Research Report AI–1993–06

Incorporating Defeasible Reasoning into an Implementation of Discourse Representation Theory

Gregory J. Izzo

Artificial Intelligence Programs
The University of Georgia
Athens, Georgia 30602–7415 U.S.A.

Incorporating Defeasible Reasoning into an Implementation of Discourse Representation Theory

Gregory J. Izzo
Artificial Intelligence Programs
University of Georgia

September 1993

1 Introduction

It was the purpose of this project to develop a systems which takes as input natural language discourse composed of simple declarative sentences, capture the meaning of the discourse in a logical representation and when prompted by a question in the discourse attempt an answer using "common sense," or defeasible reasoning.

The system was developed by integrating a version of defeasible reasoning into an existing implementation of *discourse representation theory* (DRT). d-Prolog (Nute and Lewis 1986) is an Arity Prolog implementation of Donald Nute's defeasible logic (Nute 1993). d-Prolog is an extension of Prolog that allows for not only standard Prolog facts and rules but also defeasible rules, presumptions, defeaters and a positive form of negation. The present implementation uses d-Prolog version 6.0 (released in 1993) which is able to solve some of the classic defeasible examples more intuitively than the very early version of d-Prolog described in (Nute and Lewis 1986).

The DRT foundation for the current project was a program named DRT.GLP (Covington, Nute, Schmitz and Goodman 1988). It is an implementation of DRT which converts multi-sentence discourse composed of simple declarative sentences or questions to semantically equivalent discourse representation structures (DRSs) and then to Prolog clauses. In the case of queries, the Prolog engine is called upon to prove the query. DRT.GLP is written in GULP (Covington 1989), an extension to Prolog for handling unification-based grammar. Both the original program and the program described in this report, henceforth called DefDRT, employ a unification-based top-down parser. The reader is assumed to have a basic grasp of Prolog, parsing, the use of feature structures, and unification.

DefDRT accepts as input multi-sentence discourse — possibly over several different program executions — and converts the information in the discourse into appropriate strict or defeasible facts and rules within a d-Prolog database. By asking a question the user can query the database or can investigate whether a certain inference can be proven from the database using defeasible reasoning. To achieve this final implementation, many modifications were made to what once was DRT.GLP. The modifications were necessary to deal with generic plurals and for distinguishing between sentences that express strict rules and those that express defeasible rules.

2 Background

2.1 Discourse Representation Theory

Discourse Representation Theory (DRT), introduced by Hans Kamp (1981), defines a framework for representing multi-sentence discourse as a single structure. This framework describes the translation of natural language discourse into a logical representation called a *discourse representation structure* (DRS). The framework also provides a theory of truth based on model theoretic semantics that specifies an intuitively plausible scheme for finding anaphoric antecedents. The DRS is constructed one sentence at a time by incorporating

each sentence in accordance with the context established by previous sentences. From a logical perspective the DRS is a variant of clausal logic and as such is an excellent middle ground between natural language and Prolog, which is itself a variant of clausal logic.

The DRS is an ordered pair <**U**,**Con**> consisting of the *universe of discourse* (**U**) which is a set of discourse markers (one for each entity introduced in the discourse) and a set of conditions (**Con**) which are clauses or functions describing the properties of discourse entities and the relationships between them. The simple two-sentence discourse

Cindy owns a penguin. She loves it.

would be represented as the following DRS
U = {W, X, Y, Z}
Con = {named(W,'Cindy'), penguin(X), owns(W,X), Y=W, Z=X, loves(Y,Z)}
or in the traditional diagrammatic representation

W X Y Z
named(W,'Cindy') *penguin(X)* *owns(W,X)* *Y=W* *Z=X* *loves(Y,Z)*

Truth in DRT is defined by the relationship the DRS has to a model. A DRS is true in a model if there is a *proper embedding* of the DRS in the model. This proper embedding is defined as an association of the discourse markers in the DRS with individuals in the model, such that all of the conditions in the DRS are true in the model. This idea of truth is equivalent to the idea of truth in model theory.

2.2 Defeasible Reasoning

As human reasoners, when we have precise information and specific facts we are able to draw conclusions which we feel are certain. Rules like "A penguin is a bird," or "Oxygen is consumed when a match is burned" are rules we are certain of and so could call *strict*. Unfortunately, we don't always have strict rules from which to draw conclusions. In these cases we might use "common sense" rules like "Birds fly," "A match will burn when struck" or "Buses arrive every hour" which, while not absolute, apply in normal situations. Since "common sense" rules are not absolute, conclusions drawn from them are tentative. New information may indicate that the current situation is not a typical one and might then require us to withdraw some of our tentative conclusions and draw new ones. In this case we say that the new information *defeats* our previous conclusions. This type of reasoning — drawing conclusions which may be defeated by new information — is called *nonmonotonic reasoning*.

As an example, most of us would agree that the rule "Birds fly" applies to most cases and given only the fact that "Chilly is a bird" we would conclude that Chilly can fly. Although we feel comfortable making this conclusion based on the information at hand, if we were to find out that Chilly is a penguin we might change our minds about Chilly's ability to fly. We know that although penguins are birds, penguins in general do not fly, and so presumably Chilly does not fly. This new information, therefore, defeats our previous conclusion. The fact "Chilly is a penguin" does not contradict our previous information, but it does indicate that Chilly does not fall under the category of typical birds, and so rules about typical birds might not apply.

Although there exist many attempts at formalizing this type of reasoning, we are particularly interested in Nute's (1993) formalism, *defeasible logic*. The reader is directed to (Ginsberg 1987) for a general survey of nonmonotonic formalisms and (Nute 1993) for an exposition of some of the major defeasible formalisms within nonmonotonic reasoning.

Every form of inference employs some kind of rules which allow us to take what we know and arrive at something new. Nute's formalism of defeasible logic distinguishes between several types of rules. *Strict* rules

are familiar law-like rules which do not have exceptions. They include principles of classification ("Penguins are birds," "Whales are mammals"), semantic rules or rules based on the definition of terms ("Circles are round," "A father is a male parent"), geographic principles ("Anything in France is in Europe," "Connecticut is north of Georgia"), and some exact scientific rules ("The gravitational pull between two bodies is directly proportional to the product of their masses") (Nute and Lewis 1986). We will use the general form $A \Rightarrow B$ to represent a strict rule. For example, the rule "Penguins are birds" is symbolized as *penguin(X)* $\Rightarrow$ *bird(X)*.

Defeasible rules are "common sense" rules or rules which express normal or typical relationships that have exceptions ("Birds fly," "Matches burn when struck"). A defeasible rule will be represented by the general form "$A \rightsquigarrow B$" which is read as, "If A then apparently (evidently, normally, typically) B." For example, the defeasible rule "Birds fly" is symbolized as *bird(X)* $\rightsquigarrow$ *fly(X)*.

A *presumption* is a defeasible rule that has an empty antecedent. Since it has no antecedent condition, the consequent follows immediately — unless defeated by some other fact or rule. Presumptions allow us to express defeasible facts. A presumption will be represented as "$\rightsquigarrow B$," which reads as "Presumably B."

Defeater rules are used to block or defeat a defeasible rule, but they are too weak themselves to permit inference. A typical defeater would be "A genetically altered penguin might fly." It would defeat the inference "Penguins do not fly" but it is not strong enough for us to conclude that a genetically altered penguin does fly. A defeater "$A \sim B$," is read as "If A then it might be that B."

Our goal in using any form of inference is to add to our stock of true beliefs without adding to our set of false beliefs too much. Since defeasible logic allows us to perform inference using rules which permit exception, it is necessary to be cautious in drawing conclusions. Given the defeasible rule "If P, apparently Q" and given that we know P, we cannot immediately conclude Q. It is first necessary to check all of our other rules that have *not* Q as a consequent, and determine if the antecedents to any of those rules are satisfied. If one of them is satisfied, then that rule conflicts with our previous rule. Since the antecedents of both rules are satisfied, it would be possible to derive Q and *not* Q, a contradiction, if we weren't being careful. In defeasible logic this conflict between opposing rules is resolved by only using the *superior* rule. The rule which is not superior is defeated. In Nute's formalism one rule is superior to another rule if it is more specific. The following example illustrates this idea. Suppose that we know the following things:

Chilly is a penguin.	penguin(chilly)
Chilly is a bird.	bird(chilly)
Birds typically fly.	bird(X) $\rightsquigarrow$ fly(X)
Penguins typically do not fly.	penguin(X) $\rightsquigarrow \neg$ fly(X)
Penguins are birds.	penguin(X) $\Rightarrow$ bird(X)

Using the facts "Chilly is a penguin" and "Chilly is a bird," it is possible to conclude that Chilly flies and that Chilly doesn't fly. Since applying both rules together would result in a contradiction, one of these rules should be chosen over the other. To determine which rule is superior (more specific), their antecedents are compared. In general, rule A is more specific than rule B if the antecedent of B can be derived from the antecedent of A but the antecedent of A can't be derived from the antecedent of B. In our example we can see that if something is a penguin we can derive that it is a bird, but if something is a bird we cannot derive that it is a penguin. This indicates that the rule, "Penguins typically do not fly" is more specific than "Birds fly," so the rule "Birds fly" is defeated and we conclude that Chilly does not fly. Nute's formalism specifies that in trying to derive the antecedent of one rule from the antecedent of another rule, only strict rules, defeaters and defeasible rules (excluding presumptions) are used. Nute calls this *defeasible specificity* (Nute 1993). In the case where neither rule can be determined to be superior, both rules defeat each other.

A Prolog implementation of Nute's defeasible logic called d-Prolog (Nute and Lewis 1986) is used as the defeasible inference engine in DefDRT. Besides providing a means of representing defeasible rules, d-Prolog also provides an explicit way of representing negative information. Since defeasible reasoning allows for the possibility that we are reasoning with incomplete information, the closed-world assumption cannot be made. A failure to derive something from the defeasible database simply indicates that we can draw no conclusions. Since we cannot use failure to indicate negative information, d-Prolog provides the predicate `neg`, which allows us to express negative information explicitly. The fact that Chilly is not a penguin can be represented in d-Prolog as `neg penguin(chilly)`. The `neg` predicate can be used in the body of a d-Prolog rule as well as in the consequent. For example the rule "A man is not a woman" would be represented in d-Prolog as

`neg woman(X) :- man(X)`. Note that d-Prolog treats `Goal` and `neg Goal` as completely different. Failing to derive `Goal` does not imply `neg Goal`. This distinguishes its use from the built-in Prolog predicate `not`. The query `neg Goal` succeeds in the database if either `neg Goal` exits as a fact or exists as the consequent of a d-Prolog rule whose conditions are met.

d-Prolog has specific requirements for expressing rules, facts and queries based on its Prolog foundation. The methods used to interpret natural language discourse in DefDRT were guided, in a general sense, by the fact that the final form of the interpretation needed to be usable by the d-Prolog defeasible engine.

