# Using Multiple Adaptively-Weighted Strategies for the Resolution of Demonstratives

Ralf D. Brown

10 May 1993
CMU-CS-93-148

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy*

**Thesis Committee:**
Jaime G. Carbonell, Co-chair
Masaru Tomita, Co-chair
Sergei Nirenburg
Deborah Dahl, Unisys

This research was partially supported by the Carnegie Mellon University School of Computer Science.

**Keywords:** Natural language processing, reference resolution, demonstratives

Carnegie Mellon

School of Computer Science

DOCTORAL THESIS
in the field of
Computer Science

# *Using Multiple Adaptively-Weighted Strategies for the Resolution of Demonstratives*

RALF BROWN

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

ACCEPTED:

[illegible] THESIS COMMITTEE CHAIR — 4/30/93 DATE

[illegible] DEPARTMENT HEAD — 5/14/93 DATE

APPROVED:

[illegible] DEAN — 5/14/93 DATE

# Table of Contents

# List of Figures

# Using Multiple Adaptively-Weighted Strategies for the Resolution of Demonstratives

## Abstract

The resolution of demonstratives is an interesting topic which nonetheless has received relatively little attention. As with pronominal anaphora, there is currently no comprehensive theory of demonstrative references, although there are numerous partial or microtheories. This dissertation describes a multistrategy approach to resolving demonstrative pronouns and noun phrases, using constraint, preference, and recovery strategies implementing various microtheories and heuristics. The multistrategy approach, which has proven itself for resolving pronominal anaphora, allows easy integration of new microtheories. This provides incremental improvements in performance, permitting a decreasing reliance on user intervention as the system is able to resolve a larger percentage of demonstratives. A major weakness of the multistrategy approach, the necessity to carefully choose the weights of the strategies, is overcome by automatically optimizing the weights as demonstratives are resolved. Such an adaptively-weighted multistrategy approach has been implemented in the modular and extensible MASTER-D system which is expected to be applicable to other linguistic phenomena in addition to the resolution of demonstratives.

# Chapter 1
# Introduction

One of the most common constructions in natural language is the referring expression. Referring expressions have numerous roles in language; for example, they may shorten an utterance by using a brief phrase in place of a full description, they may add more information to a previous description of the same discourse object, or they may simply allow monotony to be avoided by using varying references to the same discourse object.

An example text in which all three of the above uses are exhibited in the following article from the May 15, 1982 *The Times*:

> PRIEST IS CHARGED WITH POPE ATTACK (Lisbon, May 14)
>
> *A Spanish priest* was charged here today with attempting to murder the Pope.
>
> *Juan Fernandez Krohn, aged 32*, was arrested after *a man armed with a bayonet* approached the Pope while he was saying prayers at Fatima on Wednesday night.
>
> According to the police, *Fernandez* told the investigators today *he* trained for the past six months for the assault. *He* was alleged to have claimed the Pope 'looked furious' on hearing *the priest's* criticism of his handling of the church's affairs.
>
> If found guilty, *the Spaniard* faces a prison sentence of 15-20 years.[1]

In this text, the priest is referenced a total of eight times in seven different ways, of which five references are abbreviated versions of prior references. Each of the first three references adds more information to the reader's model of the priest, until the reader has a mental model of "Juan Fernandez Krohn, age 32, a Spanish priest who attacked the Pope with a bayonet." Finally, the article avoids monotonous repetition by using a different reference each time but one that the priest is mentioned.

As shown to some extent in the example text above, there are numerous types of referring expressions, such as pronouns, definite noun phrases, and demonstratives; similarly, many different classes of entities, such as objects, actions, and time intervals, may be referenced. Demonstratives are more speaker-oriented than most referring expressions; they act much like a pointer from the speaker to the referenced entity, and are frequently accompanied by pointing gestures when spoken. Kenneth Goodman suggested an alternate title for this dissertation, *Now Here This*, which shows three of the dimensions in which demonstratives can point: time, location, and relationship to the speaker (such as distance).

[1]This is a simplified version provided by Sergei Nirenburg.

The core of this dissertation is the MASTER-D (Multiple Adaptively-weighted STratEgies for Resolving Demonstratives) system. This system uses multiple independent but cooperating resolution strategies, each implementing a microtheory or heuristic, whose weights in voting for candidate referents are adjusted based on past performance. Error recovery is provided in the form of additional strategies which attempt to produce further candidate referents or select one of multiple candidates which are nearly equally preferred.

Several of the components of the MASTER-D system would themselves be worthy dissertation topics, and are thus present in a simplified form. Among these are world modeling and discourse analysis. The simple world modeler and discourse analyzer used by MASTER-D will frequently indicate that they are unable to give any information about a particular item. As will be shown, one of the strengths of the approach used here is that it can continue to operate successfully (though somewhat less accurately) even when given incomplete information.

Although this dissertation makes no claims that MASTER-D models human cognitive processes, there is mounting evidence that the human brain uses multiple parallel mechanisms in various activities. To dates, there is evidence that the human brain uses multiple parallel mechanisms in language and vision processing and a suggestion that multiple mechanisms are involved in motor functions.

The case of an elderly woman suffering from brain damage to both temporal lobes and small patches elsewhere suggests that the brain uses separate knowledge systems for visual and verbal knowledge [4, 24]. This woman exhibited an inability to name animals when presented with their pictures or characteristic sounds, yet had no difficulty naming other living things or inanimate objects given those cues. Testing revealed that she lacked verbal knowledge of animals' physical attributes, yet could visually match heads and bodies and identify when an animal was portrayed with the wrong color (though she was unable to name the correct color). Hart and Gordon conjecture that the verbal system contains subdomains of knowledge, allowing the woman to identify other attributes of an animal from its name even though she was unable to identify visual physical attributes of the same animal when given its name.

The visual cortex similarly uses multiple parallel processing methods whose results are later combined [44]. Each scene seen by the eyes is replicated to a dozen different areas of the visual cortex which extract movement, color, outlines, etc.

The suggestion of multiple methods in motor function comes from a patient suffering induced Parkinsonism from a contaminated dose of synthetic heroin [34]. The patient suffers the rigidity of Parkinson's disease and will "freeze" while walking. If asked to perform a slightly different motion such as raising a foot or stepping over a small obstacle, he is then able to continue walking for a period of time. One hypothesized explanation is that there are two or more parallel "circuits", each of which stops functioning when overworked; performing the alternate action allows a different circuit to take over the action of walking.

This evidence that multiple parallel mechanisms are used by the human brain lends some plausibility to the use of multiple strategies in a computer program for processing referring expressions in natural language. Plausibility, of course, does not prove that this is in fact how the humain brain processes referring expressions even though other mental processes do appear to use this method.

The remainder of this dissertation is organized into five main topic sections as follows:

1. Background: Demonstratives, The Multi-Strategy Approach, Related Work
2. MASTER-D: Architecture, Resolution Strategies, Adaptive Weighting, Implementation
3. Results: Testing Protocol, Test Results, Contributions
4. Future Work: Possible Enhancements
5. Appendices: Annotated Trace, Selected Source Code, Knowledge Base, Sample Text

# Chapter 2
# Background

As mentioned in the introduction, one of the most common constructions in natural language is the referring expression, of which demonstratives are one type. More so than most referring expressions, demonstratives are speaker-oriented, acting much like a pointer from the speaker to the referenced entity.

A restatement of the above is that demonstrative pronouns and noun phrases are examples of deictic referring expressions[2]. As with all deixis, the exact meaning of demonstratives depends on the context in which they are uttered--among other factors, where, when, and by whom (the *Origo* or zero point of Bühler's "deictic field of language" [8, 9]). This differentiates them from anaphoric pronouns and definite noun phrases, whose meaning depends on the entities in the prior discourse and only occasionally and indirectly on the context of the utterance. Webber [45] also states that the referent$_m$ (see Section 4.7, page 32) of an anaphoric NP may be something associated with the current focus, while that of a deictic NP must already be present in the focus.

In addition to their deictic properties, demonstratives also share properties with anaphoric pronouns and definite noun phrases. It is these similarities which will allow some of the strategies used by multi-strategy anaphora resolvers to apply unchanged to demonstratives. Demonstrative noun phrases, in particular, are very similar to definite noun phrases; in Alshawi's terminology, they are definite NPs with deictic determiners [1]. Thus, demonstrative NPs carry constraints such as number and gender (the latter supplied by the head of the noun phrase in English) which must be satisfied by their referents, just as is the case for definite noun phrases.

Although there are numerous other demonstrative words, discussion of demonstratives other than *this*, *that*, *these*, and *those* will largely be omitted. By narrowing the focus to those four words, the following text will be less cluttered and hopefully clearer. Further, *here*, *there*, *now*, and *then* have quite restricted uses by comparison, and would not add to the discussion.

[2]Not all linguists would agree with this statement, but it is not of any particular importance to this dissertation whether or not demonstratives are in fact deixis.

## 2.1. Attributes of a Demonstrative

Demonstratives contain a number of inherent attribute values which can be used in narrowing the space of possible referents. The prototypical attributes of English demonstratives are conceptual distance and number; demonstrative NPs may also gain attributes from the head of the NP. Many other languages have additional attributes such as visibility (e.g. Chinook [2]).

The distance attribute of English demonstratives may be either "near" or "far", depending on spatial and temporal aspects, how closely the referenced items are associated with the speaker, or even how focused the entity is in the immediately preceding discourse. Demonstratives with "near" proximity include *this*, *these*, *here*, and *now*; demonstratives with "far" proximity include *that*, *those*, *there*, and *then*. Other languages make finer distinctions such as "near"/"further"/"far" (e.g. Spanish, Navaho or Thai [10]) or "nearer to listener" (e.g. Navaho [41]).

Similarly, the number attribute may be either "singular" or "plural"; for example, *this* and *that* are singular while *these* and *those* are plural. In the case of locatives like "here", the number is usually implicitly singular since demonstrative locatives generally refer to a single location.

MASTER-D uses both the proximity and number attributes when resolving a demonstrative. It does not attempt to deal with the additional or finer distinctions used by other languages at this time, although that capability could be added quite easily.

## 2.2. Classes of Demonstratives

As might be expected from the different kinds of entities to which a reference may be made and the different ways in which they may be referenced, there are a number of classes of demonstrative references. The following taxonomy will be used as a reference in the remainder of this document. It is not meant to be a definitive classification of demonstratives; it need not be, since MASTER-D uses the class of a demonstrative as just another piece of information to be applied in the resolution process, lack of which does not make resolution impossible.

1. **Object Reference**
   A specific object or set of objects previously mentioned may be the referent of a demonstrative, as in *those people* or *this building*.

2. **Property Reference**
   Sometimes, a demonstrative refers only to a property of an object, rather than the object itself, as in
   ```
   The pale green pants look nice.
   I want a shirt in that color.
   ```

3. **Event Reference**
   Some demonstratives refer to one or a set of situations or events, either current at the time of the utterance or mentioned in the prior discourse. For example, *this meeting*, *that emergency*, or *those events*.

4. **Action Reference**
   An action may be referenced by a demonstrative, as in

```
John wanted to surprise Mary with a present yesterday.
Did he do that?
```

5. **Temporal Reference**
Demonstrative NPs may reference some point or interval of time, as in *this evening* or *that day*. Phrases of the form *this X* generally refer to a time in relation to the time in the utterance, while *that X* generally refers to a time related to one mentioned in the prior discourse, as in

```
The day Joe met Sally was not a good one for Joe.
He had cut himself shaving that morning, and things had gone
downhill from there.
```

6. **Locative Reference**
A demonstrative may refer to a particular point in space, such as

```
We camped out on that hill for several days.
```

or to a particular direction, as in

```
He went that way.
```

However, such a locational phrase need not have a referent in the prior discourse, as it may in fact be an extralinguistic reference (see below).

7. **Discourse Reference**
A demonstrative may also refer to the prior discourse itself, rather than some entity within it. The demonstrative's referent may be part or all of a discourse segment, or some aspect thereof other than the surface form, such as the speech act [45]. One of Webber's examples [45] is

```
Hey, they've promoted Fred to second vice president.
That's a lie.
```

8. **Lexical Reference**
This type of reference is distinguished from other discourse references because the referent is the actual utterance rather than the meaning of the utterance, as in

```
Decisions and feelings and positions taken today will have, I
   believe, major impact on the world.
That's a strange phrase for a business person to ever
   utter. [27]
```

9. **Logical Reference**
Many statements in natural language express logical predications or logical propositions, and these logical statements are distinct entities which may be referenced in ensuing utterances. For example,

```
Mary told me that Sue can speak French.
I wasn't aware of that.
```

10. **Non-literal Reference**
As is the case with definite NPs and anaphoric pronouns, demonstratives need not literally refer to a discourse entity. The demonstrative may instead be used in an idiomatic, metonymic, or metaphoric fashion. For example, in the idiom

```
That's the way the cookie crumbles.
```

*that* references an undesirable event or situation in the prior discourse, rather than a description of the manner in which cookies desintegrate.

11. **Quantifier/Selector**
A demonstrative is occasionally given a postmodifying restrictive relative clause. Constructions such as *those X who Y* or *those X that Y* for X null or a noun phrase and Y a relative clause create a new discourse entity of the set of all X which match the constraints of Y. The noun phrase X need not refer to any entities in prior discourse. Similarly for the construction *that which Y*. For example,

```
It's appropriate that those who understand this wondrous stuff take
on that challenge, but let us remember that there is an issue
of public trust here as well. [27]
```

or

> A third area of value derivation is *that which enables the user to achieve the desired result*.

The X may also be a prepositional phrase, as in

> [Moderator announces that only two more audience questions will be taken]
> My apologies to *those of you that were further back in the line*. [27]

12. **Placeholder**

A singular demonstrative noun phrase may specify an indeterminate member of the set evoked by the head of the NP, as in

> ...and we took a piece out of *this competitor*'s system, we took a piece out of *this competitor*'s system....

Similarly, a plural demonstrative NP may specify an indeterminate subset of the set evoked by the head noun, as in

> And frankly, back then, that was the way that you built software.
> ....
> Let's start where *these guys* are and let's see what we could innovate beyond that. [27]

13. **Comparison**

*This* and *that* may also be used in an adjectival phrase comparing two discourse entities, i.e.

> Is it really *that* big?

For the special case of a lacking referent in the prior discourse, *that* degenerates into meaning *very*, as in

> It's not *that* important.

14. **Epithets**

Demonstrative NPs may convey the speaker's opinion of or attitude toward the referent, as in *that bastard*. Epithets are also commonly combined with non-literal reference, as in

> I called New York.
> *Those idiots* lost your application again.

15. **Cataphoric or Polaroid[3] Demonstrative**

Sometimes a demonstrative pronoun or NP precedes the full description of the entity being referred to by the pronoun or NP. In this case, the demonstrative serves merely to introduce a new discourse entity, whose "shape" will become clear as the discourse continues and provides more information about that entity. A complication arises because the entity introduced by the demonstrative may be referred to by pronouns even before it has been fully-developed by the ensuing discourse. For instance [27],

> ... at the same time (and I think that *this* is an equivalent good) it must enable successful companies to flourish.

or

> *This* is my position, not Lotus Development Corporation (I don't think we would differ much).
> Basic tools, we believe, should be open.
> ...

Note that cataphoric demonstratives are not a totally separate class, but rather a variation on the other classes in the taxonomy. Thus, a cataphoric demonstrative will also belong to one of the other classes indicating the type of entity referenced by the demonstrative.

The indefinite singular *this* is always cataphoric. This usage does not refer to any existing

[3]This term is used in analogy with Hirst's "Polaroid Words" [26] to emphasize the gradual development of the meaning of such references after their actual occurrence. No implication of association with the Polaroid Corporation, which owns the trademark *Polaroid*, is intended.

entities, but serves merely to introduce a new entity as would the indefinite articles *a* and *and*. [19]

```
So, this man walks into the bar and ...
```

16. **Extralinguistic Reference**
In spoken discourse, a demonstrative may accompany a gesture such as pointing. The pointed-at object or location then becomes the referent of the object or location implied by the demonstrative. For example,

```
Put the box there (points at corner of room).
```

Similarly to cataphoic references, extralinguistic references are not a totally separate class, as the extralinguistic entity being referenced may be a member of any one of a majority of the other classes.

In addition to its role as a demonstrative, the word *that* is also used as a relativizer. In fact, the use of "that" in relative clauses is considerably more common than use as a demonstrative (see the next section). The analyzer is expected to distinguish between use in a relative clause and use as a demonstrative.

Similarly, the word *this* also has an alternate use as a singular indefinite, which was mentioned above. The parser is expected to make this distinction as well.

As previously stated, the above taxonomy is not meant to be definitive. There may be better ways to classify demonstratives or additional cases which are not covered in this taxonomy. For example, there may be a demonstrative analogue to the "pronouns of laziness" discussed by Hintikka and Kulas in the context of their Game-Theoretical Semantics [25]. Pronouns of laziness act as though the referent were lexically substituted for the pronoun and then re-interpreted in the resulting sentence, such as in so-called "paycheque sentences" like

```
The man who gave his paycheque to his wife was wiser than the man who
gave it to his mistress. [3]
```

The phrase *do that* appears to be such a construct in certain contexts.

## 2.3. Relative Frequencies of Use

The previous section stated that *that* appears more frequently in its nondemonstrative role as a relativizer than as a demonstrative. To illustrate, the count of occurrences of *this*, *that*, *these*, and *those* in demonstrative pronouns, demonstrative NPs, and nondemonstrative uses will be presented for three different texts.

The first example consists of the first ten chapters (pages 1 through 64) of the science fiction novel *Rama II* [15]. This text consists primarily of narrative which introduces the characters and sets the background (including "history" through the year 2199). The breakdown of the different uses is as follows:

| | THIS | THAT | THESE | THOSE |
|---|---|---|---|---|
| demonstrative pronoun | 16 | 16 | 1 | 4 |
| demonstrative NP | 59 | 43 | 24 | 11 |
| non-demonstrative | --- | 232 | --- | --- |

As might be expected from the large proportion of narrative text, demonstrative NPs are more common than demonstrative pronouns. In contrast, dialogues would be expected to skew more towards pronouns, particularly the pronominal use of "that" (especially "that is" referring to prior discourse). The transcript of a panel discussion on intellectual property rights and the following audience participation [27] shows this to be the case:

| | THIS | THAT | THESE | THOSE |
|---|---|---|---|---|
| demonstrative pronoun | 64 | 180 | 7 | 11 |
| demonstrative NP | 78 | 81 | 26 | 12 |
| non-demonstrative | — | 536 | — | — |

Not surprisingly, there are more occurrences in all nine categories, even though the transcript contains approximately the same amount of text as the novel excerpt (about 20,000 words). For this text, 57.1% of the demonstratives are demonstrative pronouns, more than twice the 21.3% in the previous sample. In addition, the increased proportion of demonstratives is reflected in a higher percentage of demonstrative uses of *that*--32.7% compared to 20.3% in the novel.

The third text examined here is the sequel to *Rama II*. The first thirteen chapters of *The Garden of Rama* [14] are presented in the form of a personal journal, and are mostly first-person narrative text with a few quotations. One would thus expect the distribution among the nine categories to be similar to that found in *Rama II*. The first seven chapters, pages 3 through 60, confirm the expectation with the following distribution:

| | THIS | THAT | THESE | THOSE |
|---|---|---|---|---|
| demonstrative pronoun | 6 | 14 | 2 | 0 |
| demonstrative NP | 50 | 27 | 7 | 15 |
| non-demonstrative | — | 263 | — | — |

This is indeed similar to the distribution found in *Rama II*, though the demonstratives are skewed somewhat more toward NPs (81.8% of demonstrative uses compared to 78.7%), and a smaller proportion of the occurences of *that* are demonstrative (13.5% compared to 20.3%). The distribution contrasts significantly with the panel discussion's distribution, where only 42.9% of the demonstratives were NPs and demonstrative uses accounted for fully 32.7% of the occurrences of *that*.

The first 200 of the 459 demonstratives used during the panel discussion can be classified as in the first column of Table 2-1 using the taxonomy presented earlier. Similarly, the first 200 demonstratives in the excert from *Rama II* can be classified as in the second column of Table 2-1. The totals are greater than 200 because some occurrences fall into two classes, such as the extralinguistic or cataphoric category in addition to some other category. Further, one common idiomatic construction which is always in multiple classes is "that is, X", where "that is" is a cue phrase indicating a restatement or clarification of something as "X"; the "that" can be considered either an idiom or a reference of the appropriate type to both "X" and the entity being restated.

As can be seen from the table, object references dominate. The differing genres of text and differing subject matter affect the next most-frequent classes of references; since the panel discussion is mostly dialogue, it contains many discourse references, while the novel excerpt presents a history and thus contains many temporal and event references. The discourse references in *Rama II* are due to the dialogue present in later parts of the excerpt.

It is worth noting that a number of the demonstratives in the transcript (though none of the first 200) are

| | Panel | Rama II |
|---|---|---|
| Object reference | 91 | 115 |
| Property reference | 2 | 0 |
| Discourse reference | 45 | 12 |
| Temporal reference | 8 | 23 |
| Event reference | 5 | 29 |
| Action reference | 9 | 4 |
| Locative reference | 5 | 3 |
| Lexical reference | 6 | 1 |
| Logical reference | 14 | 3 |
| Non-literal reference | 3 | 2 |
| Quantifier/selector | 6 | 4 |
| Placeholder | 4 | 0 |
| Comparison | 1 | 5 |
| Epithet | 0 | 2 |
| | | |
| Cataphoric demonstrative | 8 | 1 |
| Extralinguistic reference | 8 | 6 |

**Figure 2-1:** Relative Frequences of Types Among 200 Demonstratives

in fact relativizing uses of *that*. They occur in sentences which have been malformed such that the *that* appears as a demonstrative, even though a human can infer that it was intended to be a relativizer. The primary reasons for such malformation are interruption and mid-sentence change of the utterance.

