.6 Disjunction.

Disjunction is where one or more clauses of a sentence are joined together with *or*. The function of *or* is to show alternatives, where one clause of the sentence will be true. The clauses of a disjunction are know as disjuncts. There are three different types of disjunctive sentence, all of which will be discussed below.

### 3.2.6.1. Sentential clauses.

*(17). Mary will marry George, or he will leave London.*

In (17), *or* acts as a connector between the two sentences presenting a set of alternatives. The representation of such a sentence must list all the alternatives without making a choice between them. This is achieved using a sub-DRS template which places each alternative or disjunct in a DRS joined to the next alternative with an ∨ indicating that the truth conditions of the sentence either fit the description given by $DRS_1$, or $DRS_2$,....or, $DRS_i$.

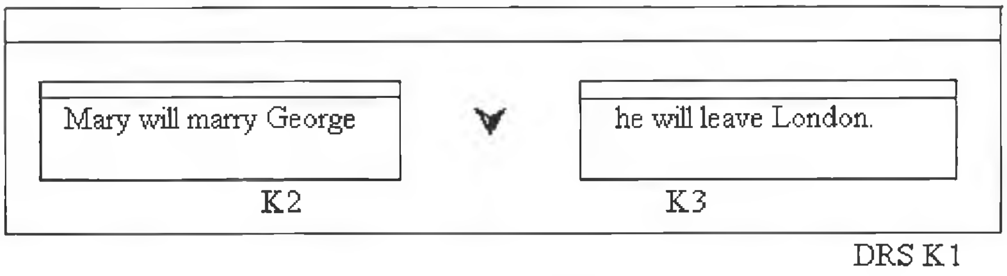


**Fig 3.23 Breaking the sentence into disjuncts.**

The representation of a disjunctive sentence presents the alternatives in DRSs each one joined by a ∨. No disjunct is subordinate to any other disjunct, each alternative is isolated from all the other alternatives, and cannot have access to the referents of any other disjunct. A disjunctive sub-DRS $K_i$ has access to the referents of $Uk_i$ and the referents of the principal DRS, but there is no access between disjunctive DRSs. Proper nouns, wherever they occur, are embedded in the DRS, as stated in 3.3.4. Processing continues as normal within each disjuctive sub-DRS, with the limited set of suitable referents available.

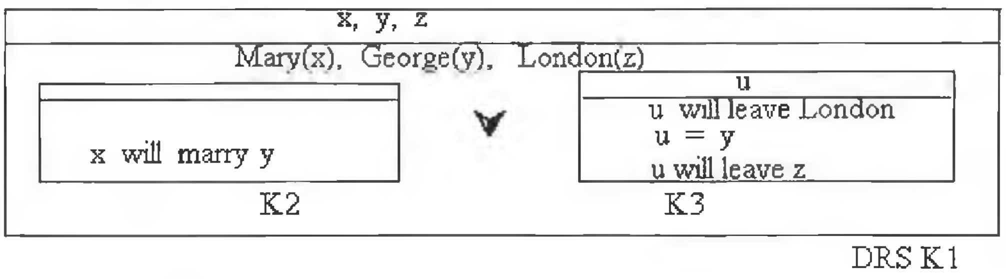


**Fig 3.24 The completed DRS with disjunctive sub DRSs.**

### 3.2.6.2 Non-Sentential Disjunctions.

The only constraint on the constituents that can be combined using *or* is that the constituents should be of the same grammatical category.

**Disjunctive NPs.**

The triggering mechanism for the disjunctive NP construction rule is a list of NPs found in the parse tree as in Fig 3.25 ( Kamp and Reyle 1991).

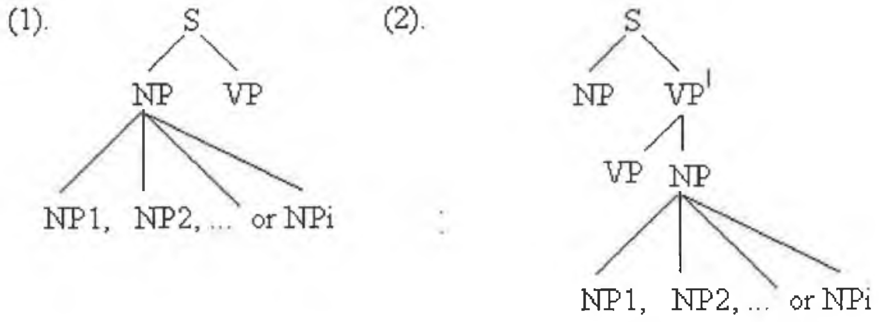


**Fig 3.25 The trigger mechanism for the disjunctive NP construction rule.**

The construction rule for disjunctive NPs states that each NP of parse tree (1) should be re-written and processed as Fig 3.26.

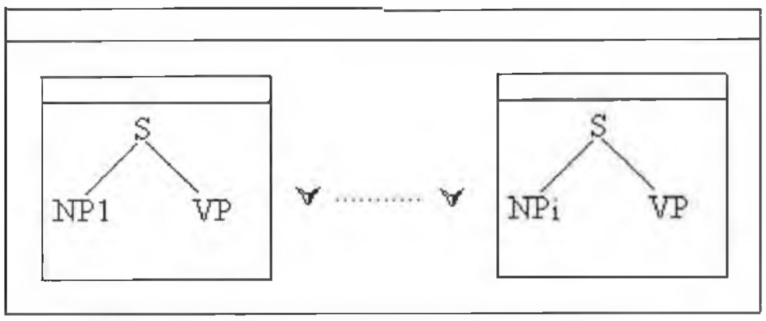


**Fig 3.26 Applying the construction rule to (1).**

*(18). A bag, a book or a record is the present.*

Sentence (18), has the same structure as parse tree (1) of Fig 3.25, therefore processing will commence with the formation of newly synthesised sentences as in Fig 3.27. These sentences are placed in sub-DRSs. The sentences are analysed as normal with no sharing of referents between the sub-DRSs as a sub-DRS $K_i$ has access to the referents of $Uk_i$ and those of the main DRS only.

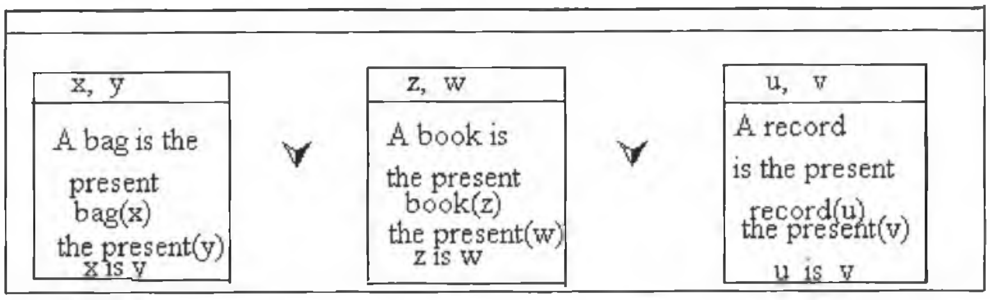

x, y
A bag is the present bag(x)
the present(y)
x is y
z, w
A book is the present book(z)
the present(w)
z is w
u, v
A record is the present record(u)
the present(v)
u is v

**Fig 3.27 The completed disjunctive DRS.**

Sentence (18) gives a list of options for *the present.* The *present* is a unique item, and such uniqueness should be represented in the sub-DRSs. In each of the disjunctive DRSs *the present* introduces a different referent and condition i.e. *present(y), present(w)* and *present(v).* There is no extra condition to state that y,w and v are equivalent, and Kamp and Reyle assert that:

> *It can be easily verified that this has no undesirable effect for the truth conditional content of the resulting DRS: its truth conditions are the same as if we had introduced only one common discourse referent for the two occurrences.* (Kamp & Reyle 1993).

The author accepts that the resulting truth conditions are exactly the same whether different or common referents are used. However, the use of different referents to represent the same object, without the addition of an extra equivalence condition causes great computational problems. How can the computer know that *present* y,w and v represent the same *present* if it has not been explicitly stated?

Adding different referents to each disjunctive DRS to represent the same noun, results in a computational explosion with each referent treated separately in subsequent processing. This duplication is crazy considering the different referents represent the same noun. The strangeness of this approach is further highlighted by sentence (19).

*(19). George, Amanda or Sue know Ada*

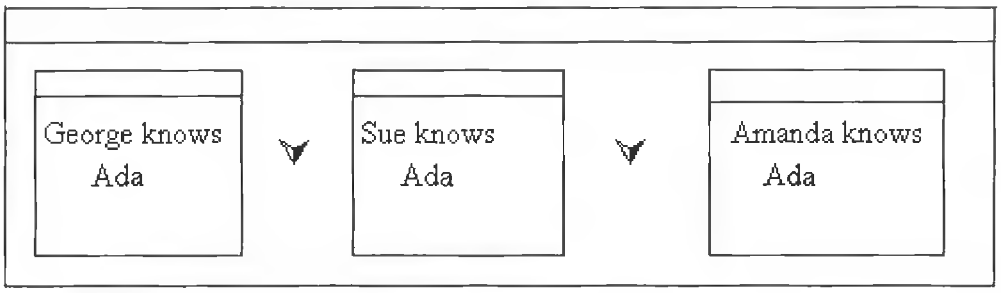


**Fig 3.28 Breaking sentence (17). up into separate sentences.**

The proper noun *Ada* appears in the three sub-DRSs Fig 3.28, each of which will yield its own discourse referent and condition. Producing multiple referents and conditions for a noun has no effect on the truth conditions of a sentence. As for Fig 3.27, the DRS of Fig 3.29 has the same truth conditions as a DRS with a common referent and condition for *Ada*.

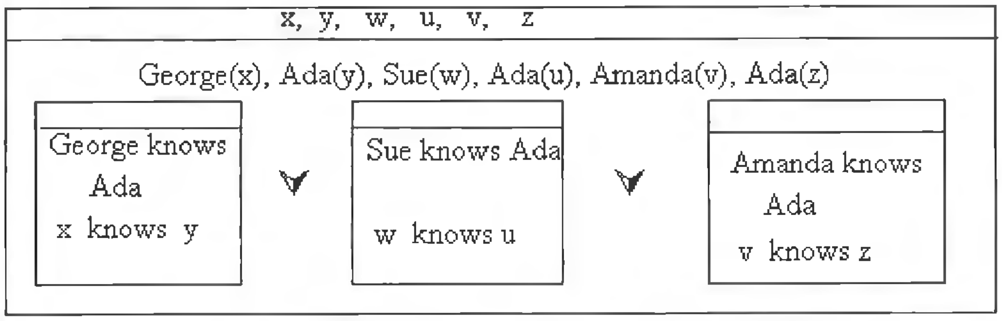


**Fig 3.29 Multiple referents and conditions for the proper noun *Ada*.**

Again the author does not agree with Kamp & Reyle's (Kamp & Reyle 1993) treatment of the non-disjunctive part of these sentences especially when proper nouns are involved. Proper nouns, in this case *Ada*, refer to their bearer the same unique bearer in each case. A unique referent and condition should therefore be introduced to represent a proper noun. A possible solution is to analyse the non-disjunctive part of disjunctive sentences before the synthesised sentences are formed, thereby eliminating duplication. Applying this to sentence (17), the existing sentence is changed by construction rules to form a new sentence which the synthesised sentences are formed.

This non-disjunctive VP forms the main part of the synthesised sentence

*(18). A bag, a book or a record is the present.*

Construction rules introduce a new referent *x* and condition *present(x)* for the present changing (18) to (20).

(20). *A bag, a book or a record is x.* *the present(x).*

The new sentences are synthesised and introduced to disjunctive sub-DRSs with the condition *the present(x)*, Fig 3.30. Processing continues as normal within the sub-DRSs.

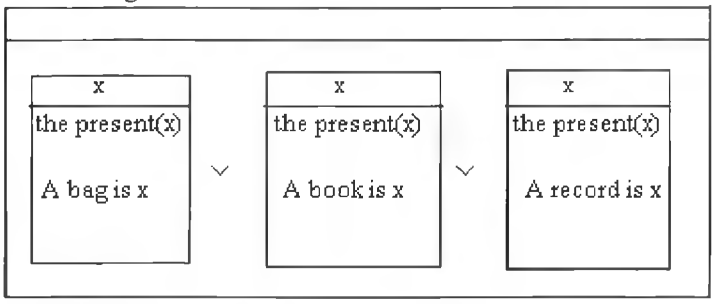


**Fig 3.30 Introducing a common referent for the non-disjunctive NP.**

A similar approach can be taken for proper nouns (20), with the discourse referent and condition introduced to the main DRS, and the original sentence changed accordingly. Parse tree (2) in Fig 3.25, is processed from left to right as normal, with the list of NPs introducing the sub-DRSs when encountered.

*(21). George lives in Durban, London or Sydney.*

Sentence (21) introduces a new referent *x* and condition *George(x)*, changing the sentence to *x lives in Durban, London, or Sydney.* The verb *owns* is left untouched. The list of NPs *Durban, London, or Sydney* triggers the introduction or sub-DRSs with synthesised sentences of the form of Fig 3.31 present in each DRS.

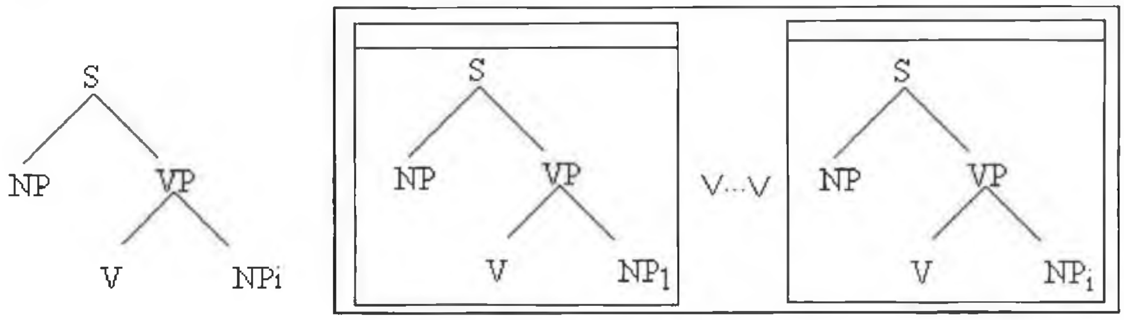


**Fig 3.31 The parse tree of the newly constructed sentences, and the DRS that results from applying the construction rule to (2).**

In Fig 3.31 $NP_i$ stands for each of the elements in the NP list. Following this procedure with sentence (21) produces a new DRS as in Fig 3.32. As usual the referents and conditional declarations of proper nouns *Durban, London* and *Sydney* are placed in the main DRS.

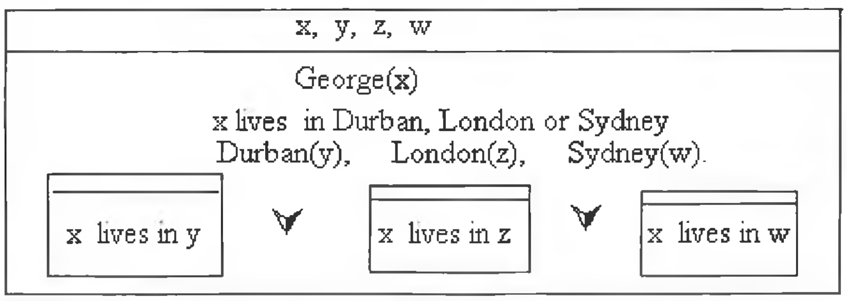

x, y, z, w
George(x)
x lives in Durban, London or Sydney
Durban(y), London(z), Sydney(w).
x lives in y
∨
x lives in z
∨
x lives in w

**Fig 3.32 Processing a sentence with disjunctive NPs in the VP.**

## Other types of disjuncts.

The trigger parse tree of the final type of disjunct is in Fig 3.33, where X is not an NP. The tree of sentence (22) contains this trigger.

*(22). John loves or hates Judy.*

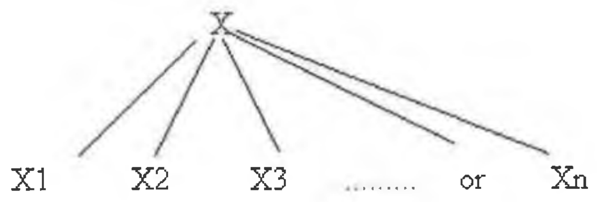


**Fig 3.33 A disjunctive parse tree, where X is not an NP.**

This parse tree introduces a set of sub DRSs each of which contains a synthesised sentence of the form of Fig 3.34.

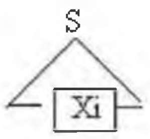


**Fig 3.34. The synthesised sentence that goes into each of the sub DRSs**

Processing of sentence (22) is as normal introducing a new referent and condition for *John*, with the sentence becoming *x loves or hates Judy.* The disjunctive trigger rule is fired by the verb list *loves or hates*, introducing the sub DRSs with the newly synthesised sentences. The author chooses to introduce only one referent and condition for Judy as discussed in the preceding section, y and Judy(y), changing the sentence to *x loves or hates y.* Processing continues as normal within the sub DRSs, Fig 3.35

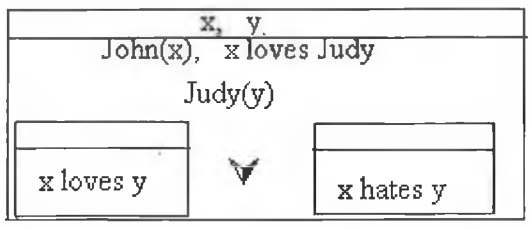


**Fig 3.35 The completed DRS.**

The revised construction rule for disjunctive NPs proposed by the author will be used extensively in this thesis. This rule analyses the non-disjunctive part of a sentence only once, this partial analysis is then used in the generation of the synthesised sentences for each disjunctive DRS (illustrated in examples of this chapter). This approach avoids the production of more than one referent for the same entity, which causes problems computationally.

### 3.2.7 Conjunction.

Conjunction is where one or more clauses of a sentence, are joined together with *and*. There are two types of conjunctive clause, sentential and constituent clauses. These may be found nested in larger sentences or may in some cases form a sentence.

#### 3.2.7.1 Sentential clauses.

*(23). Anne is going to the shop, Jill is going to the school and Ada is cooking at home.*

Sentence (23). contains sentential clauses, and is made up of three different sentences. The truth conditions for these type of sentences are simple, the sentence made of clauses $S_1$,$S_2$,...and $S_x$ is true if each one of the sentences $S_1$,$S_2$ ....$S_x$ are true. The construction rule proposes the processing of each sentence, starting with $S_i$ and i = 1. Care must be taken to fully process each sentence $S_i$ before moving onto the next conjunct $S_{i+1}$, in order to avoid the predicament of anaphoric pronouns in $S_i$ being allowed to refer to referents of $S_j$ ( where $j > i$ ) as sentence (24) illustrates.

*(24). He bought a book, Sue left the shop and George waited outside.*

*(25). George bought a book, he waited outside the shop, and Sue left soon after.*

In (24), the pronoun *He* cannot be allowed refer to *George* as the meaning of the sentence would be completely lost. To avoid such referencing, conjunctive sentences are processed one at a time with clause 1 fully processed to irreducibility before clause 2 is considered. The overall sentence is broken into component clauses, with each numbered increasingly with processing starting at clause 0. Clause 1 is only considered when clause 0 has been fully processed. Sentence (25) illustrates the point that *George,* in this case, can act as antecedent for *he.* Antecedents can be found readily from earlier conjuncts, but no linkage can be allowed between pronouns of an earlier conjunct and referents of a later conjunct. Breaking sentence (22). up into its respective clauses, and numbering each clause increasingly yields the DRS of Fig 3.36. The next step in the construction process involves the normal processing of each clause one by one, only considering clause $_{i+1}$ when clause $_i$ has been fully processed.

| |
|---|
| < Anne is going to the shop 0 > |
| < Jill is going to the school 1 > |
| < Ada is cooking at home 2 > |

**Fig 3.36 Breaking sentence (19) into clauses.**

Processing begins with clause 0. Proper noun *Anne* introduces a new referent and condition, the definite NP introduces a new condition and referent (the final step of definite NP rule is ignored as no suitable referent is readily available). This clause is now irreducible, *i* is incremented and processing moves to clause 1.

| a, s |
|---|
| anne(a) |
| the shop(s) |
| a is going to s |
| { processing moves to clause 1} |
| < Jill is going to school 1 > |
| < Ada is cooking at home 2 > |

**Fig 3.37 Processing the clauses.**

When clause 1 is fully reduced, processing moves onto the final clause 2 giving the complete DRS of Fig 3.38

| a, s, j, x, y |
|---|
| anne(a), |
| the shop(s) |
| a is going to s |
| jill(j), the school(x) |
| j is going to x |
| { process the final clause } |
| ada(y) |
| y is cooking at home |

**Fig 3.38 The completed DRS for a conjunctive sentence with sentential clauses.**

The numbering of clauses works well with simple conjuncts. Unfortunately things are not always as straightforward, as sentence (26) from Kamp and Reyle (1993) illustrates. In this sentence the conjunction is part of a larger conditional clause.

*(26). If Mary likes him and she owns a tape recorder, Fred will get it.*

The anaphoric pronoun *him* of clause 1 must be linked to a suitable referent of the discourse. A suitable referent is available in the consequent of the conditional, but at present is inaccessible to the anaphoric pronoun of the conjunctive condition, as conjunctive clauses must be processed in order and before anything else.

The DRS construction rule must be changed at this point to introduce DRS conditions ծ, with indices for each clause of a conjunction, and Ø markers for any other non conjunctive clause; indicating that these Ø clauses may be processed at any time. Given this, we are able to infer that *him* refers to *Fred* in (26) as the DRS of Fig 3.39 - 3.40 taken from (Kamp and Reyle 1993) implies.

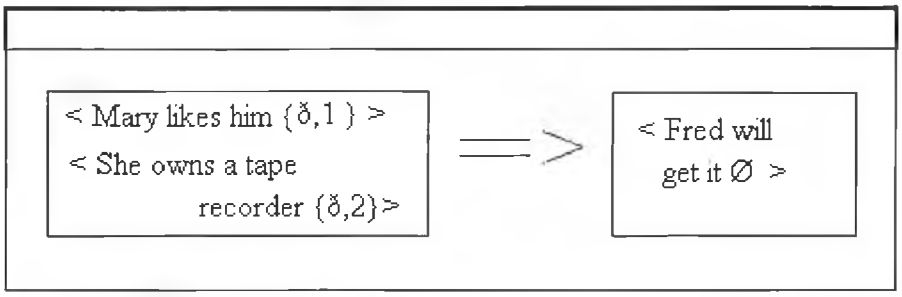


**Fig 3.39 Breaking the sentence up into its respective clauses, numbering each.**

Processing may start with: the first clause {ծ,1}.
the consequent the Ø clause.
or part of either clauses.

Ø clauses may be analysed at any time during the construction process, making the referents these clauses introduce available within the respective sub-DRS or main DRS. The male pronoun of {ծ,1} requires a male antecedent which is not available at present. A Ø clause may be able to provide the needed antecedent so partial processing starts on the only Ø clause available yielding Fig 3.40. The referent and condition of the proper noun *Fred* are as always put in the main DRS, unlike other referent types which are always declared in the DRS in which they are encountered. Clause {ð,1} can now readily access the referent y.

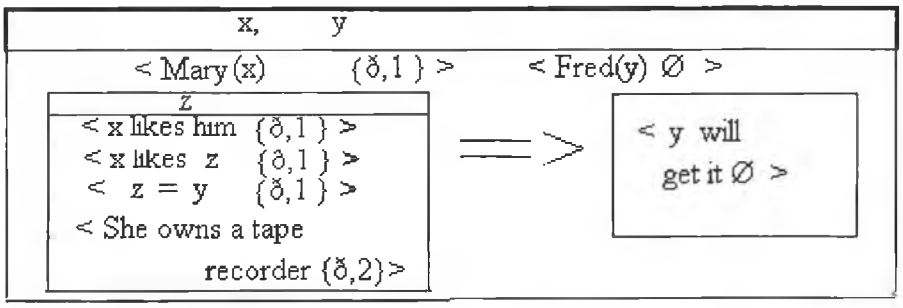


**Fig 3.40 Processing a Ø clause, produces an available suitable referent for *him*.**

Clause {ð,1} is now irreducible, so processing continues as normal incrementing the clause count moving onto {ð,2}. The new DRS construction rule proposes a processing of sentences, with non conjunctive clauses labelled as Ø clauses with conjunctive clauses labelled, numbered and processed incrementally. Sentence (23) will be broken into clauses, each one numbered and labelled with DRS clauses ð to show it is a conjunctive sentence. Processing will then continue as shown in Fig 3.36 - 3.38 with the added DRS condition ð.

### 3.2.7.2 NP clauses

*(27). John, Sue, George and Mary play tennis.*

This sentence has two meanings both of which are handled by DRT, the first one (discussed in this section) that John, Sue, George and Mary are tennis players and the second one, which will be investigated in section 3.4.1.1, that they play tennis with each other. This second type of conjunctive clause is made up of a series of NPs, $NP_1$,$NP_2$, $NP_3$,....and $NP_i$ with a parse tree of the form:

```
                    S
               /         \
            NP            VP
            /
NP1,NP2,NP3,....and NPi
```

As shown above the construction process demands that clauses should be broken up, numbered, labelled and processed incrementally, i.e. $NP_1$ VP, $NP_2$ VP,...$NP_i$ VP incrementing i as before.

| |
|---|
| < John plays tennis {ð, 1}> |
| < Sue plays tennis {ð, 2}> |
| < George plays tennis {ð, 3}> |
| < Mary plays tennis {ð, 4}> |

**Fig 3.41 Separating the NP clauses.**

There are no Ø clauses present in sentence (27), so processing continues within each clause i until that is irreducible when processing moves onto the next clause i + 1. The sequential information {ð, i} in the DRS, can be dropped when processing is complete.

| x, y, z, w |
|---|
| < John plays tennis {ð, 1}> |
| John(x) |
| < x plays tennis {ð, 1}> |
| < Sue plays tennis {ð, 2}> |
| Sue(y) |
| < y plays tennis {ð, 2}> |
| < George plays tennis {ð, 3}> |
| George(z) |
| < z plays tennis {ð, 3}> |
| < Mary plays tennis {ð, 4}> |
| Mary(w) |
| < w plays tennis {ð, 4}> |

**Fig 3.42 The first reading of sentence (26).**

A mixture of disjunction and conjunction within a sentence can be handled easily by DRT. The respective construction rules are applied one at a time, until the conditions within the DRSs are irreducible.