3 Generic Statements

3.1 Generic plurals

A generic statement is a statement that is interpreted as expressing a characteristic that is a prototypical characteristic of some homogenous group (Declerck 1991). Because generic sentences express a general relationship they are interpreted as rules. This generic reading is usually brought about by a noun phrase (NP) within a statement that lends itself to a generic (plural) reading.

The three canonical forms of generic NPs are seen in "Foxes are cunning," "A fox is cunning" and "The fox is cunning." DefDRT can handle expressions involving the first two ("foxes" and "a fox"). The singular definite form ("the fox") is not handled because in addition to being used to represent a singular entity and a generic plural, it can also be used as an anaphoric phrase: "Bill owns a dog. The dog is brown." Interpreting singular definite NPs is a complex issue in its own right, and was not explored as part of the present work.

One of the tasks of DefDRT is determining when an NP should be given a generic (plural) reading as in "A penguin is a bird," or a nongeneric (singular) reading as in "A penguin ate my fish." Although interpreting an NP as generic or nongeneric can involve semantic and pragmatic issues, DefDRT relies on sentence syntax to make this judgement.

The following sentences demonstrate the use of generic plurals.

Birds fly.
A penguin is a bird.
Penguins like fish.

Plural NPs such as "penguins" are interpreted by DefDRT as generic based simply on the fact that they are plural. A singular indefinite NP such as "a penguin," however, is more difficult to classify since it can represent a generic plural or a singular entity ("A penguin is a bird" compared to "A penguin likes Cindy"). In DefDRT singular entities are added to the current DRS as simple clauses. Generic plurals, on the other hand, force the generation of a conditional structure. For example the sentence "Birds fly" translates to "If X is a bird then X flies." It is important, therefore, to determine from sentence syntax which interpretation of a singular indefinite NP is more appropriate.

Since the determiners "every," "all" and "no" are taken to indicate a generic (plural) interpretation of a singular NP, only singular NPs which use the determiner "a" will need deliberation. The following sentences are representative of the kinds of structures involving singular indefinite NPs that the current implementation can handle, as well as their expected interpretations. (The number preceding each sentence corresponds to the rule below identifying the interpretation scheme used.)

[1] A penguin is a bird. (generic/plural)
    If something is a penguin then it is also a bird.
[2] A penguin eats a fish. (existential/singular)
    A particular penguin eats a particular fish.
[3] A boy likes Mary. (existential/singular)
    A particular boy likes Mary.
[4] A penguin loves fish. (generic/plural)
    If something is a penguin then it loves fish.
[5] Mary has a cat. (existential/singular)
    Mary has a particular cat.

[6] Farmers own a farm. (generic/plural) - every farmer ...
If something is a farmer then there is a particular farm that that farmer owns.

It can be seen from these sentences that the interpretation of a singular indefinite NP as singular or plural is based on other constituents in the sentence. After examining a variety of sentences containing singular indefinite NPs, I arrived at the following method for determining the particular interpretation.

1. The feature *specificity* is added to all relevant sentence constituents (this is not to be confused with the specificity of defeasible logic). This feature will contain the value *plural* if the constituent is known to be a generic plural ("every bird," "penguins," etc.), *singular* if the constituent is known to be a singular entity (a proper name), or is left uninstantiated if its status can't be determined by the components of the constituent alone.

2. At the sentence level, three items are compared to resolve any uninstantiated *specificity* features. These items are: the *specificity* feature from the top-level NP node, the *specificity* feature of the top level Verb Phrase (VP) node, and the *type* feature of the main verb (indicating copula or non-copula). This is the step that determines if a singular indefinite NP is interpreted as singular or plural.

The interpretations for the various combinations are shown below.

	NP-specificity	Verb-type	VP-specificity	Result
[1]	<uninstantiated>	copula	<uninstantiated>	plural
[2]	<uninstantiated>	non-copula	<uninstantiated>	singular
[3]	<uninstantiated>	<"any">	singular	singular
[4]	<uninstantiated>	<"any">	plural	plural
[5]	singular	<"any">	<uninstantiated>	singular
[6]	plural	<"any">	<uninstantiated>	plural

Once a decision has been reached, uninstantiated *specificity* features are given a value equal to the result shown above (plural or singular).

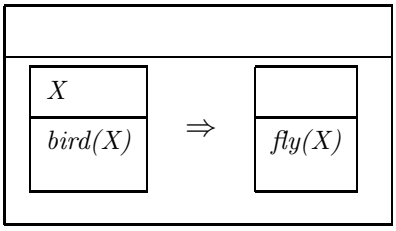
Although these criteria works well for the fragment of English that the system accepts, it would be necessary to take into account the semantics, and perhaps the pragmatics, of sentences to make the system robust enough to handle a larger fragment of English.

3.2 Strict/Defeasible Determination

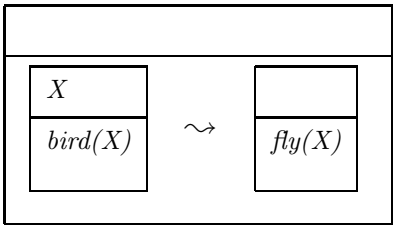
A number of authors on genericity indicate that generic statements like "Birds fly" and "A fox is cunning" don't express strict universal rules, but express "typical" relationships that allow for exceptions (Declerck 1991, Diesing 1992, Heyer 1990, Kamp 1988). In other words, generic statements tend to express defeasible rules. This implies that generic statements which represent strict rules are more of an exception than the norm. With this in mind, DefDRT takes the approach of first looking for indications that a statement is a strict rule, and failing that simply defaults to a defeasible interpretation.

Determining whether a sentence is a strict or defeasible rule is a complex issue. For human reasoners it involves a deep understanding of the real world entities mentioned in a discourse and of the relationships being expressed. Since DefDRT, for the most part, uses syntax as the means of extracting sentence and discourse meaning, the issues of deep understanding are left unexplored. In this project the resolution of the strict/defeasible question will rely on syntactic relationships alone.

DR theory specifies that the representation of a universal relationship (i.e. "Every X ..." or "If X then ...") will involve the creation of a sub-DRS that represents a conditional. This sub-DRS can be viewed as consisting of an antecedent-DRS and a consequent-DRS which is subordinate to it. For example the statement "Birds fly," which is given a plural interpretation by the criteria above, would be translated as:



Intuitively we know that this statement represents a defeasible rule since some birds don't fly. Unfortunately, DR theory treats every conditional as strict; thus the symbolization above represents "Every bird flies." To distinguish a strict rule from a defeasible one we will extend DR theory to allow other operators in a conditional structure. To symbolize a strict rule we will use the traditional $\Rightarrow$ arrow used in DR theory, but to symbolize a defeasible rule we will use the $\rightsquigarrow$ arrow. The proper symbolization of "Birds fly" is then



Recall that Nute's defeasible logic includes two other types of rules, presumptions and defeaters. The implementation of presumptions in DRT is straightforward: a presumption will be represented by a conditional structure consisting of an antecedent-DRS and a consequent-DRS joined by the defeasible operator, $\rightsquigarrow$. The difference between a presumption and a defeasible rule is that the antecedent-DRS for a presumption will be empty. DefDRT stipulates that a sentence representing a presumption begin with the word "presumably." This makes the natural language expression of a presumption easily distinguishable from other rule forms. Once a presumption is translated into DRS form, it is treated as if it were a defeasible rule.

To identify a sentence representing a defeater, DefDRT requires that a sentence representing a defeater be expressed using the modal "may" or "might." This allows us to determine that "A genetically altered penguin might not fly" is a defeater and not a defeasible rule. A defeater will be represented in a DRS by an antecedent-DRS and a consequent-DRS joined by the defeater operator $\sim$.

Adding these new types of rules to DefDRT involved changing the existing representation for a conditional. The original DRT.GLP program used the expression:

```
ifthen(DRS_antecedent,DRS_consequent)
```

In the new implementation a third argument called `Relationship` is added to this structure. The value of this argument will indicate the type of rule (strict, defeasible or defeater — recall that presumptions are treated as defeasible rules). The resulting structure is:

```
ifthen(Relationship,DRS_antecedent,DRS_consequent)
```

The main problem then becomes determining what this `Relationship` value should be. For presumptions and defeaters this determination is trivial since both of these rule types are recognized by their sentence forms. The problem, then, is determining for those sentences which are neither presumptions nor defeaters whether they are strict or defeasible. Examining the strict and defeasible interpretations of different sentence structures, some general patterns emerged.

It was found that the strict/defeasible determination must be done at the sentence level. This determination relies on the fact that certain features for the top-level NP and VP nodes have been assigned values. These features are listed below.

NP-strictness has a value of *strict* if the NP has a universal determiner ("every," "all" and "no") else it remains uninstantiated.

NP-type is used to determine if the NP contains a natural kind term or a place adjective.

NP-specificity tells whether the NP is a singular (existential) entity or if it is a generic (plural) entity.

main-verb-type tells whether the main verb is the copula or not.

VP-strictness has a value of *strict* if a direct object exists and contains a universal determiner ("every," "all" and "no"), else it remains uninstantiated.

VP-type is used to determine if the direct object (if one exists) contains a natural kind term or a place adjective.

VP-specificity tells whether the direct object (if one exists) is a singular (existential) entity or a generic (plural) entity.

The simplest structures to interpret are ones involving the determiners "every," "all" or "no." These determiners all indicate a strict relationship, so the *strictness* feature is assigned the value "strict." Since this feature is percolated up to the top-level, if the NP-strictness feature has a value of "strict" the sentence is judged to be strict and no other testing is required. For example, "Every bird flies" would be interpreted as strict and "Birds fly" would be interpreted as defeasible.