An important advantage of the multistrategy approach is that it is not necessary to classify each demonstrative in order to resolve it. If the demonstrative can be classified, the knowledge of its type can help the resolution process, but lack of that knowledge will not make resolution impossible. At worst, the system's resolution accuracy will be degraded because the strategies can not be applied in a targeted manner.

# Chapter 3
# The Multi-Strategy Approach

As the introduction stated, there is currently no complete linguistic theory of demonstratives; thus no single monolithic resolution strategy is available. Instead, there are numerous partial or microtheories and various heuristics which may be applied in varying situations. By using multiple of these knowledge sources, we may increase coverage and thus improve the accuracy with which the referents of demonstratives are determined. As additional knowledge is gained, more strategies may be implemented, producing better coverage and accuracy, and reducing the reliance on user intervention.

In addition to increased coverage, the multi-strategy approach offers easy parallelization since the resolution process is already broken into largely or entirely independent portions (the individual strategies). Depending on the number of available processors, one can distribute the evaluations of candidates in various ways:

1. one processor per group of strategies
2. one processor per strategy, evaluating all candidates with that strategy
3. one processor per candidate, applying all strategies to that candidate
4. one processor for each candidate/strategy pair

A separate processor then combines the results produced by the others.

Many different methods are possible for orchestrating the application of multiple strategies:

1. Apply all strategies to each candidate in turn.
2. Apply each strategy in turn to all candidates.
3. Group strategies and apply each group in turn to all candidates.
4. Individually apply strategies to all candidates until one candidate is sufficiently preferred.
5. Conditionally apply strategies based on the number of candidates remaining.
6. Allow one or more strategies to determine which of the remaining strategies will be applied to each candidate.
7. Allow one or more strategies to direct the order in which the remaining strategies are applied to the candidates.
8. Allow one or more strategies to direct the order in which candidates are processed by the remaining strategies.

The method of application used here is to divide the strategies (knowledge sources) into four distinct groups: constraints, preferences, and two groups of recovery actions. All strategies in the constraint group are applied; if no candidates remain, one group of recovery actions is invoked to expand the candidate space; if multiple candidates remain, all strategies in the preference group are applied. Should no single candidate attain a distinctly greater preference than the others, the second group of recovery actions is applied in an attempt to narrow the set of most-preferred candidates to a single one. (see Chapter 5 and Section 6.1 for more details)

In addition to the various manners in which strategies may be applied, there are also several possible methods for combining the results of individual strategies:

1. If two or more strategies are applicable to the candidate and give conflicting results, use only the result from the highest-weighted strategy.
2. If more than two strategies are applicable to the candidate and give conflicting results, use the result returned by the greatest number of strategies.
3. Give each strategy a number of votes proportional to its weight. The candidate receiving the greatest number of votes is considered the correct candidate.

MASTER-D employs a slightly modified voting scheme. Each strategy returns a number in the range -1.0 to 1.0 (inclusive) or, if it is a constraint strategy, a special indicator that the candidate should be ruled out[4]. The value 1.0 indicates maximum preference for the candidate, -1.0 indicates the knowledge source's certainty that the candidate is *not* the correct one, and 0.0 indicates no preference either way (possibly because the knowledge source does not apply to the candidate). Intermediate values indicate varying certainty that the candidate is or is not the correct one. The central application mechanism multiplies each returned score by the appropriate weight and sums the weighted scores.

The main disadvantage of the multi-strategy approach is its sensitivity to the weighting of the individual strategies. Since not all strategies have equal predictive power or accuracy, the strategies should not be weighted equally. Instead, the more accurate strategies should receive a greater weight than the less accurate strategies. However, finding the optimal set of weights for strategies quickly becomes too tedious to perform manually, even assuming that a fixed set of weights is optimal for all texts. The solution to this problem is to automatically reweight the strategies based on their prior performance; this adaptive weighting is discussed in Chapter 7.

The use of multiple knowledge sources has become quite popular over the past decade. Systems using such an approach include MARS, SPAR, Lucy, Capture, and Susan McRoy's word sense discriminator, all of which will be discussed in some detail in the next chapter.

[4]Constraint strategies may thus also indicate a preference among those candidates which are not ruled out.

## 3.1. Example Resolutions

A few examples will greatly aid in understanding the application of the multiple strategies in MASTER-D. For these examples, the following strategies will be used:

1. Gender C. astraint
   The gender of the candidate referent must match the gender of the demonstrative. Prefer those candidates for which the gender is identical to the demonstrative and not just compatible.
2. Number Constraint
   The number (singular/plural/etc) of the candidate must match the number of the demonstrative. Prefer those candidates for which the number is identical to the demonstrative and not just compatible.
3. Reference Type
   The type of entity represented by the candidate antecedent must match the type of entity expected in place of the demonstrative; i.e. a demonstrative filling in for an event cannot have as referent a candidate representing an action.
4. Recency Preference
   Prefer candidates which are closer to the demonstrative over those which are further away.
5. Ask the User
   When multiple candidates are equally (or nearly equally) preferred, present them to the user and allow her to select the desired referent.

### 3.1.1. Short Example

The first example is from page 59 of *Rama II* [15]. It will be resolved using strategy weights of 10 each for the preference portion of the gender and number constraints, and 10 for the recency preference strategy.

> "You wrote in your letter," the Holy Father said, without referring to any notes, "that there were some theological issues that you would like to discuss with me. I assume **these** are in some way related to your mission."

We begin by listing the candidates. In the context in which the above passage occurs, the two sentences are the first two spoken by the character, so there are no candidates from previous sentences. The candidates for resolving the *these* in the second sentence are therefore only those found in the first sentence,

> You wrote in your letter that there were some theological issues that you would like to discuss with me.

which are

Action: write a letter
Action: discuss
Object: letter
Object: theological issues
Object: speaker of utterance (me)
Object: hearer of utterance (you)

For this example, the demonstrative carries no gender information, so no candidates are eliminated by the Gender Constraint strategy. The Number Constraint strategy is applied next, and rules out "letter", "me", and "you". It also prefers "theological issues", the only plural candidate. This leaves the following candidates:

|  |  |
|---|---|
| Action: write a letter | score: 0 |
| Action: discuss | score: 0 |
| Object: theological issues | score: 10 |

The third strategy to be applied is Reference Type. Since the demonstrative is the subject of the verb *are*, it should be coreferential with a noun, i.e. its reference type for the purposes of this example is Object. Note that a system with sophistication beyond the scope of this example might create candidates such as "desired discussion" which are related to non-Object candidates but are themselves of type Object. This would be required to handle cases such as

John and Mary repeatedly discussed the design of their software. **These discussions** eventually led to the specification for the Lettuce A-B-C spreadsheet.

For the demonstrative under discussion here, the Reference Type strategy rules out all but one candidate:

|  |  |
|---|---|
| Object: theological issues | score: 10 |

Since only a single candidate remains, it is not necessary to apply any other resolution strategies. The remaining candidate is the desired referent, which leads to a reading of the second sentence as

I assume the theological issues are in some way related to your mission.

## 3.1.2. Long Example

The second example will be a somewhat simplified version of the test case whose annotated trace is shown in Appendix A:

"Copyright on a user interface would mean that each typewriter manufacturer would be forced to arrange the keys differently."

By my description, the letters of the alphabet would be tools. Of course *those* should be open.

This passage will also be resolved using strategy weights of 10 each for the preference portion of the gender and number constraints, and 10 for the recency preference strategy. The score produced by the recency preference will decrease by two points for every sentence between the sentences containing the demonstrative and the candidate antecedent.

Again, we begin by listing the candidates. In the interest of brevity, some candidates which do not affect the outcome will be omitted. Additionally, such candidates as arise from phenomena not addressed by the strategies used for this example will be omitted. The initial list of candidates is

Object: copyright
Object: user interface
Object: typewriter manufacturer
Object: keys
Object: letters of the alphabet
Object: tools
Action: force
Action: arrange
Discourse: the letters of the alphabet would be tools
Lexical: "Copyright on a user interface..."
Lexical: "the letters of the alphabet would be tools."

In this case, the Gender Constraint is unable to rule out any candidates, since all candidates have indeterminate gender. The Number Constraint strategy, on the other hand, is able to rule out several singular candidates, leaving

| | |
|---|---|
| Object: keys | score: 10 |
| Object: letters of the alphabet | score: 10 |
| Object: tools | score: 10 |
| Action: force | score: 0 |
| Action: arrange | score: 0 |
| Discourse: the letters of the alphabet would be tools | score: 0 |
| Lexical: "Copyright on a user interface..." | score: 0 |
| Lexical: "the letters of the alphabet would be tools." | score: 0 |

Note that Number Constraint has not ruled out the Action, Discourse, or Lexical candidates because these do not bear number information and are thus treated as being of indeterminate number. Indeterminate number is compatible with any other number information, though it cannot make an exact match which would be preferred with a positive score.

Applying the Reference Type strategy results in the invalidation of all non-Object candidates, leaving

| | |
|---|---|
| Object: keys | score: 10 |
| Object: letters of the alphabet | score: 10 |
| Object: tools | score: 10 |

Since multiple candidates remain after applying the constraint strategies, the preference strategy is applied next. It gives its full weight of ten points to the candidates from the immediately preceding sen<sup></sup>tence, and a lesser weight of eight points to the candidate from the next-to-last prior sentence. After the recence strategy completes, the scores are as follows:

| | |
|---|---|
| Object: keys | score: 18 |
| Object: letters of the alphabet | score: 20 |
| Object: tools | score: 20 |

Since there is a tie for most-preferred candidate, the resolution process proceeds to apply the recovery strategy, Ask the User, presenting the following two candidates:

Object: letters of the alphabet
Object: tools

Once the user selects "tools" as the desired referent, the system is able to return the result that

Of course *those* should be open.

should be read as

Of course tools should be open.

# Chapter 4
# Related Work

There has been surprisingly little work done on determining the referents of demonstrative pronouns and NPs, a[illegible]sed to anaphoric pronouns and NPs. Demonstratives are typically given only passing mention or a footnote in works on anaphora, and are rarely the main topic of a paper. For this reason, much of the related work discussed in this chapter relates to pronominal anaphora rather than demonstratives.

The MASTER-D system has been influenced by two prior pronominal anaphora resolution systems and an earlier user interface. The strongest influences are from my own previous work on the MARS system [6, 11]; the Lucy anaphora resolver [32, 38] and the KBMT-89 Augmentor interface [6, 7] have provided additional ideas. Further, Carter's SPAR system [12, 13] parallels MARS, Lucy, and MASTER-D in a number of aspects. These systems and others are discussed in more detail below. The final sections of this chapter will discuss additional systems and how they make use of the multi-strategy approach and treat or do not treat demonstratives.

## 4.1. MARS

The MARS (Multiple Anaphora Resolution Strategies) system resolves both anaphoric pronouns and definite noun phrases using multiple independent constraint and preference strategies rather than a single monolithic strategy.

After each sentence has been analyzed, the resulting parse is scanned for pronouns and definite noun phrases. Each of the pronouns and definite NPs is passed through the anaphora resolver in turn. After the entire sentence has been thus processed, the pronouns and definite NPs are added to the pool of candidate referents for subsequent sentences. Candidates which have been in the pool for more than a pre-set number of sentences are removed to prevent unbounded growth in the set of candidates to be checked for each anaphor.

The MARS anaphora resolver applies a set of constraint strategies to each candidate referent, then applies preference strategies to those candidates which pass the constraints. Each strategy has an individual weight, and may vote with less than its full weight for less-preferred candidates. The votes of all strategies are summed, and the candidates with a score within a specified threshold of the highest score are considered to be the desired (though possibly still ambiguous) referents.

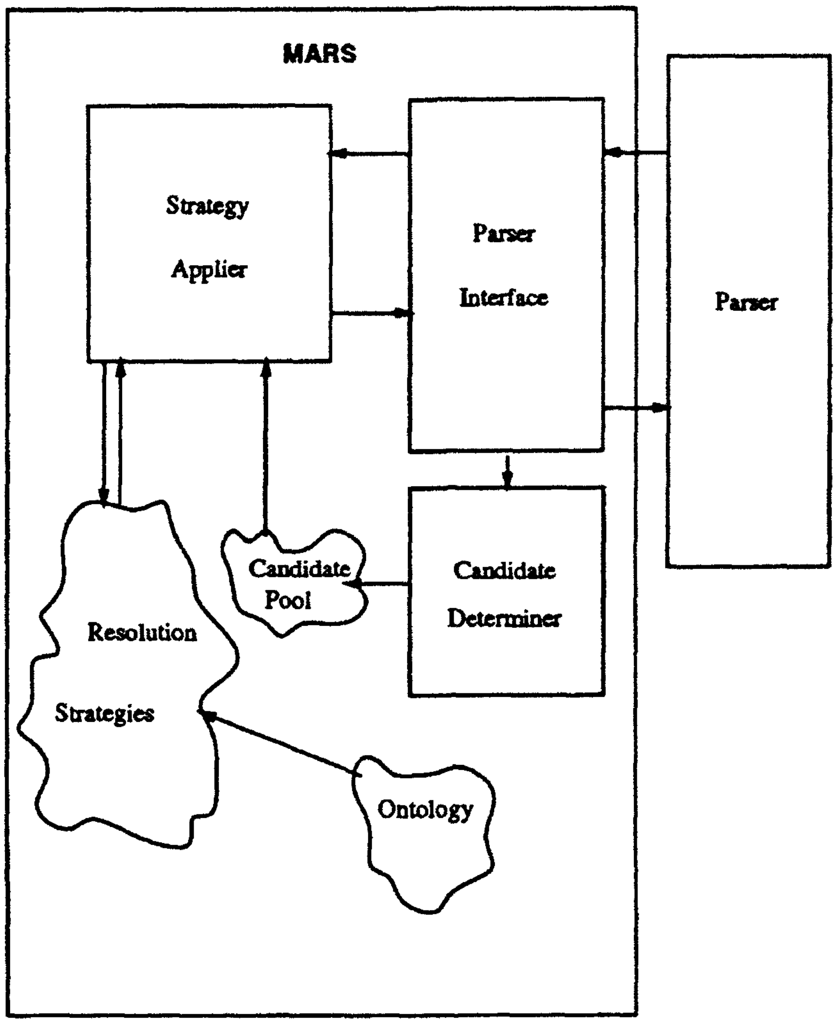


**Figure 4-1:** MARS Architecture

MARS implements three constraint strategies and four preference strategies (one of which applies to NPs only), each with an individual weight and rate of decay. The base weights and linear decay as the distance between anaphor and candidate referent increases are manually selected before use; their values remain fixed for the duration of a program run.

The constraint strategies are

1. **Local Anaphora Constraints**
   Whenever an anaphor carries a constraint such as gender, number, case, etc., the referent must satisfy those constraints. MARS implements gender, number, and animacy; if not explicitly present in the parse, these values may be inherited from the knowledge base.
2. **Case-role Semantic Constraints**
   The constraints imposed on an anaphor filling a particular case role must also be satisfied by the anaphor's referent.
3. **Precondition/Postcondition Constraints**
   Any action which changes the state of the world carries with it certain preconditions and

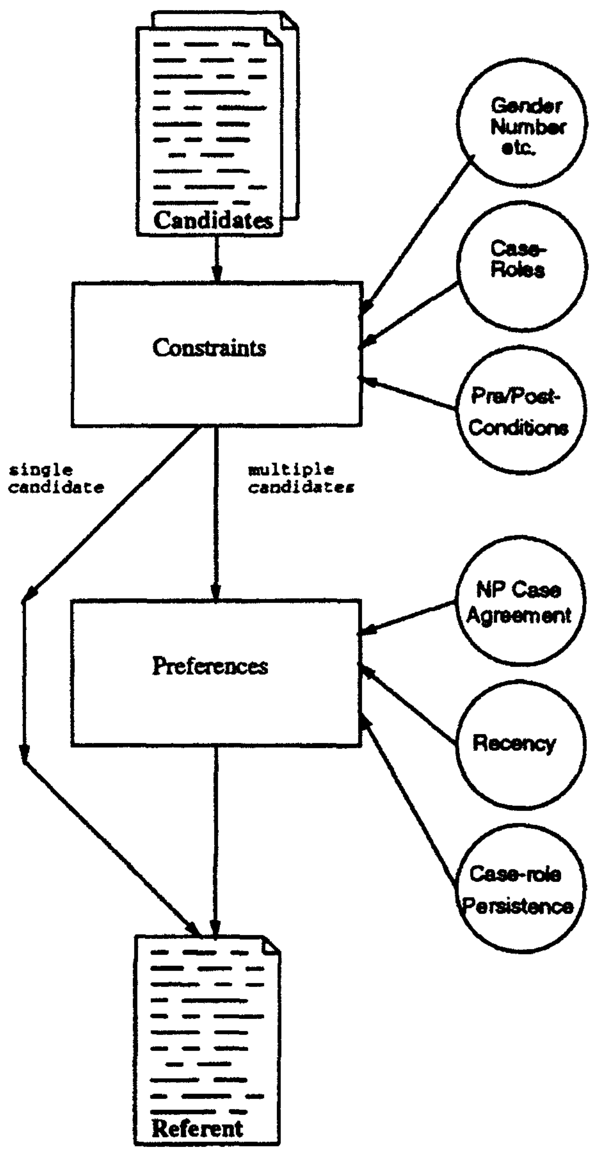


**Figure 4-2:** Resolution Process in MARS

postconditions. The postcondition of the action which a candidate referent is a part of may not violate the preconditions of the action containing the anaphor; such a violation indicates that the anaphor and candidate cannot be coreferential.

The preference strategies used by MARS are:

1. **Noun Phrase Case Agreement**
The antecedent of a definite noun phrase tends to contain many of the same fillers for its various attributes and cases. MARS models this effect by giving the highest preference to the antecedent which has the greatest commonality with the definite NP; this is implemented by scanning all the common slots and counting the number of fillers which match.

2. **Case-role Persistence**
Due to a form of "linguistic inertia", the referent of an anaphor is more likely to have filled a

case role equivalent to the one filled by the anaphor. MARS maintains separate weights for strong case-role persistence (where the underlying actions of the sentences containing the anaphor and candidate are the same) and weak case-role persistence (where the actions differ). Case-role Persistence differs from Noun Phrase Case Agreement because the matching criterion is the slot filled by the anaphor rather than the slot fillers within the anaphor; as a result, the former prefers candidates which were used similarly while the latter prefers candidates whose meanings are similar to the anaphor's.

3. **Syntactic Topicalization**
   Topicalized structures are more likely to contain antecedents for an anaphor than the remainder of the utterance containing such a structure.

4. **Intersentential Recency**
   In general, most anaphors are lexically close to their referents; thus, the likelihood that a candidate is the proper referent for an anaphor decreases as the distance between them increases.

MARS can be extended simply by adding another strategy to the list of strategies to apply. Assuming that the new strategy is independent of the existing strategies, no other changes are needed.

In addition to disambiguating pronouns and anaphoric definite noun phrases, the MARS system has a limited ability to disambiguate between multiple readings of a sentence. MARS can be placed in a special "batch" mode in which it delays output of a processed sentence until a pre-specified number of the following sentences have been processed. If a resolved anaphor is incompatible with any of the multiple readings of a sentence, those readings are eliminated before the sentence is output, thus providing additional disambiguation.

## 4.2. Lucy anaphora resolver

Another multi-strategy anaphora resolver was implemented as part of the Lucy natural-language analysis system at MCC[5] in Austin, Texas. The Lucy anaphora resolver takes an opposed approach to selecting candidate referents. Instead of collecting all possible candidates and removing unwanted ones, it allows strategies to generate new candidates for testing, stopping when a candidate's score exceeds a prespecified threshold.

Each strategy in the Lucy anaphora resolver consists of four components: a Modeler, a Constraint Poster, a Proposer, and an Evaluator [38]. The Modeler maintains a local model of the discourse, while the Constraint Poster determines constraints on the interactions between parts of a sentence. The Proposer outputs a list of candidate structures and their associated scores for a specified anaphoric reference; some strategies never propose candidates. Finally, the Evaluator returns a score (or an indication that it has no opinion) given an anaphoric reference and a previously proposed candidate antecedent.

The score for a strategy/anaphor/candidate combination consists of a pair (score,confidence). The score is a value in the range -5 to +5, while the confidence is a value in the range 0 (no confidence) to 1 (perfect

[5]Microelectronics and Computer Technology Corporation

certainty). The central strategy handler forms a composite score from the individual scores returned by each strategy.