## 3.3. The Plural - Indefinite, Definite and Quantified Plurals.

*Any natural language processing system will be severely limited in its performance unless it incorporates a systematic way of dealing with plurality.* (Link1983)

Although Link (1983) may overstate the need for a systematic plural treatment, the DRT plural proposal is based on Link's lattice theory approach (Link 1983), and is particularly concerned with plural pronominal anaphora. This improves on the Predicate Calculus approach which maps single objects to individuals, and sets of objects to predicates. Lattice theory uses lattices and rules of algebra. A lattice L is a non-empty set, which has two binary operations defined across it *join* and *meet* such that all elements of L are:

| | | |
|---|---|---|
| *Associative*: | x and ( y and z ) = ( x and y ) and z. | |
| | x or ( y or z ) = ( x or y ) or z. | |
| *Commutative*: | x and y = y and x, | x or y = y or x. |
| *Idempotent*: | x or x = x | x and x = x |
| *Absorptive*: | x = x or ( x and y ) | x = x and ( x or y ) |

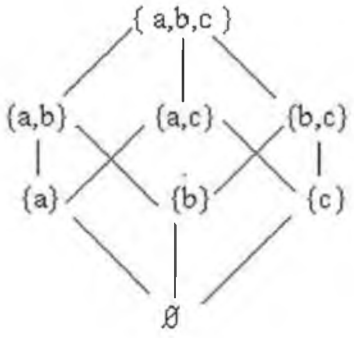


**Fig 3.43 Lattice < L, ⊆>**

The lattice L presented in Fig 3.43 has upward arcs that represent the subset relation, $\subseteq$ e.g. {a} $\subseteq$ {a,b}, {b,c} $\subseteq$ {a,b,c} and so on. Plural NPs denote sets of individuals, and are represented in the domain of individuals, which is modelled as a semi-lattice with the join operation. e.g. Fig 3.43 illustrates the possible set combinations and permutations found in the domain of individuals *a, b* and *c*. A semi-lattice is a non-empty set equipped with one binary operation, all elements of which are commutative, associative and idempotent. The author will explain how lattice theory is incorporated in DRT in the next sections.

Until now the only referent available has been the singular or atomic discourse referent. The non-atomic discourse referent represented by a capital letter is now introduced to represent plurals and sets of objects. These referents may be introduced by plural NPs e.g. ***the men, the televisions***, that introduce a non-atomic discourse referent with a corresponding condition. The plural NP *the men* introduces a new referent *X* and condition *the men(X)*, with the existing sentence changed to replace *the men* with *X*.

Plural pronouns also introduce non-atomic discourse referents. Discourse referents that serve as antecedents for singular pronouns are typically to be found among those already present in the DRS when the pronoun is encountered. This is not the case with plural pronouns and their antecedents, whose referents do not exist already in the DRS and need to be "synthesised". Plural pronouns do not have one single referent acting as an antecedent but a set of referents. Existing discourse referents are used to synthesise the set, which is given the name of a newly introduced non-atomic discourse referent. In this way the elements of the set that the plural or non-atomic referent represents may be gathered together.

Kamp and Reyle (1993) assert that within strict DRT, non-atomic discourse referents should be accompanied by the condition *non-at(discourse referent)* and represented by a capital letter, with atomic discourse referents represented by the condition *at(discourse referent)*. They also assert that on occasion these extra conditions may be omitted, with non-atomic referents represented by a capital letter, and atomic referents represented by a small letter. The author agrees with this omission, and asserts it should be permanent as the extra conditions are superfluous, and within a computational environment could lead to confusion.

### 3.3.1 Methods of set formation.

There are many different methods of set formation, of which summation and abstraction are two. Representing plural pronouns, e.g. *they* and *them*, involves the introduction of new non-atomic discourse referents. The referents must find antecedents from the set of available referents of the DRS. A non-atomic discourse referent can only have another non-atomic referent as an antecedent. This antecedent is generated by set formation techniques that introduce a new set which is made equal to the non-atomic discourse referent introduced by the plural pronoun.

#### 3.3.1.1 Summation and its application to plural NPs.

Link (Link 1983) presents a semantics of plurality, treating groups as distinct individuals in a lattice model instead of sets, and assumes there is a structural domain of discourse

> *where apart from regular atomic individuals there are complex objects or individual sums ( i-sums ) serving as denotations for plural expressions.*
>
> (Cooper *et al* 1994)

This i-sum or least upper bound (lub) is an operation on two individuals of a lattice. It is the union of the individual denotations into a new denotation representing the group e.g. in Fig 3.43, lub( {a,b}, (b,c} ) = { a,b,c }. Taking another example, imagine *Sally* and *John* are nodes of a lattice, then the i-sum of these two individuals is the new denotation of the union of *Sally and John.* Link proposes sufficient conditions to allow the interpretation of individuals as a collection by algebraically combining the denotations of the said individuals.

This operation is known as summation in DRT, and can be applied at any construction stage where there exist discourse referents that can be summed together. The method combines the referents of a number of different NPs into a single set, giving this set the name of a newly introduced non-atomic discourse referent e.g. $Z = u \oplus v$, where u and v are previously defined atomic discourse referents, and Z is the newly synthesised non-atomic discourse referent. We will discuss how summation is applied to sentences (27) and (28).

*(27). John, Sue George and Mary play tennis*

The construction rule leading to Fig 3.42 accounts for the reading of sentence (27) as an action each individual participates in on their own. There is however another reading, that which states that *John, Sue, Mary and George* are playing tennis with each other, i.e. the set of NPs are performing the action. A set consists of a group of objects and must be represented by a plural (or non-atomic) discourse referent. Summation combines the referents of a DRS into a set giving it the name of a new non-atomic discourse referent.

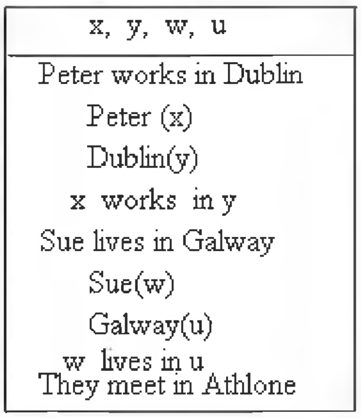
| x, y, z, w, Z |
|:---:|
| < John plays tennis {ð, 1}> |
| John(x) |
| < Sue plays tennis {ð, 2}> |
| Sue(y) |
| < George plays tennis {ð, 3}> |
| George(z) |
| < Mary plays tennis {ð, 4}> |
| Mary(w) |
| $Z = x \oplus y \oplus z \oplus w$ |
| Z play tennis |

**Fig 3.44 An application of Summation - introducing the second reading of (27).**

The construction rule for conjunctive NPs allows the interpretation of the NPs collectively as a set, but does not force this interpretation on us. The referents of the conjunctive NP can be directly copied into a summation set which is named after a new non-atomic discourse referent i.e. $Z = x \oplus y \oplus z \oplus w$ where $x$ is *John*, $y$ is *Sue*, *George* is $z$, and $w$ is *Mary;* the respective elements of the conjunctive NP. The action of playing tennis is now performed by the set Z, Fig 3.44. Any subsequent sentence containing a plural NP, and possibly requiring a non-atomic antecedent e.g. *they enjoy themselves,* will find one readily available in Fig 3.44.

*(28). Peter works in Dublin. Sue lives in Galway. They meet in Athlone.*

Conjunctive clauses are not always present in discourse to provide a ready made set. The summation construction rule presented by Kamp and Reyle (Kamp & Reyle 1993) does not give any guidelines as to what type or how many referents should be summed to form the summation set. They simply assert that summation can be applied at any stage to group more than one referent together. The number of referents involved in any one grouping is at the implementor's discretion, this produces an unsatisfactory computational algorithm as (28) will illustrate.



x, y, w, u

Peter works in Dublin
Peter (x)
Dublin(y)
x works in y
Sue lives in Galway
Sue(w)
Galway(u)
w lives in u
They meet in Athlone

**Fig 3.45 The partial DRS of (28).**

The first two sentences of the small discourse of (28) introduce the new referents *x,y,w,u* and corresponding conditions, Fig 3.45. The third sentence of (28) contains the plural pronoun *They* which introduces a new non-atomic discourse referent *V*, and the condition *V = a suitable non-atomic discourse referent* Fig 3.46. At present there is no other non-atomic discourse referent present in the DRS so one must be synthesised from the existing atomic referents of the DRS.

There are four referents *x, y, w* and *u* representing *Peter*, *Dublin*, *Sue* and *Galway* in the DRS. Summing them together results in $Z = x \oplus y \oplus z \oplus w$ where *Z* is a possible non-atomic discourse referent formed by summation. Allowing *They* to be represented by the non-atomic set Z of *Peter ⊕ Dublin ⊕ Sue ⊕ Galway* makes no sense, especially since the set representing *They* should intuitively be a set of humans i.e. *Peter ⊕ Sue*.

Outside the conjunctive clause framework, Kamp and Reyle (Kamp & Reyle 1993) do not provide us with the tools to discriminate between referents so that set inclusion or exclusion can be established. The author proposes building this into DRT, with summation being applied to form sets of two types: human and non-human referents, thereby solving the problem of set formation. The production of a human and non-human sets by summation will account for a broad band of examples.

The author proposes that a single application of summation should produce the two sets, with each set type incorporated into a DRS. If a DRS contains non-human and human referents, at its simplest a single application of summation will result in two different DRSs being developed. The two set groupings of discourse (28) are *Galway* ⊕ *Dublin* and *Peter* ⊕ *Sue*, which one will we choose to represent the plural pronoun *They* ? The author asserts that both sets should be incorporated into DRSs, thereby producing two different DRSs for (28) both shown in Fig 3.46.

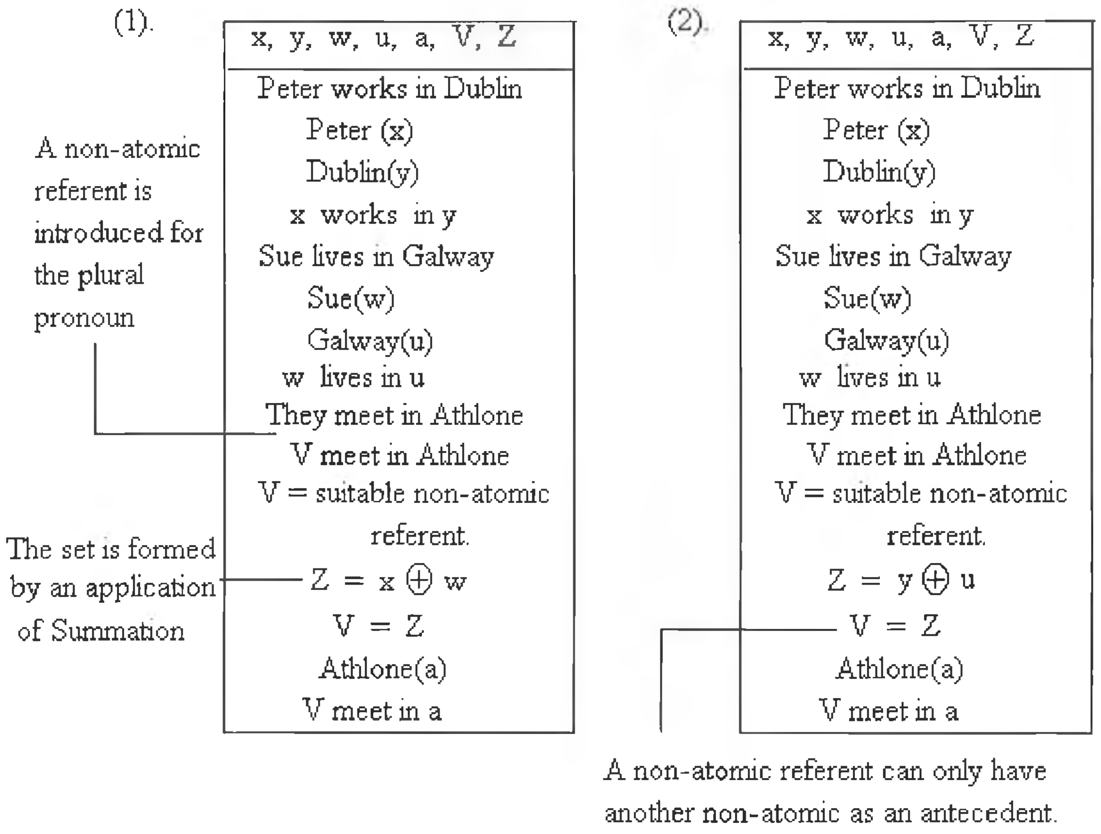


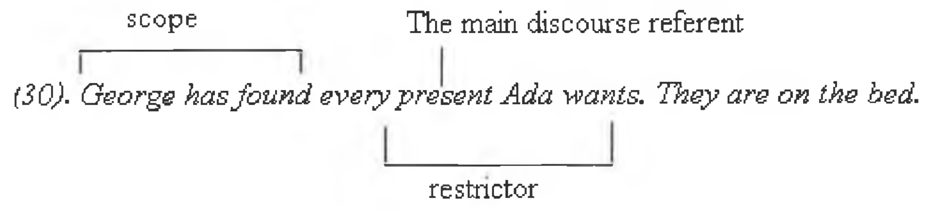
**Fig 3.46 Finding the antecedent of a plural pronoun using summation.**

Although DRS (2) of Fig 3.46 is implausible in this case, as how can *Galway* + *Dublin* meet in *Athlone?,* development of the DRS will continue within the implementation until a failure is encountered e.g. failure to find a suitable referent is encountered, or the DRS is outputted as a possible DRS. Indeed changing sentence (28) slightly leads to a plausible reading for both sets *W* of Fig 3.46, (29):

*(29). Peter works in Dublin. Sue lives in Galway. They are beautiful.*

### 3.3.1.2 Abstraction and its application to plural pronouns.

Abstraction works on duplex conditions (introduced in section 3.3.5), to form sets. This method unites the restrictor and scope of a duplex condition into one set, so that the set of objects in this set can be readily referred to.



*(30). George has found every present Ada wants. They are on the bed.*

The first sentence of (30) is represented by the duplex condition of Fig 3.47.

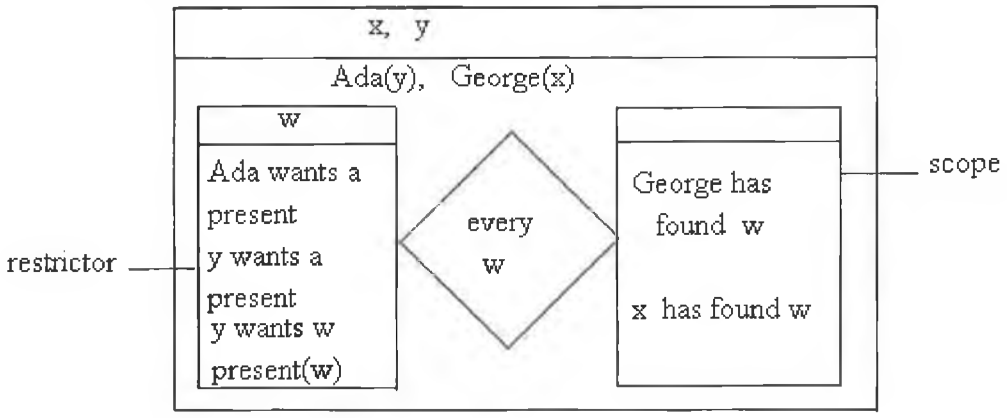


**Fig 3.47 The DRS for sentence (1), of (28).**

Moving onto the second sentence of (30), a non-atomic referent *T* is introduced to represent *They*, an antecedent for which must now be found. *They* can represent *every present Ada wants and George has found.* Summation cannot generate this meaning, as no union of referents would let *they* refer to *every present Ada wants and George has found.* A duplex condition cannot be used either, as there is no quantifier in the second sentence of (29). Abstraction is applied, forming the new set of objects by uniting the referents and conditions of the restrictor and scope of a duplex condition. A sigma sign is put outside this union to represent the sum of all such conditions together with a referent chosen from the new set of referents, this set is then given the name of a new non-atomic discourse referent, as in Fig 3.48

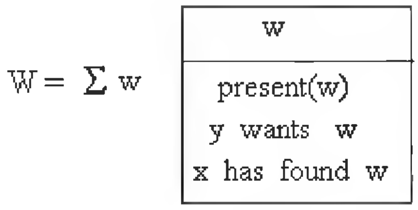


**Fig 3.48 Applying abstraction to the duplex condition of Fig 3.46**

The condition of Fig 3.48 says that the newly introduced discourse referent W stands for the set consisting of all individuals w which satisfy the conditions of the DRS beside the summation sign. Abstraction has made a non-atomic discourse referent available to the plural pronoun *They*, with the condition *T = W* linking the anaphor and antecedent. Processing continues as normal with the introduction of a new referent and condition for the definite NP *the bed.*

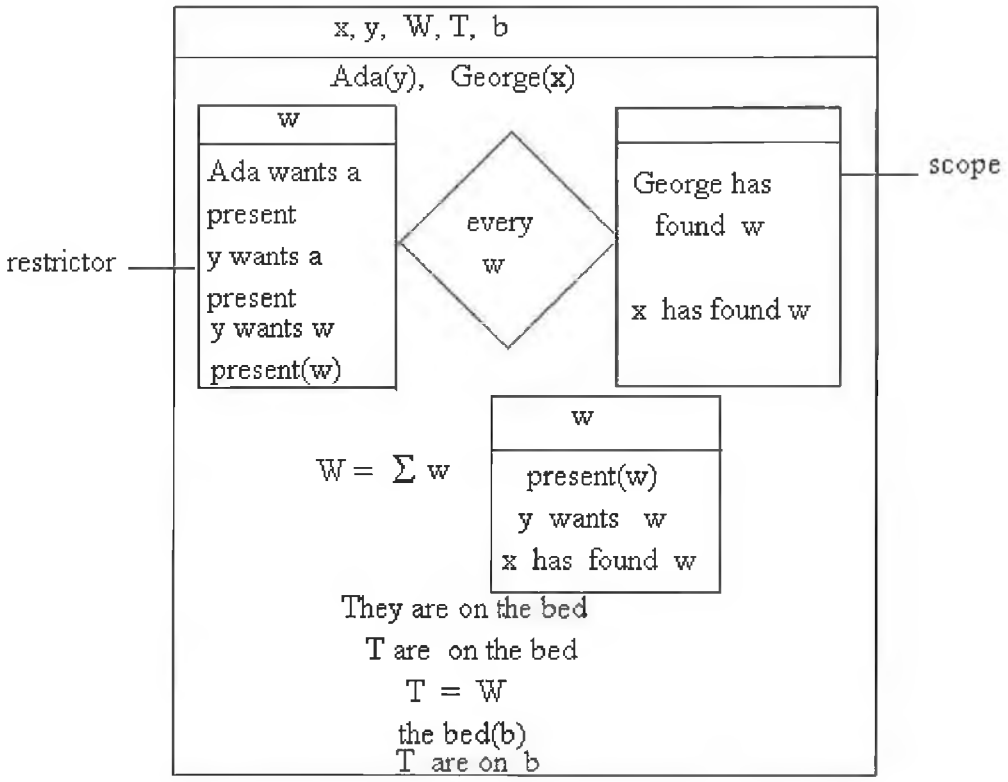


**Fig 3.49 The completed DRS with an antecedent formed by abstraction.**

Another reading of (30) is equally likely, this is the reading supplied by summation which sums all referents not involved in the duplex condition together. As discussed in the

3.4.1.1, summation sums human and non-human referents into two sets. In this case there are no non-human referents available outside the duplex condition, and therefore no non-human summation set available. The only summation set available is the human summation set which is $W = y \oplus z$, and produces the DRS of Fig 3.50.

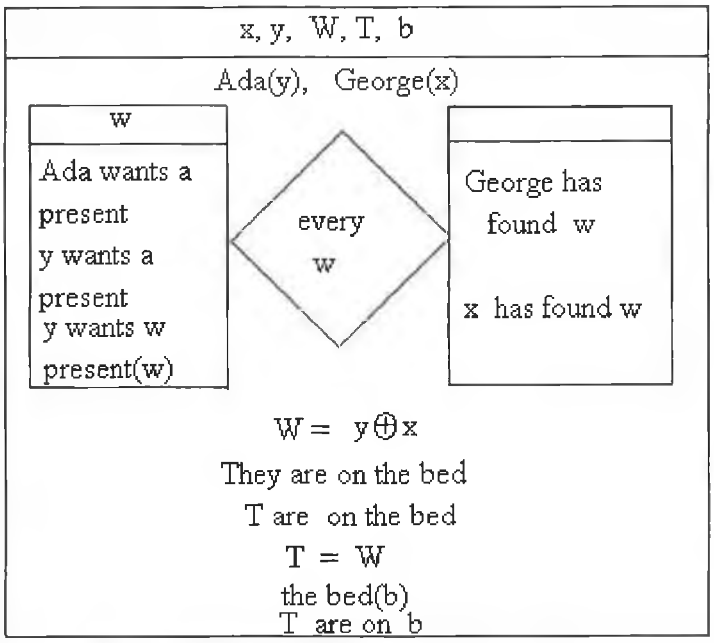


**Fig 3.50. Producing a non-atomic referent *W* by summation.**

### 3.3.2 How many meanings can a sentence have?

Sentences containing plural NPs are often ambiguous, leading to many different interpretations. This problem has been tackled by the star convention of Kamp and Reyle (Kamp & Reyle 1993) which treats a predicate with a star superscript (e.g. lawyer* ), as a predicate that can refer to both singular and plural objects (i.e one lawyer or a group of lawyers). The star convention is not discussed in this thesis as it is not included in the implementation.

These different approaches lead to many divergent representations of the same sentence, leading to a combinatorial explosion as every possible representation of every sentence of the discourse is produced. Two such representations are the collective and distributive readings of a sentence, which may be obligatory or optional in either case. Sentences (31) and (32) have both collective and distributive readings which will be discussed in the upcoming sections.

*(31). The women bought a bouquet of flowers.*

*(32). The lawyers hired a secretary* (Kamp and Reyle 1993).

All interpretations of sentences containing plural NPs begin with the introduction of a non-atomic discourse referent to represent the NP. The decision as to which reading to develop is taken from here, Fig 3.51 (a) & (b). For sentence (31), the definite plural NP introduces the condition, the *women(W)* which uses a non-atomic referent. Sentence (31) is changed accordingly showing the action the group performed *W bought a bouquet,* Fig 3.51 (a).

(a).

| W, f |
|---|
| the women(W) |
| W bought a bouquet of flowers. |

(b).

| W, f |
|---|
| the lawyers(X) |
| X hired a secretary. |

**Fig 3.51 Representing the plural NPs of sentences (31) and (32).**

### 3.3.2.1 The Collective Reading.

The author has already discussed collective readings formed by the application of summation. We will now examine the collective reading of a sentence containing plural definite NPs. The collective reading of a sentence is the reading that takes any action being performed as a group action.

The collective reading of sentence (31) is that which states that *the women* as a group, participated in the action of buying one *bouquet of flowers*. Collectively *the women* bought the *bouquet.* This type of set can be represented by an exhaustive conjunction of the individuals of the set i.e. by listing all the women in the group, but this is a tiresome approach. A collective reading is simply represented by introducing a single non-atomic discourse referent to represent the set as achieved in Fig 3.51 (a). Processing within the

DRS continues as normal with the indefinite *a bouquet of flowers* introducing a new referent, condition and changing the sentence to its irreducible form *W bought f,* Fig 3.52.

| W, f |
|---|
| the women(W) |
| W bought a bouquet of flowers |
| bouquet of flowers(f) |
| W bought f |

**Fig 3.52 A DRS representing the collective reading of (30).**

Collective readings may be optional or obligatory, but may not be represented by a duplex condition. as the main discourse referent in a duplex condition is atomic whereas it is non-atomic in a collective reading. Extending the DRS of Fig 3.51 (b) to produce the collective reading of sentence (32) results in Fig 3.53.

| X, y |
|---|
| the lawyers(X) |
| X hired a secretary |
| secretary(y) |
| X hired y. |

**Fig 3.53 A DRS representing the collective reading of (31).**

### 3.3.2.2 The Distributed Reading.

The distributive reading represents the case where every member of the set performs the same specified action. In the case of (32), the distributive reading states that each lawyer out of the set of lawyers hired a secretary of his own. Distributive readings are represented using duplex conditions. The set of individuals or entities is declared in the DRS first, then within the duplex condition a member x is chosen to distribute over the set. The scope of the duplex condition states the action that each member x of the group performed on his own.

The first step in the construction process is the introduction of a non-atomic referent to represent the plural NP as achieved by Fig 3.51. The distributive reading uses the duplex condition template, the contents of which will differ according to whether the NP is quantified or not. In Link's lattice theory [Link 1983], a distributive function D is applied to the sentences to produce the distributive representation.

This distributive reading represents the case where every member of the set performs the same specified action. The DRS template for the distributive reading of a sentence is shown in Fig 3.54.

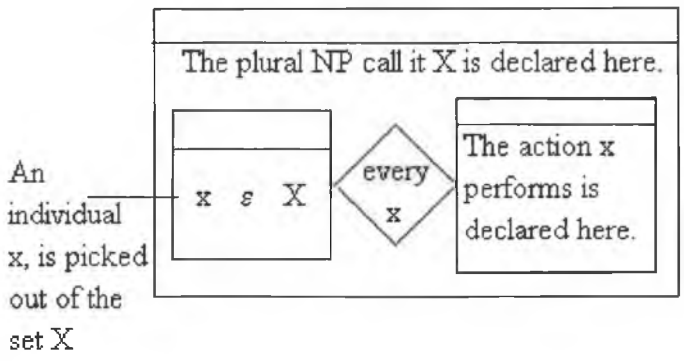

The plural NP call it X is declared here.
An individual x, is picked out of the set X
x ε X
every x
The action x performs is declared here.

**Fig 3.54 DRS template for a distributive reading.**

The general format for the production of a DRS to represent the distributive reading of quantified NPs starts (as with collective readings) with the declaration of the set, call it *X*, representing the plural NP in the main DRS. The scope of the duplex condition picks an individual *x* out of the set *X*, such that every such *x* performs the action stated in the restrictor. Analysis continues as normal within the restrictor.

In the case of (32), the distributive reading states that every lawyer out of the set of lawyers *X*, *hired a secretary*. Each member *x* of the set *X* performed the action of hiring a secretary. The restrictor of the duplex condition picks the member *x* out of the set *X*, with *every x* performing the action specified in the scope i.e. *x hired a secretary*, Fig 3.55.