In accordance with defeasible logic, a sentence which equates one natural-kind term to another is a principle of classification, and thus interpreted as strict. A natural kind term is one which indicates a class of nature that is part of some taxonomy. Some examples of natural-kind terms are birds, ostriches, lions and tigers and bears (oh my!), spotted owls, black three-toed penguins, etc. To implement this, the "type" field which is part of the lexical entry for every word in the lexicon is set to *natkind* for these nouns. This type value is added to the noun's *type* feature (which is a list) and percolated up to the top-level. If the *type* features of the top-level NP and VP nodes both contain a value of *natkind*, and the main verb is the copula, then the sentence is strict. For example, the sentence "Penguins are birds" would be strict, while the sentence "Penguins like birds" would be defeasible.

A sentence describing a geographic principle — that is, relating one place to another — is understood to be strict. For example the sentence, "Anything in France is in Europe" is understood to express a strict rule. To avoid the problems of formalizing the use of prepositional phrases in DRT, DefDRT relies on "place" adjectives to indicate geographic relations. To express the France/Europe relationship, we would use the sentence, "A French thing is a European thing." Note that in DefDRT "thing" is a non-descriptive noun that, unlike other nouns, introduces no conditions into a DRS. For example, "a French thing" would introduce the condition *french(X)* into a DRS, whereas "a French bird" would introduce *french(X)* and *bird(X)*.

In general, adding adjectives to a "strict" sentence can change its interpretation to "defeasible." For example, we would interpret "Penguins are birds" as strict but would interpret "Black penguins are old birds" as defeasible. Since adjectives can complicate the strict/defeasible interpretation process, DefDRT will only interpret as strict two types of sentence forms which contain "place" adjectives. These are sentences which strictly equate one "place" adjective with another (i.e. "A French thing is a European thing"), and sentences in which the subject contains a "place" adjective and the predicate consists solely of a "place" adjective (i.e. "A French bird is European").

To implement this, the "type" field in the lexical entry for a "place" adjective contains the value "place." This value is added to the NP's *type* feature (which is a list) and percolated up to the top-level. As an example, the sentence "A French farmer is European" would have a top-level NP *type* feature of `[place, common]` — corresponding to the "type" fields in the lexical entries for "French" and "farmer." The VP *type* feature would be `[place]`. A comparison of these features is made and, if the NP *type* feature contains "place," the VP *type* feature contains only the value "place" and the main verb in the sentence is the copula then the sentence is strict. This test will succeed for both "A French farmer is European" and "A French thing is a European thing." Note that these criteria are not intended to capture the complex interaction of meanings that adjectives can produce in a sentence. They are only intended to provide a basic means of introducing strict geographic principles into the defeasible database.

A sentence which relates a singular entity to another singular entity ("A boy owns a dog") is taken to be strict. Although the conversion of this type of sentence into an equivalent DRS representation does not generate a conditional DR structure, as a general rule the strict/defeasible test is done for each sentence regardless of its form.

All other combinations of the feature structures involved are evaluated as defeasible. The following are sample sentences with their determined strictness or defeasibility as the program currently identifies them.

Every penguin is black.	strict
Mary likes every bird.	strict
Every penguin loves fish.	strict
A penguin is a bird.	strict
Penguins are birds.	strict
A French thing is a European thing.	strict
French farmers are European.	strict
Mary is old.	strict
Mary owns a donkey.	strict
A boy likes Mary.	strict
Penguins eat fish.	defeasible
Boys like girls.	defeasible
Penguins are black.	defeasible

Note that the feature *specificity* is used in this strict/defeasible determination. This requires that all singular indefinite NPs be evaluated as generic (plural) or non-generic (singular) before the strict/defeasible test can be applied.

4 Queries

Queries are not part of the formal DR theory, but they will be understood to be requests to prove whether or not a fact or rule holds based on the other information in a DRS. We will use the inference engine provided by Prolog and extended by d-Prolog to prove a query. The natural language expression of a query is subject to the same interpretation criteria that are used for declarative sentences (i.e. specific/generic interpretations of generic NPs, interpreting the sentence as strict or defeasible). In both DRT.GLP and DefDRT the syntactic structure of a sentence indicates that it is a query. The recognized forms of a query are (using phrase structure notation)

Q –> [Do/Does] NP, VP.
Q –> [Is/Are] NP, Adj.
Q –> [Is/Are] NP, NP.

Some allowable queries expressed in natural language are

Do birds fly?
Does Chilly like fish?
Is Cindy rich?
Are penguins birds?

When a query is encountered by the conversion routines, a "query" sub-DRS is introduced into the DRS under construction. Since queries are just requests to perform an inference, any new discourse entities and conditions within them are not left in the global DRS after the query has been performed. Once the discourse containing a query has been converted to a DRS, this DRS is passed through the prologization routines. These routines convert DRS information into d-Prolog facts and clauses which, except for queries, will be added or *asserted* into the database. A query is treated as a *goal* to be proven. In DefDRT, several different methods are used to prove goals. The method used for a particular query depends on what form the prologized query has. A prologized query will take one of three possible forms: a clause, a strict rule or a defeasible rule. Examples of each are shown below

Clause	Is Tom a farmer?	`farmer(tom)`
Strict rule	Is every penguin black?	`black(X) :- penguin(X)`
Defeasible rule	Do red birds fly?	`fly(X) := red(X), bird(X)`

If the query is a clause, the defeasible inference engine is called upon to prove it. To invoke the defeasible inference engine we preface the goal with the `@` operator. `@ Goal` will succeed in d-Prolog if:

1. `Goal` succeeds,
2. or, If the strict rule `Goal :- Conditions` exists, and `@ Conditions` succeeds,
3. or, If the defeasible rule `Goal := Conditions` exists and, `@ Conditions` succeeds and the rule `Goal := Conditions` is not *defeated*. Recall that a defeasible rule X can be *defeated* by a defeasible rule Y if Y is not inferior to X.

The following are examples of DefDRT using the defeasible engine to prove simple clause queries. The examples chosen are standard problems used in the nonmonotonic literature. Note that the strange "numbers" which appear in many of the asserted rules are simply Prolog's way of representing variables. [1]

The Nixon Diamond

This example involves competing defeasible rules. In processing the query, since neither of the competing rules is more specific, each defeats the other and the system can draw no conclusion.

```
>Nixon is a Republican. Republicans are not pacifists.
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> named([1],Nixon)
Asserting>> republican([1])
Asserting>> neg pacifist(_0D58) := republican(_0D58)
-------------------------------------------------

>Nixon is a Quaker. Quakers are pacifists. Is Nixon a pacifist?
Rule type:strict
Rule type:defeasible
Rule type:strict
-------------------------------------------------
Asserting>> quaker([1])
Asserting>> pacifist(_1004) := quaker(_1004)
query is:
pacifist([1])
Testing query.
can draw no conclusion
-------------------------------------------------
```

The Tweety Triangle

Two important points are demonstrated by the following example. First, DefDRT uses the lexical information that "birds" and "penguins" are natural kind terms to interpret "Penguins are birds" as a strict rule. Second, the defeasible inference engine uses this rule to determine that "Penguins do not fly" is more specific than, and therefore superior to "Birds fly." Thus DefDRT concludes that Tweety does not fly.

```
>Birds fly. Penguins are birds. Penguins do not fly.
Rule type:defeasible
Rule type:strict
```

[1] The actual output of DefDRT includes DRT information as well as the information seen here. The DRT information was edited out for brevity. The complete DefDRT output for these examples and others can be seen in Appendix A.

```
Rule type:defeasible
-------------------------------------------------
Asserting>> fly(_15D0) := bird(_15D0)
Asserting>> bird(_1698) :- penguin(_1698)
Asserting>> neg fly(_17F8) := penguin(_17F8)
-------------------------------------------------

>Tweety is a penguin. Tweety is a bird. Does Tweety fly?
Rule type:strict
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> named([4],Tweety)
Asserting>> penguin([4])
Asserting>> bird([4])
query is:
fly([4])
Testing query.
presumably, no
-------------------------------------------------
```

The Genetically Altered Penguin

This example demonstrates how a defeater rule affects defeasible inference. "A genetically altered penguin might fly" defeats the rule "Penguins do not fly," but does not change the fact that this last rule is superior to "Birds fly." Thus, although the defeater prevents us from concluding that Chilly does not fly, the rule "Birds fly" is still inferior to "Penguins do not fly" and so DefDRT can draw no conclusions about Chilly's ability to fly.

```
>Birds fly. Penguins do not fly. Penguins are birds.
Rule type:defeasible
Rule type:defeasible
Rule type:strict
-------------------------------------------------
Asserting>> fly(_0970) := bird(_0970)
Asserting>> neg fly(_0A34) := penguin(_0A34)
Asserting>> bird(_0B98) :- penguin(_0B98)
-------------------------------------------------

>A genetically_altered penguin might fly.
Rule type:defeater
-------------------------------------------------
Asserting>> fly(_4828) :^ penguin(_4828) , genetically_altered(_4828)
-------------------------------------------------

>Chilly is a penguin. Does he fly?
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> named([0],Chilly)
Asserting>> penguin([0])
query is:
fly([0])
```

```
Testing query.
presumably, no -
-------------------------------------------------

>Chilly is genetically_altered. Does Chilly fly?
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> genetically_altered([0])
query is:
fly([0])
Testing query.
can draw no conclusion
-------------------------------------------------
```

College Student

This example illustrates the importance of using defeasible rules in determining that one rule is more specific than another. The query below uses *defeasible specificity*, which means that both strict and defeasible rules are used in determining whether one rule is more specific than another.

```
>Students do not work. Students are adults. Adults work.
Rule type:defeasible
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Asserting>> neg work(_09D0) := student(_09D0)
Asserting>> adult(_0AB8) := student(_0AB8)
Asserting>> work(_0C94) := adult(_0C94)
-------------------------------------------------

>Carl is a student. Does he work?
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> named([0],Carl)
Asserting>> student([0])
query is:
work([0])
Testing query.
presumably, no -
-------------------------------------------------
```