The knowledge sources which the Lucy anaphora resolver implements are:

1. **Recency**[6]
   Candidates which are near the anaphor are to be preferred over more distant candidates. This corresponds to MARS's Intersentential Recency preference.
2. **Syntactic Number Agreement**
   The number (singular, plural, indefinite, etc) of the candidate referent must be compatible with the number of the anaphor.
3. **Syntactic Gender Agreement**
   Similarly, the genders of the candidate and the anaphor as derived from syntax must be compatible.
4. **Knowledge Base Gender Agreement**
   For those candidates or anaphors which do not have explicit syntactic gender information, default information from the knowledge base is used instead. Again, the genders of the candidate and the anaphor must be compatible.
5. **Animacy**
   Since an animate anaphor should not have an inanimate referent or vice versa, the fifth strategy states that the animacy of the candidate should be compatible with the animacy of the anaphor. This and the previous three strategies correspond to the Local Anaphora Constraint in MARS.
6. **Type**
   Personal pronouns should not refer to events or propositions.
7. **I-Within-I**
   As stated by GB Theory, an anaphor should not coindex with a structure that contains the anaphor itself.
8. **Disjoint Reference**
   Both reflexive and nonreflexive pronouns have additional (though differing) structure-based restrictions on coreference. For example, a nonreflexive pronoun should not corefer with any of the other arguments of the verb.
9. **Semantic Type Consistency**
   The semantic interpretation of the rest of the sentence may impose additional constraints on the type of the anaphor.
10. **Knowledge Base Consistency**
    Similar to Semantic Type Consistency, this strategy does not simply apply static type constraints. Instead, semantic knowledge of context dependent phenomena is used to reason about the candidates.
11. **Cataphora**
    Certain syntactic constructions permit a pronoun to precede the full noun phrase to which it refers; this strategy proposes such noun phrases as candidates when it encounters one of those constructions.
12. **Logical Accessibility**
    Using discourse representation theory [28, 29], it is possible to rule out candidates because of their inaccessibility due to embedding within quantifiers, negations, or other such logical structures.

[6]The names of these strategies have been taken from [32].

13. **Global Focus**
When the anaphor is *it*, this strategy proposes candidates which are salient throughout a discourse, i.e. in global focus.

14. **Local Focus**
Those candidates which are locally in focus or centered should be preferred to those which are not in focus.

15. **Backward Center**
Using the local focus stack of the previous strategy, one object may be predicted as the next concept to occur in the text, and should be preferred.

16. **Duplicates**
This heuristic suggests that all occurrences of *they* within a given sentence corefer; similarly for *it*. In constrast, *one* is assumed not to corefer with any other instance of *one* within the same sentence.

The Lucy anaphora resolver does not treat demonstratives and had not yet been tested on them as of early 1991 [31]. Its support for demonstratives is limited to a demonstration of a possible method for implementing Webber's ideas on discourse deixis (see Section 4.7).

## 4.3. SPAR

David Carter's Shallow Processing Anaphor Resolver [12, 13] was designed to determine the referents of anaphors in simple short stories. It is based on the premise that natural language can be processed using limited world knowledge by maximizing the use of linguistic cues and the redundancy of language. In this respect, it parallels MARS, MASTER-D, and the Lucy anaphora resolver.

Like MARS, SPAR also uses an existing parser and an existing generator, converting the output of the parser to a format better-suited to its internal processing.

Anaphora resolution is embedded in the overall processing strategy, which also selects the proper reading of the input from among those proposed by the parser. The readings are scored based on a computed intrinsic semantic density, the ease with which anaphors can be resolved, and how many potential anaphors are unresolvable. "Ease of resolution" is approximated by subtracting points from the score for each rejected anaphor resolution rule. The reading with the highest score is the one which is accepted as the correct reading.

SPAR applies a number of strategies to determine the proper binding for an anaphor. Its main strategies are extended versions of Wilks' Preference Semantics [46, 47] and Sidner's anaphor resolution rules [42, 43]. When these approaches fail to produce a unique referent, SPAR also uses a Common Sense Inference mechanism; if even that is insufficient, it applies a number of "collective" preference criteria.

For full noun phrases, SPAR's rules closely follow Sidner's FDNP (full definite noun phrase) rules; the major differences lie in the nonimplementation of computed specification and in inferred specification. Inferred specification is performed by relaxing the information constraint to avoid common sense inference whenever possible. This relaxation is often equivalent to making the assumptions necessary for determining coreference.

SPAR's rules for resolving definite pronouns are nearly identical to Sidner's Pronoun Interpretation rules with the major exception of intrasentential candidate antecedents. SPAR also addresses some ambiguity in Sidner's rules when dealing with plural definite pronouns by stipulating that the members of a group must share a head primitive.

The Common Sense Inference mechanism in SPAR applies inference rules in an attempt to build inference chains. This process consists of the two basic operations of recursively making inferences from story assertions involving either undetermined anaphors or candidate antecedents, and attempting to find pairs of inferences forming chains linking assertions of the two different types. The successful chains thus bind one or more anaphors to candidate antecedents.

Carter gives the outline of the overall processing strategy used by SPAR (in which anaphora resolution is an integral part) shown in Figure 4-3. The tie-breaking heuristics in the last step operate on the original dependency structures. The first assumes that the dictionary is ordered such that the most common word senses appear first, such that it can select the dependency structure which selects the more common sense. The second examines the attachment point of a case, preferring verb phrase attachment over noun phrase attachment and low attachment over high attachment.

## 4.4. KBMT-89 Augmentor

The Augmentor component of the KBMT-89 machine translation system provides not only the user interface for the entire translation system, but also an interactive disambiguator when automatic disambiguation of the input is unsuccessful.

The overall architecture of the Augmentor component, and its relationship to other components of the KBMT-89 system, is shown in Figure 4-4. The user interface accepts a sentence from the user and passes it to the source language parser. The resultant parse is then passed through the format converter to produce an internal representation which is processed by the automatic augmentation and disambiguation portions of the Augmentor. Any remaining ambiguities are then processed by the interactive disambiguator with the user's help. The final result of disambiguation is then verified by the user (which involves generating a paraphrase in the source language) and then passed to the target language generator.

The interactive disambiguator first determines the points on which the remaining candidate readings differ. It then displays up to four menus at a time (see Figure 4-5) to allow the user to decide which of the cases at a point of difference is the correct one. Since four menus are displayed and the user may complete an arbitrary one of those menus, he may choose the menu which appears to be clearest or most likely to provide good disambiguation. In case the user is unsure exactly which item in a menu is the appropriate one, the augmentor permits multiple items to be selected.

After the user completes one of the menus, the disambiguator removes all candidate readings which do not contain one of the selected items at the point corresponding to the menu. The list of difference points is updated, and a new set of menus is displayed if there are still multiple candidate readings remaining.

```
Clear the Text Meaning and focus registers.

For each sentence of a text:

>  Derive one or more current fragments from each of the
>  sentence's dependency structures.

>  For each reading represented by a current fragment:

>>    Set the reading's initial score to its semantic density

>>    Apply the anaphora resolution rules (using only
>>    linguistic knowledge) to each potential anaphor,
>>    subtracting points from the score for rejected
>>    suggestions from the resolution rules and potential
>>    anaphors for which there were no acceptable suggested
>>    candidates.

>  Apply configurational constraints.

>  Accept only the highest-scoring reading(s).

>  If multiple readings, or a single reading with multiple
>  candidate anaphor referents, remain, then for each reading:

>>    Invoke the Common Sense Inference mechanism and adjust
>>    the score based on the number of inference chains
>>    completed.

>>    Invoke "collective" preference criteria if necessary.

>  Accept the highest-scoring reading, or apply weak heuristics
>  to select the desired reading in case of a score.  Merge this
>  reading into the context on the basis of the anaphora
>  resolution results.
```

Figure 4-3: Outline of SPAR Processing Strategy

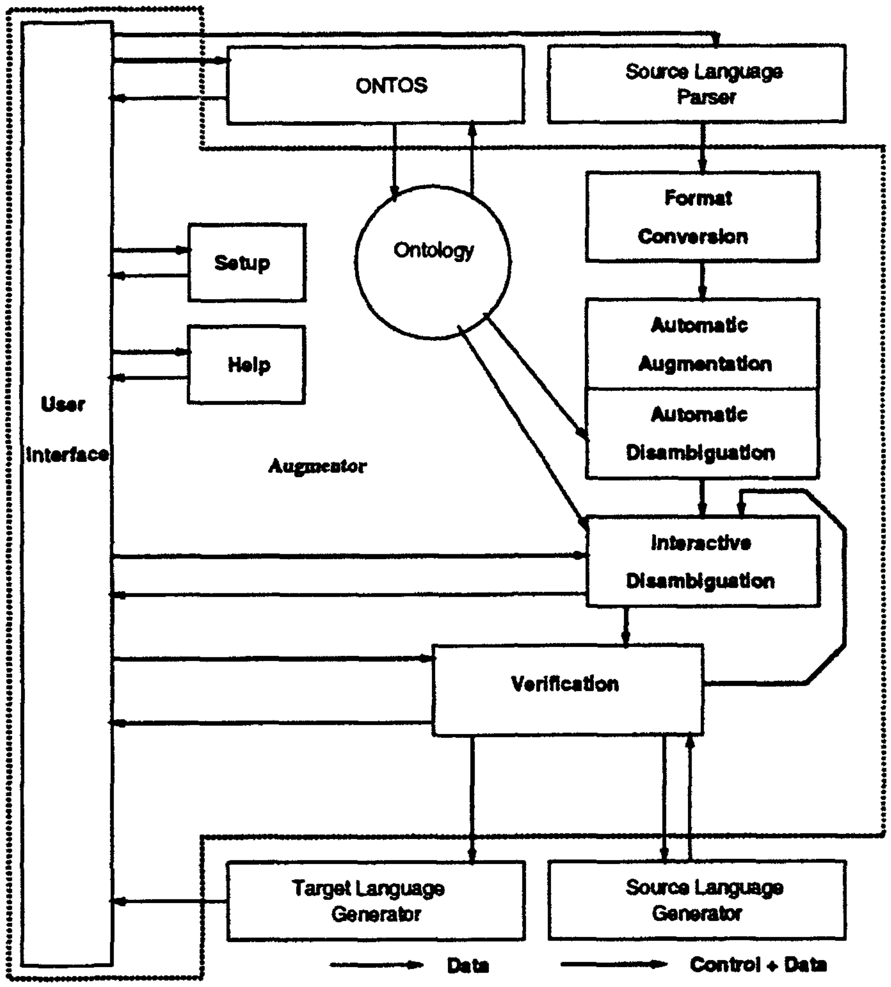
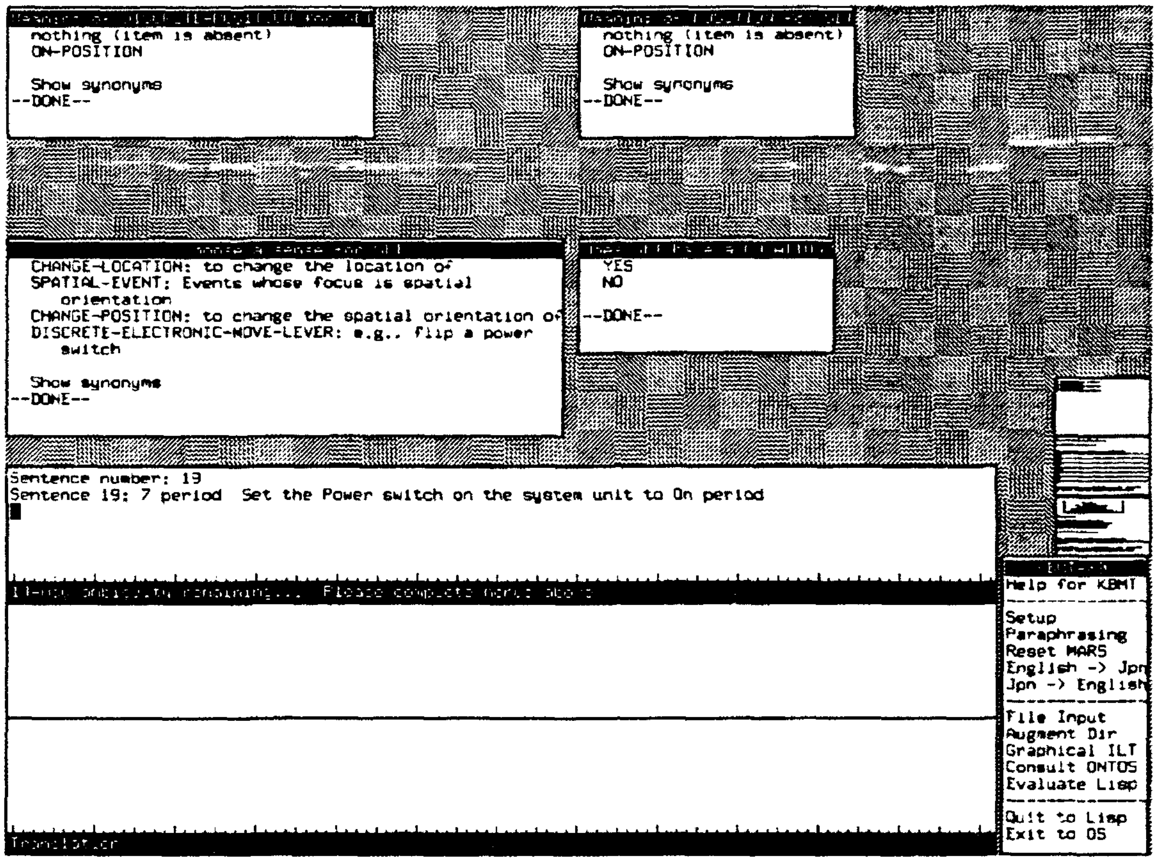


**Figure 4-4:** KBMT-89 Augmentor Architecture

## 4.5. Capture

A system which does treat demonstratives, if in a limited manner, is Alshawi's Capture [1]. Capture is a natural-language interface for data entry into a database; it understands various types of reference to the preceding text. Demonstrative handling in Capture is limited to "textual references", which form a subset of what was referred to in Section 2.2 as object references. Entities in the scope of a deixis context factor have their activation levels increased if they were mentioned sufficiently recently and frequently; implicitly referenced objects are not handled by the version of Capture described in [1].

Capture uses a marker-passing approach to activating the memory entities it generates during processing. Like MASTER-D, Capture gives its various context factors (such as recency, emphasis, deixis, subject area, and association) differing weights, and applies a decay mechanism to reduce weight with increasing distance between reference and candidate antecedent. Unlike MASTER-D, however, the weights are



```
  nothing (item is absent)
  ON-POSITION

  Show synonyms
--DONE--
```

```
  nothing (item is absent)
  ON-POSITION

  Show synonyms
--DONE--
```

```
  CHANGE-LOCATION: to change the location of
  SPATIAL-EVENT: Events whose focus is spatial
     orientation
  CHANGE-POSITION: to change the spatial orientation of
  DISCRETE-ELECTRONIC-MOVE-LEVER: e.g., flip a power
     switch

  Show synonyms
--DONE--
```

```
  YES
  NO

--DONE--
```

```
Sentence number: 19
Sentence 19: 7 period  Set the Power switch on the system unit to On period
```

**Figure 4-5:** KBMT-89 Augmentor Screen

chosen by the implementor and adjusted by hand using trial and error. For the majority of the context factors, the integer weights of all existing instances are halved each time a new instance of certain context factors is created, and any instances with zero weights are then removed.

The context factors used by Capture are

1. **Sentential Recency** The activation factors for all entities referenced explicitly or implicitly through an anaphor, or created while interpreting a sentence, are increased.
2. **Paragraph Recency** Similar to sentential recency, but includes all entities in the scope of sentential recency for the sentences already processed in the current paragraph, and any memory entities created during interpretation of the paragraph.
3. **Be-clause Emphasis** The agents of certain be-clauses become foregrounded.
4. **Syntactic-Topic Emphasis** The topics of passive sentences are foregrounded and thus have their activation levels increased.
5. **Processing History** Memory entities which are involved in memory processing have their context activation factors increased.
6. **Textual Deixis** All entities which have been mentioned recently and frequently enough in the preceding text have their activations increased.
7. **Subject Area** This context factor increases the activation of memory entities which are related to a particular subject.

8. **Associations** Memory entities closely associated with those currently in focus become activated.
9. **Task-Specific Factors**

## 4.6. Susan McRoy's Word Sense Discriminator

Susan McRoy's unnamed word sense discriminator [33] is an example of a system using the multi-strategy approach for something other than reference resolution. It uses numerous preference cues to tag each word with the correct sense from among those listed in its lexicon; for example, whether *reach* means 'achieve' or 'extend an arm'. Rather than using a fixed precedence order in applying the rules, the discriminator dynamically weighs all of the preference information on the basis of specificity.

The discriminator uses the following knowledge sources:
1. a lexicon of 10,000 unique roots making coarse sense distinctions
2. a concept hierarchy
3. collocational patterns
4. concept clusters (categorial, functional, and situational)

Word sense discrimination is distributed through much of the system rather than located entirely in a single distinct module. It begins with morphological analysis and lexical retrieval, which assigns general senses. The result of morphological analysis is then passed through a preprocessing stage which tags words with parts of speech, identifies collocations (for example, verb+particle pairs and compound noun phrases) based on a set of simple patterns, and identifies preferences indicated by activated concept clusters. The final stage occurs in the semantic interpreter, which identifies and combines preferences contributed by a variety of knowledge sources, including those
- associated with the head, argument, modifier, or their components
- arising from syntactic cues
- associated with any two of head, argument, modifier, and relation; for example, expected qualifiers for a particular head
- arising from reference resolution (not used as of [33])

It is this use of multiple knowledge sources that is of interest in this dissertation.

The semantic interpreter combines preference cues by summing the strength returned by each; the word sense with the highest total is selected. Each preference cue is assigned a strength between -10 and +10 based on its type and how well its underlying expectations are satisfied. Many of these strengths must be chosen experimentally; those with an inherent extent (such as conceptual categories) are assigned a strength based on the specificity of the particular cue--the fewer elements are subsumed, the higher the score the cue is given.

The strength value for a concept is based on the number of items in the knowledge base subsumed by the concept, ranging from a low value (0 or 1) for very general concepts to +10 for the most specific concepts subsuming only a single entity. Complex concepts created with AND, OR, and NOT, are assigned a strength based on the number of knowledge base entities subsumed by both, either, or neither subconcept. Concept clusters are scored on the basis of the specificity of the cluster (or the most specific cluster if a word sense belongs to multiple clusters).

For a preference cue of strength S, the usual combining action is to add either +S or -S to the total score for the appropriate word sense, depending on whether or not the cue is satisfied. An exception is made for some concepts, however. Very general concepts provide strong information because entities which are not members of the concept are usually grossly inappropriate; thus, the returned score is based on the specificity of the concept's complement for a score of approximately S-10 (which is more negative than -S in these cases). Very specific concepts, on the other hand, frequently overspecify a constraint. If the preference can be satisfied after a slight relaxation, the system bases its score on the specificity of the concept's complement; otherwise, it returns -S on failure. For concepts which are intermediate in specificity, the system always uses scores of +S or -S.

## 4.7. Webber's Referent$_m$

As alluded to earlier, Bonnie Webber has done work on references to discourse segments, including demonstrative references [45]. She proposes an evolving discourse model and references within that model, termed references$_m$ to distinguish them from expressions with extralinguistic referents in "the outside world". Each discourse segment has an associated referent$_m$ which has at least three properties: the segment's speech act, interpretation, and surface form. The first two of these fall into the category of Discourse Reference described in Section 2.2, while the last corresponds to Lexical Reference.

Only the referents$_m$ which are on the *right frontier* of the discourse segment heirarchy are available for use by a demonstrative. The right frontier consists of those discourse segments which are still open and the last segment to have been closed. While a referent$_m$ does not necessarily create a discourse entity, once it has been referenced it gains discourse entityhood and may subsequently be referred to by an anaphoric pronoun.

## 4.8. PUNDIT

The PUNDIT system [16, 36] treats demonstratives the same as anaphoric pronouns and NPs, although doing so is inadequate [17]. Demonstratives, by their deictic nature, generally assume (and require) more mutual knowledge than noun phrases with a definite article. Thus, one might say

```
    The highway near my office is very noisy.
```

but not

```
    That highway near my office is very noisy.
```

unless the listener is expected to know which highway the speaker has in mind. PUNDIT would allow both sentences equally because of its lack of special treatment of demonstratives. [18]

# Chapter 5
# The MASTER-D Architecture

MASTER-D consists of seven basic components: the user interface, the parser/generator interfaces, the candidate determiner, the strategy applier, the resolution strategies proper, the reweighter, and the world modeler (Figure 5-1). The user interface communicates with the user to receive input for parsing or for help in resolving a demonstrative, and to display the results of processing. The parser and generator interfaces communicate with the parser and generator in order to receive a parse to process and to create the target text after processing. The candidate determiner extracts candidate referents from each sentence processed for use in resolving demonstratives in later sentences. The strategy applier directs the application of the resolution strategies and tracks the scores for each candidate referent. The resolution strategies analyze the demonstrative/candidate pairs in various ways and return a score indicating the probability of the candidate being the actual referent of the demonstrative. The reweighter attempts to optimize the strategy weights to maximize the accuracy of the resolutions. Finally, the world modeler tracks relations between entities in the world and in the discourse.

The remaining parts of this chapter will cover the user interface, parser/generator interfaces, the candidate determiner, the strategy applier, and the world modeler in more detail. The resolution strategies are detailed in the next chapter, while the reweighter is discussed in Chapter 7.

## 5.1. User Interface

MASTER-D provides a character-based menu-driven interface to the user. This user interface provides all the necessary menus and tools for specifying the initial input, answering any queries posed by the system in selecting the proper referent for a demonstrative, etc. It has been written so that any of a number of underlying interfaces such as a character-mode terminal or an *X*-windows display may be used and selected among at run-time. At this time, only the character-mode interface is implemented; the X-windows interface consists solely of stubs.