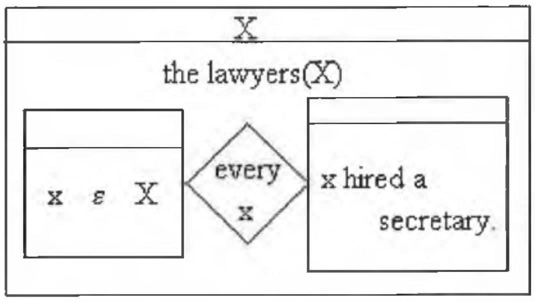


**Fig 3.55 Developing the distributive reading of sentence (32).**

Following the same process, the initial DRS representing the distributive reading of sentence (31), is shown in Fig 3.56

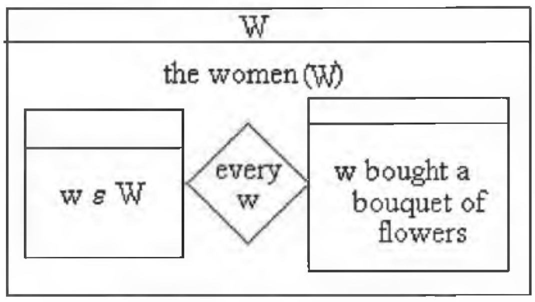
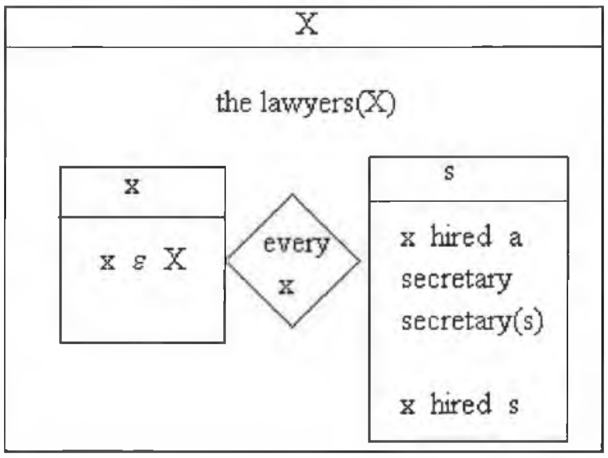


**Fig 3.56 Developing the distributive reading of sentence (30).**

Processing continues as normal within the scope of Fig 3.55 and 3.56 producing the complete DRSs of Fig 3.57 (a) and (b).

(a)



(b).

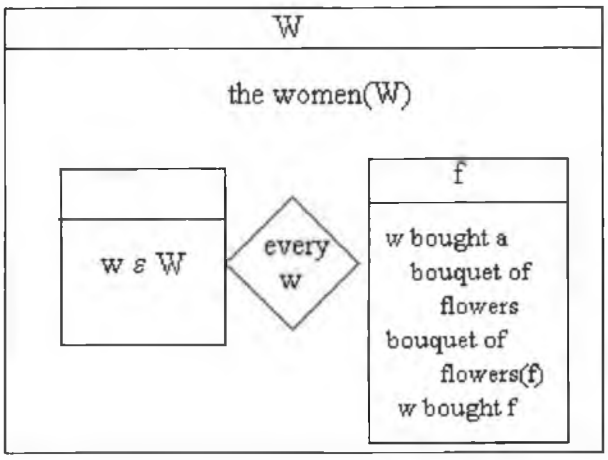


**Fig 3.57 DRSs representing the distributive readings of sentence (32) and (31).**

*(33). Three lawyers hired a secretary* (Kamp and Reyle 1991)

If an upper limit is specified as to the size of the plural NP e.g. three, four, this may be specified as a quantifier within the distributive duplex condition, Fig 3.58 or entered in the DRS, Fig 3.59 and 3.60. Sentence (33) is ambiguous between its distributive and collective reading, neither is favoured over the other. Interpreting this sentence distributively; each of the lawyers hired a secretary on his/her own or collectively, the three lawyers as a set hired a secretary produces different DRSs.

The distributive reading of (33), can be represented in a DRS in two ways. The first way, represented in Fig 3.58 treats the non-quantified NP like a quantified NP introducing a duplex condition and treating the *Three* (the determiner) as a quantifier. The restrictor of the sentence is put in the first DRS, with *three* put in the diamond DRS, and the scope *x hired a secretary* is dealt with in the final DRS.

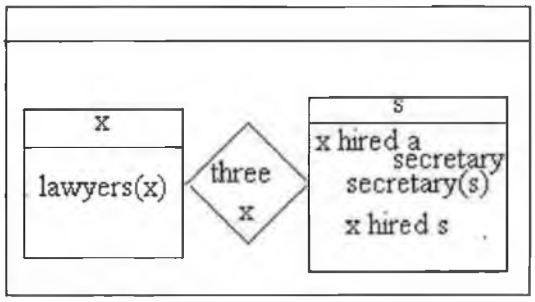


**Fig 3.58 The first representation of sentence (33) ( Kamp & Reyle 1991)**

The second distributive representation simply specifies the number of lawyers in the set *X*. Processing continues as before, with the sentence is changed to *X hired a secretary*, Fig 3.59.

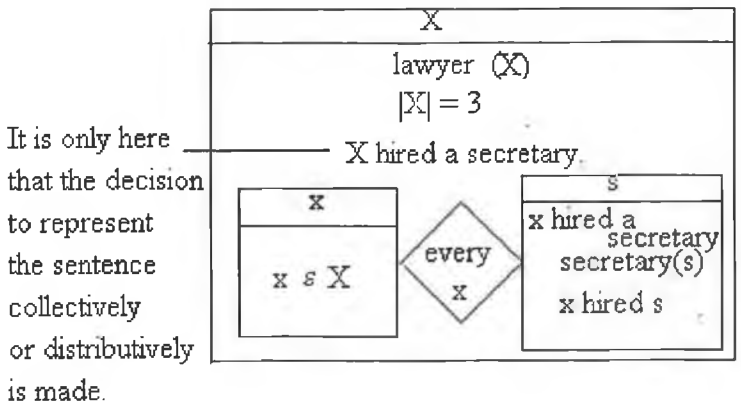

X
lawyer (X)
|X| = 3
It is only here that the decision to represent the sentence collectively or distributively is made.
X hired a secretary.
x
x ε X
every
x
s
x hired a secretary
secretary(s)
x hired s

**Fig 3.59. The second distributive DRS of (33)**

Finally the collective reading the size of set *X* is asserted and represented in Fig 3.60.

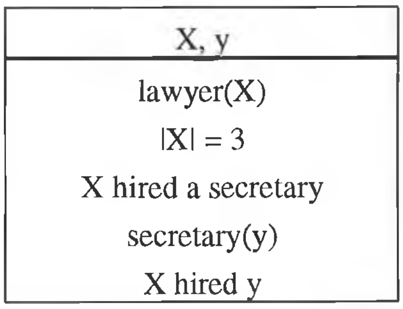


**Fig 3.60 The DRS representing the collective reading of (33).**

## 3.3.3 Interpreting plural NPs and pronouns.

Up to now, plural pronouns have introduced non-atomic referents and picked up non-atomic referents as antecedents. Plural pronouns have always refereed to other plural NPs, but this is not always the case. As in section 3.3.5 where quantified plural NPs e.g. most books, few books, introduced atomic referents to the duplex condition; so too do plural pronouns. There is no hard and fast rule as to what type of referent a plural pronoun will introduce. They can pick up and introduce atomic or non-atomic discourse referents in discourse, depending on the situation. These circumstances make the treatment of plural pronouns and NPs much more complicated than that of singular NPs.

Any plural pronoun can introduce an atomic and non-atomic discourse referent producing two possible DRSs, the only stipulation is that an antecedent be available for the new referent. If this referent is not available within a DRS, the DRS is improper and abandoned. The author will now illustrate with examples the sentence types where atomic, non-atomic and both referent types represent *they* and produce viable DRSs.

*They* can refer to an individual as in (34).

*(34). Every teacher taught a student they liked.*

where each individual teacher taught a student that *they* themselves liked.

Alternatively *they* can refer to a set:

*(27). John, Sue, George and Mary play tennis. They like it.*

In (27), the set representing *they* must be synthesised to include the individuals *John* and *Sue, George and Mary*. This set synthesis has been discussed in section 3.4.1.1

Another option presenting two alternatives in one is found in sentences of the form of (35). This type of sentence may be represented distributively or collectively; giving different referents for *they* depending on which representation is chosen.

*"they" can be interpreted as semantically singular if the antecedent NP is not read collectively.* (Reyle 1984)

*(35). The lawyers hired a secretary they liked.*

The referents introduced by *they* in sentences (34) and (35) will be discussed in the next two sections. When a plural pronoun is encountered in discourse, an atomic and non-atomic referent is introduced to represent *they* producing two possible DRS. An atomic referent can only have another atomic referent as antecedent, and vice versa for non-atomic referents. If both DRSs prove to be viable, processing continues within each one.

### 3.3.3.1 *They* as an Individual variable.

*(34). Every teacher taught a student they liked.*

This quantified sentence can only be represented by a duplex condition, Fig 3.61. Within this condition *they* can introduce an atomic or non-atomic discourse referent; both options (1), and (2) are presented producing two DRSs in Fig 3.62.

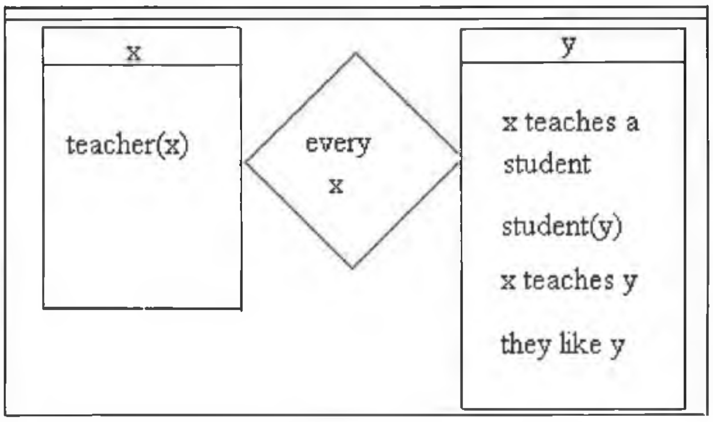
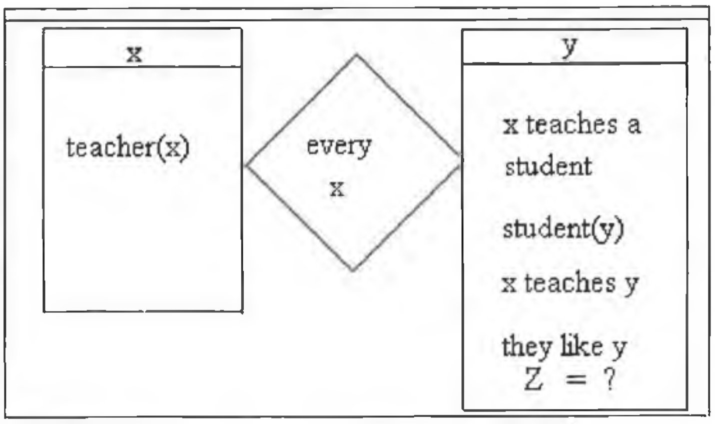
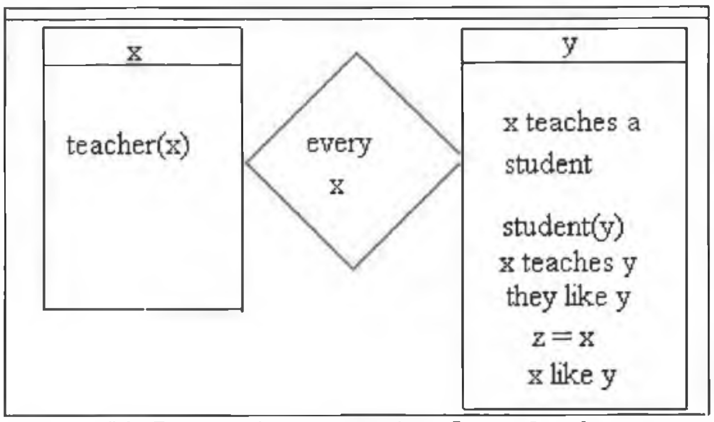


**Fig 3.61 The DRS of sentence (30).**

Option (1), introduces a non-atomic discourse referent which must find an antecedent within the duplex condition. No other non-atomic referent is available, so one would have to be synthesised by summation or abstraction. Summation would unite the referent *x* and *y* letting *Z* represent the union of *teacher* and *student.* This would not make sense, as how could *Z* the union of *teacher* and *student, like* the same *student.* Abstraction is dismissed also, as *they* occurs within the duplex condition and therefore cannot be made equal to the union of restrictor and scope. Abstraction may only be used, once the duplex condition to which it is being applied has been completed, Fig 3.62 (a). The second option, is the only viable one in this case introducing an atomic discourse referent which requires another atomic referent as antecedent; this referent is readily found in x, Fig 3.62 (b).



(a). Introducing a non-atomic referent for *they*



(b). Introducing an atomic referent for *they*

**Fig 3.62 (a) and (b), Introducing different referent types for *they*.**

### 3.3.3.2 *They* as a plural variable.

This section introduces a sentence of the form of (35), which like Fig 3.63 gives the choice of atomic or non-atomic referent introduction for *they*. Only one option is possible at any time within a single DRS, but both referent types will find an antecedent readily available.

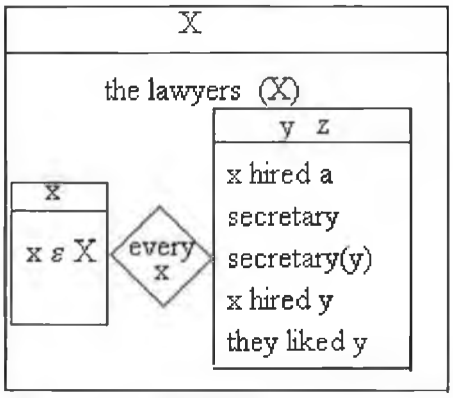
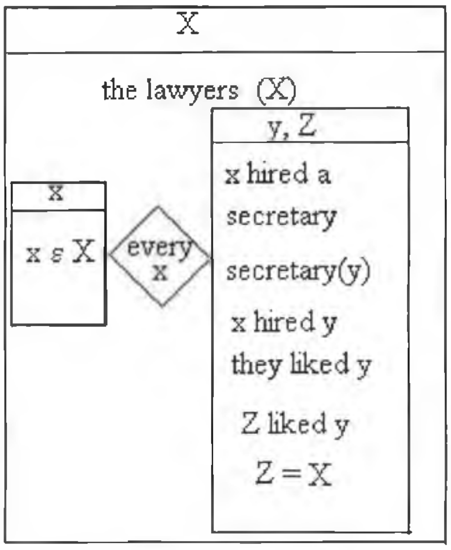


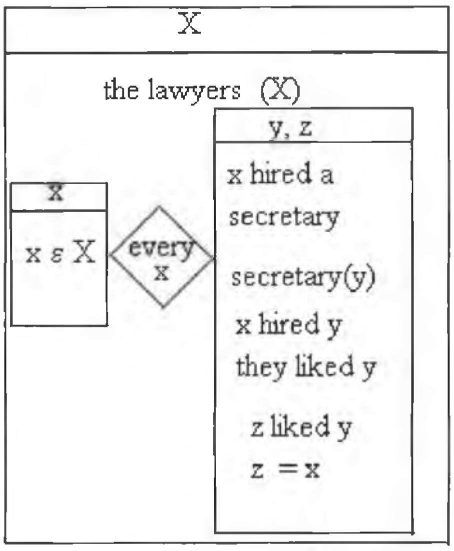
**Fig 3.63 The incomplete DRS for sentence (35).**

The distributive reading representation of sentence (35), presents the different discourse referent options for *they*. This representation introduces a non-atomic referent to represent the plural NP, *the lawyers*; with the rest of the duplex condition template filled according to the rules illustrated in section 3.4.2.2, this incomplete DRS is presented in Fig 3.63.

The plural pronoun *they* can introduce an atomic or non-atomic discourse referent. Only one of the options can be represented in a single DRS, Fig 3.63 presents both DRSs.



(a).Introducing a non atomic referent for *they*

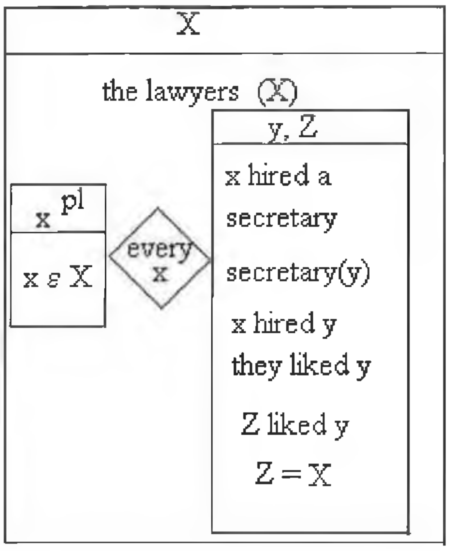


(b). Introducing an atomic referent for *they*

**Fig 3.63 The two DRSs for sentence (35).**

DRS (a), introduces an atomic discourse referent z for *they*, z must find an atomic antecedent from the set (x), x is chosen. DRS (b), introduces a non-atomic discourse referent Z which can only refer to another non-atomic referent i.e. X.

In Fig 3.63, the plural pronoun *they* introduced an atomic discourse referent. We need to keep the information that this atomic referent was introduced by a plural NP, so that only such referents can act as antecedents for such plural NPs. This is achieved by marking the atomic referent with a superscript *pl.* The pronoun construction rule must be changed to allow X and $x^{pl}$ to act as antecedents for non-atomic discourse referents. The distributive template is changed to introduce $x^{pl}$ as the individual chosen from the set. Changing Fig 3.63 to include $x^{pl}$ Fig 3.64 is produced.



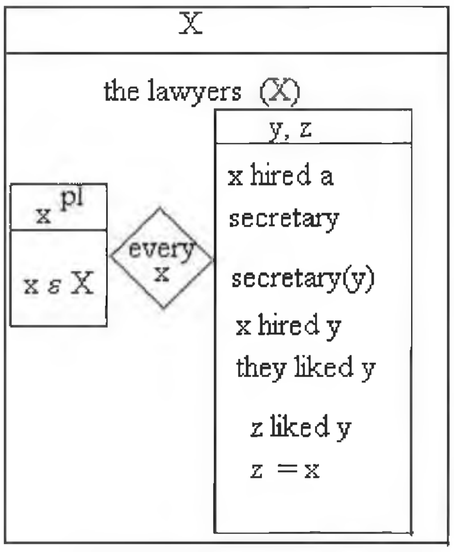


**Fig 3.64 The DRSs complete with superscript pl.**

The same rules apply to other plural pronouns e.g. them, their and these. These plural pronouns have all the same choices, atomic or non-atomic referent introduction with all the same representations. The last two sections illustrate the facts that any plural pronoun can introduce an atomic or non-atomic discourse referent to the DRS. The only stipulation being, that like refers to like when antecedents are being selected.

## 3.4 Summary

This chapter introduced the basics of DRT as proposed by Kamp and Reyle (1991). These principles will be expanded by the author in subsequent chapters; where the proposal is to model DRT so that a working implementation can be produced. This implementation will resolve all anaphora that occur within any possible discourse of a limited domain.

# Chapter 4 Implementation.

## 4.1 Introduction.

The process of implementing Discourse Representation Theory can be divided into two distinct modules, the syntax and semantics modules. The choice of which parser or grammar to use is left entirely up to the implementor by Kamp and Reyle (Kamp & Reyle 1993), DRT simply requires a syntax module to produce a parse tree to trigger the semantic construction rules. This chapter will outline the mechanisms used by the author to implement the syntax and semantic modules presently in operation in the implementation of DRT.

## 4.2. The Syntax Module.

A syntax module for a NLP application comprises of a grammar and sometimes a parser, the development of each of these will be examined in the upcoming sections. The lexicon and rules for the implementation are taken from chapters 0 and 4 of (Kamp & Reyle 1993), and can be examined in more detail in appendix A. The rules are limited to a small domain but cover all sentential forms of conjunction, disjunction, negation, quantification, indefinite, definite and quantified plurals as well as relative clauses.

Definite clause grammars (DCGs) are easily translated to Prolog (the implementation language) by means of an interpreter present in the Prolog language. They prove to be a good starting point for grammar and parser development since they also contain a parser that works exactly like Prolog itself. DCGs can be used as recognisers, where a sentence is fed into the grammar and a yes or no is outputted depending on whether it is accepted or not, or parsers, where the sentence is fed in and a parse tree is output if the string is accepted showing all the categories the words are related to. Feature structures may be built into the grammar e.g. gender, number or case, and will be returned in the parse tree. To output a parse tree the DCG is of the form of Fig 4.1, which illustrates the structure of some of the rules of the initial DCG.

```
                    s( s( NP,VP ) ) --> s( nogap,s( NP,VP ) )
        s(Gap,s( NP,VP )) --> np( Gender, Number,NP ),vp(Case,Gender,Number,VP )
np( Gender,Number,np(DET,N,OPT) ) --> det( Number,DET ), n( Gender,Number,N ),
                                   optrel( Gender,Number,OPT )
        np( Gender,Number,np(NP) ) --> pn(Gender,Number, NP )
                pn( Gen,Num, pn(Gen,Num,PN) ) --> [PN], { pn(Gen,Num,PN) }
                                    pn(male,singular,john)
```

**Fig 4.1 An extract of the DCG containing feature structures which outputs a parse tree.**

The DCG takes input of the form **s**(*Tree,Sentence*,[]) and parses the sentence using a top down approach, the input **s**(*Tree*,[*john,smokes*],[]) outputs the parse tree of the form:

**s**(np(pn(male,singular,*john*) ),vp( v(fin,intra,sing,*smokes*))).

with feature structures associated with each terminal of the grammar. This DCG is extended to associate a feature structure list with each category, not just the terminal categories. The start rule is now of the form:

```
s( [s(NP,VP), [num:Num,gap:Gap] ], [num:Num,gap:Gap] ) -->
                                  np(NP,[case:Case,gen:Gen,num:Num,gap:nogap]),
                                  vp(VP,[fin:Fin,trans:Trans,num:Num.,gap:Gap]).
```

This time the same input produces a different parse tree with feature structure lists associated with each syntactic category.

```
[s( [np( [pn(mary),[gen:fem,num:sing] ] ),
     [case:_,gen:fem,num:sing,gap:nogap]

  [vp( [v(knows),[fin:fin,trans:intra,num:sing] ] ),
      [fin:fin,trans:intra,num:sing,gap:nogap] ] ),

 [num:sing, gap:_] ]
```

As mentioned above, the DCG uses a top down parsing approach as normally used by Prolog, which will be discussed in the next section.

### 4.2.1. Top Down Parsing.

Top down parsers execute a left parse deriving sentential form by expanding the symbol found on the left hand side. They start with the start symbol S and find rules that apply to it, expanding the symbol to other categories, e.g. S $\rightarrow$ NP VP, which continues until all the non-terminals are converted to terminals. These parsers are also known as hypothesis driven parsers in that they explore a particular derivation in the belief it is right until they meet failure or success. They also parse in a depth first manner, visiting every sentential form of the derivation tree in a single vertical path. Four rules are used to implement top down parsing.

*The initialisation rule* - this introduces the start symbol S, that will be expanded in the parse.
*The success rule* - this is where all the terminals of the sentence have been derived from the S symbol, and the string is empty.

*The predict rule* - as each step of the parse introduces a new non-terminal symbol, this rule decides which grammar rules should be applied to expand these non-terminals in the hope of finding the rule that will lead to a terminal present in the string.

*The pop rule* - if a non-terminal symbol of the grammar has been fully expanded and leads to a terminal present in the string, change the string accordingly by removing the terminals. Work will now continue on the modified string.

Taking the simple grammar, S → NP VP, NP→*John*, VP → V, V → *smokes*, a top down derivation will be derived and represented in Fig 4.1 in category sequence based notation.

| Category list | String |
|---|---|
| ε | *John smokes* |

(initialisation rule - introduces the start symbol)

| | |
|---|---|
| S | *John smokes* |

( predict rule - expands the S symbol, S → NP VP )

| | |
|---|---|
| NP VP | *John smokes* |

( pop rule - NP → John, remove this terminal from the string
and the non-terminal from the category list)

| | |
|---|---|
| VP | *smokes* |

( predict rule, expands the VP category )

| | |
|---|---|
| V | *smokes* |

( pop rule - V → smokes, remove smokes from
the string and V from the category list )

| | |
|---|---|
| ε | ε |

**Fig 4.1 The Top down derivation of *John smokes*.**

categories. These parsers are also known as shift reduce parsers due to the method of replacing words by categories in the string. There are three rules in bottom up parsing:
*The success rule* - when the empty string ε is in String and the S symbol is in the Category list, this is the success state and parsing stops.

*The shift rule* - this rule captures the relationship between non-terminals and terminals i.e.non-terminal → terminal. The rule works by removing non-terminals from the string, and introducing the category they belong to in the category list.

*The reduce rule* - if all the daughters of a grammar rule are present i.e. $mother \rightarrow daughter_1, daughter_2 ...... daughter_n$ replace them in the category list by the *mother* of the rule.

Using the same grammar defined in section 4.2.1 *John smokes* will be parsed bottom up.

| Category list | String. |
|---|---|
| ε | *John smokes* |

( the shift rule - NP → John, introduce NP to the Category list and remove *John* from String)

| | |
|---|---|
| NP | *smokes* |

( the shift rule, removes the word smokes leaving the empty string, and introduces V to the Category list )

| | |
|---|---|
| NP V | ε |

( the reduce rule, replaces one non-terminal by another, VP → V)

| | |
|---|---|
| NP VP | ε |

( the reduce rule, replaces NPVP by S, S → NP VP ).

| | |
|---|---|
| S | ε |

( This is the success state, parsing stops ).