Platypus

Although the relationships below indicate that platypuses are typically duckbilled and duckbilled things are typically feathered, the conclusion that platypuses are feathered is defeated by the specific rule that platypuses are not feathered. Gertrude, however, is a platypus who is unusual in that she is feathered. If Gertrude were a typical platypus the rule that platypuses are furry would be superior and we could conclude that she is furry and not feathered. However, with the specific information that she is feathered, no rule is superior. Thus, when asked if Gertrude is furry, DefDRT responds that it can't tell.

```
>Platypuses are furry. A feathered thing is not furry.
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Asserting>> furry(_0E54) := platypus(_0E54)
Asserting>> neg furry(_0F5C) := feathered(_0F5C)
-------------------------------------------------

>A platypus is duckbilled. A duckbilled thing is feathered.
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Asserting>> duckbilled(_1118) := platypus(_1118)
Asserting>> feathered(_1278) := duckbilled(_1278)
-------------------------------------------------

>A platypus is not feathered.
Rule type:defeasible
-------------------------------------------------
Asserting>> neg feathered(_5B7C) := platypus(_5B7C)
-------------------------------------------------

>Gertrude is a feathered platypus. Is she furry?
Rule type:strict
Rule type:strict
-------------------------------------------------
Asserting>> named([2],Gertrude)
Asserting>> platypus([2])
Asserting>> feathered([2])
query is:
furry([2])
Testing query.
can draw no conclusion
-------------------------------------------------
```

Pennsylvania Dutch

The following example illustrates the use of a geographic relationship to express a strict rule. DefDRT interprets "Pennsylvanian" and "American" as place terms and as a result generates a strict rule when the two are related in a sentence. Note that the determiner "every" is used to force a strict interpretation of the sentence "Every Pa_dutch speaking person is a German speaking person." Interpreting "Pa_dutch speaking" and "German speaking" as place terms in DefDRT would cause an incorrect interpretation of "strict" for sentences combining them with "Pennsylvanian" and "American."

```
>A Pennsylvanian is an American.
Rule type:strict
-------------------------------------------------
Asserting>> american(_4324) :- pennsylvanian(_4324)
-------------------------------------------------

>A German_speaking person is not an American.
Rule type:defeasible
-------------------------------------------------
```

```
Asserting>> neg american(_5654) := german_speaking(_5654)
-------------------------------------------------

>A Pa_dutch_speaking person is a Pennsylvanian.
Rule type:defeasible
-------------------------------------------------
Asserting>> pennsylvanian(_58CC) := pa_dutch_speaking(_58CC)
-------------------------------------------------

>Every Pa_dutch_speaking person is German_speaking.
Rule type:strict
-------------------------------------------------
Asserting>> german_speaking(_7A00) :- pa_dutch_speaking(_7A00)
-------------------------------------------------

>Hans is a Pa_dutch_speaking person. Is he an American?
Rule type:strict
Rule type:strict
-------------------------------------------------
Asserting>> named([0],Hans)
Asserting>> pa_dutch_speaking([0])
query is:
american([0])
Testing query.
 The result is : yes
-------------------------------------------------

>Is Hans a German_speaking person?
Rule type:strict
-------------------------------------------------
query is:
german_speaking([0])
Testing query.
 The result is : yes
-------------------------------------------------
```

Poor Fred

Using defeasible specificity to interpret a query in the example below, no rule is judged to be superior. Thus, when asked "Is Fred poor?" DefDRT responds that it can draw no conclusions. d-Prolog allows us to use *preemption* to prohibit a defeated rule from defeating a competing rule. Given that we are trying to prove `A`, we might have the rule `A := B` and a competing rule `neg A := C`. If neither rule is superior then both are defeated and no inference can be made. If we allow preemption, `neg A := C` will be *preempted* if there is a rule superior to it of the form `A := D`. If `A := D` is superior to `neg A := C`, it defeats it and also preempts or blocks its use as a defeater of `A := B`, thus allowing `A` to be inferred. Preemption in the d-Prolog defeasible inference engine can be activated from within DefDRT by typing "preempt." Preemption is active for the second query below, which allows it to succeed. Note that "phded" below is interpreted as "has a Ph.D."

```
>A person who teaches is phded. A phded person is not poor.
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
```

```
Asserting>> phded(_3940) := teach(_3940)
Asserting>> neg poor(_8DB0) := phded(_8DB0)
--------------------------------------------------

>A person who teaches is poor.
Rule type:defeasible
--------------------------------------------------
Asserting>> poor(_60D4) := teach(_60D4)
--------------------------------------------------

>A legal_aid is law_degreed.
Rule type:defeasible
--------------------------------------------------
Asserting>> law_degreed(_58D0) := legal_aid(_58D0)
--------------------------------------------------

>A person who is law_degreed is not poor.
Rule type:defeasible
--------------------------------------------------
Asserting>> neg poor(_71EC) := law_degreed(_71EC)
--------------------------------------------------

>A legal_aid is poor.
Rule type:defeasible
--------------------------------------------------
Asserting>> poor(_4D64) := legal_aid(_4D64)
--------------------------------------------------

>Fred is a legal_aid who teaches.
Rule type:strict
--------------------------------------------------
Asserting>> named([0],Fred)
Asserting>> legal_aid([0])
Asserting>> teach([0])
--------------------------------------------------

>Is Fred poor?
Rule type:strict
--------------------------------------------------
query is:
poor([0])
Testing query.
can draw no conclusion
--------------------------------------------------

>preempt
Preemption is now enabled.

>Is Fred poor?
Rule type:strict
--------------------------------------------------
query is:
poor([0])
```

```
Testing query.
 The result is : yes
-------------------------------------------------
```

Besides clauses, DefDRT has methods for proving queries in two other forms. If the query is of the form `A :- B` (strict rule) or `A := B` (defeasible rule), then we will attempt to derive `A` from `B` using *strict deduction* or *defeasible deduction* depending on the query.

For a rule of the form `A :- B`, *strict deduction* can be stated as, "for an individual that satisfies B try to prove that the same individual also satisfies A." Given the d-Prolog rule `black(X) :- penguin(X)` (every penguin is black), we would create an individual which satisfies the antecedent (`penguin(dummy)`) and temporarily assert it into the database. Because this is a strict rule, we call upon the standard Prolog inference engine to prove the goal `black(dummy)`. Standard Prolog will only use strict rules and facts in its attempt to prove this goal. If this succeeds the program reports the success to the user. If this test fails we cannot conclude that `A :- B` does not hold, but only that `A` cannot be proven to follow from `B`. In this situation, DefDRT attempts to find a counterexample. A counterexample will be an instance that satisfies `@ B` from which we can derive `@ neg A`. If such an instance is found the system reports that a counterexample exists and that the rule `A :- B` does not hold. If a counter example cannot be found then there is insufficient evidence for or against the rule; thus the system reports that it can draw no conclusions.

If the query has the form `A := B`, then the *defeasible deduction* test is invoked. It involves assuming an individual who satisfies `B`, and then attempting to defeasibly prove `A`. It differs from *strict deduction* in that it uses the defeasible inference engine and involves another level of processing if it succeeds. If *defeasible deduction* is successful a counterexample test is performed. This involves looking for an instance that defeasibly satisfies `B` and from which we can defeasibly derive `neg A`. If no counterexample is found then the result of the query is simply "yes." If a counterexample is found the result is "yes — but exceptions exist." If the *defeasible deduction* test fails altogether, the result is "no" — `A` cannot be defeasibly derived from `B`.

The sample output below illustrates both the strict and defeasible deduction methods. The strict test is invoked below by using the determiner "every" in the query.

```
>Birds fly. Penguins are birds. Penguins do not fly.
Rule type:defeasible
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Asserting>> fly(_0970) := bird(_0970)
Asserting>> bird(_0A38) :- penguin(_0A38)
Asserting>> neg fly(_0C14) := penguin(_0C14)
-------------------------------------------------

>Is every penguin a bird?
Rule type:strict
-------------------------------------------------
query is:
bird(_31B4) :- penguin(_31B4)
Testing query.
Deductive (strict) analysis
 The result is : yes
-------------------------------------------------

>Does every bird fly?
Rule type:strict
-------------------------------------------------
query is:
fly(_2C04) :- bird(_2C04)
```

```
Testing query.
Deductive (strict) analysis
    can draw no conclusions
-------------------------------------------------

>Do birds fly?
Rule type:defeasible
-------------------------------------------------
query is:
fly(_2188) := bird(_2188)
Testing query.
Deductive (defeasible) analysis
   Yes.
-------------------------------------------------

>Chilly is a penguin.
Rule type:strict
-------------------------------------------------
Asserting>> named([7],Chilly)
Asserting>> penguin([7])
-------------------------------------------------

>Does every bird fly?
Rule type:strict
-------------------------------------------------
query is:
fly(_33A4) :- bird(_33A4)
Testing query.
Deductive (strict) analysis
   counter example found: bird([7])
 The result is : no
-------------------------------------------------

>Do birds fly?
Rule type:defeasible
-------------------------------------------------
query is:
fly(_2244) := bird(_2244)
Testing query.
Deductive (defeasible) analysis
   Yes, but exceptions exist.
-------------------------------------------------
```

This combination of testing methods provides an adequate means for querying the database for rules and clauses. The reader is directed to Appendix A for a variety of discourse examples and the processing of these by DefDRT.

5 Problems

5.1 Collective Plurals

A problem is caused by the difference in meaning of *donkeys* in the following two sentences.

Farmers like donkeys.

Farmers own donkeys.

The distinction becomes clear if we try to reason using these sentences. Given that Pedro is a farmer and Chiquita is a donkey, using the sentence "Farmers like donkeys" we can intuitively conclude that Pedro likes Chiquita. But, given the sentence "Farmers own donkeys," intuitively it does not seem to follow that Pedro owns Chiquita. Although both sentences use the plural form "donkeys," the interpretation of what it means in each sentence is different. We interpret the first sentence as expressing a universal relationship between any farmer and any donkey. In the second sentence, "donkeys" has a more restricted meaning; the verb "own" doesn't relate every farmer to every donkey (as "like" does in the first sentence), but rather relates every farmer with some particular (existential) collection of donkeys. Since DefDRT relies on syntactic structure to interpret sentences, it incorrectly interprets "Farmers own donkeys" as a generic relationship (if X is a farmer and Y is a donkey then X owns Y).