The major screen elements provided by the user interface are output windows, single- and multiple-selection menus, popup prompts and dialogs, and a status line. Figure 5-3 shows the screen display during a test run. MASTER-D has just popped up a verification dialog window, is showing three debugging windows (strategy weights, current agenda item, and top candidates), and has set the status line to indicate that it is updating various of its data structures following the resolution.

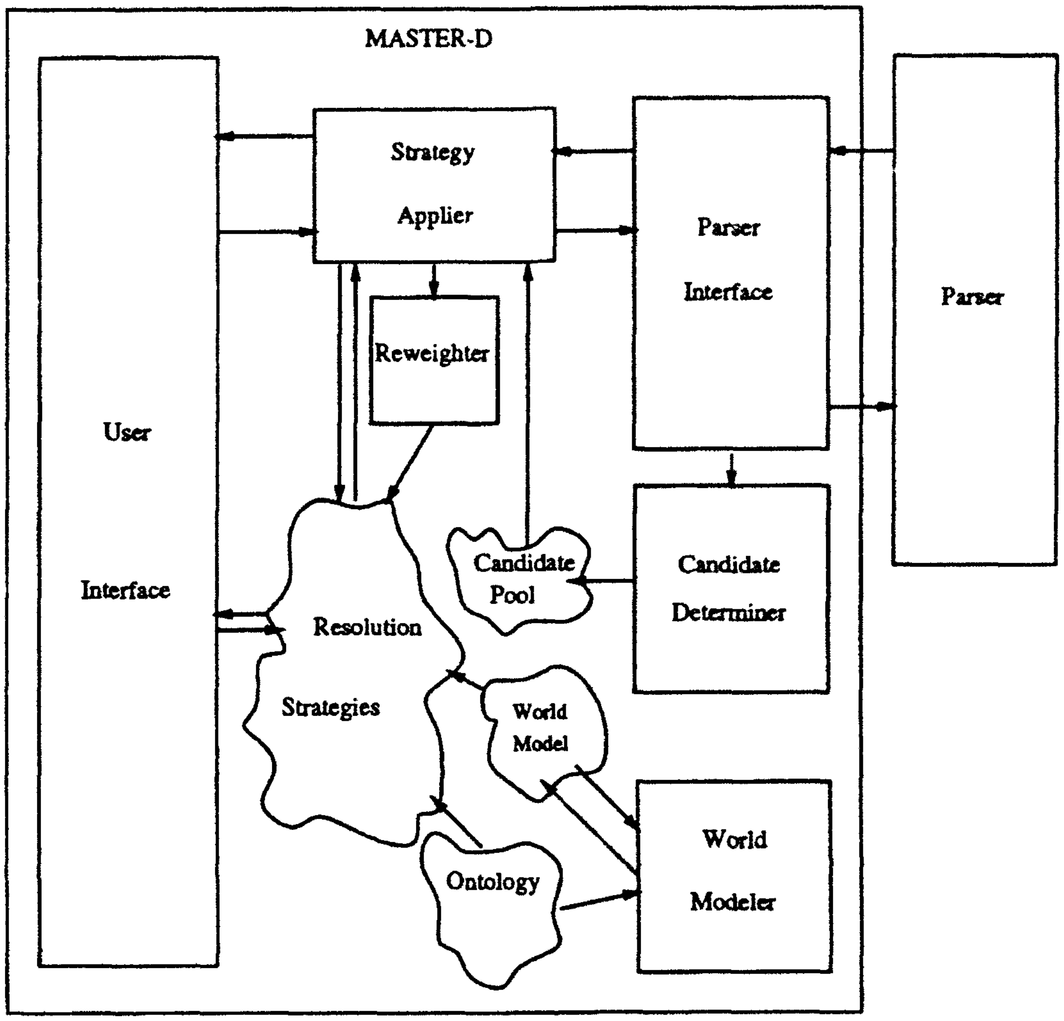


**Figure 5-1:** MASTER-D Architecture

Multiple-selection menus such as shown in Figure 5-4 are used when more than one answer may be correct or it is acceptable for the user to specify multiple items if she is unsure of which item is the appropriate one. For the example menu shown, MASTER-D will accept all candidates of any of the types selected by the user, and reject as invalid candidates of the types not selected.

## 5.2. Parser and Generator Interfaces

The MASTER-D system uses a prepared parse as input and generates a processed parse as output. Therefore, it relies on a separate parser and generator to provide full text-to-text processing, and thus must interface with them. The parser and generator interfaces are built on top of Unix sockets, with a server loop running in the parser and generator to which MASTER-D directs requests (see Figure 5-5).

The socket interface code packages up the requests and sends them to the appropriate socket for the

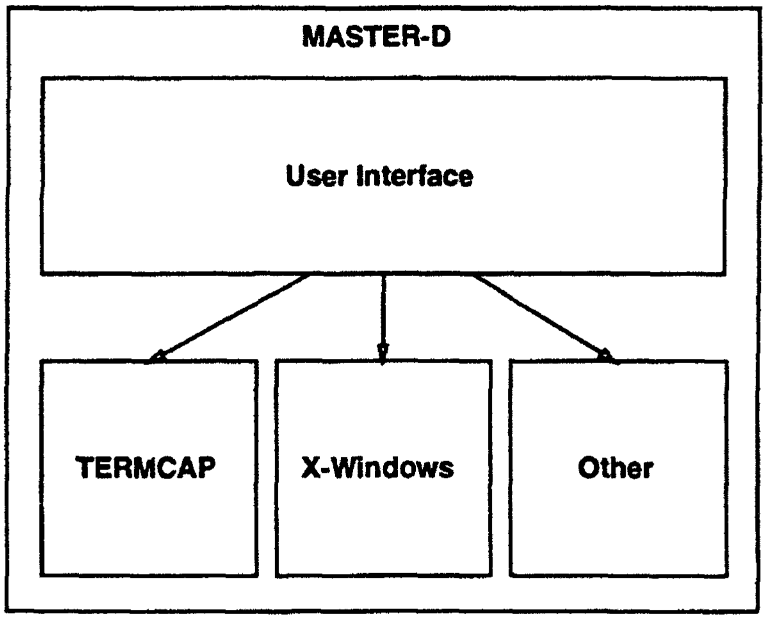
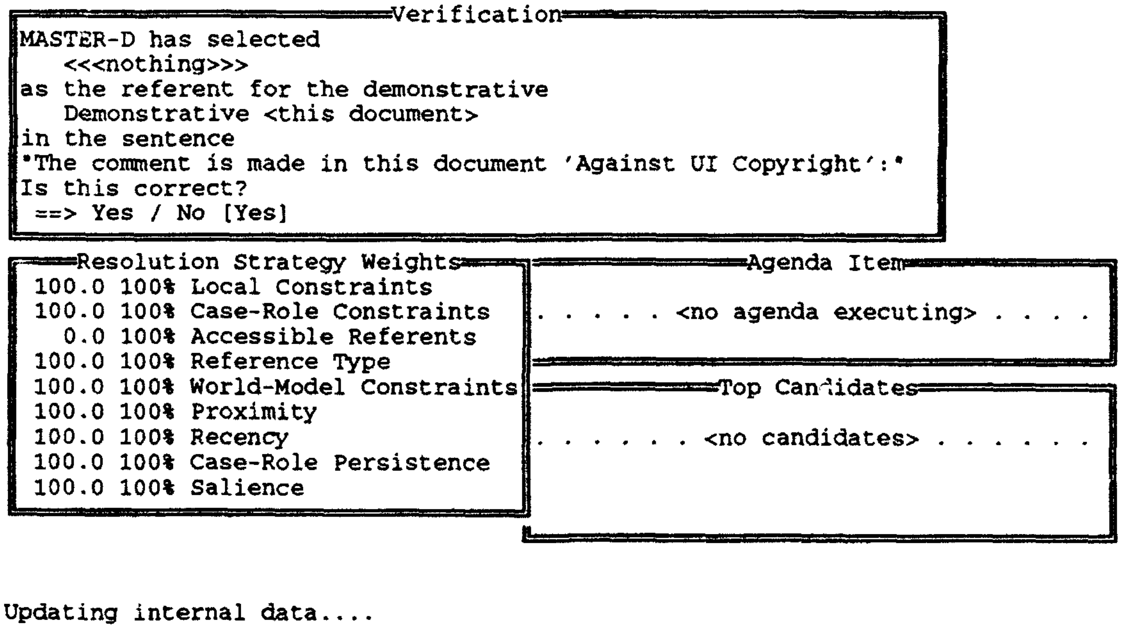


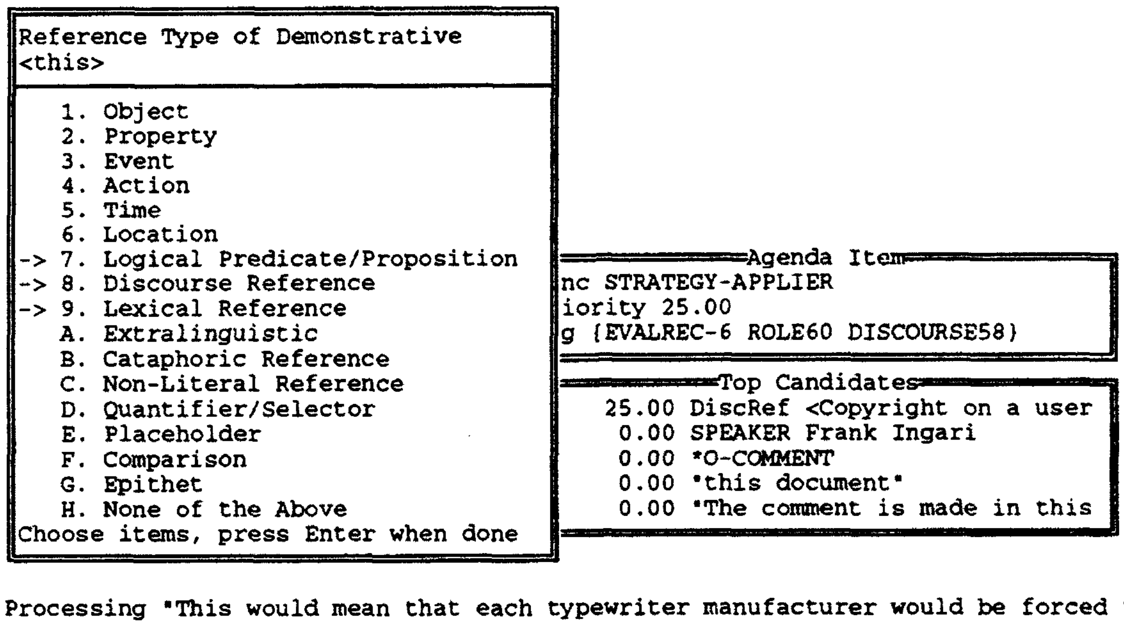
**Figure 5-2:** User Interface Architecture



```
═══════════════════════Verification═══════════════════════
MASTER-D has selected
   <<<nothing>>>
as the referent for the demonstrative
   Demonstrative <this document>
in the sentence
"The comment is made in this document 'Against UI Copyright':"
Is this correct?
 ==> Yes / No [Yes]

═══Resolution Strategy Weights═══ ══════════Agenda Item══════════
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ═════════Top Candidates═════════
 100.0 100% Proximity
 100.0 100% Recency               . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience


Updating internal data....
```

**Figure 5-3:** MASTER-D Screen Display

parser or generator, then reads the results and returns them to the caller. In both directions, Lisp data is printed out as ASCII for transmission over the network, and then read back into the Lisp system using (read) by the receiver.



```
Reference Type of Demonstrative
<this>

   1. Object
   2. Property
   3. Event
   4. Action
   5. Time
   6. Location
-> 7. Logical Predicate/Proposition    ==========Agenda Item=========
-> 8. Discourse Reference              nc STRATEGY-APPLIER
-> 9. Lexical Reference                iority 25.00
   A. Extralinguistic                  g {EVALREC-6 ROLE60 DISCOURSE58}
   B. Cataphoric Reference
   C. Non-Literal Reference            ==========Top Candidates=========
   D. Quantifier/Selector                25.00 DiscRef <Copyright on a user
   E. Placeholder                         0.00 SPEAKER Frank Ingari
   F. Comparison                          0.00 *O-COMMENT
   G. Epithet                             0.00 *this document*
   H. None of the Above                   0.00 *The comment is made in this
Choose items, press Enter when done

Processing *This would mean that each typewriter manufacturer would be forced t
```

**Figure 5-4:** A Multiple-Selection Menu

The socket servers in the parser and generator each provide three main functions that MASTER-D uses: "PARSE" or "GENERATE" to provide the actual input or output, "GC" to force a garbage collect while the parser or generator is otherwise idle, and "BYE" to terminate the connection. The "PARSE" command sends a string as its argument and returns the resulting parse. The "GENERATE" command sends a processed parse as its argument and returns the resulting text.

Further details on the socket interface are given in Section 8.6.2 in the "Implementation Details" chapter.

## 5.3. Collecting Candidate Referents

After each utterance is processed, it is examined for candidate referents, which are then placed in the pool of candidates for use by subsequent utterances. Since demonstratives may refer to a situation or implicit set as well as explicitly named objects, the candidate extractor must do more than simply selecting nouns and pronouns as was done in MARS. Among the more complex candidates are the surface form of the discourse segments (for references such as *that last sentence*), various aspects of the discourse segments such as the speech act or the time of events in the discourse, and sets implicitly formed by the items mentioned in the text.[7] Further, the recovery strategies (see Section 6.3 beginning on page 44) may place

[7]The following example provided by Deborah Dahl illustrates such set formation:

"This recipe calls for coriander, hoisin sauce, and sesame oil. *These* can all be found at your local Chinese grocery store."

*These* refers to the set of coriander, hoisin sauce, and sesame oil.

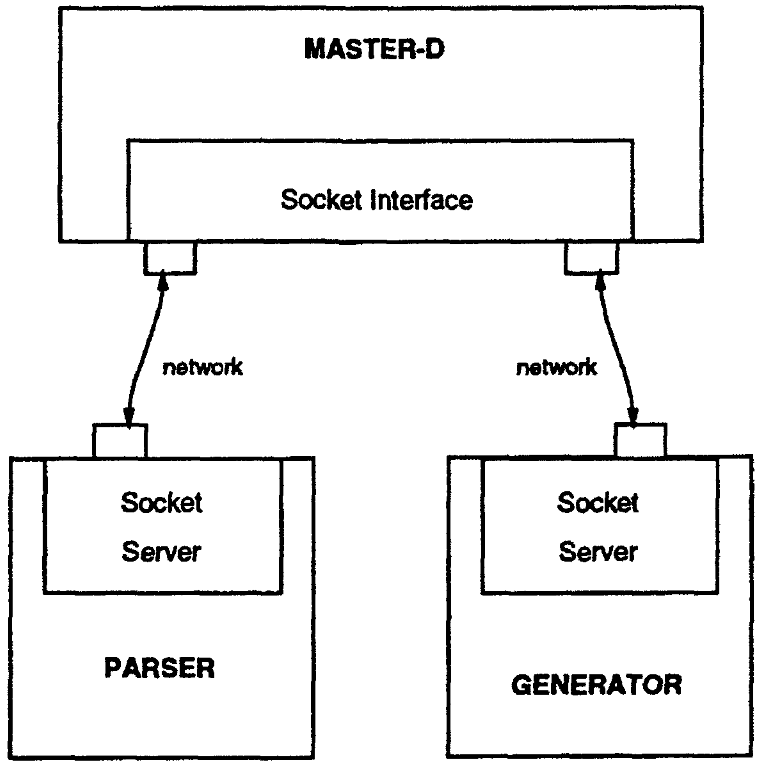


**Figure 5-5:** Parser and Generator Interface Architecture

additional candidates in the pool, such as candidates which are metonymically related to one which is already in the pool.

The current implementation does not deal with multiple antecedents to a single reference (implicit sets), but could readily be altered to do so (see Section 8.3 for implementation details). Since the candidates for a demonstrative are drawn from the candidate pool rather than generated as needed, the addition of multiple-antecedent candidates to the pool would affect only the code implementing those strategies which examine the structure of a candidate. Many multiple-antecedent candidates could be built from the simple antecedents in the candidate pool, which would slow down the system unnecessarily in most cases. Therefore, the best place to add the complex candidates would be in a recovery strategy which is invoked when a resolution fails to find any compatible referents. This arrangement permits the system to operate without the overhead of the additional candidates unless they are actually required.

Most candidates are eventually removed from the pool once the distance between the candidate and utterance is sufficiently large. There are, however, a number of candidates which are globally available and are thus never removed from the pool, such as the time and place of the utterance. The limit on the distance can be influenced by discourse analysis--once a discourse segment is closed, it becomes less likely that the candidates in that segment will be referenced again. The threshold appears to be much larger for demonstratives than for anaphoric pronouns, as references spanning twenty or more sentences are surprisingly common.

Maintaining a candidate pool is conceptually the inverse of the method used by Lucy, where candidates are produced during the attempted resolution of an anaphor. While on-the-fly candidate generation is potentially more efficient because fewer candidates must be checked, it does make the implicit assumption that resolution can stop as soon as a candidate's score exceeds a predetermined threshold. The candidate pool does not make that assumption (one of the tests performed on MASTER-D is whether that assumption holds); however, in the interests of efficiency, very old candidates are eliminated. That assumption of a limited range for references may be bypassed by setting the age at which candidates are removed to an extremely large value, larger than the size of the text to be processed. The age used by the current implementation is a count of the number of sentences prior to the current sentence plus a small multiple of the number of intervening paragraphs (to account for the segmentation of the text due to paragraph boundaries).

## 5.4. Strategy Applier

The strategy applier directs the application of the resolution strategies and maintains the total scores for each candidate, selecting the highest-ranked candidate at the end of the resolution phase.

The strategy applier starts by creating an evaluation record for each candidate in the candidate pool and placing that record on the resolution agenda. When all candidates have been placed on the agenda, the agenda handler is called to execute resolutions until the agenda is empty. The agenda handler orders the agenda items by the specified preference value, which is computed using any of the available functions from which the user may select. At this time, the functions include ranking by current total score, by number of strategies pending on the candidate, and by average score per resolution function executed.

For each item on the agenda, the agenda handler calls the associated function, which performs the actual resolution strategy application. The user may select whether all strategies should be applied at once, or whether the resolutions should be interleaved (the default). If the resolutions are interleaved, the application function only calls a single resolution strategy and then places the evaluation record back on the agenda.

Once the agenda has run to completion, the strategy applier determines the highest-ranked candidate as well as those candidates within a pre-specified threshold of the highest ranking. If there is exactly one candidate fitting this condition, it is returned as the referent of the demonstrative. If no candidates are suitable, the strategy applier invokes recovery functions which attempt to expand the candidate pool. If, on the other hand, multiple candidates fit within the threshold, the strategy applier invokes a different set of recovery functions which attempt to remove all but one of those candidates.

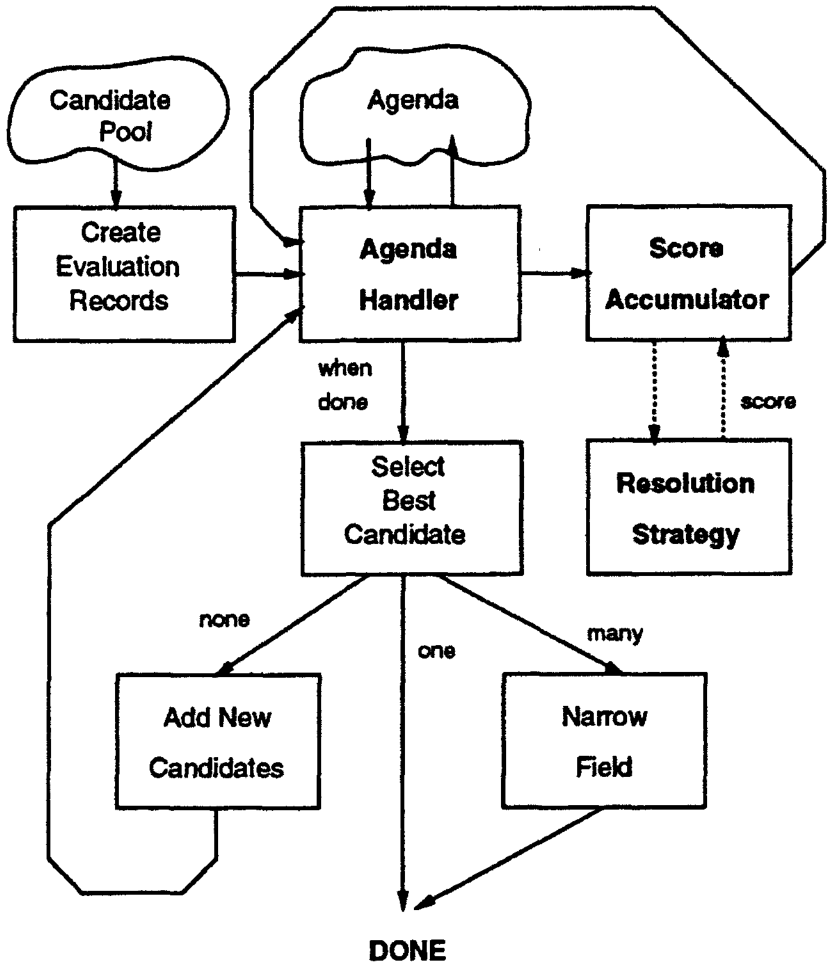


*Figure 5-6: Strategy Applier Architecture*

## 5.5. World Modeling

Since demonstratives have an inherent notion of proximity or distance, an important knowledge source will be the relative distances between the various objects in the discourse and the speaker, the listener, and each other. By maintaining a world model of the relationships between entities mentioned in the discourse, we may restrict the candidate space on the basis of distance and (for other languages) other factors such as visibility. These discourse entities need not have physical existence, so this world model corresponds roughly to Webber's discourse model [45].