**Fig 4.3 Bottom up parse of *John smokes*.**

Bottom up parsing is data driven, i.e. the words present in the string drive the derivation. It is because of this that this type of parsing has no problem with left recursive rules, given that the elements are always present before the categories are introduced, so verification has already been received meaning that false hypotheses are not a problem. However, empty production rules, i.e. $X \rightarrow \varepsilon$ do cause a problem in bottom up parsing and not in top down parsing. Empty production rules which are used to fill the gaps introduced by relative clauses, can in bottom up parsing be applied using the reduce rule at any time leading to a loop. Assume an empty production $NP \rightarrow \varepsilon$ is present in the grammar.

| Category list | String |
|---|---|
| $\varepsilon$ | *John smokes* |

( shift rule - removes John from the list)

| | |
|---|---|
| NP | *smokes* |

( the shift rule, removes the word smokes)

| | |
|---|---|
| NPV | $\varepsilon$ |

( $NP \rightarrow \varepsilon$ , the shift rule removes $\varepsilon$ from the string leaving the empty string )

| | |
|---|---|
| NP V NP | $\varepsilon$ |

( the same shift rule can be applied an infinite number of times )

| | |
|---|---|
| NP V NP NP | $\varepsilon$ |

( This continues, forming an infinite loop that will never reach the success state )

**Fig 4.4. The problem of empty production rules.**

Originally the implementation used a simple top down parser. However this ran into trouble with the NP conjunctive rule i.e. $NP \rightarrow NP$ conj NP, and the problem of left-recursion. Left recursion causes a problem for top down but not bottom up parsers, with empty production rules proving to be a problem for bottom up but not top down parsers. The drawbacks of one parser are solved by the other, therefore an ideal parser would take parts of each approach and incorporate them into a new type of parser, the left corner parser. The author will now investigate this approach.

## 4.2.3 Left Corner Parsing.

This is the parser that was finally employed in the syntax module of the implementation, and is a step towards a combined parsing technique, with grammar rules triggered by the left most element of the right-hand side of the rule, Fig 4.5

$$\text{Mother} \rightarrow \underbrace{\text{Daughter}_1}_{\text{left corner verified by bottom up approach which triggers other rules.}}, \overbrace{\text{Daughter}_2, \ldots\ldots\ldots, \text{Daughter}_n}^{\text{the remainder is parsed using the top down parsing approach.}}$$

**Fig 4.5 The left most element of a grammar rule.**

The basic cycle used in left corner parsing is that each grammar rule is triggered by the left most element of the rule, which has already been corroborated by a bottom up parse. The remaining daughters of the rule are parsed using the top down approach. The rule Mother serves as the next left corner, i.e. the remaining rules are searched for a rule that contains the Mother as a left corner and the cycle starts again. Three rules are used in a left-corner implementation, presented in a sequence based notation which presents two states linked by implication i.e.

*present state* $\rightarrow$ *final state*,

where the final state or aim of the parse must be specified at the start of parsing e.g.

*John smokes* $\rightarrow$ S

*The success rule* - a grammar rule consists of a mother and verified left corner which are the same i.e. $X \rightarrow X$.

*The shift rule* - if the rule *Category* $\rightarrow$ *word* is in the grammar, replace the *word* by the *Category* changing the string accordingly.

*The left-corner predict rule* - this rule introduces branches to the derivation,. If the left corner we are searching for is present in a grammar rule i.e.

*mother* $\rightarrow$ *left-corner daughter*$_1$*.....daughter*$_n$, branch 1 is formed by applying any rules

$X_1 \rightarrow daughter_1,.. X_n \rightarrow daughter_n$

present with branch 2 from *mother* and remainder of the *daughters* → *final state*. Using the grammar defined in section 4.2.1 *John smokes* will be parsed using the left-corner parser and sequence notation.

String → Final State

---

*John smokes* → S

---

( shift rule - replace *John* by the category NP, NP → *John* )

NP *smokes* → S

( daughter$_1$ , daughter$_2$ → final state )

---

( l-c predict, we are looking for a rule with NP as the left corner,
S → NP VP , forming two branches of derivation
linking the daughters with any available categories
daughter$_1$ NP → NP daughter$_2$ *smokes* → VP
and the mother of the rule S, with the remainder of the daughters,
none in this case leads to the final state S →S
Each of the derivations is pursued until it reaches the success state.

NP → NP      *smokes* → VP      S → S

(success rule)      (shift rule V *smokes*)      (success rule)

V → VP

---

( left-corner predict, grammar rule VP → V , with no daughters present, just have 2 cases )

V → V      VP → VP

(success rule)      (success rule)

**Fig 4.6 The left corner parse of *John smokes*.**

As mentioned previously, left recursion leads to infinite loops in top down parsers with empty production rules causing the same problem for bottom up parsers. Left-corner parsers establish the left-corner of a rule bottom up with the rest of the rule parsed top down. They do not have any problem with left recursion as the left-corner of every grammar rule is established bottom up. The top down section of the parser has as before no problem with empty production rules, but the left-corner that is established bottom up falters here. This problem is eliminated by the introduction of a link-table that explicitly states what relations are allowed in the grammar and thereby introduces more top down information to the grammar.

A link table contains the reflexive, transitive closure of non-terminals of the grammar and is restricted to include only the left-corner categories of the grammar rules. There are three different types of link included in the table, all of which provide new information: the reflexive, left-corner and transitive link.

- *The reflexive link*, links all non-terminal left-corners to themselves:
  **link(non-terminal, non-terminal)**.

- *The left-corner link*, links all mothers and left-corners of rules, i.e.
  mother $\rightarrow$ left-corner, daughter$_1$....daughter$_n$ introducing **link( mother, left-corner)**.

- *The final link* is the transitive link, if link(x,y) and link(y,z) are present in the table introduce **link(x,z)**.

The shift and left-corner predict rules must be amended to include the link table relations.

*The shift rule* (amended) - *Category* $\rightarrow$ *word*: replace the *word* by the category iff there is a link from the final state we are aiming for to the *Category* i.e. **link(final state, *Category*)** present in the link table.

*The left-corner rule* (amended) - the same conditions apply as above and the **link( final state, *mother*)** must be present in the link table.

Basically a left-corner parser is a bottom up parser which uses the left-corner relation incorporating top down information to determine which rules to try, which helps limit the search space compared to a naive bottom up parser which traverses the entire search space. The top down parser was abandoned because of left-recursion, so the author then investigated bottom up parsing only to abandon this technique due to problems with large search spaces and empty production rules. The left corner parser is a mix of top down and bottom up parsing techniques, which solves the problems of both, resulting in the author's decision to use this technique in building the syntactic parser. The DCG must be changed slightly to accept the new parser, as this parser is not inbuilt and therefore must be implemented by specifying Prolog rules. This results in a change to the terminal conditions of the DCG, changing the format of lexical entries e.g.

lex(pn([pn(john), [gen:male,num:sing] ], [gen:male,num:sing] ), john),

with parse rules added to access these entries.

### 4.2.4 Implementing the left-corner parser in Prolog.

Introducing a new parsing method involves the introduction of parsing rules, as the inbuilt top down parse is to be ignored. The left corner parser uses four Prolog rules, with the main rule **parse** taking three arguments, and calling the other three rules, finally returning a parse tree. The prolog implementation of the main rule parse is presented in Fig 4.7.

```
parse(Goal, Begin, End) :-
        shift( SubGoal, Begin, Middle),
        link( Goal, SubGoal),
        l_c( SubGoal, Goal, Middle, End ).
```

**Fig 4.7. The main rule parse of the Prolog implementation of a left corner parser.**

Fig 4.8 explains each step of **parse**:

The *Goal*, is the category depth we wish to parse to, with the variable *Tree* associated with it e.g. s(*Tree*), np(*Tree*), v(*Tree*), returned. *Begin* is the string to be parsed presented in list form e.g. [*john,smokes*], and *End* is the empty list [].

**parse**( *Goal, Begin, End* ) :-

**shift**( *SubGoal, Begin, Middle* ), - This is the **shift** rule of section 4.2.3 presented in Prolog form. A word is removed from the string *Begin,* and its category is returned as *SubGoal*. *Middle* is the changed string.

**link**( *Goal, SubGoal* ), - The **link** relation searches the link_table which explicitly states the links between non-terminals in grammar and the left corners of the non-terminal rules. The links of the table are formed as described in 4.2.3, with any duplication caused by such rules removed to avoid backtracking here.

**l_c**( *SubGoal, Goal, Middle,End* ). - This is a recursive prediction rule introducing branches to the derivation, and works as described in section 4.2.3. The left corner **l_c** rule identifies a grammar rule with the left corner *SubGoal*, *Mother* --> [*SubGoal* | *Restgoals*]. The link table is then checked for a **link** between *Mother* and a new *SuperGoal*, with the processing of *RestGoals* continuing by feeding them into the **parse_rest** rule.

Finally **l_c** is recursively called with **l_c**(*Mother, SuperGoal, Middle, End*). The **l_c** rule introduces two different branches of derivation by calling the **parse** rule and **l_c** rule, each of these derivations are pursued until the success state is reached.

**Fig 4.8 An explanation of the parse rule.**

```
parse_rest( [], String, String).
parse_rest( [Goal | RestGoal], B, E) :-  % deals recursively with
        parse( Goal, B, M),              % sub derivations
        parse_rest(RestGoal, M, E).      % generated by l_c predict
```

**Fig 4.9 The parse_rest rule.**

The fourth parse rule is the **parse_rest** rule which recursively parses the sub derivations *RestGoals* produced by the **l_c** predict rule. The rule takes three arguments with the list *SubGoal* divided into head and tail, [*Goal* | *RestGoal* ], ***parse_rest( SubGoal, Begin, End)***. The **parse** rule is called within **parse_rest** with *Goal* fed in as the first argument, and **parse_rest** is recursively called with *RestGoal*. The second argument *Begin* reflects the state of the string at that moment, with the empty string *End* as the third argument. Full code for the left corner parser, with the grammar and link-table can be seen in **appendix A**.

## 4.3 The Semantics Module.

*Hardly any of the implementations known to us really implements the DRS-construction algorithm as detailed in (Kamp 1981) and (Kamp & Reyle 1993), to the letter but is either based on a (semi-) compositional α-calculus style or a unification based reformulation*

(Cooper *et al* 1995)

The implementation presented in this thesis sets out to follow the guidelines laid down in (Kamp & Reyle 1993), extending the existing DRT theory into a viable computational algorithm.

### 4.3.1 The Basic DRS format.

As discussed in chapter 3, Discourse Representation Structures (DRSs) are the basic building blocks of DRT and are represented by flowcharts, Fig 4.1.

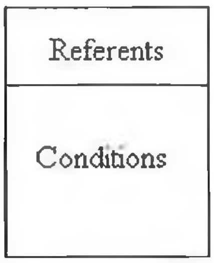


**Fig 4.10 The DRS.**

In the semantic module of this thesis, the principal DRS is represented by the template drs( [ Referents,Conditions ] ) where Referents and Conditions are lists. The change from flowchart to drs( [ Referent,Conditions ] ) format is illustrated in the change from Fig 4.11. to 4.12.

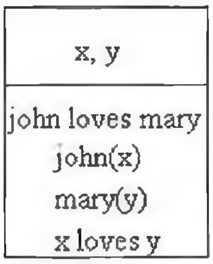


**Fig 4.11. *John loves Mary*, represented by a DRS.**

$$\text{drs( [ [ }\underbrace{\text{x,y}}_{\text{Referents}}\text{ ], }\overbrace{\text{[ john(x), mary(y), x loves y ]}}^{\text{Conditions.}}\text{ ] )}$$

**Fig 4.12 How the DRS of *John loves Mary* is represented in the semantic module.**

To make processing within the module easier referents are represented by numbers not letters, as numbers can be easily updated. The brackets are also changed to colons to avoid the Prolog interpreter mistaking the conditions for Prolog rule or fact declarations, Fig 4.13.

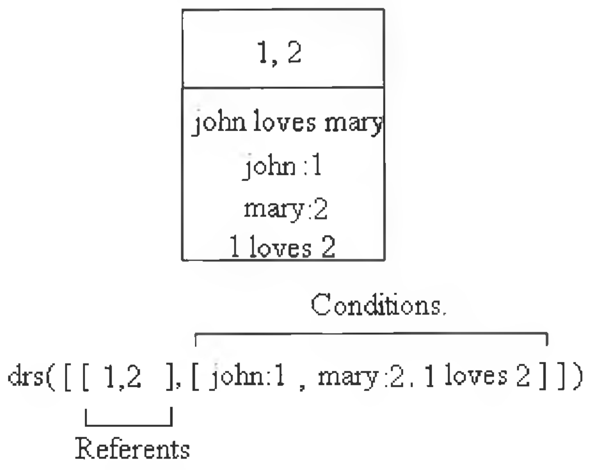


**Fig 4.13. Using numbers to represent referents - the revised Fig 4.2 and Fig 4.3**

Feature structures provided by the syntactic module e.g. [gen:male, num:sing], are included in the semantic module representation of the flowcharts in an attempt to incorporate a small amount of world knowledge into the implementation. Adding feature structures to Fig 4.13:

drs([ [1,2], [ john:1, [gen:male,num:sing], mary:2, [gen:fem,num:sing],
1, loves [fin:fin, trans:trans, num:sing], 2 ]]).

For ease of reading, the author will omit feature structures from the semantic representations of section 4.3.2.

## 4.3.2 Sub-DRSs and the semantic module.

As stated in section 3.3 of chapter 3, DRSs can be nested in one another. Conditional, negated, conjunctive, disjunctive and quantified sentences introduce different sub-DRS templates to the condition list of the principal DRS. These sentences, with the exception of negation, will be referred to as **complex sentences** within this thesis, with the remaining sentence types known as **simple sentences**. This section presents the different sub-DRS templates used within the semantic module, the prolog rules used to introduce the said templates, and an illustration from a flowchart presented in chapter 3 to semantic module representation of DRSs.

**The negated sub-DRS:** sub( not( SubReferent, SubCondition )).

Negated sub-DRSs are introduced by the **start** rule.

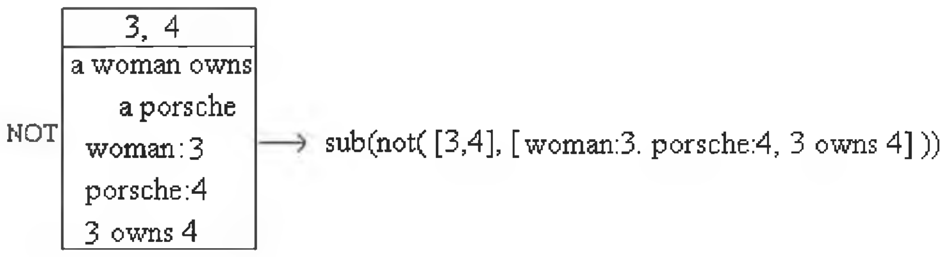


**Fig 4.14. The negated DRS of *A woman does not own a porsche*, showing the flowchart and semantic module representations.**

The negated DRS must be embedded in the Condition list of the principal DRS. We will assume that the principal DRS in Fig 4.15 is empty, with the principal DRS of Fig 4.13 extended in Fig 4.16.

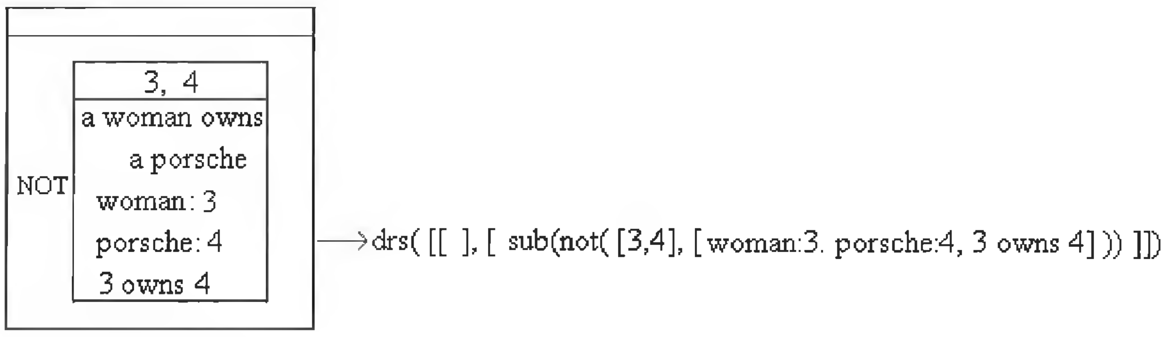

3, 4
a woman owns
a porsche
NOT
woman: 3
porsche: 4
3 owns 4
drs( [[ ], [ sub(not( [3,4], [woman:3. porsche:4, 3 owns 4] )) ]])

**Fig 4.15. The change from flowchart to semantic module representation of embedding the negated sub-DRS in an empty principal DRS.**

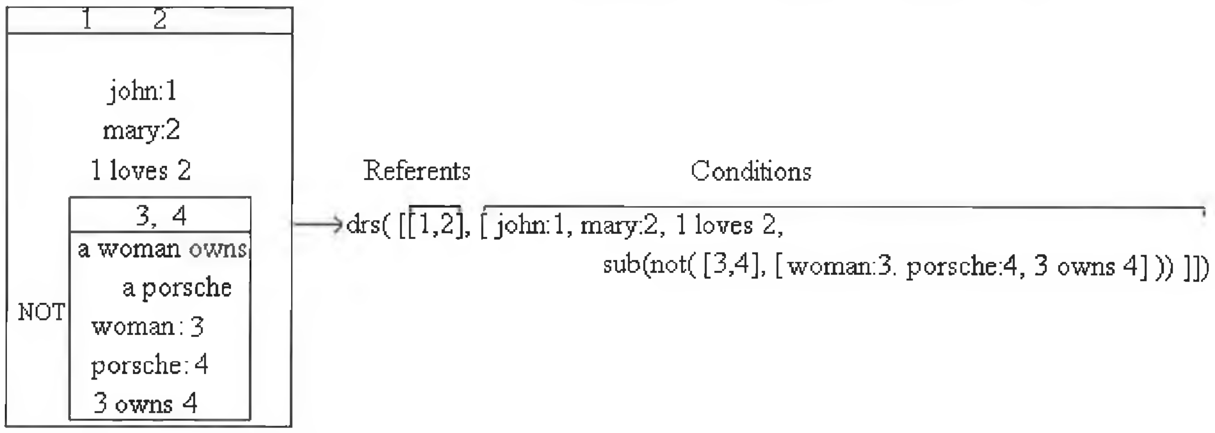
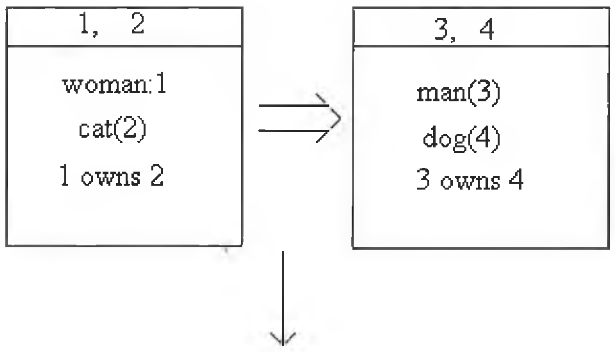


**Fig 4.16. The change from flowchart to semantic module representation of embedding the negated sub-DRS in the principal DRS of Fig 4.13.**

**The conditional sub-DRS:** sub(cond( DRS1,=>,DRS2 )).

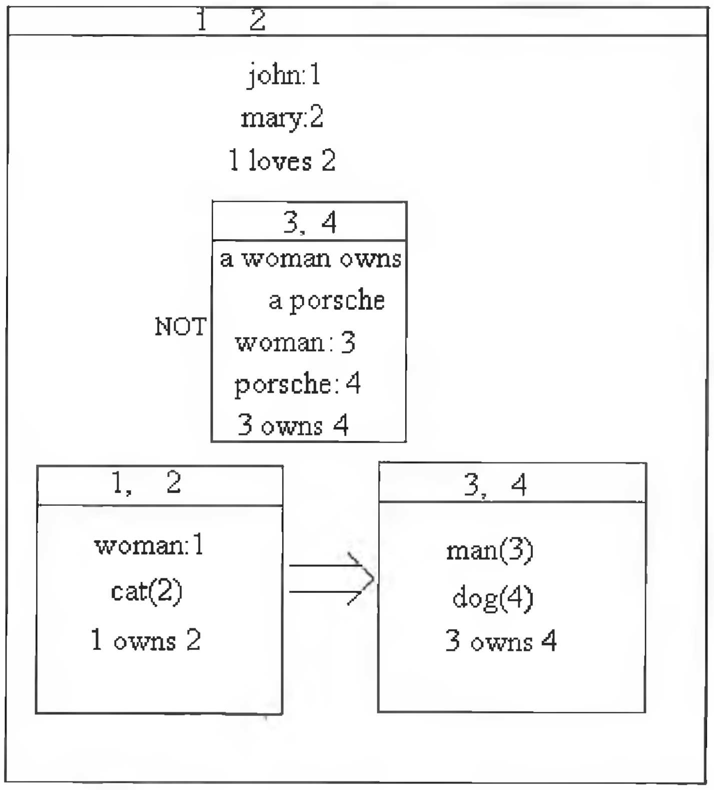
Conditional sub-DRSs are introduced by the **condition** rule.



sub(cond( drs( [ [1,2], [woman:1,cat:2,1 owns 2] ] ),⇒ ,drs([ [3,4], [man(3), dog(4), 3 owns 4]])))

**Fig 4.17. The flowchart and semantic module representation of the conditional sentence *If a woman owns a cat, then a man owns a dog.***

As with all sub-DRSs, the conditional sub-DRS is placed in the condition set of the principal DRS. Fig 4.18 illustrates the sub-DRS of Fig 4.17 being embedded in the condition list of the existing principal DRS of Fig 4.16.



drs( [[1,2], [ john:1, mary:2, 1 loves 2, sub(not( [3,4], [woman:3. porsche:4, 3 owns 4] )),

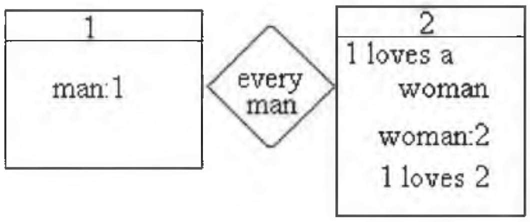
sub(cond( drs( [ [1,2], [woman:1,cat:2,1 owns 2] ] ),

⇒ ,drs([ [3,4], [man(3), dog(4), 3 owns 4]]))) ]] )

**Fig 4.18. The flowchart and semantic module representations of embedding the conditional sub-DRS in the principal DRS of Fig 4.16.**

**The Duplex condition DRS:** duplex(DRS1,Quantifier,Main-Referent,DRS2).

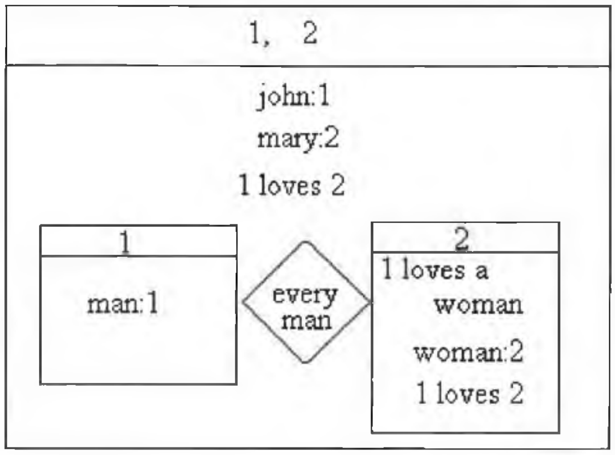
Duplex conditions are introduced by the **duplex** rule.



duplex(drs( [[1], [man:1]] ), every,man, drs( [[2], [woman:2, 1 loves 2]] ) ).

**Fig 4.19. The flowchart and semantic representation of the duplex sub-DRS of *Every man loves a woman.***

Again the duplex condition can be embedded in the condition list of the principal DRS. Fig 4.20 now represents the disjointed mini-discourse: *John loves Mary. Every man loves a woman.*



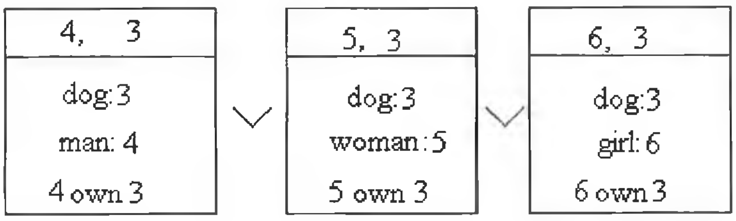
drs( [[1,2], [john:1, mary:2, 1 loves 2,

duplex( drs( [[1], [man:1]]), every, man,drs( [[2], [woman:2, 1 loves 2 ]] ) ) ]])

**Fig 4.20. The flowchart and semantic representation of embedding the duplex condition in the principal DRS of Fig 4.13.**

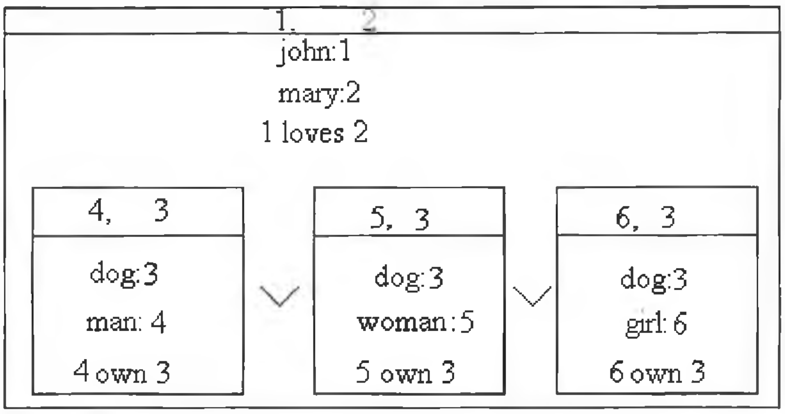
**The disjunctive DRS:** connector([DRS1,or,DRS2,or....,or,DRS5]).

Disjunctive sub-DRSs are introduced by **conj_and_disjunction** and **sentence_disj_and_conj** rules to represent disjunctive NPs and sentential disjunction.



connector( [drs([[4,3], [dog:3, man:4, 4 own 3]]), or,
drs([[5,3], [dog:3, woman:5, 5 own 3]]), or,
drs([[6,3], [dog:3, girl:6, 6 own 3]]) ]).

**Fig 4.21. The flowchart and semantic representation of *A man, a woman or a girl own a dog.***



drs( [[1,2], [john:1, mary:2, 1 loves 2,
connector( [ drs( [[4,3], [dog:3, man:4, 4 own 3]]), or,
drs( [[5,3], [dog:3, woman:5, 5 own 3]]), or,
drs( [[6,3], [dog:3, girl:6, 6 own 3]]) ]) ]])

**Fig 4.22. The flowchart and semantic representation of embedding the disjunctive sub-DRSs of Fig 4.21 in the principal DRS of Fig 4.13.**

Conjunction is treated within the same rules as disjunction, although conjunction does not introduce sub-DRSs ( see section 3.3.7 chapter 3).