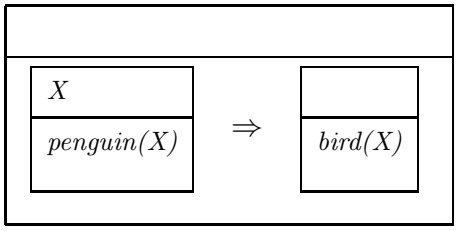
Distinguishing between the two interpretations involves more than just syntactic information. Sentences like "Penguins eat fish" and "Harry grows roses" seem to require semantic and pragmatic information in interpreting what they mean. Since DefDRT does not have access to information beyond the syntactic level, it cannot distinguish between a universal and an existential interpretation of a plural NP.

5.2 Resolving Plural Pronouns

The translation of a sentence containing a generic plural results in the creation of a sub-DRS which has the form of a conditional. For example the sentence

Penguins are birds.

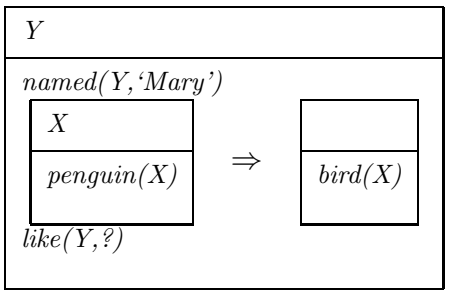
generates the following DRS.



Since DRT does not specify exact conditions for determining what information beyond the current sentence should go into the sub-DRS, the end of the sentence is usually taken as the indicator to stop adding to the sub-DRS and to resume adding information to the super-most DRS. Thus if the sentence above is followed by the sentence,

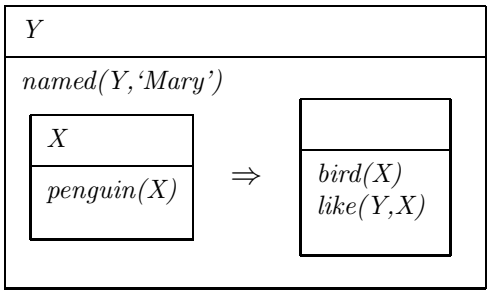
Mary likes them.

the resulting DRS is:



DR theory specifies that the search for the antecedent to an anaphoric NP should begin in the current DRS and proceeds upwards through the accessible DRSs in order of increasing dominance, checking the universe of discourse of each DRS until a suitable referent is found. Applying this procedure to the DRS above, we find that the search for the antecedent to the pronoun "them" can only access the discourse

markers in the super-most DRS, and therefore "them" can't be resolved. A solution to this problem depends on the recognition that the second sentence is related to the first, and therefore should be incorporated into the consequent of the sub-DRS as shown below.



Roberts (1987) describes a similar phenomenon which she calls *modal subordination*. She indicates that one sentence may temporarily restrict the context for the discourse through the introduction of a modal ("might," "would," "may," etc.). A sentence which follows would show its continuation of this sub-context by continuing the mood. In the discourse

If John bought a book, he'll be home reading it by now.

It'll be a murder mystery.

the second sentence by it's use of modal "will" indicates that the context established by the first (non-factual mood) continues.

A possible solution to this problem would be to introduce a separate discourse marker list into the DRT implementation. As a discourse marker is added to a DRS, a marker would also be added to this special list along with the name or number of the DRS that the marker was just added to. The search for the antecedent to an anaphoric phrase would proceed normally as described by the theory. If this search failed, the special list would be searched, and if a referent could be found the DRS that contains the anaphoric phrase would be inserted into the DRS where the referent was found. This approach is similar to the process Roberts calls *insertion*. Determining when and when not to perform insertion is a complex problem in its own right, and even solving this problem we are left with defeasible issues to resolve.

5.3 The Insertion Problem

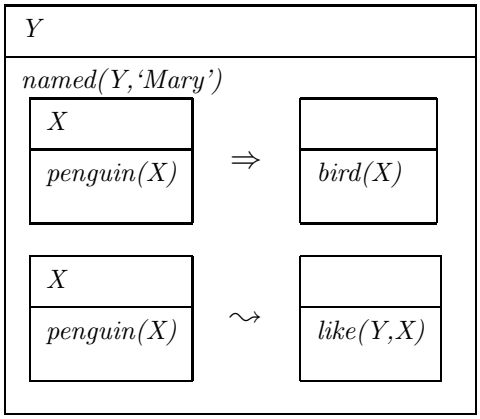
In the first sentence of the discourse

Penguins are birds. Mary likes them.

"Penguins are birds" translates into a strict rule. The second sentence "Mary likes them (penguins)" translates into a defeasible rule. By simply inserting "Mary likes them" into the consequent of the DRS generated by the first sentence, we mistakenly make the "likes" relationship a strict one. Using d-Prolog notation the proper interpretation of the discourse should be:

```
bird(X) :- penguin(X).  /* penguins are birds.(strict) */
named([1],'Mary').
like([1],X) := penguin(X)./* mary likes penguins.(defeasible) */
```

To generate this interpretation the DRS would have to include two conditional DRSs, a strict one representing "Penguins are birds" and a defeasible one representing "Mary likes penguins." This would be symbolized as



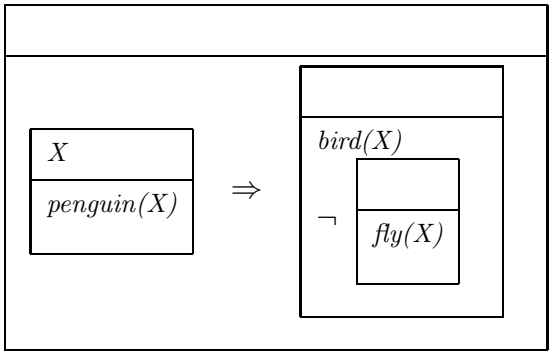
It is unclear how this DRS could be generated from only the syntactic information of the sentences in the discourse above.

5.4 Intrasentential Strict/Defeasible Mixtures

The strict/defeasible determination process in DefDRT operates under the assumption that a sentence contains either strict or defeasible information, but not both. Although this may be true of simple declarative sentences it is not true of more complex sentences. Consider the following example.

Penguins are birds that do not fly.

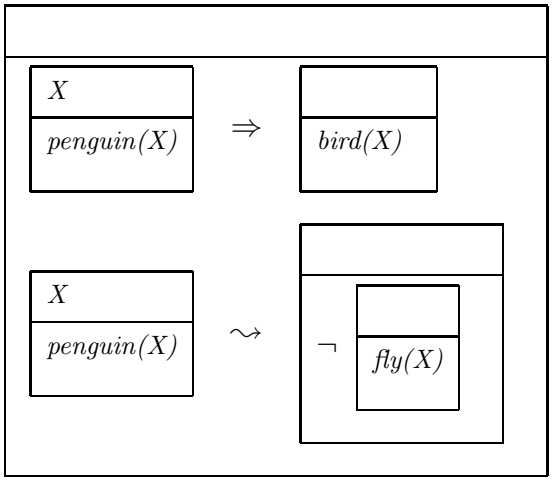
This sentence expresses the strict relationship "Penguins are birds" and the defeasible relationship "Penguins do not fly." Unfortunately, since DefDRT interprets a sentence as either strict of defeasible it cannot correctly interpret this sentence, and as a result it would be interpreted, by default, as defeasible. Note that this mixture of strict and defeasible information causes a problem similar to the one caused by insertion. Consider the DRS generated from the sentence above.



This DRS would be converted into the d-Prolog rules

```
bird(X) :- penguin(X).
neg fly(X) :- penguin(X).
```

By using the strict operator ⇒ the DRS incorrectly represents the statement "Penguins do not fly" as a strict rule. Changing the operator to the defeasible operator ⇝ incorrectly represents "Penguins are birds" as a defeasible rule. The proper representation is a DRS similar to the one given in the previous section



Once again, the problem is that this DRS structure does not follow directly from the sentence it is based on. A general solution will probably involve interpreting the semantic as well as the syntactic relationships in the sentence.

5.5 Prologization Problems

Prolog and the d-Prolog extension both require that rules have a certain logical form. A Prolog rule has an "if P then Q" form, although this is actually expressed as "Q if P." The consequent Q is called the *head* of the rule and the antecedent P is called the *body*. The body may contain multiple clauses, but the head may only consist of a single clause. The head and body of a rule are joined together with a connector that indicates the type of rule it is. The three rule connectors allowed by d-Prolog are "`:-`" (strict), "`:=`" (defeasible) and "`:^ `" (defeater). A valid d-Prolog rule can contain only one of these.

Because of these requirements, DefDRT converts DRS information into an intermediate form, then, if necessary, manipulates that form to produce usable d-Prolog facts and rules. For example, the sentence

If Chilly is a bird then penguins are birds.

is initially converted to

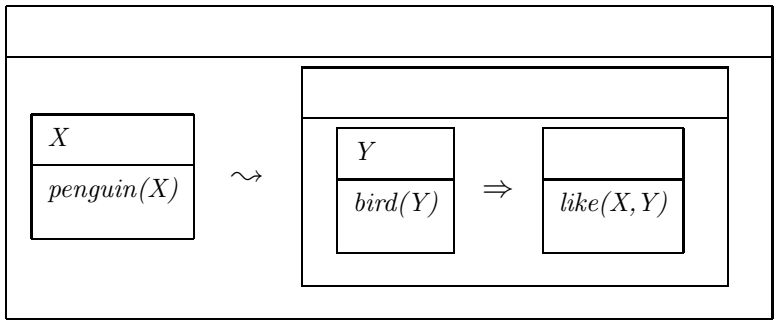
```
(bird(X) :- penguin(X)) :- bird(chilly).
```

Using *exportation* from classical logic we can transform this rule into the usable d-Prolog rule

```
bird(X) :- penguin(X), bird(chilly).
```

Although no such formal transformation exists for an intermediate form containing defeasible connectors, DefDRT informally uses the same process for the defeasible style intermediate form. Further, there are no formal methods for manipulating an intermediate representation containing both strict and defeasible connectors. Such an intermediate form can be generated by an "If ... then ..." sentence such as "If tweety flies then birds fly" or by mixing universal determiners with defeasible information. For example

Penguins like all birds.