Once a world modeler exists to maintain distance relationships, it is fairly straightforward to extend it to handle other relations. Given such a general world model, we may use those other relationships to further restrict the candidate space by ruling out candidates which would violate the assumed relations if coreferential with the particular role of the action containing the demonstrative.

To support the world modeler, the knowledge base must contain descriptions of the changes in relations between objects caused by each action, similar to the preconditions and postconditions used by MARS. An inference module within the modeler will use those descriptions to determine the changes in the world model caused by the utterances being processed.

Once again, the incremental nature of the multistrategy approach becomes useful. Since the world modeler is used to restrict the candidate space. an incomplete set of action descriptions merely degrades its performance, and does not completely invalidate it. For those entities for which the modeler either had no information or could not keep the relations updated, it simply returns a marker indicating that the information is not available. Any strategies using the modeler as a knowledge source can then return a vote which indicates either no preference or no confidence in the returned score. MARS used this method in its precondition/postcondition constraint strategy; when the knowledge base had no constraints for either the anaphor or the candidate referent, the strategy returned a "no preference" value.

# Chapter 6
# The Resolution Strategies

As has been stated previously, the strategies in MASTER-D are divided into four types in three classes: constraints, preferences, and recovery. As shown in Figure 6-1, these classes of strategies are applied sequentially until only a single candidate remains (i.e. if constraints rule out all but one candidate, no preferences or recovery strategies are applied). As will be discussed in Section 6.4, this is a generalization of the approach used by other multi-strategy resolution systems.

## 6.1. Constraint Strategies

The first class of strategies are the constraints, which may rule out candidate referents. Since the constraint strategies may be in error, however (i.e. if the utterance violates a constraint in making the reference), we allow the possibility of resuming the evaluation of a candidate which is ruled out by a constraint.

In some cases, a constraint strategy may determine that a particular candidate is not only acceptable, but is in fact preferred over other candidates. In such cases, the constraint strategy may cast a positive vote for the candidate, instead of a simple valid/invalid determination. For example, the Gender Constraint strategy not only rules out candidates whose gender is incompatible with the demonstrative, it also returns a positive score for (prefers) those candidates whose gender is an exact match; candidates which do not have the identical gender value but are compatible are ruled to be valid but receive a score of zero (no preference).

- **Local Constraints**
  Like anaphoric pronouns and definite NPs, demonstratives carry information such as number (*that* vs. *those*) and gender (*that man* vs. *that woman*) which their referents must agree with. In addition, the referent of a demonstrative NP must be a subclass or instance of the head noun of the demonstrative (*a red ball...that ball*). Finally, demonstratives which are the subject of a "be"-verb derive constraints from the object of the verb ("this is an excellent wine").
- **Case-role Constraints**
  As with anaphoric pronouns, any constraints imposed by the case role of the demonstrative must be compatible with the constraints imposed on candidate antecedents.
- **Accessible Referents**
  Not all segments of the prior discourse are accessible from the current utterance, so candidate referents contained in such inaccessible segments need not be considered any further. This strategy corresponds roughly to the Lucy discourse system's Logical Accessibility strategy. For example, any candidates in a closed discourse segment are generally inaccessible, such as candidates in an interruption once the main conversation resumes.
- **Reference Type**
  If the type of reference made by a demonstrative NP can be determined reliably without actually resolving it, referents which are not of the same or another compatible type may be

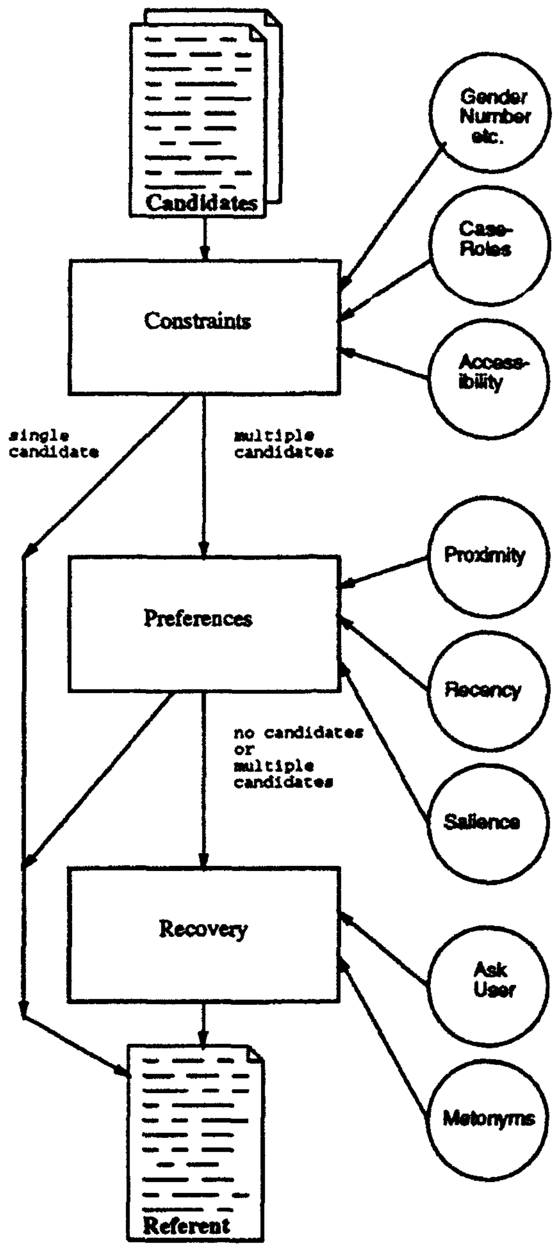


**Figure 6-1: Applying Resolution Strategies in MASTER-D**

ruled out immediately. For instance, *that time* could refer to a time or an event but not a location.

- **World-Model Constraints**
  By keeping track of the relationships between objects in the world, as modified by the propositions of the prior discourse, it may be possible to rule out candidates on the basis of violating constraints on the objects in the model. This is a considerably more general analogue to the precondition/postcondition constraints used by MARS, which were static assertions stored in the knowledge base.

It should be noted at this point that the world-model constraints may need to be demoted to a strong preference, as other strategies occasionally override the invalidation on the basis of the world model. For example, in

```
Although most stayed sealed, some of the envelopes opened
   due to defective adhesive.
Those were carefully resealed.
```

world-model constraints are completely compatible with other strategies such as recency, while in

```
Although some of the envelopes opened due to defective
   adhesive, most stayed sealed.
*Those were carefully resealed.
```

they conflict. Even though the world model indicates that the referent could not be the set of envelopes evoked by "most stayed sealed", that is in fact the referent which seems preferred, making this second example anomalous.

## 6.2. Preference Strategies

The second class of strategies are the preferences. These are heuristics, and therefore do not absolutely rule out any candidates. Since they vary in predictive power, each preference is given a separate weight which indicates the maximum amount it may contribute to the overall score for a candidate (the weighting is discussed in Chapter 7).

- **Proximity**
  The referent of a demonstrative should have the same proximity value as the demonstrative. Thus, "this car" should not be matched with a candidate which is considered to be remote from the speaker's point of view, such as a car which is several city blocks away. In languages other than English, additional distinctions beyond "near" versus "far" may be inherent in the demonstrative, and should also be taken into account. The world modeler described previously maintains the relationships between objects in the discourse from which the proximity values of candidates may be derived.

- **Recency**
  References and their antecedents are generally near each other lexically; thus, the likelyhood of a candidate being the referent of a demonstrative decreases as the distance between candidate and demonstrative increases. An example of recency effects is seen in the "envelopes" sentences above. This strategy corresponds to Intersentential Recency in MARS. For example, if everything else is equal, one would expect *those* in

  ```
  ....keys....
  ....tools....
  Those should be open.
  ```

  to refer to *tools* rather than *keys*.

- **Case-role Persistence**
  There is a pervasive form of linguistic inertia which manifests itself as a preference to maintain the same case role for an object. Thus, a candidate should be given preference if it fills the same role as the demonstrative, especially if the underlying action is the same. This strategy corresponds to the Case-role Persistence Preference in MARS.

- **Salience**
  Certain constructions (both syntactic and semantic) can make an object "stand out" and thus make it more likely to be the antecedent of a later reference. A specific example is topicalization, as described above for MARS; focus also plays an important role in making an entity salient. This strategy is a superset of both the MARS Syntactic Topicalization Preference and Lucy's Local Focus strategy. An entity which is in focus will also be more salient and thus more likely to be referenced multiple times; MASTER-D makes use of this behavior to declare a multiply-referenced entity to be in focus, i.e. in

```
While I am very much behind the Lotus position, I am
  speaking tonight as an individual, not as an officer
  of Lotus.
I say that by choice.
I wasn't asked to say that.
```

having determined that the first *that* references the first sentence, both it and the *that* become coreferential and are therefore considered to be in focus. The second *that* will then be given a positive salience score on the first sentence than on other sentences in the prior context.

## 6.3. Recovery Strategies

The third class of strategies are the recovery strategies, which are only applied when the constraints and preferences fail to produce a single most-preferred referent. Different strategies are applied depending on whether all candidates have been disqualified or multiple candidates remain.

- **Metonyms**
  When the constraints rule out all candidates, we may expand the space of candidates by considering metonymic references, e.g.

  ```
  I called the IRS.
  Those beaurocrats still haven't processed your tax return.
  ```

- **Relax Constraints**
  If there is a suspicion that one or more candidates were erroneously discarded, we may wish to reconsider those candidates which had the greatest preference before a strategy ruled them out, especially those which were almost able to pass the constraint. Goodman [20, 21] describes how to use such constraint relaxation to recover from reference identification failures.

  An example in which the usual gender constraints are violated is the German sentence (marked for gender)

  Der Kaiser$^M$, *das*$^N$ ist ein weiser Mann$^M$. [3]

  The Emperor, that is a wise man.

  In this example, the neuter *das* refers to the masculine NP *der Kaiser.*

- **Ask the User**
  When multiple candidates remain, let the user complete the selection process through an interactive dialogue. Similarly, if no candidates remain, ask the user to identify the referent in the preceding text.

Constraint relaxation is not explicitly implemented in the current system, but the mechanism used to apply the strategies provides some implicit constraint relaxation capability. Should all candidates be ruled out by one constraint or another, all candidates which had previously been invalidated will be reactivated. Resolution then continues until one or more candidates have been completely processed or all candidates have been ruled out by a second constraint (in which case all candidates are again reactivated).

### 6.4. Resolution Strategy Invocation

All of the above strategies are called individually by a central controller, which applies the various strategies to each candidate referent and maintains a running total of the scores returned by the strategies (see Section 7). Constraints are applied first, followed by preferences for those candidates which were not ruled out by the constraints. If there is not a unique referent remaining after those strategies, the recovery strategies are invoked. Since several of these strategies are actually groups of strategies (such as salience, which encompasses both local and global focus, among others), it may be more effective to treat each substrategy of those strategies as an independent strategy with a separate weight, and invoke the substrategies individually.

The three-step approach just outlined may be seen as a generalization of the approach used by MARS, the explicit statement of the approach used by SPAR, and an analogue of the approach used by Lucy. MARS used only the first two steps, not having an explicit concept of recovery strategies (although "ask the user" was implemented as part of the KBMT-89 Augmentor into which MARS was incorporated). SPAR uses all three classes of strategies, although it does not call its recovery strategy such explicitly. Recovery is attempted by invoking a backup inference mechanism using weak, very general rules if the normal mechanisms fail to bind all active anaphors. Lucy can be considered to interleave constraints and preferences, while effectively making most of its strategies recovery strategies. Since Lucy starts with an empty candidate space, every strategy which proposes new candidates is in effect recovering from an unsuccessful resolution by expanding the candidate space.

### 6.5. Scope

Since there is a wide variety of types of demonstrative reference, and time was constrained, MASTER-D only attempts to resolve a subset of those types, as shown in Figure 6-2. The heading "Partially Supported" in that figure shows those types of references for which MASTER-D contains only rudimentary support. For example, metonymic demonstratives are limited to those relations which may be readily derived from the knowledge base, such as "part for whole".

MASTER-D also restricts itself to the "near" vs. "far" distinction used by English, and does not attempt to address the additional distinctions present in other languages. Such finer distinctions could be added at a later time.

**Supported Reference Types**

- Object
- Property
- Event
- Action
- Locative

**Partially Supported Types**

- Logical
- Temporal
- Lexical
- Discourse
- Non-literal (metonyms only)

**Unsupported Types**

- Extralinguistic
- Quantifier/selector
- Comparison
- Cataphoric
- Epithet
- Placeholder

**Figure 6-2:** Type of Demonstrative References Supported by MASTER-D

# Chapter 7
# Adaptive Strategy Weighting

The main drawback of the multi-strategy approach is its sensitivity to the weighting of the strategies. Once there are more than a few strategies, determining the optimum weighting by hand becomes infeasible. To overcome this drawback, the MASTER-D system attempts to optimize the strategy weights automatically. Adaptive weighting is not a new idea; it dates back at least to Samuel's checkers program [39, 40]. However, it has not previously been used in conjunction with the multistrategy approach to reference resolution.

As currently implemented, MASTER-D assumes that the optimal weight for a strategy is the same regardless of the type of demonstrative, i.e. pronouns and full noun phrases. The system could fairly readily be modified to store multiple weights keyed to the type of demonstrative, but that would slow down weight convergence by splitting the reweighting among multiple values. Since the available test set was relatively small, this modification was not undertaken, but it could be performed in the future.

## 7.1. Strategy Weighting

Each strategy receives an individual weight which specifies the maximum effect the strategy may have on the overall score for a candidate referent. A strategy may return a score between -1 and +1, inclusive, or the symbol `invalid`. A score of zero means that the strategy had no opinion on the candidate, while increasingly positive scores indicate that the strategy was increasingly confident that the candidate is the correct referent, and increasingly negative scores indicate increasing confidence that the candidate is *not* the correct referent. The value `invalid` indicates that the strategy considers the candidate to be impossible due to constraint violations. The central strategy applier performs the actual strategy weighting by multiplying the returned raw score by the strategy's current weight before adding it to the total score (thus yielding a strategy score in the range -W to +W, inclusive, for a strategy of weight W). This separation of the strategy weight and the strategy's raw scoring simplifies other portions of the system, such as the reweighting mechanism.

MASTER-D makes a further separation of the raw score into the strategy's basic confidence based on the context, and the amount that the confidence is attenuated by the distance between demonstrative and candidate antecedent; this is done for additional flexibility. In the current implementation, the attenuation factor is fixed at load-time by specifying an attentuation function and speed factor in the strategy's definition; future versions may permit the reweighting mechanism to adjust either or both in attempting to optimize the system's performance. Since the strategy applier immediately applies the attenuation function to the base score returned by the resolution strategy, this separation is essentially transparent to the remainder of the system, and resolution strategies can be considered to return a single confidence score which depends on both the context and the distance between demonstrative and antecedent.

One case where this separation is of particular import is the Recency Preference strategy. To all external appearances, this strategy returns a confidence score which decreases as the number of parses intervening between the demonstrative and candidate increases. In actuality, the scoring function for this preference uniformly returns 1.0 (maximum preference) in all cases, and it is the decay function which actually produces the decreasing scores. In a version of the system where the reweighting method affects the decay functions as well as the strategy weights, this allows the reweighter to adjust the effective range of the Recency Preference strategy. For the tests described in Chapter 10, the decay speed was set to 0.25, which limits the range of recency effects to the previous eight parses (any candidate prior to that point receive a maximum negative preference). Should this prove to be too small, changing the decay function or reducing the decay speed would increase the effective range.

The central controller which applies the strategies converts a score of `invalid` into a highly negative score. This prevents further consideration of the candidate as long as there are candidates which have not been declared invalid by one or more strategies (see Section 7.2). Should all other candidates be deemed invalid, processing will continue; another score of `invalid` will again temporarily remove the candidate from consideration.

The weighting system described above is quite similar to that used by the MARS anaphor resolver except for the splitting of the raw score into a basic confidence and attenuation factor. In MARS, each strategy is given a weight which indicates the maximum positive contribution toward the overall score. Negative contributions are not limited, however, allowing a strategy which is the equivalent of a Lucy discourse system "fading infinite set generator" (such as recency) to return arbitrarily negative scores. A MARS strategy can also return `invalid` if it determines that a candidate violates one or more constraints. In MARS, a single strategy score of `invalid` converts the overall score to `invalid`, preventing any further consideration of the candidate.

The Lucy discourse system strategies return a pair consisting of a score and a confidence factor. Although arbitrary combinations of score and confidence are possible, the Lucy system only uses a small set of weighting patterns. For example, "finite set generators" and "fading infinite set generators" as described in [32] use a fixed value for the confidence score, while filters use only two (score,confidence) pairs such that the contribution of the strategy is either zero or highly negative. All three of these patterns may be created by a single numeric score such as MASTER-D uses, and in fact a suitable combination function could create a single score from the (score,confidence) pair in the patterns used by Lucy.

Even though returning a pair (score,confidence) is more general than a simple score, and may be slightly more accurate, a simple score lends itself more readily to automatic reweighting (see Section 7.3). As detailed above, a single number can produce most of the weighting patterns which are actually used in Lucy, so the loss of generality is much less than it might appear at first glance.

## 7.2. Strategy Ordering

Because the system will potentially examine all of the candidates for the preceding context, a method for ordering the candidates is highly desirable. The obvious ordering strategem is to order the agenda by the current total score of each candidate. The primary purpose of ordering is to defer further processing of candidates which have been declared invalid, since all strategies will be applied to each candidate which is not declared invalid. When the score `invalid` is converted into a highly negative value, ordering by current total score achieves the desired result. In addition, those candidates which have already been declared the preferred antecedent by one strategy will be processed first.

Two alternate processing orders are also available as options in MASTER-D. Processing by current weight tends to apply all strategies to a candidate at once, since an early positive score makes the candidate the highest-weighted among the incompletely processed candidates, where it usually remains until all strategies have been applied. Therefore, the user may also select to process candidates in order by remaining strategies or by average score per strategy applied. The former guarantees strict interleaving, since the second strategy will not be applied until all candidates have had the first strategy applied. The latter preserves the concept of processing the most-preferred candidates first while having much less of a tendency to apply all strategies to a candidate at once. However, the exact sequence in which the resolution strategies are applied to the various candidates is not of importance when all strategies are applied to all candidates, as is the case when reweighting is in effect. Only when the resolution process is stopped early, such as when a candidate reaches a particular threshold score, does the order of strategy application have an effect; such an early completion is not possible when reweighting is active because a strategy must be applied to all candidates to determine its accuracy. Whether it will be possible to stop the resolution process when a candidate reaches a particular threshold without adversely affecting accuracy remains to be determined.

## 7.3. Adaptive Reweighting

As mentioned at the beginning of this chapter, the greatest weakness of the multistrategy approach lies in assigning weights to the individual strategies, which in prior systems has been done by hand. However, the strategy weights can not be expected to be optimal initially, so a method for adjusting the weights automatically is desirable. After each utterance, we wish to reduce the weight of strategies which voted against the actual referent, and increase the weight of those which voted for it. As a result, the more accurate strategies will become increasingly influential in the determination of the referent for a demonstrative, while less reliable strategies will become less important.

MASTER-D provides a number of reweighting methods, from which the user selects one in the system's setup. The currently implemented methods are setting the weight based on the overall accuracy of the strategy, adjusting the weight by a fixed percentage after each resolution, and adjusting the weight by an exponentially decreasing amount after each resolution.

Setting the weight based on the overall accuracy is conceptually the simplest method, but determining the accuracy proves to be somewhat tricky, as will be discussed shortly. Once a strategy's accuracy has been computed, a simple linear function converts the accuracy into an actual weight. Initially, this function was

$$\frac{accuracy-50}{50} \times initial\_weight$$

yielding weights from `-initial_weight` to `+initial_weight` and a score of 0 for a 50-percent accurate strategy. This produced undesirable results, so the function was later modified to

$$\frac{accuracy-20}{80} \times initial\_weight$$

which yields weights from $-0.25 \times$ `initial_accuracy` to `+initial_accuracy` and a score of 0 for a 20-percent accurate strategy.

Adjusting the weight by a fixed percentage simply means changing the current weight based on how the strategy voted on the current resolution. If the strategy voted for the desired referent, its weight is increased; if it voted against, the weight is decreased; and if it offered no opinion, the weight remains unchanged. The change is five percent of the current weight by default, but the size of the change may be modified in the system configuration.

Incrementally adjusting the weight functions the same as fixed-percentage reweighting, except that the amount of the change is precomputed and does not depend on the strategies current weight. The amount by which strategy weights will be changed is the smaller of the initial change (30 by default) and

$$initial\_change / \frac{resolutions+1}{decay\_factor}$$

where `decay_factor` controls how quickly the amount of change decreases, and is 10 by default (thus, the first ten resolutions use the initial change, and the change is reduced to half the initial amount by the twentieth resolution).

In order to simplify the testing of other aspects of the system, the scores are rescaled (normalized) after every reweighting to keep the sum of the weights constant. For example, if there are eight strategies and each begins with an initial weight of 100, a candidate receiving full preference from all eight strategies will always receive a score of 800 (ignoring rounding errors), no matter how the strategies have been reweighted. This rescaling permits checks for thresholds at which candidates may be accepted or rejected automatically without further processing.