## 4.3.3. Constructing a DRS.

DRS construction starts when the syntactic analysis of a sentence of the discourse *Sentence_Syn*, is passed into the **construction** rule.

```
construction(Referent, Sentence_Syn, Existing_Drs, New_Drs, Condition, Refcount):-
constructing( Referent, Sentence_Syn, Existing_Drs, New_Drs, main,
                                                          Condition,Refcount).
                                                       ↑
                                                       Flag
```

- *Referent* is the number of the next referent to be assigned.
- *Sentence_Syn* is the syntactic analysis of a sentence of the discourse.
- *Existing_Drs* is the principal DRS built so far. This receives the value *drs( [[], []] )* if the principal DRS is empty or processing has just begun.
- *New_Drs* is the rule output incorporating the new information introduced by *Sentence_Syn* and the information already stored in the *Existing_Drs*.
- *Condition* controls looping within the module.
- *Refcount* is the number of the next referent to be assigned when *construction* or *constructing* is exited.
- *Flag* i.e *Flag* = main, acts as a signal indicating where new information should be placed in the DRS.

The **construction** rule calls a three part recursive rule **constructing,** which treats simple and complex sentences. All recursive rules need terminating conditions, and so *Condition* controls the number of iterations of **constructing**. The first call to **constructing** gives *Condition* the value [s(unanalysed,unanalysed)], indicating that both the NP and VP part of the *Sentence_Syn* are unanalysed as yet:

NP part

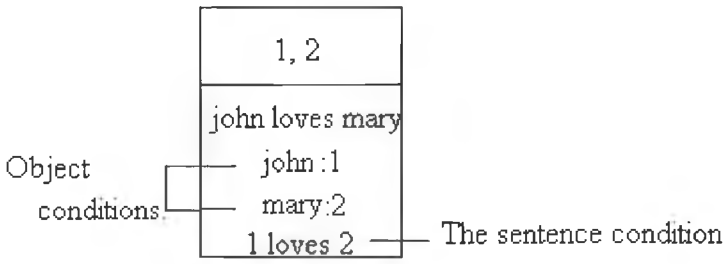
|

*[s(unanalysed, unanalysed)].*

|

VP part

The construction rules of DRT use triggers to recognise parts of sentences and introduce referents and conditions to the DRS. As each trigger is fired depending on which part of the sentence the trigger applies to, the NP or VP part of *Condition* is changed to *analysed.* Looping stops when *Condition* is [s(analysed, analysed)], implying that the sentence is irreducible or fully processed. The need for *Condition* is further explained by revisiting Fig 4.11



The author asserts that excluding sub-DRS templates, there are two different types of DRS conditions present in any DRS:

**object conditions**: which are introduced when a trigger rule fires e.g *john:1*. These conditions also change the existing sentence syntax e.g. when *john:1* is introduced the sentence is changed from *john loves mary* to *1 loves mary*. Object conditions are added to the DRS until *Condition* is [s(analysed, analysed)] , and the sentence is irreducible.

**sentence conditions**: When *Condition* is [s(analysed, analysed)], the sentence is irreducible, but the remaining sentence syntax contains the **sentence condition** e.g. *1 loves 2*. This essential condition is nested in other syntactic information which must be removed before the condition can be inserted into the DRS, this which achieved by **constructing**.

As mentioned previously, there are two different types of sentence in this implementation, **simple** and **complex** sentences. The analysis of simple sentences forms the backbone of the semantic implementation, as all sentences may be reduced to this type. Complex sentences are treated by breaking them up into simple sentences, analysing these parts, and joining the new structures together into a sub-DRS template which will be incorporated into the principal DRS. To cope with simple and complex sentences which introduce object and sentence conditions, the **constructing** rule is three pronged:

**Prong 1**. Establishes the sentence type, simple or complex, and treats accordingly.

**Prong 2.** Introduces sentence conditions to simple sentences.

**Prong 3.** Assigns the new information to the *New_Drs* for complex sentences.

Entry to each part of **constructing** is controlled by *Condition.*

**Prong 1:**

```
constructing(Referent,Sentence_Syn,Existing_Drs,Finished_Drs,Flag,Condition,Refcount) :-

    (( duplex( Referent, Existing_Drs, Sentence_Syn, Flag, Condition, Finished_Drs,_) )
      ;
     ( not( same( Condition, finished ) ),
       conj_and_disjunction(Referent,Sentence_Syn, Existing_Drs,New_Drs,
                                             Flag, Refcount),
        same(NewCond, finished) )
     ;
     ( start(Referent, Sentence_Syn, New_Sentence_Syn, Existing_Drs, New_Drs, Flag,
                                   NewFlag, Condition, NewCond ),
       ( number(Referent);
        ( not(number(Referent)),!, fail )),
       Referent_two is Referent + 1, ! )
    ;
    ( plural_definiteNPs( Referent, Referent_two, Sentence_Syn, New_Sentence_Syn,
                                   Condition, NewCond, Existing_Drs, Flag, NewFlag, New_Drs) )
    ;
    ( not( same( Condition, finished )),
      sentence_disj_and_conj(Referent,Sentence_Syn, Existing_Drs, New_Drs,
                                             Flag,Refcount),
```

**same**(*NewCond,* finished) )

;

( **condition**( *Sentence_Syn, Existing_Drs, Finished_Drs, NewFlag, Referent, Referent_two, NewCond, Condition*))),

**constructing**( *Referent_two, New_Sentence_Syn, New_Drs, Finished_Drs, NewFlag, NewCond, Refcount* ).

Prong 1. of **constructing** consists of an or-loop of rules, where each or-clause contains a rule designed to recognise a sentential component and introduce a sub-DRS to the DRS being developed, aside from **start** which works on simple sentences. Since sentences of any type can eventually be broken up into smaller simpler sentences the **start** rule - which encodes the bulk of the construction rules of DRT - is used at some stage in most processing.

As simple sentence analysis forms the essence of the semantic implementation presented in this thesis the author will begin the semantic discussion at this point, before moving onto each of the complex rules of the or-loop of **prong 1** in turn.

### 4.3.3.1. Analysing simple sentences within constructing

The first call to **constructing** will have *Referent* as 1, *Condition* as [s(unanalysed,unanalysed)] and *Flag* as main. As stated in chapter 3, the construction rules of DRT use triggers to recognise parts of sentences, and introduce referents and conditions to the DRS according to which triggers have been fired. The **start** rule contains an or-list of triggers with a corresponding set of actions to be executed when any trigger is fired. The rule takes the *Sentence_Syntax* and passes it through the triggers to see if any element of the sentence e.g. indefinite NP, definite NP, can introduce a new referent and condition. **start** succeeds if one of the triggers fires, and the connected actions are executed. At present the author is assuming that the *Sentence_Syntax* passed into **constructing** is simple and **start** will succeed where the other complex rules of the or-clauses fail. The structure of **start** is:

**start**( *Referent, Sentence_Syntax, New_Sentence_Syntax, Existing_Drs, New_Drs, Flag, NewFlag, Condition, NewCondition* ) :-

(( trigger$_1$(cr_pn.....                    ),          ______________ cr_pn is the construction

```
   Action to be performed if trigger1 is fired )
 ;


 ( trigger2(cr_pn,....                ),          ______________
   Action to be performed if trigger2 is fired )
 ;
 ( trigger3(cr_pro,....               ),
 . Action to be performed if trigger3 is fired )
 ;
 .
 .
 (triggeri(cr_id....                  ),
  Action to be performed if triggeri is fired ) ........... ).
```

rule for proper nouns. All trigger rules have two parts, one to recognise the NP part of a sentence and the other for the VP part

The **trigger** rule is a matching fact declared as:

**trigger**( *Trigger_name, Syntax, Syntax,*
*DRS, DRS,*
*New_Sentence_Syntax, New_Sentence_Syn,*
*Flag, Flag).*

An instance of the indefinite NP trigger is shown below:

```
                                   If Sentence_Syn is the same as the shown syntax
                                          the trigger fires.

          (trigger(cr_id, Sentence_Syn,
                        [s([ np( [det(a), [num:sing]], [n(B), [gen:C,num:sing]],
A new DRS is               [optrel( [] ), _], _],VP), S_Featstr ],
formed, DRS  ——    drs([ [Referent], [B:Referent, [gen:C, num:sing]]]), DRS ,
with the specified [s( Referent ,VP), S_Featstr],  New_Sentence_Syn,Flag,NewFlag),
conditions.
                        Referent replaces the NP part of the sentence
                        to give the New_Sentence_Syn
Change the
NP part       Condition = [s( E,F )],                                  Flag stays
of Condition  NewCondition = [s(analysed, F )],                        the same.
to analysed.  drs_conc(Existing_DRS, DRS, New_Drs, Flag))
                   |
              Add the newly formed DRS to the condition list of the Existing_DRS
```

- This states that if the *Sentence_Syntax* is the same as [s([np([det(a).....] the trigger fires as the syntax of the two sentences is equivalent. A new referent, the number of which is specified by *Referent,* and condition ( *B:Referent* e.g. *woman:1* ) are introduced to the *DRS*. Changes made to *DRS* are incorporated into the *Existing_DRS* at a later stage.

- The *Sentence_Syntax* is changed by replacing the trigger element, in this case the indefinite NP declaration i.e. [np( [det(a), [num:sing]], [n(B), [gen:C, num:sing]], [optrel([]),_],_] by the newly introduced *Referent* producing *New_Sentence_Syntax.*

The *Flag* of the trigger rule has two values *main* or *sub*, where *main* states that the DRS additions are simple and should be placed in the main universe of the *Existing_DRS*, and *sub* states that the new conditions and referents should be incorporated into a sub-DRS which was just joined with the *Existing_DRS*. These changes are made by the **drs_conc** rule:

**drs_conc**( *DRS1, DRS2, New_Drs, Flag*) :-

which concatenates conditions and referents of different DRSs together depending on the value of *Flag*, producing the *New_Drs* which is output by **start**.

The value of *Flag* can be changed within **start** by the negation trigger rule, which recognises the **do not** of the negated *Sentence_Syn*, changes the *Flag* to *sub* and introduces a negated sub-DRS template. All referents and conditions introduced by this sentence from this point on will be incorporated into the negated sub-DRS.
As illustrated in section 3.3.4 of chapter3, referents and conditions of proper nouns are always put into the principal DRS condition list, regardless of whether they occur in a sentence that introduces a sub-DRS or not. To account for this within the proper noun trigger rules, the **drs_conc** rule call always presents *Flag* as main:

**drs_conc**( *Existing_DRS, DRS, New_Drs*, main)

This is so that the proper noun declarations contained in *DRS* will always be placed in the condition list of the *Existing_DRS* and not a sub-DRS within *Existing_DRS*.

**start** is non-recursive with one iteration firing at most one trigger, therefore taking at least two iterations to completely reduce a sentence. The recursive **constructing** rule, calls **start** until the value of *Condition* changes to [s(analysed,analysed)] and the second prong of **constructing** can be called.

Taking an example: *(1). A man loves Sue.*

Assume the syntactic analysis exists and this is the initial call, therefore the Existing_Drs is empty i.e. drs( [ [], [] ]):

```
                                        Sentence_Syntax
                                              |
construction( 1, [s([np([det(a), [num:sing]], [ n(man), [gen:male,num:sing]],[optrel([]),_]),
                                        [case:_, gen:male, num:sing, gap:nogap ]],
              [vp( [ v(loves), [fin:fin, trans:trans, num:sing]],
                   [np([pn(sue), [gen:fem,num:sing] ]), [case:_, gen:fem, num:sing]] ),
                                        [fin:fin, trans:trans, num:sing, gap:nogap] ]),
                                        [num:sing, gap: nogap]],
              drs( [ [], [] ]), New_Drs, [s( unanalysed, unanalysed )], Refcount ) :-
                         |                   |
                    Existing_Drs         Condition.
```

```
  constructing(1,[s([np([det(a),[num:sing]],[n(man),[gen:male,num:sing]],[optrel([]),_]),
                                        [case:_, gen:male, num:sing, gap:nogap ]],
              [vp([ v(loves), [fin:fin, trans:trans, num:sing]],
                   [np([pn(sue), [gen:fem, num:sing] ]), [case:_, gen:fem, num:sing]]),
                                        [fin:fin, trans:trans, num:sing, gap:nogap] ]),
                                        [num:sing, gap: nogap]],
              drs( [ [], [] ]), New_Drs, main, [s( unanalysed, unanalysed )], Refcount ).
```

Prong 1. of **constructing** is executed, as the sentence is simple i.e. contains no complex components, and all other rules fail until **start** is encountered:

```
  start(1, [s( [np( [det(a), [num:sing]], [ n(man), [gen:male, num:sing] ], [optrel( [] ),_]),
                                        [case:_, gen:male, num:sing, gap:nogap ]],
          [vp( [ v(loves), [fin:fin, trans:trans, num:sing]],
               [ np( [pn(sue), [gen:fem, num:sing]]), [case:_, gen:male, num:sing]]),
                                        [fin:fin, trans:trans, num:sing, gap:nogap] ]),
                                        [num:sing, gap: nogap]],
       New_Sentence_Syntax, drs([[], []]), New_Drs, main, NewFlag,
                                      [s(unanalysed,unanalsyed)], NewCondition).
```

The triggers are processed in order one by one with the proper noun trigger firing first:

```
( trigger( cr_pn, [s( [np( [det(a), [num:sing], [n(man), [gen:male, num:sing]],
                          [optrel( []),_], [case:_,gen:male,num:sing,gap:nogap]],
                     [vp( [v(loves), [fin:fin, trans:trans, num:sing]],
                          [np( [pn(sue), [gen:fem, num:sing]],
                               [case:_,gen:fem, num:sing, gap:nogap ]]),
                      [fin:fin,trans:trans, num:sing,gap:nogap]]), [num:sing, gap:nogap]],
                   [s( NP, [vp(V, [np([pn(A), [gen:B, num:C]]), _]), VPfeatstr]), Sfeatstr],

              drs( [[1], [sue:1, [gen:fem, num:sing]]]), DRS,

              [s( [np([det(a), [num:sing,], [n(man), [gen:male, num:sing]],
                      [ optrel( [] ), _], [case:_, gen:male, num:sing, gap:nogap]],
                   [vp( [v(loves), [fin:fin, trans:trans, num:sing]], 1),
                       [ fin:fin, trans:trans, num:sing, gap:nogap]]), [num:sing, gap:nogap]],
              New_Sentence_Syn,
              main, NewFlag),
      Condition = [s(unanalysed, unanalysed)],
      NewCondition = [s( unanalysed, analysed )],
      drs_conc( Existing_DRS, DRS, New_Drs, main))
```

This trigger fires as the syntax of the two sentences presented is equivalent introducing:

- a referent specified by *Referent* which is 1 and condition *sue:1* to *DRS*
  *DRS* = drs( [[1], [sue:1, [gen:fem, num:sing]]])
- a *New_Sentence_Syntax* which is formed by replacing the recognised category with *Referent.*

```
New_Sentence_Syntax = [s( [np([det(a), [num:sing,], [n(man), [gen:male, num:sing]],
                                [ optrel( [] ), _],
                                    [case:_, gen:male, num:sing, gap:nogap]],
                          [vp( [v(loves), [fin:fin, trans:trans, num:sing]], 1),
                                    [ fin:fin, trans:trans, num:sing, gap:nogap]]),
                          [num:sing, gap:nogap]]
```

- The VP part of *Condition* is changed to *analysed* as the proper noun occurs in this part of the sentence producing *NewCondition.*

  *NewCondition* = [s( unanalysed, analysed )].

- The new conditions and referents introduced to the temporary variable *DRS* must now be incorporated into the *Existing_DRS* to give *New_Drs.* As this is a proper noun construction rule, all information will be introduced to the main condition list, and not a sub-DRS.

  *New_Drs* = drs( [[1], [sue:1, [gen:fem, num:sing]]]).

The next step within the **start** or-clause of **constructing** is to increment *Referent* in preparation for the next iteration: *Referent_two is Referent + 1.* This completes the operations of this or-clause, and dropping out of the clause and the or-loop leads to a second call to constructing:

```
    constructing( Referent_two, New_Sentence_Syn, New_Drs, Finished_Drs, NewFlag,
                                                                  NewCond, Refcount).
```

Filling in the values we have just generated:

```
    constructing( 2, [ s( [np( [det(a), [num:sing]], [n(man), [gen:male, num:sing]],
                                  [optrel([]),_], [case:_,gen:male,numLsing,gap:nogap]],
                              [vp( [v(loves), [fin:fin, trans:trans,num:sing]], 1),
                                [fin:fin,trans:trans,num:sing,gap:nogap]]), [num:sing,gap:nogap]],
                    drs( [[1], [sue:1, [gen:fem, num:sing] ]), Finished_Drs, main,
                                   [s( unanalysed, analysed )], Refcount).
```

**Prong 1** of **constructing** is executed as long as *Condition* is not *[s(analysed,analysed)]* or *finished.* This is the case, so it is called again. The sentence syntax is simple as before, and all other rule invocations fail until **start** is encountered where the indefinite NP construction rule is executed producing another *New_Drs* and a *Condition* of [s(analysed,analysed)].

```
( trigger( cr_id, [s( [np( [det(a), [num:sing]], [n(man), [gen:male, num:sing]],
                             [optrel([]),_], [case:_,gen:male,numLsing,gap:nogap]],
                            [vp( [v(loves), [fin:fin, trans:trans,num:sing]], 1),
                              [fin:fin,trans:trans,num:sing,gap:nogap]]), [num:sing,gap:nogap]],
                  [s( [np( [det(a), [num:sing]], [n(B), [gen:C, num:sing]], [optrel([]),_]),_],VP),
                                                                           Sfeatstr],
            drs( [[2], [man:2, [gen:male, num:sing]]]), DRS,

            [s(2, [vp([v(loves), [fin:fin, trans:trans, num:sing]],1),
                    [fin:fin, trans:trans, num:sing, gap:nogap]]), [num:sing, gap:nogap]],
            New_Sentence_Syn,
            main, NewFlag),
    Condition = [s( unanalysed, analysed )],
    NewCondition = [s( analysed, analysed )],
    drs_conc( Existing_Drs, DRS, New_Drs, Flag ))
```

This time **start** is exited with:

```
  New_Sentence_Syn = [s( 2, [vp( [v(loves), [fin:fin, trans:trans, num:sing]], 1),
                                  [fin:fin, trans:trans, num:sing, gap:nogap]]),
                                                   [num:sing, gap:nogap]]

  New_Drs = drs( [ [1,2], [sue:1, [gen:fem, num:sing], man:2, [gen:male, num:sing]]])

  Condition = [s(analysed,analysed)].
```

Again *Referent* is incremented, and the or-loop is exited resulting in a third call to **constructing**. Filling in the values obtained from **start** results in:

```
  constructing( 3, [s( 2, [vp( [v(loves), [fin:fin, trans:trans, num:sing]], 1),
                             [fin:fin, trans:trans, num:sing, gap:nogap]]),
                                                   [num:sing, gap:nogap]],
                drs([ [1,2], [sue:1, [gen:fem, num:sing], man:2, [gen:male, num:sing]] ]),
                Finished_Drs, main, [s(analysed,analysed)], Refcount).
```

The *New_Sentence_Syn* produced by **start** is irreducible, but the sentence condition contained in *New_Sentence_Syn* must be removed and put into the DRS being developed. This is achieved by the second prong of **constructing** which is called when *Condition* is [s(analysed, analysed)].

Sentences containing relative clauses and gap information contribute more that one sentence condition to the DRS being developed, an example of this is shown on the next page.

*John loves a woman who does not love him* contributes two sentence conditions to a DRS:

- *John loves a woman*
- *A woman does not love him.*

*Mary saw the woman who George kissed* also contributes two sentence conditions to a DRS:

- *Mary saw the woman*
- *George kissed the woman.*

Prong 2 of **constructing** contains a set of or-clauses that present an irreducible sentence syntax coupled with actions to be executed if the sentence syntax presented and the *Sentence_Syntax* are equivalent. In this way sentence conditions can be isolated and inserted one at a time to the DRS.

**An extract of Section (1) of Prong 2:**

```
constructing( Referent, Sentence_Syn, Existing_Drs, The_Drs, Flag,
                                          [s(analysed,analsyed)], Refcount) :-
  same(Refcount, Referent),
  (( same(Sentence_Syntax,[s( [np( Ref1, [s( Ref2, [vp( [v(Verb), Info], Ref3),_]),
                                     [_,gap:gap(np)]]),_],        This sentence
                     [vp([v(Verb2), Info2], Ref4),_]), [_, gap:nogap]]),   syntax is one of
                                              many produced by sentences
   Existing_Drs = drs([Refs, Cons]),                    that contain a NP gap, which
   Flag = main,                                       produce two different sentence
   drs_conc( drs( [[], [Ref1, Verb2, Info2, Ref4 ]]),       conditions.
                  drs( [Refs, Cons] ), Temp_Drs, main),
   drs_conc( drs( [[], [Ref2, Verb, Info, Ref3 ]]), Temp_Drs, Finished_Drs, main))
  ;
  ( same(Sentence_Syntax, [s( [np( Ref1, [vp( [v(V), Info]), _]),_],     This sentence syntax
                          [vp([v(V2), Info2], Ref2),_]),_]),   is one of many produced by
   Existing_Drs = drs([Refs, Conds]),                     relative clauses, which
   Flag = main,                                         produce two different
   drs_conc(drs( [ [], [Ref1,V,Info, Ref2]]),               sentence conditions.
                    drs( [Refs, Conds]), Temp_Drs, main),
   drs_conc( drs( [ [], [Ref2,V2,Info2]]), Temp_Drs, Finished_Drs, main))
  ;
  .                                                  This is the syntax
  .                                                  produced by start
```

```
( same(Sentence_Syntax, [not( Ref1,                    ______________for negated sentences
                              [vp(V,[np(Ref2, [vp(D,Ref3),F]),_]),G]),H]),
  Existing_Drs = drs( [Refs, Conditions]),
  is_last(Conditions, sub(not(SubRef, SubCond))),
  Flag = sub,
  drs_conc(drs( [ [Refs, Conditions]), drs([[], [Ref1,V,Ref2]]), Temp_Drs,sub),
  drs_conc( Temp_Drs,drs([[], [Ref2,D,Ref3]]), Finished_Drs, sub))
;
.
.
( same(Sentence_Syntax, [s(Ref2, [vp( [v(Verb), Info], Ref3), _] ), _]),        This is the
  Existing_Drs = drs([ Ref1, Cond1]),                                    same as Sentence_Syntax
  Flag = main,
  drs_conc(drs([[],[Ref2,Verb,Info,Ref3]]),drs( [Ref1, Cond1]), Finished_Drs,main))
```

Generally the procedure followed in this part of **constructing** is to assign *Refcount* the value of *Referent,* and search the or-loop for a sentence syntax the same as *Sentence_Syn.* When this is found the sentence condition or conditions within the *Sentence_Syn* are isolated and put into *Temp_Drs* before being incorporated into *Finished_Drs.*

Resuming the processing of sentence (1), *Sentence_Syntax* matches the syntax presented in the final clause of **section (1) prong 2** of **constructing** producing *New_Drs,* the output of **constructing**.

```
The_Drs = drs( [ [1,2], [ sue:1, [gen:fem, num:sing], man:2, [gen:male, num:sing],
                          2, loves, [fin:fin, trans:trans, num:sing, gap:nogap], 1] ])
```

*Refcount = 3,*

When this or-loop is exited, **section (2) prong 2** includes a search for suitable referents of any pronoun conditions i.e. suitable_referent:R of the *Finished_Drs.* If as in this case there are no such conditions present *The_Drs* is output, and this non-recursive part of **constructing** is exited. The treatment of pronouns within this implementation will be discussed in the next section.

#### 4.3.3.1.2 The treatment of pronouns and negated sentences.

*(2). A man loves Sue. She does not love him.*

To illustrate the resolution of pronouns within the semantic module the author will extend the *Finished_Drs* produced in section 4.3.3.1 which represents *A man loves Sue*:

```
drs( [[1,2], [ sue:1, [gen:fem, num:sing], man:2, [gen:male, num:sing],
                        1, loves, [fin:fin, trans:trans, num:sing, gap:nogap],2]]),
```

to include the new sentence representation *She does not loves him.* As before the **construction** rule is called but with the *Referent* as 4 taken from the *Refcount.*

```
construction( 3, [s( [np( [pro(she), [case:(+), gen:fem, num:sing]],
                  [case:(+), gen:fem, num:sing, gap:nogap]],
              [vp( [aux(does), [num:sing]], not,
                  [vp([v(love), [fin:nonfin,trans:trans,num:_]],
                      [np([pro(him), [case:(-), gen:male, num:sing, gap:nogap]]),
                      [fin:nonfin, trans:trans, num:_, gap:nogap]] ),
                  [fin:nonfin, trans:_, num:sing, gap:_]]),
                                  [num:sing, gap: nogap]],
```

Within the lexicon, *case* is given a positive value, + to signal that this pronoun should be placed in the subject position whereas a negative value, - indicates that the pronoun should be placed in the object position

Within **constructing,**
known as
*Existing_Drs* ------ drs([[1,2],[sue:1,[gen:fem,num:sing], man:2, [gen;male, num:sing],
1, loves, [fin:fin, trans:trans, num:sing], 2]]), *DRS,*
[(s(unanalysed,unanalysed)], *Refcount).*

The **construction** rule calls **constructing**, and since *Sentence_Syntax* is simple **start** will be invoked to introduce the referents and conditions to *DRS*. The pronoun trigger rule of **start** is fired, introducing a new referent 3 and conditions: suitable_referent:3, [gen:fem, num:sing] to *DRS*. Unlike other word categories, pronouns are not themselves introduced to *DRS* but are replaced by the phrase - **suitable_referent**, which acts as a catchall for all types of pronouns. Incorporating these new conditions into the *Existing_Drs* gives:

```
New_Drs = drs( [[1,2], [sue:1, [gen:fem, num:sing], man:2, [gen:male, num:sing],
                              2, loves, [fin:fin, trans:trans, num:sing], 1,
                              suitable_referent:3, [case:(+),gen:fem, num:sing] ]]).
```

```
New_Sentence_Syntax = [s( 3, [vp( [aux(does), [num:sing]], not,
                                  [vp([v(love), [fin:nonfin,trans:trans,num:_]],
                                      [np([pro(him),[case:(-), gen:male, num:sing]]),
                                          [case:(-), gen:male, num:sing, gap:nogap]]),
                              [fin:nonfin, trans:trans, num:_, gap:nogap]] ),
                          [fin:nonfin, trans:_, num:sing, gap:_]]), [num:sing, gap: nogap]],
```

```
Condition = [s(analysed, unanalysed)]
```

Incrementing *Referent* and passing the new sentence syntax into the second **constructing** rule, calls the negation **trigger**:

```
(trigger(cr_neg, [s( 3, [vp( [aux(does), [num:sing]], not,
                         [vp([v(love), [fin:nonfin,trans:trans,num:_]],
                             [np([pro(him), [case:(-), gen:male, num:sing, gap:nogap]]),
                             [fin:nonfin, trans:trans, num:_, gap:nogap]] ),
                         [fin:nonfin, trans:_, num:sing, gap:_]]), [num:sing, gap: nogap]],
         [s(X, [vp([aux(A), [num:B]], not, Y),_]),Z], drs( sub(not([],[]))), DRS,
         [not( X,Y ), Z], New_Sentence_Syn, sub, NewFlag),

 NewCondition = Condition,

 drs_conc(Existing_Drs, DRS, New_Drs, sub))
```

This rule:

- introduces the empty sub-DRS template to the *Existing_Drs,* and sets the *NewFlag* to sub, so that any new referents and conditions will be added to the sub-DRS.