This structure would generate the intermediate form

```
(like(X,Y) :- bird(Y)) := penguin(X)
```

Since this form is similar to the ones above, it might be possible to perform a similar manipulation and generate a valid d-Prolog rule. In this case, however, since the operators are not all the same it seems reasonable to use the weaker of the two in the final representation. This would give us

```
like(X,Y) := bird(Y), penguin(X)
```

We might generalize this process to be

`(p :? q) :? r` becomes `p := q,r`
If either occurrence of `:?` is the defeasible operator `:=`.[2]

Although this transformation has an intuitive appeal it lacks a formal foundation. Even if we use this transformation, we cannot create a valid d-Prolog rule from the sentence

If birds fly then robins fly.
`(fly(X) := robin(X)) :- (fly(Y) := bird(Y))`

It is apparent that this representation requires a larger set of defeasible transformation procedures.

The underlying source of these prologization problems is the lack of a formal set of defeasible equivalence transformations. These formal transformations are essential for creating a d-Prolog database involving the more complex defeasible relationships which can be expressed in natural language.

6 Conclusions

In looking for a way to incorporate defeasible reasoning into the original DRT implementation I have solved some problems and discovered others. The defeasible operators ($\rightsquigarrow$, $\sim$) introduced into DR theory, although convenient, do not share the model theory foundation that the original $\Rightarrow$ operator does. There seems to be as of yet no universally accepted semantics for defeasible rules. In Nute's formalism, rules are described as "policies for forming and revising beliefs" which as such don't have truth values. The search for a semantics for DRT extended with defeasible operators is left unexplored.

English discourse is notoriously economic. For example, the sentence "Presumably, a penguin who both eats fish and likes every bird might be happy" expresses a great deal of information in a single sentence. The problem in interpreting this from a defeasible reasoning point of view is that we must distinguish between the strict relationships, defeasible relationships, presumptions and defeaters. A syntactic parse is only part of the information needed to find these relationships. It is apparent that the formalism used in DefDRT for converting syntactic information into d-Prolog rules just scratches the surface.

One of the major problems encountered in this project was the lack of equivalence rules for transforming DRS output into usable d-Prolog facts and rules. DefDRT can create complex intermediate rules from a DRS which are useless in the d-Prolog database because of their form. The process of translating from natural language to d-Prolog would benefit greatly from a set of defeasible equivalence rules.

DefDRT is capable of accepting natural language discourse over multiple executions and converting it into appropriate facts and rules in a defeasible database. The information in this database is also written out to a text file that can be read in at the start of a future discourse session. Several future directions seem appropriate to extend DefDRT even further:

- Implement a table of identity based on the suggestions in (Covington, Nute, Schmitz and Goodman 1988). This will allow the system to recognize when two different names refer to the same individual.
- If during a query a clause or rule is proven to be true, assert it to the defeasible database. This would capture the intuitively attractive idea that the system should learn as a result of a query.

[2]This method was suggested by Donald Nute.

- Add truth maintenance procedures for ensuring that new information doesn't invalidate the existing database. In the event that a user enters contradictory information, the user would be given a choice by the system of removing the rules from the database which cause the contradiction, ignoring the current discourse, or asserting the new information and allowing the contradiction.

6.1 References

Asher, N. (1986) Belief in discourse representation theory. *Journal of Philosophical Logic* 15.2:127–189.

Covington, M. (1989) *GULP 2.0: An extension of Prolog for unification-based grammar.* ACMC Research Report AI–1989–01, University of Georgia.

Covington, M., and Schmitz, N. (1988) *An implementation of discourse representation theory.* ACMC Research Report 01–0023, University of Georgia.

Covington, M.; Nute, D.; and Vellino, A. (1988) *Prolog programming in depth.* Glenview, Illinois: Scott, Foresman.

Covington, M.; Nute, D.; Schmitz, N.; and Goodman, D. (1988) *From English to Prolog via discourse representation theory.* ACMC Research Report 01–0024, University of Georgia.

Declerck, R. (1991) The origins of genericity. *Linguistics* 29:79–102.

Diesing, M. (1992) Bare plural subjects and the derivation of logical representations. *Linguistic Inquiry* 23.3:353–380.

Ginsberg, M., ed. (1987) *Readings in nonmonotonic reasoning.* Los Altos, CA: Morgan Kaufmann.

Goodman, D. (1989) *An implementation of an extension to discourse representation theory: translating natural language to discourse representation structures to Prolog clauses.* Master's Thesis, University of Georgia.

Guenthner, F. (1987) Linguistic meaning in discourse representation theory. *Synthese* 73.3:569–598.

Herzog, O., and Rollinger, C.-R., eds. (1991) *Text understanding in LILOG: integrating computational linguistics and artificial intelligence: final report on the IBM Germany LILOG project.* New York: Springer-Verlag.

Heyer, G. (1990) Semantics and knowledge representation in the analysis of generic descriptions. *Journal of Semantics* 7.1:93–110.

Johnson, M., and Klein, E. (1986) *Discourse, anaphora and parsing.* CSLI Research Report 86–63, Stanford University.

Kamp, H. (1981) A theory of truth and semantic representation. In J. Groenendijk, T. Janssen, and M. Stokhof, eds., *Formal methods in the study of language*, 277–332. University of Amsterdam.

Kamp, H. (1985) Unpublished discourse representation theory project description, University of Texas, Austin.

Kamp, H. (1988) Conditionals in DR theory. In J. Ph. Hoepelman, *Representation and reasoning: proceedings of the Stuttgart conference workshop on discourse representation, dialogue tableaux and logic programming*, 66–124. Tubingen: Max Niemeyer Verlag.

Kamp, H. (undated manuscript) *From discourse to logic: introduction to model theoretic semantics of natural language, formal logic and discourse representation theory.* Institute for Computational Linguistics, University of Stuttgart.

Nute, D. (1993) Defeasible logic. To appear in D. Gabbay and C. Hogger, eds., *Handbook of logic for artificial intelligence and logic programming*, Vol. III, Oxford, 1993.

Nute, D., and Lewis, M. (1986) *A user's manual for d-Prolog.* ACMC Research Report 01–0017, University of Georgia.

Roberts, C. (1987) *Modal subordination, anaphora, and distributivity.* Ph.D. Thesis, University of Massachusetts at Amherst.

Smith, H. (1989) *Constrained clauses in discourse representation theory.* Master's Thesis, University of Georgia.

Spencer-Smith, R. (1987) Semantics and discourse representation. *Mind and Language* 2.1:1–26.

6.2 DefDRT Sample Executions

The Nixon Diamond

```
>Nixon is a republican. Republicans are not pacifists.
[nixon,is,a,republican,.,republicans,are,not,pacifists,.]
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1]]
gender([1],m,sg,nixon)
named([1],Nixon)
gender([1],m,sg,republican)
republican([1])
NOT:
  []
  IF:
    [_0D58]
    gender(_0D58,_0E14,pl,republicans)
    republican(_0D58)
  THEN:defeasible
    []
    IF:
      [_0D58]
      gender(_0D58,_0EB4,pl,pacifists)
      pacifist(_0D58)
    THEN:defeasible
      []
-------------------------------------------------
Asserting>> named([1],Nixon)
Asserting>> republican([1])
Asserting>> neg pacifist(_0D58) := republican(_0D58)
-------------------------------------------------

>Nixon is a quaker. Quakers are pacifists. Is Nixon a pacifist?
[nixon,is,a,quaker,.,quakers,are,pacifists,.,is,nixon,a,pacifist,?]
Rule type:strict
Rule type:defeasible
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1]]
gender([1],m,sg,quaker)
quaker([1])
IF:
  [_1004]
  gender(_1004,_10CC,pl,quakers)
  quaker(_1004)
THEN:defeasible
  []
```

```
  IF:
    [_1004]
    gender(_1004,_116C,pl,pacifists)
    pacifist(_1004)
  THEN:defeasible
    []
QUERY:
  [[1]]
  gender([1],_4BEC,sg,pacifist)
  pacifist([1])
-------------------------------------------------
Asserting>> quaker([1])
Asserting>> pacifist(_1004) := quaker(_1004)
query is:
pacifist([1])
Testing query.
   using @@ Goal
can draw no conclusion
-------------------------------------------------
```

Tweety (actually Chilly) Triangle

```
>Birds fly. Penguins are birds. Penguins do not fly.
[birds,fly,.,penguins,are,birds,.,penguins,do,not,fly,.]
Rule type:defeasible
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2]]
IF:
  [_12AC]
  gender(_12AC,_1320,pl,birds)
  bird(_12AC)
THEN:defeasible
  []
  fly(_12AC)
IF:
  [_13F0]
  gender(_13F0,_14B8,pl,penguins)
  penguin(_13F0)
THEN:strict
  []
  IF:
    [_13F0]
    gender(_13F0,_1558,pl,birds)
    bird(_13F0)
  THEN:strict
    []
IF:
  [_1A38]
  gender(_1A38,_389C,pl,penguins)
  penguin(_1A38)
THEN:defeasible
  []
  NOT:
```

```
    []
    fly(_1A38)
------------------------------------------------
Asserting>> fly(_1188) := bird(_1188)
Asserting>> bird(_1250) :- penguin(_1250)
Asserting>> neg fly(_13B0) := penguin(_13B0)
------------------------------------------------

>Chilly is a penguin. Chilly is a bird. Does Chilly fly?
[chilly,is,a,penguin,.,chilly,is,a,bird,.,does,chilly,fly,?]
Rule type:strict
Rule type:strict
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2],[3],[3],[3]]
gender([3],_14B0,sg,chilly)
named([3],Chilly)
gender([3],_14B0,sg,penguin)
penguin([3])
gender([3],_14B0,sg,bird)
bird([3])
QUERY:
  []
  fly([3])
------------------------------------------------
Asserting>> named([3],Chilly)
Asserting>> penguin([3])
Asserting>> bird([3])
query is:
fly([3])
Testing query.
   using @@ Goal
presumably, no -
------------------------------------------------

>Do old birds fly?
[do,old,birds,fly,?]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2],[3],[3],[3]]
QUERY:
  []
  IF:
    [_3478]
    gender(_3478,_60E8,pl,birds)
    bird(_3478)
    old(_3478)
  THEN:defeasible
    []
    fly(_3478)
------------------------------------------------
query is:
fly(_1078) := bird(_1078) , old(_1078)
Testing query.
```

```
defeasible analysis using @
Deductive (def) analysis
   Yes.
-------------------------------------------------
```