Assigning an accuracy to each strategy proves to be less than straightforward. Because of the large number of candidates which are processed during a resolution, it is not feasible to simply look at the candidate which proved to be the actual referent. Doing so tends to yield either an excessively low accuracy or an excessivly high accuracy. If the strategy is considered to have voted correctly only if it gave the highest score to the correct candidate, the accuracy will be underestimated whenever a strategy gives multiple candidates high scores and when the correct candidate received the second- or third-highest score. Conversely, if the strategy is considered to have voted correctly as long as it did not vote against the correct candidate, most strategies will typically receive high accuracy ratings even if they pick scores at random. Taking all incorrect candidates into account produces similar difficulties in rating accuracy. Because of these problems, the MASTER-D system provides multiple user-selectable accuracy functions, which are described in Section 8.2.

Another potential problem is skewing of the results by correlated strategies. If two or more strategies' scores are correlated (even if the strategies are otherwise entirely independent), then it is possible for the correlated strategies to vote incorrectly but override the correct votes of other strategies. In an unsupervised environment, these incorrect votes would then erroneously be considered correct, affecting the reweighting of all strategies. The current implementation of MASTER-D avoids this problem by always asking the user for verification that the resolution was correct whenever adaptive reweighting is enabled. The system can thus properly identify correct and incorrect votes even when the incorrect votes outweigh the correct votes.

# Chapter 8
# Implementation Details

This chapter provides details on the actual implementation used for a number of the components in the MASTER-D system. The first two components discussed are the core of the MASTER-D system: the strategy applier and strategy reweighter. The remaining sections of the chapter cover the implementation of components which are in a sense peripheral to the system, but nonetheless important. These components are the candidate determiner, the world modeler, the definition of new strategies, and the Unix interfaces to the display and to other processes in which the parser and generator are running.

The MASTER-D system consists of approximately 16,200 lines of Lisp code, plus a knowledge base which may be as large as required to represent the domain model and world modeler's inference data. The knowledge base currently contains 524 entries (frames) in a file which is 2338 lines in length.

## 8.1. Strategy Applier

The strategy applier is the heart of the system. It performs the actual invocation of the resolution strategies, the ordering thereof, the tallying of the scores for each candidate, and the selection of the highest-ranked candidate(s). The reader may wish to refer to the source code in Section B.1, beginning on page 123, while reading this section.

The agenda handler used by the strategy applier exports the following functions of importance to this discussion: `queue-agenda-item` and `run-entire-agenda`. The former places a new item on the agenda with the specified priority, while the latter repeatedly executes the first item on the agenda until the agenda contains no more items. An agenda item is a triple consisting of a priority value (used for ordering items on insertion), a function, and a list of arguments to pass to the function. The agenda handler invokes an agenda item by calling the function with the specified arguments; on return, the agenda item is discarded unless the function returns one of the special values `:quit` or `:requeue`. If the function returned `:quit`, all remaining items are removed from the agenda, effectively terminating execution. If the function returned `:requeue` or a list whose first element is `:requeue`, then the current item is placed back on the agenda either with its original priority value or with the priority specified by the second element of the returned list.

The strategy applier is indirectly invoked by `resolve-demonstrative`. This function creates a resolution record for the specified demonstrative and candidate referents, then creates one evaluation record for each candidate referent. As each evaluation record is created, it is added to the agenda as an argument to the function `strategy-applier`. When all candidates have been preprocessed in this manner, `run-entire-agenda` is called to invoke the strategy applier on all candidates for the current demonstrative. Once `run-entire-agenda` returns, all candidates have been evaluated, and the resolution record created as the first step of `resolve-demonstrative` is examined to determine the candidates considered best by the resolution strategies. If exactly one candidate remains, no further action is necessary; if no candidates remain, `initiate-recovery0` is invoked to attempt an expansion of the candidate space; finally, if multiple candidates remain, `initiate-recovery2` is invoked to disambiguate among the remaining candidates. After performing recovery actions, if necessary, `resolve-demonstrative` performs a variety of housekeeping, including invoking the strategy reweighter.

Function `strategy-applier` is designed such that it can be interrupted after each resolution strategy, permitting the processing of the candidates to be interleaved rather than completely processing one candidate before proceeding to the next. This interruption is made possible by using the `:requeue` feature of the agenda handler and storing a list of uncalled strategies in the evaluation record. It is actually implemented by having `strategy-applier` return with the `:requeue` flag; when it is subsequently re-invoked, it continues by processing the next strategy from the list of uncalled strategies.

## 8.2. Reweighting

The reweighter attempts to optimize the strategy weights by adjusting them after each resolution based on the outcome of that resolution. The reader may wish to refer to the source code in Section B.2, beginning on page 136, while reading this section.

The main function of the reweighter is `strategy-reweighter`, which is called with a completed resolution. It calls `reweight-strategy` on each constraint and preference strategy, then `rescale-strategies` to scale the resultant strategy weights such that the total weight remains constant. After rescaling, `strategy-reweighter` performs various housekeeping, including removing outdated voting information associated with each strategy.

The `reweight-strategy` function begins by determining the accuracy of the given resolution strategy, then invokes the user-specified reweighting method. On return from the reweighting method, it accumulates statistics on how the strategy's weight has changed over the course of all resolutions.

As was mentioned earlier, determining the accuracy of a strategy is not as straightforward as it first appears. The approach taken by the current implementation of MASTER-D (which could be improved) is to allow the user to select one of five accuracy functions which compute the accuracy in slightly different fashions. Each function awards a value between 0.0 and 1.0 points per demonstrative to each strategy depending on how it scored the candidates, and the overall accuracy is computed by dividing the total accuracy points by the number of demonstratives.

The first function looks at the absolute ranking of the correct referent among all candidates. It awards correctness points for each resolution as follows:

- 0.00 if the unweighted score of the correct referent is negative or `invalid`
- 0.50 if all candidates received a score of zero
- 1/N if all N candidates received the same nonzero score
- 1.00 if the correct referent has the highest unweighted score
- 0.50 if the correct referent has the second-highest unweighted score
- 0.33 if the correct referent has the third-highest unweighted score
- 0.25 if the correct referent has the fourth-highest unweighted score
- 0.00 otherwise

The rationale is to consider the strategy to have been "half-right" if the two highest-scoring candidates would have to be selected to include the correct one, "one-third-right" if the three highest are needed, etc. The score of 0.50 in case of all zero scores handles the case of a strategy which does not provide an opinion on any of the candidates and which therefore does not affect the outcome of the resolution one way or the other.

The second accuracy function is the same as the first one with the exception to an adjustment in ranking. When multiple candidates receive the same unweighted score, the ranking is reduced by one, such that a score of 1.0 would be ranked second-highest and a score of 0.7 would be ranked fourth-highest among 1.0, 1.0, 0.7, 0.7, 0.7, 0.7, and 0.6. Ranking by first occurrence in descending order proved to overstate the accuracy of strategies which return many duplicate scores, while ranking by last occurrence greatly understated their accuracy. Reducing the ranking by one is a compromise between those two alternatives, but most likely a more effective method is possible.

The third accuracy function simply examines the value returned by the strategy for the correct candidate, and assigns correctness points as follows:

- 0.00 if the correct referent's unweighted score is negative or `invalid`
- 0.50 if all candidates received the same score
- 1.00 if the correct referent has a positive score
- 0.50 if the correct referent's score is zero and no other candidate received a positive score
- 0.00 otherwise

The final two accuracy functions examine the relative ranking of the correct referent. The relative rank is computed by first deleting all duplicate scores and then determining the ranking within the remaining set of duplicate scores. The fourth accuracy function uses the same rules for awarding correctness points as the first strategy function, while the fifth function uses the same rules as the first except for the initial check for a negative or `invalid` score.

## 8.3. Candidate Determiner

The candidate determiner is the primary means by which new candidates are added to the candidate pool. Each candidate is represented by a FrameKit [35] frame, and the candidate pool is simply a list of the frame names for the candidates. The frame representing a candidate may be the actual frame used in the canonicalized parse, or it may be a frame especially constructed for the candidate pool by the candidate determiner. The latter type generally include a reference to some frame in the canonicalized parse.

The current implementation of the candidate determiner finds and adds five types of candidates: noun, lexical, discourse, logical proposition and predications, and property. To find the noun candidates, it traverses the tree formed by the canonical parse, extracting those frames which are stored in a pre-specified list of slots such as `agent` and `instrument`. Lexical candidates are found by traversing the tree a second time and checking for the surface strings or root forms the parser indicated as producing a particular portion of the parse. The discourse candidate is formed from the entire sentence; an enhancement would be to use discourse analysis to form candidates from entire discourse segments. Logical propositions and predications are found by traversing the parse tree a third time, checking for frames whose `is-a` link points at a subclass of the ontological frame `*logical-proposition*`. Finally, properties are found by traversing the parse tree yet again and checking for fillers of a set of slots known to represent properties.

For lexical and discourse candidates, the candidate determiner forms special frames which set the `is-a` and `$objtypes$` slots appropriately--to `*lexical-reference*` and `lexical-ref` or `*discourse-reference*` and `discourse-ref`, respectively. The lexical candidate also specifies the surface form in the `input` slot, while the discourse candidate also specifies the appropriate portion of the text through the `discourse-referent` slot.

Similarly, special frames are formed for logical and property candidates, which are `is-a` `*proposition*` and `*property*`, respectively. A logical candidate's frame points at the appropriate portion of the original parse in its `logical-prop` slot. A property candidate's frame contains the property slots which were found in the parse, a `property-of` slot pointing at the original frame, and a `property-type` slot containing a list of all the property slots.

Section 5.3 mentioned the possibility of multiple-antecedent candidates. These could be made part of the candidate pool by defining a new frame similar to those used for lexical and discourse candidates which indicates that particular type of antecedent and contains references to all of the antecedents for the candidate.

## 8.4. World Modeler

The simple world modeler used by MASTER-D acts on precondition and effect information stored in the ontology. Whenever the preconditions of an action are satisfied, the world model is updated according to the directives stored in the effect data.

The preconditions are processed one at a time until one is violated or all preconditions have been processed. A precondition specifies a relationship which either must exist or must not exist.[8] To permit relationships with unspecified objects in the world model, a variable may be specified, which is then bound to the object actually found. The bound variable may then be used in subsequent precondition statements to indicate the object to which it has been bound.

Should all preconditions be satisfied, the effect data is then processed one effect at a time. Each effect statement may either bind a variable for subsequent use, assert or retract a relation, or add or delete an object in the world model. Variables bound by the precondition statements may also be used in the effect statements.

A portion of the ontology used by MASTER-D is listed in Appendix C.

## 8.5. Adding New Strategies

The system is designed so that strategies may easily be added. The module `strategies.lisp` (see Appendix B.3 beginning on page 145) contains the declarations of the resolution strategies to be used as well as the top-level functions for those strategies.

To add a new strategy once it has been implemented, it must merely be added to the appropriate list of strategies: constraints, preferences, recovery by expansion, or recovery by restriction. The entry for the new strategy specifies a user-readable name, the function to invoke, and a variety of optional information; the structure containing these settings also includes fields used internally by MASTER-D for housekeeping and statistics. Optional information which may be specified for the strategy includes

- a non-default initial weight
- the initial decay function and decay speed to adjust for the distance between demonstrative and referent; these may be modified by some reweighting mechanisms
- a non-default distance function
- a function to be called before processing a demonstrative to prime the strategy
- a function to be called after the demonstrative has been resolved to update strategy-internal data
- a list of the substrategies invoked by the strategy, for use by the strategy applier when it is told to apply substrategies individually

[8]Nonexistence is defined as in Prolog, i.e. the relation is not known to exist; it may in fact exist but be unknown due to incompleteness of the world model.

```
(make-strategy
        :name "Accessible Referents"
        :func 'accessible-referents
        :init-weight 0.0  ; current only returns 0 and invalid
        :weight 0.0       ; must be same as init-weight
        :priming-func 'prime-accessibility
        :update-func 'update-accessibility
        )

(defun accessible-referents (demonstrative candidate
                             &aux access score)
   (setf access (or (global-candidate-p candidate)
                    (referent-accessible-p demonstrative
                             candidate)
                    ))
   (if access
      (if (numberp access)
         (setf score access)
      ;else
         (setf score 0.0)
      )
   ;else
      (setf score 'invalid)
   )
   score
)
```

**Figure 8-1:** Sample Strategy Definition

From the point of view of the strategy applier, the new interface to the new strategy consists solely of the strategy's main function, its optional priming and updating functions, and the distance and decay functions. For the strategy reweighter, the entire interface to the new strategy consists of the parameters and statistics stored in the strategy's data structure. Therefore a new strategy may be added without updating either the strategy applier or strategy reweighter.

The MASTER-D system provides a number of functions designed to support the resolution strategies, but a strategy can be entirely self-contained and never call other functions. Some of the support functions available to a resolution strategy are

- ontology access: retrieve an item from the knowledge base, determine the class of an object or reference, etc.
- world model access: determine what, if any, relations exist between two objects.
- object typing: determine the type of an item, i.e. Object, Action, Event, etc.
- substrategy applier: perform the default substrategy evaluation when strategies are treated monolithically even though they contain substrategies.

An example strategy definition is show in Figure 8-1. This strategy implements the accessibility constraint under the name "Accessible Referents", with a strategy weight of zero (which disables automatic reweighting for the strategy), using `accessible-referents` as its main function, `prime-accessibility` as the function to prepare the strategy for a new sentence, and `update-accessibility` as the function to process any changes after the sentence has been resolved. The function `accessible-referents` in turn calls a function that examines the discourse model to determine whether the specified candidate may be referenced by the demonstrative, and returns the appropriate score depending on the result. The function is already prepared for a preference value to be returned in addition to a simple yes/no response, but that has not yet been implemented. The current implementation is merely a very rudimentary version of a focus stack [22], which relies on information supplied by the parser to determine when to push and pop focus frames.

## 8.6. Low-Level Input/Output

Any program which performs input or output eventually must call one or more functions which actually invoke the operating system or some other service provider to do the actual data transfer, unless the program accesses the hardware directly. In MASTER-D, the user interface and parser/generator interfaces perform two different kinds of I/O, and thus two modules are required to allow them to perform the actual I/O. These two modules contain the low-level user-interface primitives for the display device being used (currently only Unix "termcap" is supported) and for performing network communications using the Unix socket system.

### 8.6.1. Unix TERMCAP Interface

The character-mode user interface is written in fairly portable Common Lisp (only four small functions and a few constants are implementation-dependent). It uses the Unix "termcap" system to determine the control sequences required to effect the various primitive actions performed in placing or removing menus, updating the status line, etc. To make use of this system, a special module interprets the "termcap" entry for the terminal being used and emits appropriate command sequences to effect the actions requested by the MASTER-D user interface code.

The termcap module begins by extracting the values of the TERM and TERMCAP environment variables. Under CMU Common Lisp, the program's environment is available as the value of the `extensions::*environment-list*` variable; other Lisp implementations may require that the environment be retrieved from a different variable or function, or that a function be written specifically for this purpose. The TERM and TERMCAP variables are then used to retrieve the terminal capability entry for the display being used; which is parsed to produce an association list of capability names and their values.

The various output requests made by the user interface code eventually funnel down to two functions in the termcap modules: `putchar` to display a character and `tputs` to send a terminal control sequence. The `tputs` function interprets the control sequence specification to generate the appropriate stream of characters to execute the control sequence with the specified parameters.

Since many of the functions such as popup windows and dialogs require the screen to be restored afterwards, and Unix does not provide a method for reading the current contents of the display, the termcap module maintains a secondary copy of the display's contents in memory. For each screen position, both the character and its attributes (boldface, reverse video, etc.) are stored in an array which is referenced when the current display contents must be retrieved. The `putchar` function updates the secondary copy with the character and the current set of attributes at the same time it sends the character to be displayed on the terminal.

### 8.6.2. Unix Socket Interface

The Unix socket interface functions by encapsulating requests and responses inside a simple protocol. The protocol begins each exchange with a line specifying the number of arguments to the command, followed by a line containing the command and finally zero or more lines containing the arguments. The reply to the command consists of a line containing either "ACK" if successful or "NAK" followed by an error message if unsuccessful. If successful, the return value follows the ACK on a separate line.

A line in the sense used in the previous paragraph may actually be transferred in multiple pieces and reassembled by the receiver. Each packet of data transferred over the Unix socket consists of a length byte followed by the actual data. This simplifies the socket-reading code as it knows at all times how much data may be read from the socket. To prevent overflowing the operating system's buffers, each packet is kept to a maximum of 127 data bytes (configurable); long lines are transparently split by the sender and reassembled by the receiver. The high bit of the length byte is used to indicate a continuation, and informs the receiver that more data will be appended to the current line.

Since Unix sockets only pass raw streams of bytes, all Lisp forms which must be transferred via the socket interface are first converted to ASCII text with the `format` function and then reconverted to Lisp forms with the `read` function. Thus there is a limitation on the forms which may be used as arguments to a command or as return values: the type of form must be known to the Lisp reader. For MASTER-D as currently implemented, this limitation has no effect, since it only passes Lisp atoms and lists between itself and the servers.

# Chapter 9
# Testing Protocol

Probably the most important portion of a dissertation is demonstrating the superiority of its methodology over existing systems, or its efficacy when there are no systems with comparable coverage. Such a demonstration requires extensive testing and performance evaluation. Thus, the methods by which effectiveness will be measured should be established at the outset.

## 9.1. Performance Measurement

Performance is measured as the percentage of correctly-resolved demonstratives. In computing the percentage, instances for which the candidate space was reduced to two or three referents, one of which is the correct one, count as one-half and one-third, respectively, of a correct answer. However, these instances are not counted as even partially correct should there have been only two or three candidates before resolution.[9]

The effectiveness of reweighting will be guaged in relation to a baseline accuracy determined by running the system with reweighting disabled. For simplicity, the strategies will each be given a weight of 100 for the baseline test, and as an initial weight in all other tests unless otherwise indicated.

In addition, the overall effectiveness of the system will be compared against a minimum (null) heuristic. Unfortunately, the minimum heuristic for demonstratives is not as straight-forward as for anaphoric pronouns, where selecting the lexically most recent candidate is correct a significant percentage of the time. Because of the wide variety of possible referents for a demonstrative, the minimum heuristic must either take into account the type of reference (thus effectively incorporating the Reference Type strategy) or risk greatly underperforming. Limiting the examination to demonstratives which reference objects allows examination of the system's effectiveness on a majority of the demonstratives with the same heuristic used as a benchmark for pronominal anaphora, without implicitly including one of the system's strategies in the minimum heuristic. Therefore, the comparison for accuracy will be between MASTER-D and the heuristic of selecting the lexically most recent object.

[9]This provision has no effect in practice, since only the first one or two sentences woud have so few candidates. A typical demonstrative in the later portions of a lengthy text will typically have 200 or more candidates from which to select a referent.

## 9.2. Test Data

Lack of a suitable parser with a broad coverage caused the testing corpus to be scaled back somewhat. The initial plan had been to use three or more samples of different styles of text, each with some 200 demonstratives. As the parses needed to be written by hand rather than automatically generated, only a single text was used, containing 98 demonstratives in 182 sentences. This text is listed in Appendix D.

Naturally, a test set of less than 100 demonstratives (of which only 68 are deemed to be in the scope of the resolver's capabilities) is insufficient for some of the tests one would like to perform on the system, and will produce inconclusive results on other tests. Despite its relatively small size, however, this test set proved to be near the upper limit of feasibility due to the slowness of the test machine. Test runs ranged in duration from one hour 35 minutes to more than three hours.

## 9.3. Questions to be Answered

Naturally, there are many different questions one can attempt to answer when testing a system. Some of the questions that might be answered are

1. How much does adaptive reweighting improve performance?
2. Which reweighting algorithm is optimal?
3. Does the order in which strategies are applied affect the accuracy of the system? If so, does the optimal order depend on the current strategy weights?
4. Is it more effective to separate the substrategies of non-monolithic strategies into separately-weighted strategies than to keep them combined?
5. Do strategy weights converge to different values for different types of text (dialogue, narrative, etc.)? For different styles of the same type?
6. Do different classes of demonstrative references need different strategy weights? If so, how much is performance affected if a demonstrative is incorrectly classified?
7. How much does each resolution strategy contribute to the overall performance?
8. Are the strategies in fact independent?
9. Are there sets of strategies which exhibit synergy?
10. Is there a threshold score above which one can immediately discard all other candidates?

The ideal answers to the first two questions would be that adaptive reweighting greatly improves performance and that a simple, computationally cheap reweighting algorithm is either optimal or so close to optimal that it is not worth using the actual optimal algorithm. In reality, the efficacy of adaptive reweighting depends considerably on how closely the initial strategy weights approximate the optimum strategy weights. If the optimum happens to have weights which are very nearly equal, then reweighting will appear to have little effect because the original weighting will already perform very well. However, this case is not the most probable, because recency effects and local constraints such as number and gender are likely to have a much larger impact on the overall accuracy of resolution than most other strategies.

Because the resolution strategies are meant to be independent, it seems improbable that the order of application will have an effect, provided that all strategies are applied to all candidates. In the presence of pruning, however, changing the order of evaluation may indeed change the outcome, as a candidate may be prematurely discarded which would have been kept under a different evaluation order.

There is no clear-cut *a priori* answer on the matter of separating out substrategies into separately-weighted strategies since two opposing effects apply. On the one hand, the finer reweighting possible with separately-weighted substrategies may substantially increase the accuracy of the resolution process. On the other hand, the additional variables introduced with the additional weights may increase the time needed to converge on the optimum weights to the point where convergence is not achieved on reasonably-sized texts.