In this case *Existing_Drs* = drs([ [1,2], [ sue:1, [gen:fem,num:sing],
man:2, [gen:male,num:sing],
2, loves, [fin:fin, trans:trans, num:sing], 1,
suitable_referent:3, [case:(+),gen:fem, num:sing],
sub(not( [], [] )) ]]).

- changes the *Sentence_Syntax* by removing the **auxiliary not** syntactic information. The new positive sentence *New_Sentence_Syntax* is encased in a **not** i.e.[not(NP,VP)] signalling a negated sentence.

  *New_Sentence_Syntax* = [not(3, [vp( [v(love), [fin:nonfin, trans:trans, num:_]],
  [np([pro(him),
  [case:(-),gen:male,num:sing,gap:nogap]]),
  [fin:nonfin, trans:trans, num:_, gap:nogap]],
  [num:sing, gap:nogap]]

- does not change the value of *Condition,* as neither the NP or VP part of the sentence has been analysed. *Condition* = [s(analysed,unanalysed)], as no real changes have been made the value of *Referent* stays the same.

The recursive **constructing** rule is called again with *Condition* as [s(analysed,unanalysed)]. This time none of the trigger rules will fire, as the *Sentence_Syntax* is headed by **not**. The **sentence_in_sentence** rule located in the final clause of the **start** trigger or-loop, looks after these sentence types in the same way as **start** producing a new DRS. In this case the *New_Drs*, *New_Sentence_Syn* and *Condition* produced are:

*New_Drs* = drs([ [1,2], [ sue:1, [gen:fem,num:sing],
man:2, [gen:male,num:sing],
2, loves, [fin:fin, trans:trans, num:sing], 1,
suitable_referent:3, [case:(+),gen:fem, num:sing],
sub(not( [4], [suitable_referent:4,
[case:(-),gen:male,num:sing]] )) ]]).

*New_Sentence_Syn* = [not( 3, [vp( [v(love), [fin:nonfin, trans:trans, num:_]],4),
[fin:nonfin, trans:trans, num:_, gap:nogap]],
[num:sing, gap:nogap]]

*Condition* = [s(analysed, analysed)]. with *Flag* = sub

The second prong of **constructing** can now be called as *Condition* is [s(analysed, analysed)]. *Flag* is sub, so any referents and conditions introduced at this stage will be put in the sub-DRS. that was last added to the existing DRS. The first section prong (2) of **constructing** produces:

```
Finished_Drs = drs([ [1,2], [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                              2, loves, [fin:fin, trans:trans, num:sing], 1,
                              suitable_referent:3, [case:(+),gen:fem, num:sing],
                              sub(not( [4], [suitable_referent:4,
                                              [case:(-),gen:male,num:sing]
                                     3 love [fin:nonfin, trans:trans, num:_], 4] )) ]]).
```

Finally the suitable_referent:_ conditions of *Finished_Drs* need to be removed and replaced with suitable referents that occur in *Finished_Drs.* This is achieved by the section (2) of prong 2 of **constructing**

**prong 2 section(2)**

```
same(Finished_Drs, drs([_,The_Conds])), % picks the condition list  out of Finished_Drs
                                         % and calls it The_Conds.

% is_last checks to see if the negated sub-DRS template was the last thing entered in
% TheConds. The or-loop collects all the conditions to be searched into one variable
% The_Conditions.

   (( is_last(sub(not(_,SubCond)), TheConds),
     % member_ref removes the sub-DRS template from TheConds to give The_MainC
     member_ref( sub(not(_,SubCond)), TheConds, [], The_MainC,_,_),
     % Now need to rejoin the sub-DRS conditions SubCond with TheMainC
     % resulting in The_Conditions so an extensive search for the
     % condition suitable_referent:_ can take place.
     conc( TheMainC, SubCond, The_Conditions) );
    ( not( is_last(sub(not(_,_)), The_Conds)),
     % The sub-DRS template was not the last thing entered, just call the main
```

```
        % conditions The_Conditions.
        same( The_Conds, The_Conditions) )),

    % find, counts the number of unresolved pronouns, suitable_referent:_ conditions present.
    % in The_Conditions
    find(The_Conditions, suitable_referent:_,0,The_Count),
    % If Count > 0, replace these referents.

    (( not(same( The_Count, 0)),
      % There are suitable_referent:_ conditions to be replaced so call change_the_drs
      % to replace them and call the new DRS, The_Drs.

      change_the_drs( The_Count, Finished_Drs, NewFlag, Referent,The_Drs ));

     (not(same( The_Count, 0)),!,
      % If change_the_drs fails, it is because the sentence is complex and all
      % suitable_referent:_ conditions are replaced within the complex rules.
      not(change_the_drs( The_Count, Finished_Drs, NewFlag, Referent,The_Drs));

     (same(The_Count, 0),!,
      % There are no conditions to replace so The_Drs is Finished_Drs.
      same(Finished_Drs, The_Drs ) )).
```

Continuing the processing of sentence (2) through section (2) of prong 2:

```
    The_Conds = [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                        2, loves, [fin:fin, trans:trans, num:sing], 1,
                        suitable_referent:3, [case:(+),gen:fem, num:sing],
                        sub(not( [4], [suitable_referent:4, [case:(-),gen:male,num:sing]
                                    3 love [fin:nonfin, trans:trans, num:_], 4] )) ]]).
```

As we can see the negated sub-DRS is the last thing to have been added to *The_Conds*, **is_list** recognises this and calls **member_ref** to divide *The_Conds* into sub-DRS conditions *SubCond*, and the main conditions *The_MainC*:

```
   SubCond = [suitable_referent:4, [case:(-),gen:male,num:sing],
                                    3, love, [fin:nonfin, trans:trans, num:_], 4]
```

```
The_MainC = [sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                         2, loves, [fin:fin, trans:trans, num:sing], 1,
                                  suitable_referent:3, [case:(+),gen:fem, num:sing]]
```

The next step is to join these condition lists up into *The_Conditions* having eliminated the sub-DRS template. *The_Conditions* contains all the suitable_referent:_ conditions to be resolved and referents accessible from the current position in the DRS are available to resolve them.

```
The_Conditions = [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                         2 loves, [fin:fin, trans:trans, num:sing], 1
                         suitable_referent:3, [case;(+),gen:fem, num:sing],
                         suitable_referent:4, [case:(-),gen:male,num:sing]
                         3 love [fin:nonfin, trans:trans, num:_], 4]
```

**find** counts the number of suitable_referent:R conditions present in *The_Conditions,* and *The_Count* is incremented by two each time such a condition is found to account for the condition and its feature structures. Passing *The_Conditions* into **find** results in *The_Count* of four. All replacements of suitable_referent:R conditions occurs in **change_the_drs** which produces the final DRS output of **constructing.** This rule is recursive with each iteration decreasing *The_Count* by two, until all suitable_referent:R conditions have been replaced in *Finished_Drs.*

```
change_the_drs( 0, Finished_Drs,_, _, The_Drs) :-
                  same(Finished_Drs, The_Drs).

change_the_drs( Count, Finished_Drs, Flag, Referent, Referent,The_Drs) :-
                  new_drs(Finished_Drs, Flag, _,_ , continue, Referent, TempDrs),
                  NewCount is Count - 2,
                  change_the_drs(NewCount, TempDrs, Flag, Referent, The_Drs).
```

**change_the_drs** recursively calls **new_drs** to resolve the pronouns until *Count* is zero, when the new information is passed into *The_Drs.* One iteration of **new_drs** succeeds in resolving one set of pronoun conditions with the help of **suitable_ref** and **changing_DRS**. One call to **change_the_drs** can account for many different DRSs being produced as all possible suitable referents can replace a suitable_referent:R condition must be allowed to do so producing many possible DRSs.

```
new_drs( Finished_Drs, _, _, [], end,_, The_Drs ) :-
    same( The_Drs, Finished_Drs),!.

new_drs( Finished_Drs, Flag, DRS, Tail, Loop Referent, The_Drs ) :-

    suitable_ref(  Finished-Drs, Suitable_list, Rfeaturestructures, Referent, Head, Tail).

    % suitable_ref picks out all the suitable referents of the DRS that could replace a
    % suitable_referent:_ condition, and puts them into Suitable_list.

    (( same( Suitable_list, []),!, fail   % There are no suitable_referent:_ conditions in
                                          % Finished_Drs  to be replaced so new_drs fails.
     );

     ( changing_DRS( Suitable_list, Rfeaturestructures, Finished_Drs, DRS, Flag,
                                                                        Tail),
        % changing_DRS picks one suitable referent of Suitable_list and replaces the
        % specified suitable_referent:_ condition with it.
        (( same( Rest, [] ),
           % Rest, Tail and Loop control iterations of new_drs
           % when Rest = [], one suitable_referent:_ condition has been replaced.
          new_drs( DRS,_,_,[],end,_, The_Drs ) );

         ( same( Loop, continue,),
           new_drs( Drs, Flag, _, _, continue, _, The_Drs) ) ) ) ).
```

The **suitable_ref** rule:

**suitable_ref**( *Finished_Drs, Suitable_list, Rfeaturestructures, Referent, Head, Tail).*

takes the *Finished_Drs* and searches to see if there are any suitable_referent:R conditions present in the condition list. If these conditions are present the suitable referents (i.e. those referents of the DRS with the same feature structures as suitable_referent:R) are isolated and put into *Suitable_list.* If there are no suitable_referent:R conditions present in the existing DRS, *Suitable_list* is assigned to [].

If a suitable_referent:R condition represents a plural pronoun i.e. the feature structures associated with the condition are: [case:_, gen:_, num:plur]:

- the condition list of the *Finished_Drs* is searched to see if there are any existing non-atomic referents i.e. **Referent: Setdeclaration** - *6:[table:1, book:2]* or **Plural definite NP** conditions, *the_man(1), [gen:male, num:plur]*, present. If there are any such referents present they are put into *Temp_list,* and if no such conditions are found *Temp_list* is given the value of [].

- **summation** is applied summing all the human referents into one set, and non-human elements into another set, with the two sets placed in *Sum_list.* These synthesised sets represent the antecedent of a plural pronoun.

- finally *Sum_list* and *Temp_list* are concatenated by **drs_conc** to give the set of non-atomic suitable referents, *Suitable_list.*

*Suitable_list* contains a set of suitable referents for any one suitable_referent:R condition of the DRS. This list is accessed element by element, with each element taking the place of the suitable_referent:R condition specified by **suitable_ref** yielding R:Element, i.e.

*Suitable_list* = [3,4,5] and *suitable_referent:6*,
replacing this condition results in three possible conditions

6:3, 6:4, and 6:5
all of which must be entered into a DRS producing three possible DRSs.

**changing_DRS**( *Suitable_list, Rfeaturestructures, Finished_Drs, DRS, Flag, Tail).*

**changing_DRS** replaces suitable_referent:R in the *Finished_Drs* with an element from the *Suitable_list,* provided this element is not in the *UnavailableReferent* list. The *UnavailableReferent* list contains the list of referents of the DRS that must be excluded from consideration as suitable referents. These elements include other suitable_referent:R conditions, and those involved in sentence conditions with the R of the present suitable_referent:R condition. If these referents were allowed to be considered as suitable referents, we would be allowing sentences like *john likes him* to mean *john likes john* where *john* represents the same person in each case.

*Suitable_list* can have four different values within **changing_DRS**:

- the empty list, [] implying that there are no suitable referents in the DRS and therefore failure results as the discourse the DRS represents is invalid.

- a set representing a plural pronoun which contains two non-atomic sets - human and non-human. An element of the set list is taken to replace the suitable_referent part of the plural pronoun condition. Backtracking will select another element of the list.

- a plural referent representing a plural definite NP - more detail in section 4.3.3.1.3.

- singular referents that are chosen one at a time to replace the suitable_referent part of suitable_referent:R.

Within **changing_DRS, mixup**(*Element, Suitable_list*), picks an element from the *Suitable_list* to represent the suitable_referent:R. The next step replaces the suitable_referent:R with *Element*:R to give the new condition which is joined with the rest of the conditions of the DRS being developed, and **changing_DRS** and **new_drs** are exited.

Applying all this to the example presently being processed means processing continues within **section 2, prong 2** of **constructing** where *The_Count* is four, and **change_the_drs** is called to replace the suitable_referent:R conditions with suitable referents.

```
                 The_Count                      Finished_Drs
                     |                               |
change_the_drs( 4, drs([ [1,2], [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                            2,loves, [fin:fin, trans:trans, num:sing], 1
                            suitable_referent:3, [gen:fem, num:sing],
                            sub(not( [4], [suitable_referent:4, [gen:male,num:sing]
                            3 love [fin:nonfin, trans:trans, num:_], 4] )) ]]),
                                                                    sub, The_Drs ).
```

- **change_the_drs** calls **new_drs** which replaces one suitable_referent:R condition in the DRS.

- **suitable_ref** forms the *Suitable_list* of referents for suitable_referent:3 which is the first pronoun condition to be encountered within the DRS: *Suitable_list* = [sue:1].
- **changing_DRS** replaces the suitable_referent:3 with 3:1 to give the *TempDrs,* and **new_drs** is exited *The_Count* is decreased by two to give *NewCount = 2.*

```
TempDrs = drs([ [1,2], [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                          2, loves, [fin:fin, trans:trans, num:sing], 1,
                          3:1, [gen:fem, num:sing],
                          sub(not( [4], [suitable_referent:4, [gen:male,num:sing],
                                   3, love, [fin:nonfin, trans:trans, num:_], 4] )) ]])
```

- Since *NewCount* is not zero there are still suitable_referent:R conditions to be replaced in the *TempDrs* and **new_drs** is called again to replace them yielding *The_Drs* which is output by **constructing**.

```
The_Drs = drs( [ [1,2], [ sue:1, [gen:fem,num:sing], man:2, [gen:male,num:sing],
                          2, loves, [fin:fin, trans:trans, num:sing], 1,
                          suitable_referent:3, [gen:fem, num:sing],
                          sub(not( [4], [4:2, [gen:male,num:sing],
                          3, love, [fin:nonfin, trans:trans, num:plur], 4] )) ]]).
```

In this case there was only one element present in *Suitable_list* implying that there is only one possible *The_Drs.* This is not always the case as the next example will illustrate:

(4.3). *John loves Ulysses. Anne loves Pollyanna. They rotate.*

The author assumes that *DRS* for (4.3) has been produced by **constructing**:

```
DRS = drs( [[1,2,3,4,5], [john:1, [gen:male, num:sing], ulysses:2, [gen:neut, num:sing],
                          1, loves, [fin:fin, trans:trans, num:sing], 2,
                          anne:3, [gen:fem, num:sing], pollyanna:4, [gen:neut, num:sing],
                          3, loves, [fin:fin, trans:trans, num:sing], 4,
                          suitable_referent:5, [case:(+), gen:_, num:plur],
                          5, rotate, [fin:fin,trans:intrans,num:_]] ]).
```

Processing continues within section (2), prong 2 of **constructing**, as any suitable_referent:R conditions of *DRS* need to be resolved.

- **suitable_ref** is called producing the *Suitable_list* for suitable_referent:5. There is no existing non-atomic referent present in the condition list, so the non-atomic referents are synthesised by **summation**.

```
Suitable_list = [ [non-at, 6: [anne:3, john:1], 5:6, [case:(+), gen:_, num:plur]],
                  [non-at, 6: [pollyanna:4, ulysses:2],
                                    5:6,[case:(+),gen:_, num:plur]] ]
```

The non-at is placed at the head of the set list to indicate that what is to follow is a set.

- **changing_DRS** introduces one of the suitable referents of *Suitable_list* at a time to replace the suitable_referent:5 condition to give two possible DRSs as output of **constructing.**

```
The_Drs = drs( [ [1,2,3,4,5,6],
                 [ john:1, [gen:male, num:sing], ulysses:2, [gen:neut, num:sing],
                   1, loves,[fin:fin, trans:trans, num:sing], 2
                   anne:3, [gen:fem, num:sing], pollyanna:4, [gen:neut, num:sing],
                   3, loves, [fin:fin, trans:trans, num:sing], 4,
                   6: [ anne:3, john:1], 5:6, [case:(+), gen:_, num:plur]],
                   5 rotate, [fin:fin, trans:intrans, num:_] ]]).

The_Drs = drs( [ [1,2,3,4,5,6],
                 [john:1, [gen:male, num:sing], ulysses:2, [gen:neut, num:sing],
                   1, loves,[fin:fin, trans:trans, num:sing], 2
                   anne:3, [gen:fem, num:sing], pollyanna:4, [gen:neut, num:sing],
                   3, loves, [fin:fin, trans:trans, num:sing], 4,
                   6: [ pollyanna:4, ulysses:2], 5:6, [case:(+), gen:_, num:plur]],
                   5 rotate, [fin:fin, trans:intrans, num:_] ]]).
```

This system combines everything possible to give every reading possible for every sentence of the discourse, because of this the implementation may output a small amount nonsensical readings. These readings are a result of not having enough world knowledge built into the system, and nonsensical results will have to be eliminated by the human operator when the results are output.

#### 4.3.3.1.3. The treatment of definite NPs within simple sentences.

**Singular definite NPs.**

Within this semantic module, singular definite NPs are treated like pronouns in that we presume that they are familiar entities. The author appreciates that definite NPs can be used in a multitude of ways, but asserts that within this implementation which uses a small fragment of English, an attempt can be made to regard them as familiar. The NPs are treated within **start**, which presents two different triggers for a definite NP:

**(1).** which introduces the **suitable_referent:R** condition to the DRS being developed, and will result in a search for the most suitable referent.

**(2)** if **(1)** fails, trigger **(2)** which introduces the condition definite NP:Referent (i.e. the_man:4), will be fired.

*(3). Bill knows a woman. The woman owns a book.*

```
DRS = drs([[4,3,2,1],   [book:3,[gen:neut,num:sing],
                          woman:2,[gen:fem,num:sing],
                          bill:1,[gen:male,num:sing],
                          1, knows,[fin:fin,trans:trans,num:sing], 2,
                          4, owns, [fin:fin,trans:trans,num:sing], 3,
                          4:2, [gen:fem,num:sing] ]] ) ? ;
no
```

No problem finding a suitable referent for the definite NP above, so trigger (2) does not fire. If Sentence (4). is processed when the Existing_DRS is empty, trigger (2) would fire:

*(4). Bill knows the woman.*

```
DRS = drs([[4,3,2,1],
                         [the_woman:2,[gen:fem,num:sing],
                          bill:1,[gen:male,num:sing],
                          1, knows,[fin:fin,trans:trans,num:sing], 2] )? ;
no
```

**Plural definite NPs - *they* and *them*.**

**plural_definiteNPs**(*Referent, NewReferent, Sentence_Syntax, New_Sentence_Syn,*
*Cond, NewCond, Existing_DRS,*
*Flag, NewFlag, New_DRS* ).

These NPs are treated by the rule **plural_definiteNPs** contained in the or-loop of **prong 1** of **constructing**. This rule, like **start,** contains an or-loop of triggers, and actions to be performed if the said triggers are fired. As discussed in chapter 3 section 3.4, depending where the NP is situated in a sentence, a sentence can have a collective, distributive, or both collective and distributive readings producing many possible DRSs for one *Sentence_Syntax*, all of which must be generated by **plural_definiteNPs**. A typical trigger or-clause of **plural_definiteNPs** which produces both the collective and distributive readings is shown below:

```
(( % Is the Sentence_Syntax the same as the sentence syntax presented?, if so place the plural
  % noun in a non-atomic DRS condition.
  trigger(cr_id, Sentence_Syntax,
              [s([np([ det(the), [num:plur] ],[ n(B), [gen:C,num:plur]],
                        [ optrel( [] ),_]),_)], W), Z],
          drs([ [Referent], [Referent:[B:Referent, [gen:C, num:plur]]]]), DRS,
          _, _Flag, NewFlag),

  % In this case the NP part of Cond has been analysed
  Cond = [s(unanalysed,G)],
  TempCond = [s(analysed,G)],
  % Both collective and distributive readings are available with this Sentence_Syntax.
  % plural_definiteNPs outputs one of these readings at a time , with the other one
  % available by backtracking.
    (( % Collective Reading - this reading is represented by introducing a non-atomic referent
      % to represent the plural noun, as achieved in DRS with processing continuing as
      % normal after this - section 3.4.2.1
      same(New_Sentence_Syn, [s(R,W),Z]),  % remove the NP information from
                                           % Sentence_Syntax

      NewCond = TempCond,   % The NewCond simply carries the information
                            % that the NP part of Sentence_Syn has been analysed,
                                % and processing will continue as normal outside
                                % **plural_definiteNPs** and within **constructing**.

**drs_conc**( *Existing_DRS, DRS, New_DRS, Flag*))
;
( % Distributive Reading - The non-atomic set is *DRS,* we now intend to distribute over
 % the set producing a duplex condition- section 3.4.2.2
 *Ref2* **is** *Referent + 1*,   % Ref2 is going to be the element picked out of set *Referent.*
 **same**(*New_Sentence_Syn,* [s(Ref2,W),Z]),     % introduce *Ref2* to the sentence syntax
                                                  % so that it will be involved in future
                                                  % conditions.

 **same**(*DRS,* drs([[*Referent*], *TheSet*])), % isolate the set list - *TheSet.*
 **same**(*DRS1*,drs([[],[*Ref2:TheSet*]])),  % mimicking x e X the first part of the
                                             % distributive sub-DRS template
                                             % chapter 3, section 3.4.2.2
                                             % introduce *Ref2* : *TheSet.*

 *Ref3* **is** *Ref2 + 1*,        % the set is called *Referent, Ref2* is an element of the set
                            % and future processing will lead to *Ref3.*

 % Need to analyse the rest of the sentence to form *DRS2* of the duplex condition that
 % is the action every element, *Ref2* of the set performed.
 **construction**(*Ref3,New_Sentence_Syn,Existing_DRS,DRS2,TempCond,*
                                                                  *NewReferent*),
 % Putting the duplex condition together.
 **same**(*TempDRS1*,duplex(*DRS1*,every,*Ref2*,*DRS2*)),
 % The original set declaration, *Referent:TheSet* must be added before the
 % *Existing_DRS.*
 **drs_conc**(*Existing_DRS, DRS, Temp_mainDRS*),

 % The main condition information must be joined with the duplex condition.

 **drs_conc**(*Temp_mainDRS, TempDRS1, The_drs,_*),

 % Need to remove any suitable_referent:R conditions replacing them with suitable
% referents, and outputting New_DRS.

changing_conditions( The_drs, Flag, New_DRS, Ref4),

% All processing of the sentence is complete, so assign NewCond the value
% finished - so that constructing will be exited.
NewCond = finished)))
```

This treatment of plural NPs which introduce collective and distributive readings will be illustrated by example (5) which is processed by **construction** where the *Existing_DRS* is empty.

*(5). The men love a woman that they know.*

```
construction( 1,     [s([np([det(the),[num:plur]],[n(men),[gen:male,num:plur]],
                      [optrel([]),_H]),
                    [case:_I,gen:male,num:plur,gap:nogap]],
                  [vp([v(love),[fin:nonfin,trans:trans,num:plur]],
                      [np([det(a),[num:sing]],[n(woman),[gen:fem,num:sing]],
                       [optrel([relpron(that),[gen:fem,num:_F]],
                           [s([np([pro(they),[case:(+),gen:_D,num:plur]]),
                              [case:(+),gen:_E,num:plur,gap:nogap]],
                              [vp([v(know),[fin:nonfin,trans:trans,num:plur]],
                                     [np(gap),
                                   [case:_A,gen:_B,num:_C,gap:gap(np)]]),
                              [fin:nonfin,trans:trans,num:plur,gap:gap(np)]]),
                           [num:plur,gap:gap(np)]]),
                         [gen:fem,num:sing]]),
                    [case:_G,gen:fem,num:sing,gap:nogap]]),
                     [fin:nonfin,trans:trans,num:plur,gap:nogap]]),
                  [num:plur,gap:nogap]],

      drs([[],[]]), DRS, [s(unanalysed,unanalysed)], _).
```

This call to **constructing** produces seven different ***DRSs***, each representing a different reading of sentence (5).