Genetically Altered Penguin

```
>A genetically_altered penguin might fly.
[a,genetically_altered,penguin,might,fly,.]
Rule type:defeater
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2],[3],[3],[3]]
IF:
  [_4E94]
  gender(_4E94,_8038,sg,penguin)
  penguin(_4E94)
  genetically_altered(_1318)
THEN:defeater
  []
  fly(_1318)
-------------------------------------------------
Asserting>> fly(_1318) :^ penguin(_1318) , genetically_altered(_1318)
-------------------------------------------------

>Chilly is genetically_altered. Does he fly?
[chilly,is,genetically_altered,.,does,he,fly,?]
Rule type:strict
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2],[3],[3],[3]]
genetically_altered([3])
QUERY:
  []
  fly([3])
-------------------------------------------------
Asserting>> genetically_altered([3])
query is:
fly([3])
Testing query.
   using @@ Goal
can draw no conclusion
-------------------------------------------------
```

College Student

```
>Students do not work. Students are adults. Adults work.
[students,do,not,work,.,students,are,adults,.,adults,work,.]
Rule type:defeasible
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
```

```
------------------------------------------------
[]
IF:
  [_09D0]
  gender(_09D0,_0A44,pl,students)
  student(_09D0)
THEN:defeasible
  []
  NOT:
    []
    work(_09D0)
IF:
  [_0AB8]
  gender(_0AB8,_0B5C,pl,students)
  student(_0AB8)
THEN:defeasible
  []
  IF:
    [_0AB8]
    gender(_0AB8,_0BFC,pl,adults)
    adult(_0AB8)
  THEN:defeasible
    []
IF:
  [_0C94]
  gender(_0C94,_0D2C,pl,adults)
  adult(_0C94)
THEN:defeasible
  []
  work(_0C94)
------------------------------------------------
Asserting>> neg work(_09D0) := student(_09D0)
Asserting>> adult(_0AB8) := student(_0AB8)
Asserting>> work(_0C94) := adult(_0C94)
------------------------------------------------

>Carl is a student. Does he work?
[carl,is,a,student,.,does,he,work,?]
Rule type:strict
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[[0],[0]]
gender([0],m,sg,carl)
named([0],Carl)
gender([0],m,sg,student)
student([0])
QUERY:
  []
  work([0])
------------------------------------------------
Asserting>> named([0],Carl)
Asserting>> student([0])
query is:
work([0])
Testing query.
   using @@ Goal
```

```
presumably, no -
-------------------------------------------------
```

Platypus

```
>Platypuses are furry. A feathered thing is not furry.
[platypuses,are,furry,.,a,feathered,thing,is,not,furry,.]
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1]]
IF:
  [_0E3C]
  gender(_0E3C,_0E80,pl,platypuses)
  platypus(_0E3C)
THEN:defeasible
  []
  furry(_0E3C)
NOT:
  []
  IF:
    [_0F44]
    gender(_0F44,n,sg,thing)
    feathered(_0F44)
  THEN:defeasible
    []
    furry(_0F44)
-------------------------------------------------
Asserting>> furry(_0E3C) := platypus(_0E3C)
Asserting>> neg furry(_0F44) := feathered(_0F44)
-------------------------------------------------

>A platypus is duckbilled. A duckbilled thing is feathered.
[a,platypus,is,duckbilled,.,a,duckbilled,thing,is,feathered,.]
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0],[1],[1],[1]]
IF:
  [_1118]
  gender(_1118,_115C,sg,platypus)
  platypus(_1118)
THEN:defeasible
  []
  duckbilled(_1118)
IF:
  [_1278]
  gender(_1278,n,sg,thing)
  duckbilled(_1278)
THEN:defeasible
  []
  feathered(_1278)
-------------------------------------------------
```

```
Asserting>> duckbilled(_1118) := platypus(_1118)
Asserting>> feathered(_1278) := duckbilled(_1278)
--------------------------------------------------

>A platypus is not feathered.
[a,platypus,is,not,feathered,.]
Rule type:defeasible
--------------------------------------------------
Cleaned-up (simplified) DRS:
--------------------------------------------------
[[0],[0],[1],[1],[1]]
NOT:
  []
  IF:
    [_5B7C]
    gender(_5B7C,_5DAC,sg,platypus)
    platypus(_5B7C)
  THEN:defeasible
    []
    feathered(_5B7C)
--------------------------------------------------
Asserting>> neg feathered(_5B7C) := platypus(_5B7C)
--------------------------------------------------

>Gertrude is a feathered platypus. Is she furry?
[gertrude,is,a,feathered,platypus,.,is,she,furry,?]
Rule type:strict
Rule type:strict
--------------------------------------------------
Cleaned-up (simplified) DRS:
--------------------------------------------------
[[0],[0],[1],[1],[1],[2],[2]]
gender([2],f,sg,gertrude)
named([2],Gertrude)
gender([2],f,sg,platypus)
platypus([2])
feathered([2])
QUERY:
  []
  furry([2])
--------------------------------------------------
Asserting>> named([2],Gertrude)
Asserting>> platypus([2])
Asserting>> feathered([2])
query is:
furry([2])
Testing query.
   using @@ Goal
can draw no conclusion
--------------------------------------------------
```

Pennsylvania Dutch

```
>A Pennsylvanian is an American.
[a,pennsylvanian,is,an,american,.]
Rule type:strict
--------------------------------------------------
```

```
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_4324]
  aux_place(_4324,pennsylvanian)
  gender(_4324,_5D10,sg,pennsylvanian)
  pennsylvanian(_4324)
THEN:strict
  [_4324]
  aux_place(_4324,american)
  gender(_4324,_72E0,sg,american)
  american(_4324)
------------------------------------------------
Asserting>> american(_4324) :- pennsylvanian(_4324)
------------------------------------------------

>A German_speaking person is not an American.
[a,german_speaking,person,is,not,an,american,.]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
NOT:
  []
  IF:
    [_5654]
    gender(_5654,n,sg,person)
    german_speaking(_5654)
  THEN:defeasible
    [_5654]
    aux_place(_5654,american)
    gender(_5654,_92A8,sg,american)
    american(_5654)
------------------------------------------------
Asserting>> neg american(_5654) := german_speaking(_5654)
------------------------------------------------

>A Pa_dutch_speaking person is a Pennsylvanian.
[a,pa_dutch_speaking,person,is,a,pennsylvanian,.]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_58CC]
  gender(_58CC,n,sg,person)
  pa_dutch_speaking(_58CC)
THEN:defeasible
  [_58CC]
  aux_place(_58CC,pennsylvanian)
  gender(_58CC,_9518,sg,pennsylvanian)
  pennsylvanian(_58CC)
------------------------------------------------
Asserting>> pennsylvanian(_58CC) := pa_dutch_speaking(_58CC)
------------------------------------------------
```

```
>Every Pa_dutch_speaking person is German_speaking.
[every,pa_dutch_speaking,person,is,german_speaking,.]
Rule type:strict
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_7A00]
  gender(_7A00,n,sg,person)
  pa_dutch_speaking(_7A00)
THEN:strict
  []
  german_speaking(_7A00)
------------------------------------------------
Asserting>> german_speaking(_7A00) :- pa_dutch_speaking(_7A00)
------------------------------------------------

>Hans is a Pa_dutch_speaking person. Is he an American?
[hans,is,a,pa_dutch_speaking,person,.,is,he,an,american,?]
Rule type:strict
Rule type:strict
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[[0],[0]]
gender([0],m,sg,hans)
named([0],Hans)
gender([0],n,sg,person)
pa_dutch_speaking([0])
QUERY:
  [[0]]
  aux_place([0],american)
  gender([0],_0E6C,sg,american)
  american([0])
------------------------------------------------
Asserting>> named([0],Hans)
Asserting>> pa_dutch_speaking([0])
query is:
american([0])
Testing query.
   using @ Goal
 The result is : yes
------------------------------------------------

>Is Hans a German_speaking person?
[is,hans,a,german_speaking,person,?]
Rule type:strict
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[[0],[0]]
QUERY:
  [[0]]
  gender([0],n,sg,person)
  german_speaking([0])
------------------------------------------------
```

```
query is:
german_speaking([0])
Testing query.
   using @ Goal
 The result is : yes
-------------------------------------------------
```

Poor Fred

```
>A person who teaches is phded. A phded person is not poor.
[a,person,who,teaches,is,phded,.,a,phded,person,is,not,poor,.]
Rule type:defeasible
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[]
IF:
  [_3940]
  gender(_3940,n,sg,person)
  teach(_3940)
THEN:defeasible
  []
  phded(_3940)
NOT:
  []
  IF:
    [_8DB0]
    gender(_8DB0,n,sg,person)
    phded(_8DB0)
  THEN:defeasible
    []
    poor(_8DB0)
-------------------------------------------------
Asserting>> phded(_3940) := teach(_3940)
Asserting>> neg poor(_8DB0) := phded(_8DB0)
-------------------------------------------------

>A person who teaches is poor.
[a,person,who,teaches,is,poor,.]
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[]
IF:
  [_60D4]
  gender(_60D4,n,sg,person)
  teach(_60D4)
THEN:defeasible
  []
  poor(_60D4)
-------------------------------------------------
Asserting>> poor(_60D4) := teach(_60D4)
-------------------------------------------------
```

```
>A legal_aid is law_degreed.
[a,legal_aid,is,law_degreed,.]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_58D0]
  gender(_58D0,_5B00,sg,legal_aid)
  legal_aid(_58D0)
THEN:defeasible
  []
  law_degreed(_58D0)
------------------------------------------------
Asserting>> law_degreed(_58D0) := legal_aid(_58D0)
------------------------------------------------

>A person who is law_degreed is not poor.
[a,person,who,is,law_degreed,is,not,poor,.]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
NOT:
  []
  IF:
    [_71EC]
    gender(_71EC,n,sg,person)
    law_degreed(_71EC)
  THEN:defeasible
    []
    poor(_71EC)
------------------------------------------------
Asserting>> neg poor(_71EC) := law_degreed(_71EC)
------------------------------------------------

>A legal_aid is poor.
[a,legal_aid,is,poor,.]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_4D64]
  gender(_4D64,_4F94,sg,legal_aid)
  legal_aid(_4D64)
THEN:defeasible
  []
  poor(_4D64)
------------------------------------------------
Asserting>> poor(_4D64) := legal_aid(_4D64)
------------------------------------------------

>Fred is a legal_aid who teaches.
[fred,is,a,legal_aid,who,teaches,.]
```

```
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0]]
gender([0],m,sg,fred)
named([0],Fred)
gender([0],m,sg,legal_aid)
legal_aid([0])
teach([0])
-------------------------------------------------
Asserting>> named([0],Fred)
Asserting>> legal_aid([0])
Asserting>> teach([0])
-------------------------------------------------

>Is Fred poor?
[is,fred,poor,?]
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0]]
QUERY:
  []
  poor([0])
-------------------------------------------------
query is:
poor([0])
Testing query.
   using @@ Goal
can draw no conclusion
-------------------------------------------------

Preemption is now enabled.

>Is Fred poor?
[is,fred,poor,?]
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[0],[0]]
QUERY:
  []
  poor([0])
-------------------------------------------------
query is:
poor([0])
Testing query.
   using @ Goal
 The result is : yes
-------------------------------------------------
```