As was shown in Section 2.3, the relative frequencies of use differ between various kinds of text. Therefore, one would expect the optimum strategy weights to differ between different kinds of text, and perhaps even between different authors of the same kind of text. Similarly, one would not be surprised to find that the referencing patterns for different types of references (object, event, action, time, etc.) are sufficently different to require different strategy weights; it is also quite possible that the optimum weights will differ depending on the location of the demonstrative in the parse (subject, object, theme, etc.) While the current system is not capable of supporting multiple weights per strategy, such support could be added with the appropriate modifications to the strategy applier and reweighter. Unfortunately, the limited test corpus does not allow questions about difering optimum weights to be answered at this time. Further, each criterion on which the weights are differentiated (reference type, location in parse, or possibly something else) increases the required test size because the differentiation effectively splits the test set into groups of tests, one per criterion value.

Determining whether the strategies are truly independent requires an analysis of the scores they produce to check for correlation (or anti-correlation) between strategies. Correlated strategies can cause problems with unsupervised reweighting, as was discussed in Section 7.3, which is the reason why the current system always queries the user to determine the actual antecedent to a demonstrative. Two strategies which seem likely to be correlated are proximity and salience, since salient discourse entities tend to be conceptually nearer and thus more likely to use a "near" demonstrative, and nearer objects tend to be more salient.

To determine whether any synergistic effects are present, one must check whether certain groupings of strategies give confidence beyond their individual reliabilities when all members of the grouping prefer the same candidate. This may be done with statistical analysis similar to that used in determining correlations, but synergy need not imply correlation or vice versa. Synergistic sets of strategies may well disagree on the desired candidate in most cases, yet be exceedingly accurate in those cases where all members of the synergistic set agree on a single candidate.

Answering the final question listed above involves examining the final scores for the candidates in each resolution. If the most-preferred candidate's score is always above some value that is rarely exceeded by less-preferred candidates, then the answer is Yes, such a threshold exists. Strategy reweighting complicates matters, however, and it may not be possible to make any determination until the strategy weights have converged and reweighting is disabled.

## 9.4. Biases

When testing any system, one must always be aware of the potential biases which could skew the results. The known biases for the tests performed on the system fall into several classes as listed below, some of which skew the results toward greater accuracy and some toward lesser accuracy. The biases which will be discussed in this section involve

1. Parses
2. World model
3. Ontology
4. Reweighting

Because the parses were hand-written, they can contain more accurate information than automatically-generated parses in certain areas. However, they were hand-written by a non-linguist, and thus are missing various nuances, and in a few cases entire subclauses. While strategies which benefit from tags and similar parser hints in the parse (such as Salience Preference) will show better performance, the lack of various details in the parses is likely to be detrimental to MASTER-D. The overall effect on accuracy is expected to be neutral or slightly negative.

The world model used for the tests described in the next chapter is quite limited. Thus, the world modeler will in most cases be unable to determine relationships between objects which a more complete world model would be able to track. This results in an underrepresentation of the world modeler's efficacy. The sample text used in testing causes a further reduction in apparent efficacy because it is not a narrative and thus provides few opportunities to use relations established earlier in the text.

The ontology used in testing is also fairly limited, including little knowledge not required to represent the items actually present in the sample text. While this will have the effect of focusing any metonymic searches on items which are likely to be relevant, the limited knowledge will probably be detrimental to other portions of the system.

None of the reweighting methods is particularly sophisticated, so the efficacy of reweighting is expected to be significantly underestimated by the results of using the existing methods. In particular, none of the reweighting methods addresses the decay function, which affects the adjusted scores of several of the resolution strategies; this omission means that a strategy whose scoring decays at the wrong speed cannot have its weight properly optimized. Further, one of the reweighting methods (weighting by accuracy) is strongly dependent on the values returned by the accuracy function. Since even slight variations in the accuracy function can have large effects on the computed accuracies, the particular reweighting method must be tested with each of the various accuracy functions available to the user.

# Chapter 10
# Performance and Test Results

This chapter presents the results of performing accuracy and timing tests on the MASTER-D system. The test corpus, consisting of 182 parses containing 98 demonstratives, which was used for these tests is listed in Appendix D.

In order to keep the number of candidates for each demonstrative to a manageable level, the candidate management was configured to remove any candidates which were more than 20 sentences "old". Paragraph breaks counted as ten sentences, so at most ten sentences from the previous paragraph--and none from any prior paragraphs--were included in the candidate pool. In one case, this resulted in the desired antecedent being removed from the pool before the demonstrative was reached.

## 10.1. Accuracy without Reweighting

When run without reweighting, the MASTER-D system gives an indication of the accuracy of the resolution strategies for the default or manually-assigned weights for the strategies. This may then serve as the baseline for comparison with the various reweighting methods.

The results of executing the resolver on the test corpus with reweighting disabled are shown in Figures 10-1 and 10-2. Figure 10-1 tallies the outcome of the resolutions, broken down separately for demonstrative pronouns and NPs. For each resolution, four different outcomes are possible:

1. **correct** The desired referent is returned.
2. **part.correct** The desired referent is one of a small set of returned candidates. The partially correct resolutions are further categorized by the size of the returned set (i.e 1/4 means that four candidates were returned).
3. **incorrect** The desired referent is not among the returned candidates. The special case of a NIL return value is listed on the following line.
4. **unhandled** The demonstrative reference is beyond the scope of the resolver, such as intrasentential or extralinguistic references; however, the resolver may (and often does) correctly indicate that the referent is not in its candidate pool by returning NIL.

The second table, Figure 10-2, lists a more precise assesssment of the resolver's accuracy based on whether the demonstrative is of a type that can be handled. This table considers the demonstrative to have been correctly resolved if the set of candidates was narrowed to no more than three, or a demonstrative with no referent in the candidate pool is resolved to NIL. The "weighted correctness" lines consider the

| | | | | | |
|---|---|---|---|---|---|
| demonstrative pronouns | | | | | |
| | correct: | 18 | 28.1% | | |
| | part.correct: | 12 | 18.8% | | |
| | | | | 1/2 | 5 |
| | | | | 1/4 | 2 |
| | | | | 1/5 | 2 |
| | | | | 1/6 | 1 |
| | | | | 1/7 | 1 |
| | | | | 1/8 | 1 |
| | incorrect: | 17 | 26.6% | | |
| | | | | NIL | 3 |
| | unhandled: | 17 | 26.6% | | |
| | total: | 64 | 100% | | |
| demonstrative NPs | | | | | |
| | correct: | 7 | 20.6% | | |
| | part.correct: | 1 | 2.9% | | |
| | | | | 1/3 | 1 |
| | incorrect: | 13 | 38.2% | | |
| | | | | NIL | 3 |
| | unhandled: | 13 | 38.2% | | |
| | total: | 34 | 100% | | |

**Figure 10-1:** Summary of Resolutions without Reweighting

resolution to have been half correct if the final result was a set of two referents and one-third correct if the result was a set of three referents.

For comparison, the following accuracies are obtained using the naive heuristic of choosing the lexically nearest noun or pronoun from the preceding sentence:

| | | |
|---|---|---|
| demonstrative pronouns | 14 | 21.9% |
| demonstrative NPs | 4 | 11.8% |
| overall | 18 | 18.4% |

| | | | |
|---|---|---|---|
| demonstrative pronouns | | | |
| | in resolver's scope: | 47 | (73.4%) |
| | at least 1/3 correct: | 23 | 48.9% |
| | weighted correctness: | 20.5 | 43.6% |
| | out of scope: | 17 | (26.6%) |
| | correctly resolved: | 8 | 47.0% |
| | total correct: | 31 | 48.4% |
| demonstrative NPs | | | |
| | in resolver's scope: | 21 | (61.8%) |
| | at least 1/3 correct: | 8 | 38.1% |
| | weighted correctness: | 7.33 | 34.9% |
| | out of scope: | 13 | (38.2%) |
| | correctly resolved: | 1 | 7.7% |
| | total correct: | 9 | 26.5% |
| overall | | | |
| | in resolver's scope: | 68 | (69.4%) |
| | at least 1/3 correct: | 31 | 45.6% |
| | weighted correctness: | 27.83 | 40.9% |
| | out of scope: | 30 | (30.6%) |
| | correctly resolved: | 9 | 30.0% |
| | total correct: | 40 | 40.8% |

**Figure 10-2:** Resolution Accuracy without Reweighting

| | | | | | |
|---|---|---|---|---|---|
| demonstrative pronouns | | | | | |
| | correct: | 20 | 31.2% | | |
| | part.correct: | 11 | 17.2% | | |
| | | | | 1/2 | 3 |
| | | | | 1/3 | 1 |
| | | | | 1/4 | 2 |
| | | | | 1/5 | 2 |
| | | | | 1/6 | 1 |
| | | | | 1/7 | 1 |
| | | | | 1/8 | 1 |
| | incorrect: | 16 | 25.0% | | |
| | | | | NIL | 0 |
| | unhandled: | 17 | 26.6% | | |
| | total: | 64 | 100% | | |
| demonstrative NPs | | | | | |
| | correct: | 7 | 20.6% | | |
| | part.correct: | 2 | 5.9% | | |
| | | | | 1/3 | 2 |
| | incorrect: | 12 | 35.3% | | |
| | | | | NIL | 3 |
| | unhandled: | 13 | 38.2% | | |
| | total: | 34 | 100% | | |

**Figure 10-3:** Summary of Resolutions with Fixed-Percentage Reweighting

## 10.2. Accuracy with Reweighting

Comparing the accuracy without reweighting to the accuracy with various reweighting functions indicates how effective the reweighting functions are. A more accurate measure may be obtained by using the final weights as the initial weights for a second run over the same input. The second run will use the optimized weights for all demonstratives, while the first run began with less optimal weights and had to adjust them towards more optimal values.

Figure 10-3 and the second column of Figure 10-6 show the results when using a fixed change of five percent of the current weight. If the strategy is deemed to have voted correctly, its weight is increased by five percent; if incorrectly, it is decreased by five percent, and if noncommittal, the weight remains unchanged. Due to the rescaling performed after reweighting, the effective weight change may differ. The final strategy weights are shown in Figure 10-14; as is expected, the three strongest strategies are local

| | | | | | |
|---|---|---|---|---|---|
| demonstrative pronouns | | | | | |
| | correct: | 19 | 29.7% | | |
| | part.correct: | 12 | 18.8% | | |
| | | | | 1/2 | 6 |
| | | | | 1/4 | 1 |
| | | | | 1/5 | 2 |
| | | | | 1/6 | 1 |
| | | | | 1/7 | 1 |
| | | | | 1/8 | 1 |
| | incorrect: | 16 | 25.0% | | |
| | | | | NIL | 3 |
| | unhandled: | 17 | 26.6% | | |
| | total: | 64 | 100% | | |
| demonstrative NPs | | | | | |
| | correct: | 7 | 20.6% | | |
| | part.correct: | 2 | 5.9% | | |
| | | | | 1/3 | 2 |
| | incorrect: | 12 | 35.3% | | |
| | | | | NIL | 3 |
| | unhandled: | 13 | 38.2% | | |
| | total: | 34 | 100% | | |

**Figure 10-4:** Summary of Resolutions with Incremental Reweighting

constraints, reference type, and recency, and these strategies have the highest weights at the end of processing.

Incremental reweighting was used in the run summarized in Figure 10-4 and the third column of Figure 10-6. For this method, the strategy weights are adjusted up or down by an absolute amount which decreases as more demonstratives are processed. This increment is initially 30 points; once ten demonstratives have been processed, the increment is set to 30 divided by one-tenth the number of previous resolutions. Thus, when the final demonstrative in the test corpus is processed, the increment is only 3.1. As with all reweighting methods, the strategy weights are rescaled (normalized) after reweighting, so the effective change in a strategy's weight depends on the changes in all other weights; the final weights are shown in Figure 10-14.

The test corpus was processed a second time with incremental reweighting, using the final strategy weights of the first run as the initial weights for the second run. The results of this second run are summarized in Figure 10-5 and the last column of Figure 10-6. The second run produced a slight

| | | | | | |
|---|---|---|---|---|---|
| demonstrative pronouns | | | | | |
| | correct: | 15 | 23.4% | | |
| | part.correct: | 19 | 29.7% | | |
| | | | | 1/2 | 7 |
| | | | | 1/3 | 4 |
| | | | | 1/4 | 1 |
| | | | | 1/5 | 3 |
| | | | | 1/6 | 1 |
| | | | | 1/7 | 1 |
| | | | | 1/8 | 1 |
| | | | | 1/9 | 1 |
| | incorrect: | 13 | 20.3% | | |
| | | | | NIL | 3 |
| | unhandled: | 17 | 26.6% | | |
| | total: | 64 | 100% | | |
| demonstrative NPs | | | | | |
| | correct: | 7 | 20.6% | | |
| | part.correct: | 4 | 11.8% | | |
| | | | | 1/2 | 1 |
| | | | | 1/3 | 3 |
| | incorrect: | 10 | 29.4% | | |
| | | | | NIL | 3 |
| | unhandled: | 13 | 38.2% | | |
| | total: | 34 | 100% | | |

**Figure 10-5:** Summary of Resolutions with Incremental Reweighting, Second Run

additional improvement in performance, though the number of demonstrative pronouns resolved uniquely to the correct referent decreased somewhat. The low weights for several of the strategies caused a larger number of candidates to pass the high-score threshold of 98% of the highest score, and this turned some previously-unique resolutions into resolutions returning multiple candidates. Conversely, some resolutions which formerly excluded the desired referent could include it because the smaller weight for some strategy which had previously caused it to be ruled out allowed it to fall within the high-score threshold. This is precisely the idea behind adaptive reweighting–lessen the importance of the incorrect strategies so that the correct strategies can override them.

In addition to the four test runs detailed above, three additional runs were performed using the weight-by-accuracy method and three different accuracy functions (modified absolute rank, relative rank, and relative

| | No Reweighting | | Fixed | | Incremental | | Second Incr. | |
|---|---|---|---|---|---|---|---|---|
| demonstrative pronouns | | | | | | | | |
| in resolver's scope: | 47 | (73.4%) | 47 | (73.4%) | 47 | (73.4%) | 47 | (73.4%) |
| at least 1/3 correct: | 23 | 48.9% | 24 | 51.1% | 25 | 53.2% | [illegible] | 55.3% |
| weighted correctness: | 20.5 | 43.6% | 21.83 | 46.4% | 22 | 46.8% | 19.83 | 42.2% |
| out of scope: | 17 | (26.6%) | 17 | (26.6%) | 17 | (26.6%) | 17 | (26.6%) |
| correctly resolved: | 8 | 47.0% | 8 | 47.0% | 8 | 47.0% | 8 | 47.0% |
| total correct: | 31 | 48.4% | 32 | 50.0% | 33 | 51.6% | 34 | 53.1% |
| demonstrative NPs | | | | | | | | |
| in resolver's scope: | 21 | (61.8%) | 21 | (61.8%) | 21 | (61.8%) | 21 | (61.8%) |
| at least 1/3 correct: | 8 | 38.1% | 9 | 42.9% | 9 | 42.9% | 11 | 52.4% |
| weighted correctness: | 7.33 | 34.9% | 7.66 | 36.4% | 7.66 | 36.4% | 8.5 | 40.5% |
| out of scope: | 13 | (38.2%) | 13 | (38.2%) | 13 | (38.2%) | 13 | (38.2%) |
| correctly resolved: | 1 | 7.7% | 1 | 7.7% | 1 | 7.7% | 1 | 7.7% |
| total correct: | 9 | 26.5% | 10 | 29.4% | 10 | 29.4% | 12 | 35.3% |
| overall | | | | | | | | |
| in resolver's scope: | 68 | (69.4%) | 68 | (69.4%) | 68 | (69.4%) | 68 | (69.4%) |
| at least 1/3 correct: | 31 | 45.6% | 33 | 48.5% | 34 | 50.0% | 37 | 54.4% |
| weighted correctness: | 27.83 | 40.9% | 29.5 | 43.4% | 29.66 | 43.6% | 28.33 | 41.7% |
| out of scope: | 30 | (30.6%) | 30 | (30.6%) | 30 | (30.6%) | 30 | (30.6%) |
| correctly resolved: | 9 | 30.0% | 9 | 30.0% | 9 | 30.0% | 9 | 30.0% |
| total correct: | 40 | 40.8% | 42 | 42.9% | 43 | 43.9% | 46 | 46.9% |

**Figure 10-6:** Resolution Accuracy with Various Reweighting Methods

value); a final run made after correcting the problems discussed here will be presented later. These runs aborted due to a short-coming in the user interface code, which proved unable to deal with menus more than one screen (60 lines) in length. Before these runs aborted, however, it was already clear that the weight-by-accuracy method was producing significantly poorer results than no reweighting. These poor results appear to be due primarily to negative weights for better strategies such as recency. On the run using the modified absolute ranking as the accuracy function, the constraint strategies all tended toward a 50 percent accuracy rating, resulting in zero weights for their preference portions. A summary of the partial results is shown in Figures 10-7 and 10-8, but it does not adequately reflect the subjective feeling that the resolver was producing much more ambiguous results--several of the incorrect resolutions returned sets of more than 20 candidates, none of which was the desired referent. The most ambiguous result for any of the other reweighting methods contained fourteen candidates, none of which was correct, but all other resolution results with those methods contained ten or fewer candidates. A massively ambiguous result (at least 30 candidates remaining) was the cause of each run's premature termination.

The problems with weight-by-accuracy point out a shortcoming in the accuracy functions: what is needed is less an overall accuracy computation than one which takes into account only those resolutions on which

```
                          Absol.Rank   Rel. Rank    Rel. Value
demonstrative pronouns
    correct:               4  19.0%     8  19.0%     8  18.2%
    part.correct:          2   9.5%    10  23.8%    11  25.0%
    incorrect:            14  66.7%    14  33.3%    15  34.1%
    unhandled:             1   4.8%    10  23.8%    10  22.7%
    total:                21  100%     42  100%     44  100%
demonstrative NPs
    correct:               3  30.0%     3  15.0%     3  15.0%
    part.correct:          0   0.0%     2  10.0%     2  10.0%
    incorrect:             4  40.0%     8  40.0%     8  40.0%
    unhandled:             3  30.0%     7  35.0%     7  35.0%
    total:                10  100%     20  100%     20  100%
```

**Figure 10-7:** Summary of Weight-by-Accuracy Tests

the strategy offered an opinion. A partial attempt at effecting this modification was the addition of the 50% accuracy score for resolutions on which all returned values were zero, but this proved to be inadequate. The accuracy function should store both the number of applicable resolutions and the total accuracy points for those resolutions, rather than relying on a global variable to count the total resolutions. This modification was made for the final test run.

Another factor which may affect resolution accuracy is the handling of substrategies. Two of the strategies (Local Constraints and Salience) consist of multiple substrategies, and these substrategies have thus far been ignored, instead reweighting the strategies as a monolithic whole. MASTER-D has the option to treat substrategies as separate strategies for the purposes of adjusting strategy weights. Figure 10-9 compares the incremental adjustment method when applied to substrategies both monolithically and individually. For this small test set, the results are too nearly equal to make any definitive observation about which approach is better--in fact, the results are identical except for two resolutions in which the monolithic approach returned two candidates and the individual approach returned three candidates.

One final test run using the weight-by-accuracy method and the relative-rank accuracy function was made after two modifications were performed. The first modification has already been mentioned: changing the accuracy computation to disregard resolutions where the strategy did not apply and provided no opinion. The second modification was a slight change in the function used to convert the accuracy percentage into an actual strategy weight. Instead of linear function from 0 to 100 into -W to +W (where W is the initial strategy weight, 100 points by default), the new function is a linear function returning -0.25W to +W. This shifts the zero point from 50 percent down to 20 percent and prevents the strategy weights from going negative for the test set being used. The results of the run with these two modifications are shown in Figures 10-10 and 10-11. These figures show this test run to have been slightly better on pronouns and slightly worse on NPs than the incremental adjustment method. A much larger test set would be required to definitively state whether this particular variation of weight-by-accuracy is better or worse than incremental reweighting.

| | Absolute Rank | | Relative Rank | | Relative Value | |
|---|---|---|---|---|---|---|
| demonstrative pronouns | | | | | | |
| in resolver's scope: | 20 | (95.2%) | 32 | (76.2%) | 34 | (77.3%) |
| at least 1/3 correct: | 4 | 20.0% | 12 | 37.5% | 12 | 35.3% |
| weighted correctness: | 4 | 20.0% | 9.83 | 30.7% | 9.83 | 28.9% |
| out of scope: | 1 | (4.8%) | 10 | (23.8%) | 10 | (22.7%) |
| correctly resolved: | 0 | 0.0% | 5 | 50.0% | 5 | 50.0% |
| total correct: | 5 | 23.8% | 17 | 40.5% | 17 | 38.6% |
| demonstrative NPs | | | | | | |
| in resolver's scope: | 7 | (70.0%) | 13 | (65.0%) | 13 | (65.0%) |
| at least 1/3 correct: | 3 | 42.9% | 4 | 30.1% | 4 | 30.1% |
| weighted correctness: | 3 | 42.9% | 3.33 | 25.6% | 3.33 | 25.6% |
| out of scope: | 3 | (30.0%) | 7 | (35.0%) | 7 | (35.0%) |
| correctly resolved: | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| total correct: | 3 | 30.0% | 4 | 20.0% | 4 | 20.0% |
| overall | | | | | | |
| in resolver's scope: | 27 | (87.1%) | 45 | (72.6%) | 47 | (73.4%) |
| at least 1/3 correct: | 7 | 25.9% | 16 | 35.6% | 16 | 34.0% |
| weighted correctness: | 7 | 25.9% | 13.16 | 29.2% | 13.16 | 28.0% |
| out of scope: | 4 | (12.9%) | 17 | (27.4%) | 17 | (36.2%) |
| correctly resolved: | 0 | 0.0% | 5 | 29.4% | 5 | 29.4% |
| total correct: | 7 | 22.6% | 18.16 | 29.3% | 18.16 | 28.4% |

**Figure 10-8:** Resolution Accuracy with Weight-by-Accuracy Reweighting

Some general observations were made during the course of testing. The most-ambiguous resolutions were generally those involving lexical and logical references, since those types of candidates have only rudimentary support in the current implementation. Further, these resolutions were essentially unaffected by changes in strategy weights precisely because of the rudimentary nature of the system's support for them. In general, if a demonstrative was resolved correctly with reweighting disabled, it was also resolved correctly with each of the successful reweighting methods, though there were a few which became ambiguous or even failed to resolve correctly after reweighting.