**Collective reading (1).** The men love a woman that they know.

```
DRS = drs([ [2,1,3],
            [woman:2, [gen:fem,num:sing],
             1:[men:1,[gen:male,num:plur]],
             1, love, [fin:nonfin,trans:trans,num:plur], 2,
             3, know, [fin:nonfin,trans:trans,num:plur], 2,
             3:1, [case:_J,gen:_D,num:plur]]] )? ;
```

**Distributive reading (1).** Every man out of the set of men loves a woman the set of men know.

```
DRS = drs([ [1],
            [ 1:[men:1,[gen:male,num:plur]],
              duplex(drs([ [], [ 2:[1:[men:1,[gen:male,num:plur]] ]]]),
                     every,2,
                  drs([[3,4],[woman:3,[gen:fem,num:sing],
                       2, love, [fin:nonfin,trans:trans,num:plur], 3,
                          4, know, [fin:nonfin,trans:trans,num:plur], 3,
                          4:1, [case:_J,gen:_D,num:plur]]]))]]) ? ;
```

**Distributive reading (2).** Every man out of the set of men loves a woman that he knows.

```
DRS = drs([[1],
            [ 1:[men:1,[gen:male,num:plur] ],
              duplex(drs([[],[2:[1:[men:1,[gen:male,num:plur]]]]]),
                     every,2,
                  drs([[3,4],[woman:3,[gen:fem,num:sing],
                          2, love,[fin:nonfin,trans:trans,num:plur],3,
                          4,know,[fin:nonfin,trans:trans,num:plur],3,
                          4:2,  [case:_J,gen:_D,num:plur]]]))]]) ? ;
```

**Collective reading (2).** The set of men love a women that they know

```
DRS = drs([ [2,1,3],
            [ 2:[men:2, [gen:male,num:plur]],
              woman:1, [gen:fem,num:sing],
              2, love,[fin:nonfin,trans:trans,num:plur],1,
              3, know,[fin:nonfin,trans:trans,num:plur],1,
              3:2, [case:_J,gen:_D,num:plur]]]) ;
```

The referents are declared in a different order to Collective reading (1).

```
DRS = drs([ [2,1],                                              ________Distributive reading (3).
             [ 2:[men:2,[gen:male,num:plur]],                           Same meaning
              woman:1,[gen:fem,num:sing],                               as Distributive reading
             duplex(drs([[],[3:[2:[men:2,[gen:male,num:plur]]]]]),      (1), but conditions
                      every,3,                                          are declared in
                   drs([ [1,4], [woman:1,[gen:fem,num:sing],            a different order
                              3,love,[fin:nonfin,trans:trans,num:plur],1, which
                              4,know,[fin:nonfin,trans:trans,num:plur],1,  may lead
                              4:2, [case:_J,gen:_D,num:plur]]]))]]) ? ;   to different
                                                                        readings in the future

DRS = drs([ [2,1],                                              ________Distributive reading (4).
        [ 2:[men:2,[gen:male,num:plur]],                        Same meaning as dist reading(2)
            woman:1,[gen:fem,num:sing],                            but conditions are declared in
            duplex(drs([ [], [ 3:[2:[men:2,[gen:male,num:plur]]] ]]),  a different order.
                      every,3,
                  drs([ [1,4], [woman:1,[gen:fem,num:sing],
                             3, love, [fin:nonfin,trans:trans,num:plur],1,
                             4, know,[fin:nonfin,trans:trans,num:plur],1,
                             4:3, [case:_J,gen:_D,num:plur]]]))]]) ? ;

DRS = drs( [ [3,1,2],                                           _________Collective reading (3).
             [ 3:[men:3, [gen:male,num:plur]],                  Same meaning as collective
              woman:1,[gen:fem,num:sing],                       reading (2) but different
              3, love, [fin:nonfin,trans:trans,num:plur],1,     referent declarations.
              2, know, [fin:nonfin,trans:trans,num:plur],1,
              2:3, [case:_J,gen:_D,num:plur]]])?;

no.
```

### 4.3.3.2. Analysing complex sentences.

The author will now discuss how complex sentences are treated within the semantic module. These sentences are analysed within **prong 1** of **constructing,** where inside the or-clauses each complex sentential component calls a complex rule to break the sentence into a simple form so it can be treated by **start** and **change_the_drs**. Each complex rule will be discussed and analysed by taking them in the order in which they occur in **prong 1**.

### 4.3.3.2.1. The Quantified sentence - the duplex condition.

The sub-DRS template for quantified sentences is of the form:

**duplex**(*DuplexDRS1,Quantifier,Referent,DuplexDRS2*).

and is introduced by the **duplex** rule within **constructing**.

**duplex**( *Referent, Existing_Drs, Sentence_Syntax, Flag, Condition, Finished_Drs, NewCondition*) :-

% produce the duplex DRS.
**every_rule**(*Referent, Sentence_Syntax, Existing_Drs, DuplexDRS, Flag, Condition, NewCondition,NewRef*),
% Resolve any suitable_referent:R conditions.
**changing_conditions**( *DuplexDRS, Flag, Temp_DRS, NewRef* ),

% A duplex condition can produce a non-atomic set by abstraction. This set may
% be needed at a later stage in processing, the author chooses to produce the set
% now, inserting it as a non-atomic condition into the *New_DRS*.

*Referent_two* is *NewRef* + 1,

**same**(*Temp_DRS*, drs([*Ref,Conditions*])),

**member_ref**(duplex(A,B,C,D), *Conditions*, [],_,_,_), % isolate the duplex
% condition of *Conditions*.

% **abstraction** produces a *NewCondset* by putting all the conditions
% of the duplex condition into a new sub-DRS- *Referent_two*:*NewCondset*.

**abstraction**(duplex(A,B,C,D), *Referent_two, NewCondset*),

**conc**(*Conditions*,[*NewCondset*],*NewConditions*), % Add the set to *Conditions*.

**same**(*New_DRS,* drs([*Ref,NewConditions*])). % Add the new condition list to
% *New_DRS*

Like **start, every_rule** contains a set of triggers and actions to be performed if these triggers are fired. The difference being that a quantified sentence introduces two different DRSs joined by a quantifier and main referent Fig 4.10, 4.11. Within **every_rule**, the triggers are structured as follows:

- check to see if *Sentence_Syntax* contains a determiner noun category where the determiner is a quantifier.

- Introduce a condition and referent to *DuplexDRS1* representing the noun.

- The *Referent* replaces the NP of *Sentence_Syntax* that contains the recognised determiner noun categories to give *TempSentence_Syn.*

- The appropriate part of *Condition* is changed to analysed to give *TempCondition*, and *Referent* is incremented to give *Referent_two.*

- *DuplexDRS2* still needs to be generated, a call to **constructing** will generate it, the call is of the form:

  **constructing**( *Referent_two, TempSentence_Syn,* drs([[],[]]), *DuplexDRS2, Flag,*
  *TempCondition).*

  Complex *Sentence-Syntax* trigger rules call **sub-sentence** which copes with NP gaps and complex relative clauses.

- **same**( *DRS,* duplex(*DuplexDRS1,Quantifier,Referent,DuplexDRS2)* ) - produces the *DRS* which is incorporated into the *Existing_Drs* condition list by calling **drs_conc** producing the *DuplexDRS.*

**every_rule** produces the *DuplexDRS.* The next step is to replace any suitable_referent:R conditions present in the respective DuplexDRSs, which is achieved by **changing_conditions. changing_conditions** takes into account which suitable referents are accessible from either of the DuplexDRSs, and forms a *TemporaryDRS* which contains all suitable referents accessible from the

DuplexDRS we are currently looking at, where **cond_drs** resolves all suitable_referent:R conditions within this DuplexDRS by calling **change_the_drs**.

- *DuplexDRS* contains the duplex condition and other main DRS conditions. **member_ref** isolates these different DRS conditions:

  The duplex condition: duplex(drs([*Ref1,Cond1*]), *Quantifier, MainRef,* drs([*Ref2,Cond2*]) )

  The main conditions and referents: *Conds Refs.*

  The suitable_referent:R conditions of DuplexDRS1 have access to suitable referents of *DuplexDRS1* and *Conds* so **cond_drs** is called with a *TemporaryDRS* containing the main referents and conditions, and *DuplexDRS1,* with the conditions are resolved to produce *NewDuplexDRS1*.

  DuplexDRS1

  |

  **cond_drs**( drs([*Refs, Conds*]), drs([*Ref1, Cond1*]), *Flag, NewDuplexDRS1,NewRef* ).

  |

  *Refs* are the main referents of DuplexDRS, and *Conds* are the main DuplexDRS conditions.

  The suitable_referent:R conditions of DuplexDRS2 have access to the suitable referents of DuplexDRS2, DuplexDRS1 and the main *Conds* and *Refs.* The referents *Refs* and *Ref1* are concatenated into *TempRef,* and *Conds* and *Cond1* are concatenated into *TempCond* encompassing all of the suitable referents outside *DuplexDRS2*:

  **cond_drs**( drs(*[TempRef,TempCond*]), drs([*Ref2, Cond2*]), *Flag, NewDuplexDRS2, NewRef*).

  The resolved DRSs of *NewDuplexDRS1,* and *NewDuplexDRS2* are joined into a new duplex condition *Temp_DRS* as **changing_conditions** is exited.

- The next step involves an application of **abstraction** to the duplex condition of *Temp_DRS* producing a non-atomic set *NewCondset* which contains all the conditions of the duplex condition. This set may be needed in future processing, and is added to the condition list of *Temp_DRS* producing the *NewConditions.*

- Finally the main *Refs* of *Temp_DRS,* and the list *NewConditions* form the *New_DRS.*

*Condition* is changed to *finished* within **every_rule**, to indicate that all processing has been completed within this or-clause. This occurs within all complex rule or-clauses, activating the third prong of **constructing,** and exiting the module where this prong simply outputs the *New_DRS* formed.

The author will now illustrate the production of a ***DRS*** for the quantified sentence (6). As aspects of scope are ignored within the semantic module only one scope representation of (6) is produced, with the two ***DRS*** representations resulting from backtracking which rearranges the application of semantic rules, and therefore the introduction of conditions to ***DRS***.

*(6). Every man loves a woman.*

Assuming that the *Existing_DRS* of **construction** is empty, the following *DRSs* are produced:

```
DRS = drs( [ [3], [duplex( drs( [[1], [ man:1, [gen:male, num:sing] ]]),
                              every, man,
                         drs( [[2], [ woman:2, [gen:fem, num:sing],
                                     1, loves, [fin:fin, trans:trans, num:sing], 2 ]])),
                         3: [ 1: [ man:1, [gen:male, num:sing],
                         woman:2, [gen:fem, num:sing],             ______ The non-atomic
                         1, loves, [fin:fin, trans:trans, num:sing], 2 ] ]  ]))?;     set declaration.
                                                                          The set is called 3.

DRS = drs( [ [1,4], [ woman:1, [gen:fem, num:sing],
                      duplex( drs( [[2], [ man:2, [gen:male, num:sing] ]]),
                              every, man,
                           drs( [[ ], [ 2, loves, [fin:fin, trans:trans, num:sing], 1 ]])),
                          4: [ 2: [ man:2, [gen:male, num:sing],
                          2, loves, [fin:fin, trans:trans, num:sing], 1 ] ]  ]))?;

no.
```

#### 4.3.3.2.2. Conjunction and disjunction NPs within sentences.

**conj_and_disjunction**( *Referent, Sentence_Syntax, Existing_Drs, New_Drs, Flag, Refcount).*

**conj_and_disjunction** works on NP conjunction and disjunction. The conjunctive and disjunctive NPs are found at the beginning or end of the *Sentence_Syntax,* but wherever they are found they need to be isolated from the rest of the sentence, so that the remaining sentence syntax can be analysed once, and not every time a conjunct or disjunct is encountered, viz. chapter 3, sections 3.3.6 & 3.3.7.

- The non-conjunctive or disjunctive part of *Sentence_Syntax* - *NonDisjConj,* is separated from the disjunctive or conjunctive part.

  *Connector* is found in *Sentence_Syntax* and is *or* or *and* depending on whether the *Sentence_Syntax* is a disjunctive or conjunctive NP.

- If *NonDisjconj* contains a transitive verb, the referents and conditions are introduced into a DRS by passing *NonDisjConj* into **put_in_right_place** which develops two DRSs *DisjDrs* and *StartDrs.*

  If a proper noun is present in the *NonDisjConj* information the referents and conditions introduced are put into *StartDrs,* which holds all principal DRS information. If *Connector* is *or* all other information is put into *DisjDrs*, but if *Connector* is *and* all information is put into *StartDrs*. This is because conjunction does not introduce sub-DRSs whereas disjunction does (chapter 3 section 3.3.6).

  *NonDisjConj* has been analysed producing a *NewNonDisjConj* with the referents and conditions introduced by it placed in *StartDrs* and *DisjDrs* respectively.

- Sentences are synthesised with NP disjunction or conjunction joined with the analysed *NewNonDisjConj.* Processing continues using **start** and **construction** calls, making sure that the referent and condition information is put into the correct DRS. Any suitable_referent:R conditions are analysed within **construction,** with a disjunctive DRS only having access to the referents of the disjunctive DRS being developed *DisjDrs,* and the main DRS *StartDrs.* Accessibility is taken into account by the formation of careful calls to **construction.**

All information is put into the *StartDrs* if *Connector* is *and,* and the a different *DisjDrs* for each disjunctive NP clause if the *Connector* is *or*, with proper noun information always placed in *StartDrs.*

- The final processing step for *Connnector = or* involves incorporating the newly formed disjunctive DRSs into the main DRS information:

```
drs([ Refs,
        [......mainconditions.....connector(DisjDRS1,or,DisjDRS2,or,.......DisjDRS5) ] ]).
                                                          |
                                                  Disjunctive sub-DRS template.
```

Conjunctive information is simply incorporated into the main DRS *Start_Drs* during processing, and does not introduce a sub-DRS template.

Let's take some examples to inspect the results produced by the above code fragments, taking a disjunctive example (5), and then the same example expressed conjunctively (6).

*(5). Bill owns Ulysses or Pollyanna.*

produces one ***DRS*** when the *Existing DRS* is empty:

```
DRS  = drs( [ [1],
              [ bill:1, [gen:male, num:sing],
                connector( [ drs( [[2], [ulysses:2, [gen:neut, num:sing],
                                        1, owns, [fin:fin, trans:trans, num:sing], 2]]),
                             or,
                             drs( [[3], [pollyanna:3, [gen:neut, num;sing].
                                        1, owns, [fin:fin, trans:trans, num:sing], 3]]) ])
                                                                                  ]])?;
no
```

Changing (5) slightly results in the conjunctive sentence (6):

*(6). Bill owns Ulysses and Pollyanna.*

Again assume that the *Existing_DRS* is empty.

```
DRS = drs( [[3,2,1],
                        [ pollyanna:3, [gen:neut, num:sing],
                          ulysses:2, [gen:neut, num:sing],
                          1, owns, [fin:fin, trans:trans, num:sing], 2,
                          1, owns, [fin:fin, trans:trans, num:sing], 3,
                          bill:1, [gen:male, num:sing] ] ])?;

no
```

#### 4.3.3.2.3. Conjunctive and disjunctive sentences.

**sentence_disj_and_conj**(*Referent, Sentence_Syntax, Existing_Drs, New_Drs, Flag, Refcount).*

This rule works on sentential disjunction and conjunction. Conjunction produces one DRS with each conjunct fully analysed before moving onto the next sentential conjunct. When all conjuncts have been analysed the new DRS information is put into *New_Drs.* Disjunction introduces disjunctive DRSs, *DisjDrs,* which are eventually incorporated into the *Existing_Drs* producing the *New_Drs.*

*Sentence_Syntax* is analysed breaking the sentence up into the respective conjunctive or disjunctive sentential components. The **constructing** and **start** rules introduce referents and conditions to *DisjDrs* and the conjunctive DRS, taking care to place proper nouns in the main DRS being developed, and resolve pronouns considering only suitable referents accessible from the conjunctive DRS or *DisjDrs* being developed.

When processing is complete the *DisjDrs* formed are placed into the disjunctive sub-DRS template shown in 4.3.3.2.2, and joined to the main DRS by **drs_conc** to produce the *New_Drs.*

We will take the following examples to illustrate the mechanics of the semantic implementation of disjunctive and conjunctive sentences (7) and (8).

*(7). John owns a cat or Mary owns a dog.*

```
DRS = drs( [[4,1],
             [mary:4,[gen:fem,num:sing],
                 john:1,[gen:male,num:sing],
                 connector([drs([[2],[cat:2,[gen:neut,num:sing],
                                    1, owns, [fin:fin,trans:trans,num:sing], 2]]),
                           or,
                          drs([[5],[dog:5,[gen:neut,num:sing],
                                   4,owns,[fin:fin,trans:trans,num:sing],5]])])]])? ;
no
```

*(8). John owns a cat and Mary owns a dog.*

```
DRS = drs([ [4,3,2,1],   [dog:4, [gen:neut,num:sing],
                          mary:3, [gen:fem,num:sing],
                          cat:2, [gen:neut,num:sing],
                          john:1, [gen:male,num:sing],
                          1, owns, [fin:fin,trans:trans,num:sing],2,
                          3, owns, [fin:fin,trans:trans,num:sing],4]])? ;

no.
```

#### 4.3.3.2.4. Conditional sentences.

**condition**( *Sentence_Syntax, Existing_Drs, New_Drs, NewFlag, Referent, Referent_two, NewCondition, Condition).*

- This rule contains the conditional trigger rule which checks *Sentence_Syntax* to see if it contains [s(if, *Sentence1*, then, *Sentence2*)]. If this is present two calls are made to **constructing,** one to analyse *Sentence1* and the other to analyse *Sentence2,* producing *DRS1* and *DRS2* respectively.

- *CondDRS* is declared as sub(cond( *DRS1,* =>, *DRS2* )). *NewCondition = finished* indicating that when this rule is exited all processing will be complete for *Sentence_Syntax* and the third prong of **constructing** will output the *New_Drs.*

- **changing_conc_drs** is called to resolve any suitable_referent:R conditions present in *CondDRS.*

```
    changing_cond_drs( Existing_Drs, CondDRS, New_Drs ) :-

            same( CondDRS, sub(condDRS1, =>, DRS2))),
            cond_drs(Existing_Drs, DRS1,_, New_DRS1),

            % The second DRS can have access to the referents of DRS1 and the
            % Existing_Drs
            % Join the conditions of both of these DRSs into TempConds.
            same(Existing_Drs, drs([_,MainConds])),
            conc( MainConds, Conds, TempConds),

            % Resolve the suitable_referent:_ conditions in DRS2, to give NewDRS2
            cond_drs( drs([[], TempConds ]), DRS2, _, NewDRS2),

            % Form the NewCondDRS
            same(NewCondDRS, sub(cond(NewDRS1,=>,NewDRS2))),
            same(Existing_Drs, drs([MainRef, MainConds])),

            % Add the NewCondDRS to the MainConds
            conc(MainConds, [NewCondDRS], NewConditions),
            same(NewDRS, drs([MainRef, NewConditions])),
```

**changing_cond_drs** rule takes the *Existing_DRS* and a *CondDRS* which contains two DRSs, *DRS1* and *DRS2*. The aim of the rule is to produce a *New_DRS* with *CondDRS* incorporated into the *Existing_Drs* and all suitable_referent:R conditions resolved.

The suitable_referent:R conditions of *DRS1* must be resolved first of all. These conditions have access to suitable referents in *DRS1* and *Existing_Drs.* **cond_drs** is called using *DRS1* and *Existing_Drs* as arguments producing *New_DRS1* with all such conditions resolved.

The suitable_referent:R conditions of DRS2 have access to the referents of DRS2, DRS1 and *Existing_Drs.* A *TempDRS* is formed by concatenating the referents and conditions of *DRS1* and *Existing_Drs,* **cond_drs** takes this *TempDRS* and *DRS2* as arguments and resolves any such conditions within *DRS2* producing *NewDRS2.*

Finally the *New_CondDRS* is formed by placing *NewDRS1* and *NewDRS2* into the conditional sub-DRS template: sub(cond( *NewDRS1,=>, NewDRS2)),* which is joined with *Existing_DRS* using **drs_conc** to produce *New_Drs.* As before all the possible suitable referents that can replace a suitable_referent:R condition are allowed to do so producing all possible *NewDRSs.*

*(9). If a man knows a woman, then she knows him.*

Assuming *Existing_DRS* is empty, processing (9), results in ***DRS:***

```
New_DRS = drs( [ [], [ sub( cond( drs([ [2,1], [ woman:2,[gen:fem,num:sing],
                                                  man:1,[gen:male,num:sing],
                                                  1, knows, [fin:fin,trans:trans,num:sing], 2]]),

                             =>,

                                 drs( [[4,3], [ 3, knows, [fin:fin,trans:trans,num:sing], 4,
                                                4:1, [case:(-),gen:male,num:sing],
                                                3:2, [case:(+),gen:fem,num:sing]]]))]]) ? ;


no
```

## 4.4 Conclusions

This chapter presented an implementation of Discourse Representation Theory as presented in Kamp and Reyle (1993), with extensions by the author in the areas of disjunction, summation set formation, and definite NPs. These extensions, and many more were necessary to facilitate the usage of a theoretical approach to DRT in a strict computational environment, thereby avoiding:

- condition and referent declaration duplication within disjunctive DRSs, which is harmless in a theoretical sphere but superfluous in a computational domain, viz. chapter 3 section 3.3.6.2

- formation of incorrect summation sets. Kamp and Reyle (1993) suggest that every referent present in a DRS can be summed to form a new non-atomic referent set. This approach produces only one summation set, and concentrates on world knowledge to eliminate nonsensical referent combinations within the set.

  Computationally this ungainly method can lead to senseless readings as non-human and human referents are joined into one unusual set. The author attempts to incorporate world knowledge into the procedure by placing human, and non-human referents in separate sets. Consequently each application of summation produces two summation sets, and two possible DRSs. Each DRS produced is developed until the discourse is fully processed and the DRS is output, or the DRS becomes improper and failure occurs.

- problems with definite NPs and familiarity. There are many different types and treatments of definite NPs. Non-generic definite NPs when encountered in discourse re-introduce familiar entities to the environment. Kamp and Reyle (1993) provisionally assert that processing of indefinite NPs as new entities, and definite NPs as familiar entities forms the backbone of DRT. They claim that definite NPs are treated like pronouns in that they are familiar to the speaker and hearer of a discourse, and should be linked with available accessible suitable referents. Subsequently Kamp and Reyle (1993) revise this approach as suitable referents are not always available in the DRS, they assert that definite NPs should simply introduce a referent and condition without the suitable referent search.

  The author agrees that suitable referents are not always available, but claims that after the initial referent and condition of the definite NP are introduced to the DRS, a search for a suitable referent should take place. If this search fails the conditions are left as they are, but if a suitable referent or set of suitable referents are found an extra condition linking the NP and referent should be added to the DRS being developed. This search is achieved by the author in the implementation.

The next chapter will discuss the areas of the implementation that would benefit from expansion and improvement.

# Chapter 5 Conclusions.

## 5.1 Conclusions.

This thesis presented:

- **Chapter 1 -** This chapter defines the terminology used in the discussion of anaphora, the problem of anaphora, and the traditional approaches used in anaphora resolution.

  Anaphors are conversational shorthand, in that they refer to entities explicitly mentioned in text e.g.

  *(1). Mary owns a dog. She loves it.*

  *She* and *it* are the anaphors and can be linked with the antecedents (i.e. preceding entities) *Mary* and *dog*. Humans use world knowledge i.e. our knowledge of particular events and situations, to isolate and expand anaphors into entities they refer to in text. This world knowledge is implicit in conversation, as in order to communicate, and be understood the *speaker* and *hearer* must share the same knowledge of the world and environment. In (1) world knowledge implies that the female anaphor *She* refers to a previously mentioned female human entity, this is readily found in *Mary*. The information that *She* should be linked with a human, and that humans usually own dogs suggesting that *it* is *a dog*, is also provided by world knowledge which is not explicitly mention anywhere in (1). In order to model anaphora resolution i.e. expand the anaphor, world knowledge must be incorporated in an attempt to model the reasoning behaviour of humans.

  This chapter also discusses the earliest approach to anaphora resolution that of Predicate Calculus. This calculus provides truth conditions for every sentence of a discourse, but unlike human reasoning is unable to provide a link or common variable connection between sentences of the discourse. The calculus works well for single sentences but fails in a multisentential domain.

- **Chapter 2 -** With the failure of Predicate Calculus as applied to discourses, this chapter discusses the historical manipulations of anaphora that led to contemporary thinking, and Discourse Representation Theory (DRT). Historical approaches were built on Predicate Calculus, and attempt to build on the calculus so that it may be applied to multisentential domains. A common theme running through all the methods designed to resolve anaphors is that different treatment given to indefinite NPs. An analysis may regard the NP as a variable, a referring, non-referring or quantifying article. How the indefinite NP is regarded forms the core of these methods. This NP is singled out for special treatment as it can range across sequences of sentences, behaves differently to other NPs when in a conditional and is capable of anteceding a pronoun across a relative clause when other NPs cannot.

This chapter introduces the awkward donkey sentence, as discussed by Geach (Geach 1962), the development of methods to deal with such sentences, and the behaviour of indefinite NPs within the said sentences. Ultimately leading to a discussion of the dynamic systems, with a discussion of File Change Semantics (Heim 1982), and Discourse Representation Theory (Kamp 1981). These systems are deal with discourses, and differ from those developed previously in that they present a more dynamic view of semantics. In dynamic semantic theories new sentences in discourse are interpreted in the context of what has gone before, and in this way new information is built into an existing structure. The core philosophy behind these theories is that *meaning is change* (Cooper et al 1995). Discourse Representation Theory (DRT) is discussed further in chapter 3.

- **Chapter 3** - This chapter presents an overview, and a discussion of the dynamic multisentential system of DRT as presented by Kamp and Reyle (Kamp and Reyle 1993) with extensions provided by the author. The overview discusses the basic mechanisms, construction rules and antecedent prediction methods of DRT covering all syntactic structures and plural NPs.

- **Chapter 4 -** This chapter presents, and discusses an implementation of a DRT extended theory in Prolog, where the discussion is divided into two modules - the syntactic and semantic modules. The syntactic module discusses the parsing methods that were available to the author, and explains why a left corner parser was chosen. The semantic module divides all sentences into simple and complex sentences, where a complex sentence is a conditional, quantified, conjunctive, disjunctive sentence, or any sentence containing a plural NP. The analysis of simple sentences with the reduction of complex sentences to simple sentences is discussed with examples in the semantic module.

## 5.2 Suggestions for further work.

The implementation works well within the small domain specified, but would benefit from improvement as:

- the selection of accessible suitable referents for definite NPs, plural, or singular pronouns repeatedly results in sets of possible referents. Each element of the suitable referent set must be placed in the existing DRS producing a new DRS. This simple operation can lead to the production of multiple DRSs, normally anything from two to eight DRSs are produced by a search for a suitable referent. A computational explosion is the consequence of this as even a simple three sentence discourse can lead to at least eight or ten possible DRSs, some of which contain nonsensical readings e.g. section 4.3.3.1.2

A computational explosion also results from ambiguous sentences where every possible reading of every sentence of the discourse must be represented in a DRS, section 4.3.3.1.2. e.g.

*Everyone in the room speaks at least two languages.*

(Sag, 1991).

The ambiguity here, is do they speak the same two languages or does everybody know different languages.