Strict and Defeasible Deduction Tests

```
>Birds fly. Penguins are birds. Penguins do not fly.
[birds,fly,.,penguins,are,birds,.,penguins,do,not,fly,.]
```

```
Rule type:defeasible
Rule type:strict
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
IF:
  [_0970]
  gender(_0970,_09E4,pl,birds)
  bird(_0970)
THEN:defeasible
  []
  fly(_0970)
IF:
  [_0A38]
  gender(_0A38,_0ADC,pl,penguins)
  penguin(_0A38)
THEN:strict
  []
  IF:
    [_0A38]
    gender(_0A38,_0B7C,pl,birds)
    bird(_0A38)
  THEN:strict
    []
IF:
  [_0C14]
  gender(_0C14,_0CAC,pl,penguins)
  penguin(_0C14)
THEN:defeasible
  []
  NOT:
    []
    fly(_0C14)
------------------------------------------------
Asserting>> fly(_0970) := bird(_0970)
Asserting>> bird(_0A38) :- penguin(_0A38)
Asserting>> neg fly(_0C14) := penguin(_0C14)
------------------------------------------------

>Is every penguin a bird?
[is,every,penguin,a,bird,?]
Rule type:strict
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
QUERY:
  []
  IF:
    [_31B4]
    gender(_31B4,_5934,sg,penguin)
    penguin(_31B4)
  THEN:strict
    [_31B4]
    gender(_31B4,_6E0C,sg,bird)
    bird(_31B4)
```

```
------------------------------------------------
query is:
bird(_31B4) :- penguin(_31B4)
Testing query.
Deductive (strict) analysis
 The result is : yes
------------------------------------------------

>Does every bird fly?
[does,every,bird,fly,?]
Rule type:strict
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
QUERY:
  []
  IF:
    [_2C04]
    gender(_2C04,_534C,sg,bird)
    bird(_2C04)
  THEN:strict
    []
    fly(_2C04)
------------------------------------------------
query is:
fly(_2C04) :- bird(_2C04)
Testing query.
Deductive (strict) analysis
    can draw no conclusions
------------------------------------------------

>Do birds fly?
[do,birds,fly,?]
Rule type:defeasible
------------------------------------------------
Cleaned-up (simplified) DRS:
------------------------------------------------
[]
QUERY:
  []
  IF:
    [_2188]
    gender(_2188,_4114,pl,birds)
    bird(_2188)
  THEN:defeasible
    []
    fly(_2188)
------------------------------------------------
query is:
fly(_2188) := bird(_2188)
Testing query.
Deductive (defeasible) analysis
   Yes.
------------------------------------------------

>Chilly is a penguin.
[chilly,is,a,penguin,.]
```

```
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[7],[7]]
gender([7],_4A68,sg,chilly)
named([7],Chilly)
gender([7],_4A68,sg,penguin)
penguin([7])
-------------------------------------------------
Asserting>> named([7],Chilly)
Asserting>> penguin([7])
-------------------------------------------------

>Does every bird fly?
[does,every,bird,fly,?]
Rule type:strict
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[7],[7]]
QUERY:
  []
  IF:
    [_33A4]
    gender(_33A4,_5AEC,sg,bird)
    bird(_33A4)
  THEN:strict
    []
    fly(_33A4)
-------------------------------------------------
query is:
fly(_33A4) :- bird(_33A4)
Testing query.
Deductive (strict) analysis
   counter example found: bird([7])
 The result is : no
-------------------------------------------------

>Do birds fly?
[do,birds,fly,?]
Rule type:defeasible
-------------------------------------------------
Cleaned-up (simplified) DRS:
-------------------------------------------------
[[7],[7]]
QUERY:
  []
  IF:
    [_2244]
    gender(_2244,_41D0,pl,birds)
    bird(_2244)
  THEN:defeasible
    []
    fly(_2244)
-------------------------------------------------
query is:
fly(_2244) := bird(_2244)
```

```
Testing query.
Deductive (defeasible) analysis
   Yes, but exceptions exist.
-------------------------------------------------
```

6.3 DefDRT User Notes

DefDRT is written in Arity Prolog 6.0. The phrase structure rules within DefDRT are written in GULP 2.0 (Covington 1989), an extension to Prolog for handling unification–based grammar. Defeasible reasoning in DefDRT is provided by d-Prolog, an extension to Arity Prolog which is an implementation of Defeasible Logic (Nute 1993). To run DefDRT the user will require a copy of Arity Prolog 6.0 and the following files:

The DefDRT program (DEFDRT.GLP)
GULP for Arity Prolog (GULP.DEF)
d-Prolog for Arity Prolog (DPROLOG.DEF)
The driver program (DEFRUN.ARI)

To load DefDRT, start up Arity Prolog by typing

```
API
```

At the Arity Prolog prompt, type

```
[defrun].
```

This will consult the file DEFRUN.ARI which will load the other three files mentioned above. The user should only use the versions of GULP and d-Prolog mentioned above since they have been modified slightly to prevent any duplication of the predicates "member" and "list." In GULP.DEF "member" and "list" have been changed to "g_member" and "g_list." In DPROLOG.DEF, they are "d_member" and "d_list."
Once DEFRUN has finished, the user will type

```
go.
```

to start DefDRT. The user will then be prompted for an input file name and an output file name. The input file is a file previously generated by DefDRT. By specifying an input file, the user can continue a discourse from a previous session. The user will either type in the DOS file name of a previous discourse session or just hit ENTER if no input file is to be used.

The output file is used to store the d-prolog facts and rules generated from the discourse, as well as the final form of the DRS. The user must enter an output file name. If the output file name and the input file name are the same, then upon ending the DefDRT program the file will contain the previous discourse information with the current discourse results appended to it. Note that at the beginning of the discourse session the output file is cleared if it already exists or is created if it doesn't exist. The last line of the output file will contain the final state of the DRS, and the current value of the Skolem number. The output file should not be edited since the final DRS is written out to the file as a single, very long line. An editor is likely to either break it into separate lines in the file or truncate it.

DefDRT expects each sentence in a discourse to end with either a period or a question mark. The user is free to use upper and lower case letters in a sentence, since DefDRT converts every character in a sentence to lower case. If a word in a sentence is unknown to DefDRT, the user will be prompted to enter a replacement word.

When the user hits the ENTER key, the sentences typed by the user are processed into facts and rules, and queries are proved. When this process completes, DefDRT waits for the user to type in additional sentences which will be treated as being part of the same discourse. A discourse session ends when the user types the command:

```
quit.
```

This forces DefDRT to write out the DRS to the output file, clear its working memory of any clauses generated from the discourse, and returns control back to the Arity environment. At this point the user may start a new discourse session (or continue a previous one) by typing

```
go.
```

or may exit Arity Prolog completely.

DefDRT recognizes the following command words when they are entered at the DefDRT prompt:

`QUIT` Writes the DRS to the output file, clears the working memory of clauses generated from the discourse, closes any open files, and returns the system to the Arity Prolog prompt.

`PREEMPT` Toggles preemption on and off. Preemption is an option provided by d-Prolog for allowing defeated rules to be preempted.

`HELP` Displays a brief description of these commands.

The types of sentences and questions that DefDRT recognizes are listed below in phrase structure form.

S–> [no], NP1, V [every], NP2.
S–> NP, VP.
S–> NP, [may/might], VP.
S–> NP, [Does/Do], [not], VP.
S–> NP, [may/might], [not] VP.
S–> NP, [is/are] Adj.
S–> NP, [may/might], [be], Adj.
S–> NP, [is/are], [not], Adj.
S–> NP, [may/might], [not], [be], Adj.
S–> NP, [is/are], NP.
S–> NP, [may/might], [be], NP.
S–> NP, [is/are], [not], NP.
S–> NP, [may/might], [not], [be], NP.
S–> [if], S, [then], S.
S–> [presumably], S.
Q –> [do/does] NP, VP.
Q –> [is/are] NP, Adj.
Q –> [is/are] NP, NP.

DefDRT Lexicon

PROPER NOUNS
Nixon, Fred, Carl, Hans, Pedro, Tom, Bob, Bill, Chiquita, Maria, Mary, Nancy, Gertrude, Chilly, Opus, Tweety.
COMMON NOUNS
bandersnatch, boojum, farmer, knight, lady, knave, student, adult, speaker, job, legal_aid, Republican, pacifist, Quaker, thing, person, people.
NATURAL KIND COMMON NOUNS
man, boy, woman, girl, donkey, penguin, platypus, bird, animal, fish.
ADJECTIVE
German_speaking, Pa_dutch_speaking, poor, phded, law_degreed, big, furry, feathered, duckbilled, hungry, green, rich, genetically_altered, old, happy, poor, black, yellow, sick.
PLACE ADJECTIVES
French, German, American, Pennsylvanian, Connecticut.
TRANSITIVE VERBS
see, love, like, own, have, beat, feed, eat, admire, fight, insult.
INTRANSITIVE VERBS
bark, eat, bray, fly, swim, chirp, work, teach.
DETERMINERS
a, an, every, all, no.
PRONOUNS
he, him, she, her, it.
MISCELLANEOUS
is, are, be, do, does, not, both, and, who, whom, which, that, if, then, may, might, presumably.

Note that the plural forms of the appropriate nouns and verbs are also allowed.