These inherent ambiguities make the system in its present state nonviable as an instrument in analysing large discourses, e.g. if a three sentence discourse can result in eight DRSs, a larger discourse would result in hundreds of possible DRSs.

The implementation needs to be revised, eliminating nonviable sentence readings and unsuitable set elements before they are added to the existing DRS. This could be achieved by examining surrounding sentences, introducing a type of focus or ranking to gauge the sentence readings and set elements, with only the two highest ranking set elements or readings considered. One sentence would therefore produce at most four DRSs. This could potentially halve the number of DRSs produced, and greatly reduce the computational effort required to process a medium sized discourse. Alternatively more processing could be introduced at the syntactic level, introducing more world knowledge to the system.

- in forming the implementation, it was decided to omit the treatment of scope relations between quantifiers of sentences. This decision excludes one form of ambiguity from the implementation, with the disadvantage of a resulting 2-dimensional mechanism rather than 3-dimensional approach. For a more rounded implementation this ambiguity can be introduced to the system.

Theoretically the DRT system of Kamp and Reyle (1993) works well, but merely provides guidelines to build on. The system is basic and needs to incorporate a greater degree of world knowledge with the early elimination of nonsensical readings before they produce nonviable DRSs.

## References.

Allwood, J. Dahl, O. and Andersson, L-G. (1977), *Logic in Linguistics,* Cambridge Textbooks in Linguistics.

Asher N. and Wader H. (1986), *BUILDRS: An implementation of DR theory and CFG,* in COLING '86, p540-46

Bates, M. and Weischedel R. (1993), *Challenges in Natural Language Processing,* Studies in NLP series, Cambridge University Press.

Beardon, C, Holmes, G. and Lumsden, D (1991), *Natural Language and Computational Linguistics , an introduction,* Ellist Horwood.

Burnstein, M and Schank, R. (1985), *Artificial Intelligence Modelling Memory for Language Understanding,* From Van Dijk, T. (ed.), *Handbook of Discourse Analysis,* London Ontario Academic Press.

Cann, R. (1993), *Formal semantics. An introduction.* Cambridge University Press.

Carter, D. (1987), *Interpreting anaphors in natural language text,* Chichester: E Horwoods New York Halsted Press.

Charniak E, (1977), *Inference and Knowledge in Language Computation,* From Elcock and Mitchie, *Machine Intelligence 8,* p541-75.

Chierchia, G. (1992), *Anaphora and Dynamic Binding,* Linguistics and Philosophy 15.

Chierchia, G. and Mc Connell, S. (1990), *Meaning and Grammar. An Introduction to Semantics,* MIT Press.

Chierchia, G. (1995), *Dynamics of Meaning - Anaphora, Presuppositions and the theory of Grammar,* Universtiy of Chicago Press - forthcoming.

Cohen, R. (1987), *Analysing the Structure of Argumentative Discourse.* Computational Linguistics Volume 13 no 1-2, Jan-Jun.

Cooper, R (1979), *The Interpretation of pronouns* in Heny, F and Schnelle, H (eds), *Selections from the 3rd Groningen Round Table, Syntax and Semantics (10),* New York Academic Press, p 61-92.

Cooper, R et al (1994), *Describing Approaches,* FraCaS - A Framework for Computational Semantics, LRE 62-051 Deliverable D8.

Cooper, R et al (1995), *Evaluating the State of the Art,* FraCaS - A Framework for Computational Semantics, LRE 62-O51 Deliverable D10.

Coulthard, M. (1985), *An introduction to Discourse Analysis,* Longman.

Covington, M. Schmitz D. N. and Goodman D, (1988), *From English to Prolog via Discourse Representation Theory,* Advanced Computational Methods Centre Research Report 01-0024, University of Georgia.

Crouch, D. Kamp, H. and van Genabith, J. (1994), *Specification of Linguistic Coverage,* FraCaS - A Framework for Computational Semantics LRE 62-051 Deliverable D2

Egli, U. (1979), *The Stoic Concept of Anaphora* in Bauerle, R. Egli, U. and Stechow, A. *Semantics from Different Points of View,* Berlin Springer.

Evans, G. (1977), *Pronouns; quantifiers of relative clauses,* Canadian Journal of Philosophy 7.

Evans, G (1980), *Pronouns,* Linguistic Inquiry 11, MIT Press.

Gal, A. Lapalme, G. Saint Dizier, P. and Somers, H, (1991), *Prolog for Natural Language Processing,* Chichester: Wiley.

Gamut, L. T. F. (1990), *Logic, Language and Meaning, volume 1,* University of Chicago.

Gamut, L. T. F. (1990), *Logic, Language and Meaning, volume 2,* University of Chicago.

Gazdar, G and Mellish, C. (1989), *Natural Language Processing in Prolog. An introduction to Computational Linguistics,* Wesley.

Grice, H. P. (1975), *Logic and Conversation* in Cole, P. R and Morgan, J. L. eds (1975) *Studies in Syntax, Volume 3,* NewYork: Academic Press.

Grishman, R. (1986), *Computational Linguistics, An introduction,* Cambridge University Press.

Grosz, B. J. (1986), *Attention, Intentions, and Structure of discourse,* Computational Linguistics Volume 12, no 3, July-Sept.

Guenthner, F. Lehmann, H. and Schonfeld, W. (1986). *A theory for representation of knowledge.* International Business Corporation.

Heim, I. (1981), *The Semantics of Definite and Indefinite Noun Phrases in English,* PhD Thesis, University of Massachusetts, Amherst. Distributed as *Arbeitspapier* 73, SFB 99, Konstanz. - Published (1988). NewYork: Garland.

Hinkelman, E. and Traum, D.R. (1992), *Conversation Acts in Task Oriented Spoken dialogue,* Computational Linguistics Special issue: Computational Approaches to Non-Literal Language, Volume 8 no 3 August.

Hobbs, J (1986), *Resolving pronoun references* in Grosz, B. J, Sparc Jones, K and Webber, B. (eds), *Readings in Natural Language Processing,* Morgan Kaufman Publishing Inc.

Hobbs, J. and Shieber, S. (1987), *An algorithm for General Quantifier scoping,* Computational Linguistics, Volume 13, Numbers 1-2, January-June.

Johnson, M. and Klein E. (1986), *Discourse, anaphora and parsing,* From COLING '86, p669-75

Kamp, H. (1981), *A Theory of Truth and Semantic Representation,* in Jeroen A. G. Groenendijk, T. M. V. Janssen & Martin B. J. Stokhof (eds.), *Formal Methods in the Study of Language,* Mathematical Centre Tract 135, Amsterdam, pp 277-322. - Reprinted in: Jeroen A. G. Groenendijk, T. M. V. Janssen & Martin B. J. Stokhof (eds.), *Truth, Representation and Information* ( = GRASS Series No. 2), Dordrecht, pp. 227 - 322.

Kamp, H. and Reyle, U. (1993), *From Discourse to Logic. Introduction to Model Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Kluwer Academic Publishers.

Kartunnen, L., Zwicky, A. (1985), *Introduction to Natural Language Processing* in Dowty, D. Kartunnen, L. Zwicky, A (eds), *Natural Language Parsing,* Cambridge University Press.

Laird, J. (1986), *Reasoning without Logic,* From Cognitive Science Series - Myers, T., Brown, K., and Mc Gonigle, B, (1986), *Reasoning and Discourse Processes.*

Lewis, D. (1979), *Scorekeeping in a Language Game,* From Bauerle,R. Egli U. and Stechow A, *Semantics from Different Points of View,* Berlin Springer.

Link, G. (1983), *The logical analysis of plurals and mass terms: A lattice-theoretical approach,* In Bauerle, R et al (eds), *Meaning, Use, and Interpretation of Language,* de Gruryter, Berlin p 302-23.

Link, G. (1991), *First order axioms for the logic of plurality,* presented at the Third European Summer School in Language, Logic and Information, Saarbrucken, August 12-23, 1991.

Link, G. (1986), *Generalized Quantifiers and Plurals,* University of Munich and CSLI Report no CSLI-86-67.

Link, G, and Lonning J. (1991), *A chapter in Lattice Theory,* From *Algebraic Semantics* I: *Plurals and Mass Terms,* presented at the Third European Summer School in Language, Logic and Information, Saarbrucken, August 12-23, 1991.

Lonning, J. T. (1987), *Mass Terms and Quantification,* Linguistics and Philosophy 10, p 1-52.

Maier, H (1994), *IAN - A testbed for incremental syntax-semantic analysis of natural language,* Final year project, School of Computational Linguistics, Dublin City University.

Partee, B. (1976), *Montague Grammar,* New York Academic Press.

Parsons, B. (1978), *Pronouns as paraphrases,* University of Massachusetts, Amherst.

Pereira, F. C. N and Shieber, S. M. (1987), *Prolog and Natural Language Analysis,* CSLI Lecture Notes no 10.

Pereira, F. C. N. and Pollack, M. E. (1991), *Incremental Interpretation,* Artificial Intelligence 50, p 37 - 82, Received December 1989, Revised May 1990.

Renkema, J. (1993), *Discourse Studies, An Introductory Textbook,* John Benjamins Publishing Company.

Reape, M. (1992), *Introduction to Some Common Parsing Algorithms,* Department of Computer Science, Trinity College Dublin.

Roberts, C. (1991), *Distributivity,* presented at the Third European Summer School in Language, Logic and Information, Saarbrucken, August 12-23, 1991.

Roeper, P. (1991), *Semantics for mass terms with quantifiers,* presented at the Third European Summer School in Language, Logic and Information, Saarbrucken, August 12-23.

Rooth, M. (1986), *NP interpretation in Montague Grammar, File Change Semantics and Situation Semantics.* Centre for Study of Language and Information, Report no CSLI-86-51, Dordecht, Holland.

Russell, B (1905), *Descriptions* in Linsky L (1972), *Semantics and the Philosophy of language, A collection of readings,* Urbana, Ill: University of Illinois Press 1972.

Sag, I. (1991), *Linguistic Theory and Natural Language Processing,* From Natural Language Speech Symposium Proceedings, Brussels.

Sedogbo, C. (1987), *A DRT system in Prolog,* in *Natural Language in* Dahl, V. and Samint-Dizier, P. (eds), *Understanding and Language Programming,* p 185-201.

Shieber, S. M. (1986), *An Introduction to Unification Based Approaches,* CSLI Lecture Notes no 4, Stanford University Press.

Smaby, R. (1979), *Ambiguous coreference with quantifiers* in Guenthner, F and Schmidt, S (eds), *Formal Semantics and Pragmatics for Natural Language,* Dordrecht Reidel, p 37 - 75.

van Genabith, J. (1995), Lecture Notes on parsing presented at Dublin City University, Spring 1995.

Varile, G. B. (1983), *Charts: a Data Structure for parsing*, From King, M (1983), *Parsing Natural Language,* Academic Press.

Webber, B (1986), *So what can we talk about now?* in Grosz, B. J, Sparc Jones, K and Webber, B. (eds), *Readings in Natural Language Processing*. Morgan Kaufman Publishing Inc.

# Appendix A

```
% This file kamp4.pl contains the grammar and lexicon which is parsed by the
%  left_corner parser of left_corner.pl.

?-op(500,xfy,--->). % declares the operator.

% This  left corner recursive grammar, this file is parsed by
% "left_corner" the parser which
% also uses the  link table in file "link_table".

% gap has two values gap(np) and nogap

s(S) ---> [s(S,[num:_,gap:nogap])].

% np(Case,Gender,Num,Gap)].
% vp(Fin,Trans,Num,Gap)].

s([s(COND1,S1,COND2,S2),[num:_,gap:nogap]],
                        [num:_,gap:nogap]) ---> [ cond(COND1),
                                                  s(S1,[num:_,gap:nogap]),
                                                  cond(COND2),
                                                  s(S2,[num:_,gap:nogap])].

s([s(S1,CONNECTOR,S2),[num:_,gap:nogap]],
                        [num:_,gap:nogap]) ---> [ s(S1,[num:_,gap:nogap]),
                                                 connector(CONNECTOR),
                                                 s(S2,[num:_,gap:nogap])].

s([s(NP,VP),[num:Num,gap:Gap]],
          [num:Num,gap:Gap]) --->
                                     [ np(NP,[case:_,gen:_,num:Num,gap:nogap]),
                                      vp(VP,[fin:_,trans:_,num:Num,gap:Gap])].

np([np(PN),[case:_,gen:Gen,num:Num,gap:nogap]],
          [case:_,gen:Gen,num:Num,gap:nogap]) --->
                                                  [ pn(PN,[gen:Gen,num:Num]) ].
```

```
np([np(DET,N,OPT),[case:_,gen:Gen,num:Num,gap:nogap]],
                    [case:_,gen:Gen,num:Num,gap:nogap]) --->
                                        [ det(DET,[num:Num]),
                                          n(N,[gen:Gen,num:Num]),
                                        optrel(OPT,[gen:Gen,num:Num]) ].

np([np(N,OPT),[case:_,gen:Gen,num:Num,gap:nogap]],
            [case:_,gen:Gen,num:Num,gap:nogap]) --->
                                   [ n(N,[gen:Gen,num:Num]),
                                       optrel(OPT,[gen:Gen,num:Num])].

np([np(PRO), [case:Case,gen:Gen,num:Num,gap:nogap]],
           [case:Case,gen:Gen,num:Num,gap:nogap]) --->
                                  [ pro(PRO,[case:Case,gen:Gen,num:Num])].

np([np(gap), [case:_,gen:_,num:_,gap:gap(np)]],
           [case:_,gen:_,num:_,gap:gap(np)]) ---> [[]]. % This is the empty np
                                                 % productions.

np([np(NP1,NP2,CONNECTOR,NP3),[case:Case,gen:_,num:_,gap:nogap]],
                         [case:Case,gen:_,num:_,gap:nogap]) --->
                                  [ np(NP1,
                                       [case:Case,gen:_,num:_,gap:nogap]),
                                   special_np(NP2,
                                       [case:Case,gen:_,num:_,gap:nogap]),
                                  connector(CONNECTOR),
                                  np(NP3,
                                       [case:Case,gen:_,num:_,gap:nogap])].


special_np([np(NP1,NP2,NP3),[case:Case,gen:_,num:_,gap:nogap]],
                      [case:Case,gen:_,num:_,gap:nogap]) --->
                             [ np(NP1,[case:Case,gen:_,num:_,gap:_]),
                               np(NP2,[case:Case,gen:_,num:_,gap:_]),
                                        np(NP3,[case:Case,gen:_,num:_,gap:_])].

% vp(Fin,Trans,Num,Gap)].

% intransitive verb rule:

vp([vp(V),[fin:Fin,trans:intra,num:Num,gap:nogap]],
        [fin:Fin,trans:intra,num:Num,gap:nogap]) --->
                                            [v(V,[fin:Fin,trans:intra,num:Num])].
% transitive verb rule:
vp([vp(V,NP),[fin:Fin,trans:trans,num:Num,gap:Gap]],
              [fin:Fin,trans:trans,num:Num,gap:Gap]) --->
                                            [v(V,[fin:Fin,trans:trans,num:Num]),
                                             np(NP,[case:_,gen:_,num:_,gap:Gap])].

vp([vp(AUX,NOT,VP),[fin:Fin,trans:_,num:Num,gap:_]],
                    [fin:Fin,trans:_,num:Num,gap:_]) --->
                                                  [aux(AUX,[num:Num]),
                                                   neg(NOT),
                                                  vp(VP,[fin:Fin,trans:_,num:_,gap:_])].

optrel([optrel([]),_],[gen:_,num:_]) ---> [[]].

optrel([optrel(PRON,V),[gen:Gen,num:Num]],
                    [gen:Gen,num:Num]) --->
                        [relpron(PRON,[gen:Gen,num:Num]),
                           vp(V,[fin:_,trans:_,num:_,gap:nogap])].

optrel([optrel(PRON,S),[gen:Gen,num:Num]],
                    [gen:Gen,num:Num]) --->
                        [relpron(PRON,[gen:Gen,num:Num]),
                           s(S,[num:_,gap:gap(np)])].

lex(connector(and),and) :- !.
lex(connector(or),or) :- !.
```

```
% For conjunction:

lex(conj(and),and).

% For negation:
lex(neg(not),not).

% For conditionals:
lex(cond(if),if).
lex(cond(then),then).
% For disjunctions.
lex(disj(or),or).

lex(relpron([relpron(that),[gen:neut,num:_]],[gen:neut,num:_]),that).
lex(relpron([relpron(who),[gen:male,num:_]],[gen:male,num:_]),who).
lex(relpron([relpron(who),[gen:fem,num:_]],[gen:fem,num:_]),who).
lex(relpron([relpron(which),[gen:neut,num:_]],[gen:neut,num:_]),which).

lex(det([det(the),[num:sing]],[num:sing]),the).
lex(det([det(the),[num:plur]],[num:plur]),the).
lex(det([det(some),[num:sing]],[num:sing]),some).
lex(det([det(some),[num:plur]],[num:plur]),some).
lex(det([det(a),[num:sing]],[num:sing]),a).
lex(det([det(every),[num:sing]],[num:sing]),every).
lex(det([det(all),[num:plur]],[num:plur]),all).
lex(det([det(most),[num:plur]],[num:plur]),most).

lex(n([n(book),[gen:neut,num:sing]],[gen:neut,num:sing]),book).
lex(n([n(woman),[gen:fem,num:sing]],[gen:fem,num:sing]),woman).
lex(n([n(man),[gen:male,num:sing]],[gen:male,num:sing]),man).
lex(n([n(stockbroker),[gen:male,num:sing]],[gen:male,num:sing]),stockbroker).
lex(n([n(stockbroker),[gen:fem,num:sing]],[gen:fem,num:sing]),stockbroker).
lex(n([n(books),[gen:neut,num:plur]],[gen:neut,num:plur]),books).
lex(n([n(women),[gen:fem,num:plur]],[gen:fem,num:plur]),women).
lex(n([n(men),[gen:male,num:plur]],[gen:male,num:plur]),men).
lex(n([n(stockbrokers),[gen:male,num:plur]],[gen:male,num:plur]),stockbrokers).
lex(n([n(stockbrokers),[gen:fem,num:plur]],[gen:fem,num:plur]),stockbrokers).
lex(n([n(ulysses),[gen:neut,num:sing]],[gen:neut,num:sing]),ulysses).
lex(n([n(porche),[gen:neut,num:sing]],[gen:neut,num:sing]),porsche).
lex(n([n(porche),[gen:neut,num:plur]],[gen:neut,num:plur]),porsche).

lex(pn([pn(john),[gen:male,num:sing]],[gen:male,num:sing]),john).
lex(pn([pn(anna),[gen:fem,num:sing]],[gen:fem,num:sing]),anna).
lex(pn([pn(elaine),[gen:fem,num:sing]],[gen:fem,num:sing]),elaine).
lex(pn([pn(jones),[gen:male,num:sing]],[gen:male,num:sing]),jones).
lex(pn([pn(anna_karenina),[gen:fem,num:sing]],[gen:fem,num:sing]),anna_karenina).
lex(pn([pn(bill),[gen:male,num:sing]],[gen:male,num:sing]),bill).
lex(pn([pn(mary),[gen:fem,num:sing]],[gen:fem,num:sing]),mary).
lex(pn([pn(smith),[gen:male,num:sing]],[gen:male,num:sing]),smith).

% trans - transitive, intra - intransitive].
% fin:fin - fin:finite, infin:fin - infin:finite

lex(v([v(abhors),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),abhors).
lex(v([v(adores),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),adores).

lex(v([v(knows),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),knows).
lex(v([v(likes),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),likes).
lex(v([v(loves),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),loves).
lex(v([v(owns), [fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),owns).

lex(v([v(hit),      [fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),hit).
lex(v([v(fascinates),[fin:fin,trans:trans,num:sing]],
                    [fin:fin,trans:trans,num:sing]),fascinates).
```

```
lex(v([v(adore),[fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),adore).
lex(v([v(know), [fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),know).
lex(v([v(own),  [fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),own).
lex(v([v(love), [fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),love).
lex(v([v(like), [fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),like).
lex(v([v(fascinate),[fin:nonfin,trans:trans,num:plur],
                              [fin:nonfin,trans:trans,num:plur]]),fascinate).

lex(v([v(hit),  [fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),hit).
lex(v([v(abhor),[fin:nonfin,trans:trans,num:plur]],
                              [fin:nonfin,trans:trans,num:plur]),abhor).

lex(v([v(knows),  [fin:fin,trans:intra,num:sing]],
                              [fin:fin,trans:intra,num:sing]),knows).
lex(v([v(rotates),[fin:fin,trans:intra,num:sing]],
                              [fin:fin,trans:intra,num:sing]),rotates).

lex(v([v(know),  [fin:nonfin,trans:intra,num:plur]],
                              [fin:nonfin,trans:intra,num:plur]),know).
lex(v([v(rotate),[fin:nonfin,trans:intra,num:plur]],
                              [fin:nonfin,trans:intra,num:plur]),rotate).

% Case is nominative + for he,she and they and non-nominative for him,her and
% them].

lex(pro([pro(it),[case: _,gen:neut,num:sing]],[case: _,gen:neut,num:sing]),it).
lex(pro([pro(he),[case: +,gen:male,num:sing]],[case: +,gen:male,num:sing]),he).
lex(pro([pro(she),[case: +,gen:fem,num:sing]],[case: +,gen:fem,num:sing]),she).
lex(pro([pro(they),[case: +,gen:_,num:plur]], [case: +,gen:_,num:plur]),they).
lex(pro([pro(him),[case: -,gen:male,num:sing]],[case: -,gen:male,num:sing]),him).
lex(pro([pro(her),[case: -,gen:fem,num:sing]],[case: -,gen:fem,num:sing]),her).
lex(pro([pro(them),[case: -,gen:_,num:plur]],[case: -,gen:_,num:plur]),them).

lex(aux([aux(does),[num:sing]],[num:sing]),does).
lex(aux([aux(do),[num:plur]],[num:plur]),do).
```

```
% This is the left_corner parser - left_corner.pl

% This is the parser, it is called with parse(Goal,[sentence],[]).
% This file is used in conjunction with link_table.pl which explicitly
% states the links between categories in the grammar, and the grammar
% kamp4.pl. An example query is parse(s(Tree),[jones,loves,mary],[]).

?-op(500,xfy,--->).

parse(Goal,B,E) :-
        shift(SubGoal,B,M),
        link(Goal,SubGoal),     % the link fact is found in file link_table.
                                % which needs to be compiled with this file.
        l_c(SubGoal,Goal,M,E).

shift(Cat,[Word|Rest],Rest) :-
        lex(Cat,Word).

shift(Cat,String,String) :-
        Cat ---> [[]].

l_c(Goal,Goal,String,String). %shift


l_c(SubGoal,SuperGoal,B,E) :-
        Mother ---> [SubGoal|RestGoals],
        link(SuperGoal,Mother),
        parse_rest(RestGoals,B,M),
        l_c(Mother,SuperGoal,M,E).

parse_rest([],String,String).  % deals recursively with the sub derivations.
parse_rest([Goal|RestGoal],B,E) :-    % generated by l_c predict.
        parse(Goal,B,M),
        parse_rest(RestGoal,M,E).

% Next rules, check the link table and attempt to stop the problem of
% looping when the head of a rule h ---> [].
```

**% This is the link table used by the parser - link_table.pl**

```
% This file is used in conjunction with the parser "left_corner" and the
% grammar "kamp4", this file is called by "left_corner".

% This is a link table, it explicitly states the links between non terminals
% in grammars and the left corners of the non terminal rules.

% Reflexive links, links for every category to themselves.
% x element of nontermials -> link(x,x) [Josef lecture notes]

link(s(_),s(_)) :- !.
link(s(_,_),s(_,_)) :- !.
link(np(_,_),np(_,_)) :- !.
link(vp(_,_),vp(_,_)) :- !.
link(pn(_,_),pn(_,_)) :- !.
link(det(_,_),det(_,_)) :- !.
link(n(_,_),n(_,_)) :- !.
link(optrel(_,_),optrel(_,_)) :- !.
link(pro(_,_),pro(_,_)) :- !.
link(conj(_),conj(_)) :- !.
link(neg(_),neg(_)) :- !.
link(aux(_,_),aux(_,_)) :- !.
link(v(_,_),v(_,_)) :- !.
link(relpron(_),relpron(_)) :- !.
link(cond(_),cond(_)) :- !.
link(disj(_),disj(_)) :- !.
link(connector(_),connector(_)) :- !.
link(special_np(_,_),special_np(_,_)) :- !.


% The left corner links :
% for all x all y ( x -> y..... i.e y is lft corner of rule, y el of nonterminal
% ---> link(x,y)   [ josef lecture notes]
```

```
link(s(_),cond(_)) :- !.
link(s(_),np(_,[_,_,_,gap:nogap])) :- !.
link(s(_),pn(_,_)) :- !.
link(s(_),det(_,_)) :- !.
link(s(_),n(_,_)) :- !.
link(s(_),pro(_,_)) :- !.

link(s(_,[_,gap:_]),pn(_,_)) :- !.
link(s(_,[_,gap:_]),det(_,_)) :- !.
link(s(_,[_,gap:_]),n(_,_)) :- !.
link(s(_,[_,gap:_]),pro(_,_)) :- !.

link(special_np(_,_),pn(_,_)) :- !.
link(special_np(_,_),det(_,_)) :- !.
link(special_np(_,_),n(_,_)) :- !.
link(s(_),s(_,_)) :- !.
link(s(_,_),cond(_)) :- !.
link(s(_,[_,gap:_]),np(_,[_,_,_,gap:nogap])) :- !.

link(np(_,[_,_,_,gap:nogap]),pn(_,_)) :- !.
link(np(_,[_,_,_,gap:nogap]),det(_,_)) :- !.
link(np(_,[_,_,_,gap:_]),n(_,_)) :- !.
link(np(_,[_,_,_,gap:_]),pro(_,_)) :- !.
link(np(_,[_,_,_,gap:Gap]),np(_,[_,_,_,gap:Gap])) :- !.

% link(vp(_,[_,_,_,gap:nogap]),v(_,_)) :- !.
link(vp(_,_),v(_,_)) :- !.
link(vp(_,_),aux(_,_)) :- !.
link(optrel(_,_),relpron(_,_)) :- !.
link(special_np(_,_),np(_,[_,_,_,gap:_])) :- !.


% The transitive links:
% for all x all y all z [( link(x,y) and link(y,z) -> link(x,z))].
```

```
link(special_np(_,_),pro(_,_)) :- !.
link(special_np(_,_),np(_,[_,_,_,gap:_])) :- !.

% Not all the links were added in again due to duplication.
```