A few particular erroneous resolutions were noted as occurring on all test runs except the original weight-by-accuracy runs (which produced a considerable amount of nonsense). For two demonstratives, all of the candidates presented in the result set were lexical (surface string) candidates, even though the desired referent was not; in one case, the system knew from user interaction that the desired referent was not a lexical item. For that resolution, the result set consisted of either two or three items (depending on the reweighting method in use), one of which was the surface form of the desired candidate. This error may stem from the rudimentary support for lexical items, for which only the recency and reference type strategies apply (focus applies only after a demonstrative has already referenced the lexical item).

| | No Reweighting | | Incr.Monolith | | Incr.SepSubstr | |
|---|---|---|---|---|---|---|
| demonstrative pronouns | | | | | | |
| in resolver's scope: | 47 | (73.4%) | 47 | (73.4%) | 47 | (73.4%) |
| at least 1/3 correct: | 23 | 48.9% | 25 | 53.2% | 25 | 53.2% |
| weighted correctness: | 20.5 | 43.6% | 22 | 46.8% | 21.66 | 46.1% |
| out of scope: | 17 | (26.6%) | 17 | (26.6%) | 17 | (26.6%) |
| correctly resolved: | 8 | 47.0% | 8 | 47.0% | 8 | 47.0% |
| total correct: | 31 | 48.4% | 33 | 51.6% | 33 | 51.6% |
| demonstrative NPs | | | | | | |
| in resolver's scope: | 21 | (61.8%) | 21 | (61.8%) | 21 | (61.8%) |
| at least 1/3 correct: | 8 | 38.1% | 9 | 42.9% | 9 | 42.9% |
| weighted correctness: | 7.33 | 34.9% | 7.66 | 36.4% | 7.66 | 36.4% |
| out of scope: | 13 | (38.2%) | 13 | (38.2%) | 13 | (38.2%) |
| correctly resolved: | 1 | 7.7% | 1 | 7.7% | 1 | 7.7% |
| total correct: | 9 | 26.5% | 10 | 29.4% | 10 | 29.4% |
| overall | | | | | | |
| in resolver's scope: | 68 | (69.4%) | 68 | (69.4%) | 68 | (69.4%) |
| at least 1/3 correct: | 31 | 45.6% | 34 | 50.0% | 34 | 50.0% |
| weighted correctness: | 27.83 | 40.9% | 29.66 | 43.6% | 29.33 | 43.1% |
| out of scope: | 30 | (30.6%) | 30 | (30.6%) | 30 | (30.6%) |
| correctly resolved: | 9 | 30.0% | 9 | 30.0% | 9 | 30.0% |
| total correct: | 40 | 40.8% | 43 | 43.9% | 43 | 43.9% |

**Figure 10-9:** Comparison of Monolithic and Individual Reweighting of Substrategies

Another erroneous resolution could be corrected, but only by applying one of the recovery strategies before the main resolution strategies. In the example of

```
I'm a musician.
That's what I was doing before....
```

A metonymic reference is made to the profession of *a musician*, i.e. playing music. Since metonyms are not examined unless no other candidate is suitable, the presence of another candidate which can even remotely be matched to the demonstrative blocks the use of metonyms, preventing correct resolution of the demonstrative. Some method of determining likely metonymic references would be desirable in order to allow the additional candidates reached through a metonymic relation to be added to the candidate pool prior to the start of the resolution process, but only when needed in order to avoid an unacceptable increase in processing time.

Finally, tuning the ontology specifically for the text being used would probably allow several additional demonstratives to be correctly resolved.

| | | | | | |
|---|---|---|---|---|---|
| demonstrative pronouns | | | | | |
| | correct: | 20 | 31.3% | | |
| | part.correct: | 12 | 18.8% | | |
| | | | | 1/2 | 5 |
| | | | | 1/4 | 2 |
| | | | | 1/5 | 2 |
| | | | | 1/6 | 1 |
| | | | | 1/7 | 1 |
| | | | | 1/8 | 1 |
| | incorrect: | 15 | 23.4% | | |
| | | | | NIL | 3 |
| | unhandled: | 17 | 26.6% | | |
| | total: | 64 | 100% | | |
| demonstrative NPs | | | | | |
| | correct: | 7 | 20.6% | | |
| | part.correct: | 1 | 2.9% | | |
| | | | | 1/3 | 1 |
| | incorrect: | 12 | 35.3% | | |
| | | | | NIL | 3 |
| | unhandled: | 13 | 38.2% | | |
| | total: | 34 | 100% | | |

**Figure 10-10:** Summary of Resolutions with Corrected Weight-by-Accuracy

## 10.3. Strategy Weights

This section will explore the actual weights of the resolution strategies, which so far have been ignored in favor of other aspects of the resolution process such as the results. Figure 10-12 shows the statistics produced at the end of the run using the Incremental Adjustment reweighting method. The "Reweights" column indicates how many times the strategy's weight was adjusted upward and how many times it was adjusted downward, ignoring changes produced by normalizing the strategy weights. This column also gives an indication that the strategies other than Local Constraints, Accessible Referents, Reference Type, and Recency are only infrequently used.

The graph in Figure 10-13 shows the evolution of the strategy weights for three strategies. The upper curve is the Reference Type strategy, the middle curve is the Recency preference, and the lower curve is the Salience strategy. Strategy weights were rounded to the nearest point before plotting, so some of the finer weight variations later in processing are not seen. Because of the normalization process, the weights

```
                          No Reweighting  By Accuracy  Incremental
demonstrative pronouns
   in resolver's scope:   47    (73.4%)   47   (73.4%)  47    (73.4%)
   at least 1/3 correct:  23     48.9%    25    53.2%   25     53.2%
   weighted correctness:  20.5   43.6%    22.5  47.9%   22     46.8%

   out of scope:          17    (26.6%)   17   (26.6%)  17    (26.6%)
   correctly resolved:     8     47.0%     8    47.0%    8     47.0%

   total correct:         31     48.4%    33    51.6%   33     51.6%

demonstrative NPs
   in resolver's scope:   21    (61.8%)   21   (61.8%)  21    (61.8%)
   at least 1/3 correct:   8     38.1%     8    38.1%    9     42.9%
   weighted correctness:   7.33 34.9%      7.33 34.9%     7.66 36.4%

   out of scope:          13    (38.2%)   13   (38.2%)  13    (38.2%)
   correctly resolved:     1      7.7%     1     7.7%    1      7.7%

   total correct:          9     26.5%     9    26.5%   10     29.4%

overall
   in resolver's scope:   68    (69.4%)   68   (69.4%)  68    (69.4%)
   at least 1/3 correct:  31     45.6%    33    48.5%   34     50.0%
   weighted correctness:  27.83 40.9%     29.83 43.9%   29.66 43.6%

   out of scope:          30    (30.6%)   30   (30.6%)  30    (30.6%)
   correctly resolved:     9     30.0%     9    30.0%    9     30.0%

   total correct:         40     40.8%    42    42.9%   43     43.9%
```

**Figure 10-11:** Resolution Accuracy with Corrected Weight-by-Accuracy

```
                                 Initial   Final    Reweights
      Strategy                   Weight    Weight     Up/Down  Accuracy
-----------------------------------------------------------------
Local Constraints               100.000   286.532   77/   2    94.94%
Case-Role Constraints           100.000    23.221    0/   0    51.90%
Accessible Referents              0.000     0.000    0/   0    50.00%
Reference Type                  100.000   263.770   71/   4    94.30%
World-Model Constraints         100.000    23.221    0/   0    50.00%
Proximity                       100.000    36.245   12/   0    42.30%
Recency                         100.000    97.293   63/  13    70.67%
Case-Role Persistence           100.000    29.235   16/   0    41.67%
Salience                        100.000    40.486   21/   0    16.98%
-----------------------------------------------------------------
Total Rescaling Factor: 0.23
=============================
```

**Figure 10-12:** Strategy Weight Statistics at End of a Test Run

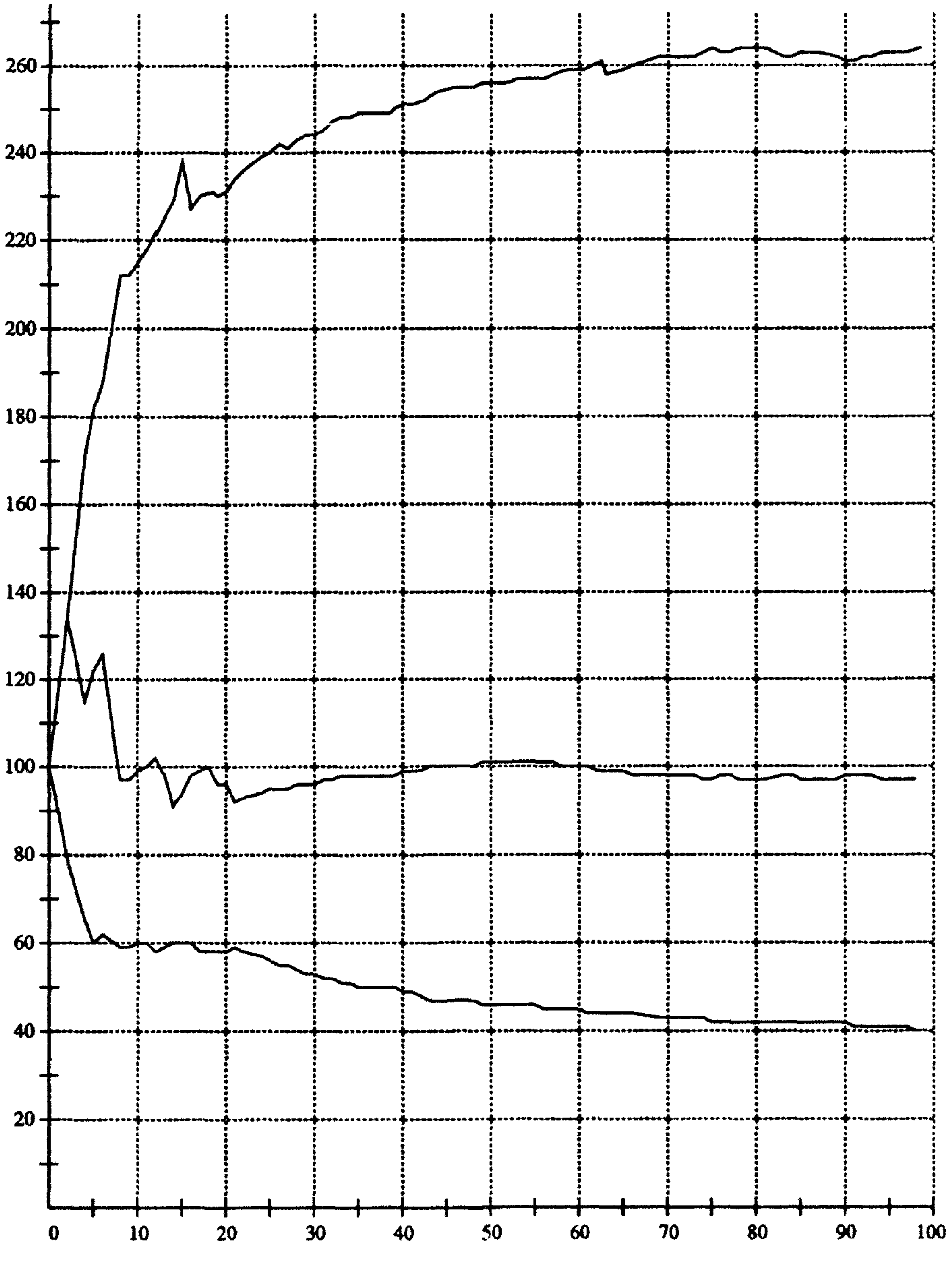


**Figure 10-13:** Change in Strategy Weights over Time

shown in the graph are influenced not only by the reweighting of the individual strategies, but also by the overall trend in weights. Thus, the gradual decline in Salience's weight (the lower curve) after 40 resolutions is due as much to an average increase in weight of the other strategies as to changes to Salience.

| Strategy | Fixed | Increm. | 2nd Inc | By Acc1 | By Acc2 | ByAcc (corr) |
|---|---|---|---|---|---|---|
| Local Constraints | 359.6 | 286.5 | 391.4 | 0.0 | 270.3 | 122.7 |
| Case-Role Constraints | 8.0 | 23.2 | 2.8 | 0.0 | 6.0 | 139.2 |
| Reference Type | 295.1 | 263.8 | 341.6 | 0.0 | 282.4 | 131.6 |
| World-Model Constraints | 8.0 | 23.2 | 2.8 | 0.0 | 0.0 | 139.2 |
| Proximity | 17.5 | 36.2 | 6.3 | -203.4 | 138.2 | 75.6 |
| Recency | 69.2 | 97.3 | 43.3 | -134.1 | -5.0 | 87.9 |
| Case-Role Persistence | 16.7 | 29.2 | 4.1 | -152.0 | -52.1 | 53.1 |
| Salience | 25.9 | 40.5 | 7.8 | -310.6 | 46.1 | 40.5 |

**Figure 10-14:** Final Strategy Weights

Figure 10-14 summarizes the strategy weights at the end of various test runs. The first five columns show the weights after using the fixed-percentage adjustment, the incremental change method using first the default initial weights and then the result of the first run, and weight-by-accuracy with the modified absolute rank and relative value accuracy functions, respectively. One can easily see that the first three columns show similar results while the two weight-by-accuracy columns are very different. The final column shows weight-by-accuracy after making the two modifications to avoid the problems encountered in the first two tests; this run used the relative rank accuracy function. For this column, the Case-Role Constraints and World-MOdel Constraints strategies have the highest weights because they never applied for the test set being used, and thus are computed to have been correct 100% of the time. Contrast this with the first three columns, where these two strategies had the lowest weights because their weights were never changed while all other strategies had their weights increased more often than decreased.

## 10.4. Speed

The following measurements of CPU times and elapsed times were made on an IBM RT PC with 12MB RAM running the Mach operating system and X Windows, using CMU Common Lisp version M2.9. The IBM RT is, by current standards, a rather slow machine (5-6 VAX MIPS); more recent workstations could execute the software about an order of magnitude more quickly. More RAM would noticeably speed up processing, as the IBM RT suffered a fair amount of disk thrashing while executing MASTER-D, particularly during garbage collection.

Figure 10-15 shows the execution profile statistics generated by MASTER-D at the end of a typical run on the test corpus. Because the statistics were gathered using functions with a granularity of 0.01 second, some inaccuracies are possible for the more quickly executing functions (such as the anomalous case of the world modeler, which shows CPU time greater than real time). The strategy applications happen so quickly that it is likely that a significant portion of the time credited as Resolution Overhead was in fact spent in the actual application of resolution strategies.

The different phases of execution listed in Figure 10-15 are:

```
      Phase                    Real Time          CPU time      Calls
---------------------------------------------------------------------
Parser Interface                36.61 sec         23.55 sec       368
Generator Interface            133.49 sec        114.40 sec       184
Canonicalization                21.27 sec         14.19 sec       366
Reweighting                      3.18 sec          0.90 sec       100
Strategy Application          1028.63 sec        943.50 sec    130860
Recovery Strategies             18.43 sec         13.61 sec        39
Resolution Overhead            677.61 sec        598.84 sec        98
World Model                    205.57 sec        227.71 sec     14725
Ontology                        24.72 sec         18.47 sec      1725
Candidate Management           149.84 sec        140.24 sec       366
Corpus Access                    0.00 sec          0.00 sec         1
Garbage Collection            1081.55 sec        106.53 sec         8
User Interface                2280.36 sec        369.91 sec      1121
Other                           80.74 sec         59.50 sec         1
---------------------------------------------------------------------
        Total                 5742.23 sec       2631.46 sec
```

**Figure 10-15:** Execution Profile for a Test Run

1. Parser Interface
   The parser interface retrieves a parse either from a parser running in a separate process, or from a file as for these tests.

2. Generator Interface
   The generator interface either transmits the resolved parse to the generator running in a separate process, or stores it in a file as for these tests.

3. Canonicalization
   Before the parse can be used, it must be canonicalized, and the resolved parse must be decanonicalized before being output.

4. Reweighting
   The amount of time required to adjust the strategy weights after every resolution varies according to the reweighting method, and is listed under this heading.

5. Strategy Application
   The category counts the amount of time actually spent applying the resolution strategies to the candidates.

6. Recovery Strategies
   How much time was spent in attempting to determine additional candidates for a failed resolution or disambiguate an ambiguous resolution; most of the time for the Ask the User strategy is charged to User Interface below.

7. Resolution Overhead
   The overhead for resolving a demonstrative includes the time to set up the agenda, actually run the agenda, compute candidate scores, and perform administrative tasks such as updating strategy voting histories.

8. World Model
   Maintaining the world model and checking for world model relations requires the time listed under this category.

9. Ontology
   The Ontology category includes the time needed to load the ontology and perform various lookups on it.

10. Candidate Management
    How much time was spent removing old candidates from the pool and adding new candidates after every parse was processed.
11. Corpus Access
    This category counts the time spent accessing a corpus of sentences to be parsed, and is not applicable in this case.
12. Garbage Collection
    The time spent in the Lisp garbage collector.
13. User Interface
    The time spent displaying menus, status lines, prompts, etc., and waiting for user input.
14. Other
    Anything else not covered by one of the previous categories.

The initial implementation of the strategy applier was much slower due to overhead in the DIBBS (Device Independent BlackBoard System [30]) system, which was used to implement the agenda. On determining that the system was unacceptably slow, DIBBS was replaced with a simple agenda handler optimized for speed rather than the generalized agenda handling supported by DIBBS, and the DIBBS "units" were replaced with equivalent structures. Additionally, some frames which had been DIBBS units were changed into regular FrameKit frames. These changes resulted in a speedup of nearly one second of CPU time per strategy application. At that early stage, with only the simple, computationally inexpensive strategies implemented, this represented an order of magnitude speedup even though the entire system (except FrameKit and DIBBS, but including the new agenda handler) was interpreted rather than compiled. The speedup ratio would have been even greater if the system had been using compiled Lisp at the time.

Some additional speedup was achieved by replacing several key FrameKit functions with versions which do not perform demon checking. Demon checks in FrameKit are very expensive as every access to a frame involves an `is-a-p` call which can potentially involve many levels of inheritance before returning. In the earlier KBMT-89 system, an equivalent replacement resulted in an overall speedup by a factor of 2.5 in the Augmentor portion. Since this replacement was made in MASTER-D at the outset, the exact speedup from this change is not known.

At the end of development, a near-total replacement of FrameKit resulted in an overall speedup by approximately a factor of two over the tweaked FrameKit, and reduced garbage collections by a factor of five. The FrameKit replacement removed all of the generality provided by FrameKit which was not actually required for MASTER-D, such as facets and views, multiple inheritance types, and relations with automatically-maintained inverses. The replacement code provides a subset of FrameKit's interface and uses similar (but simplified) internal data structures. The reduction in garbage collections is due in part to the simplified data structure (there are two fewer levels of nested lists) and in part to the provision of a new function which returns the fillers of a slot without making a copy for those callers (almost all) who do not destructively modify the returned list of fillers.

## 10.5. Conclusions

The multiple-strategy approach is effective in resolving demonstrative references, and is improved with the addition of adaptive reweighting. As is shown in Figure 10-16, MASTER-D is more than twice as effective as a naive heuristic even without reweighting, and achieves a further eleven to eighteen percent increase in correct or partially correct resolutions with reweighting[10]. Even on demonstrative references which are not within the scope of the resolver's capabilities, the correct result (that there is no available referent) is frequently returned.

Although two of the three reweighting methods which were implemented were successful and increased the resolver's accuracy, the third method (weight by accuracy) did not work as anticipated and required modification. The negative strategy weights assigned to strategies with accuracies less than 50 percent caused wild swings in resolutions, particularly when the Recency Preference's weight became negative. After modification to return negative weights only for accuracies below 20 percent (which never occur on the test set used) and to compute the accuracies based only on those resolutions for which a strategy offered an opinion, the third method worked approximately as well as the other two.

Further work is warranted on determining an appropriate point at which to purge candidates. One of the demonstratives in the test corpus was not resolvable because its referent had already been removed from the candidate pool due to its age. A more flexible limit which takes factors such as salience and focus into account would likely reduce the incidence of such premature purging without unduly increasing the size of the candidate pool and slowing execution.

Not all of the questions raised in Section 9.3 have been answered, but the following answers are available:

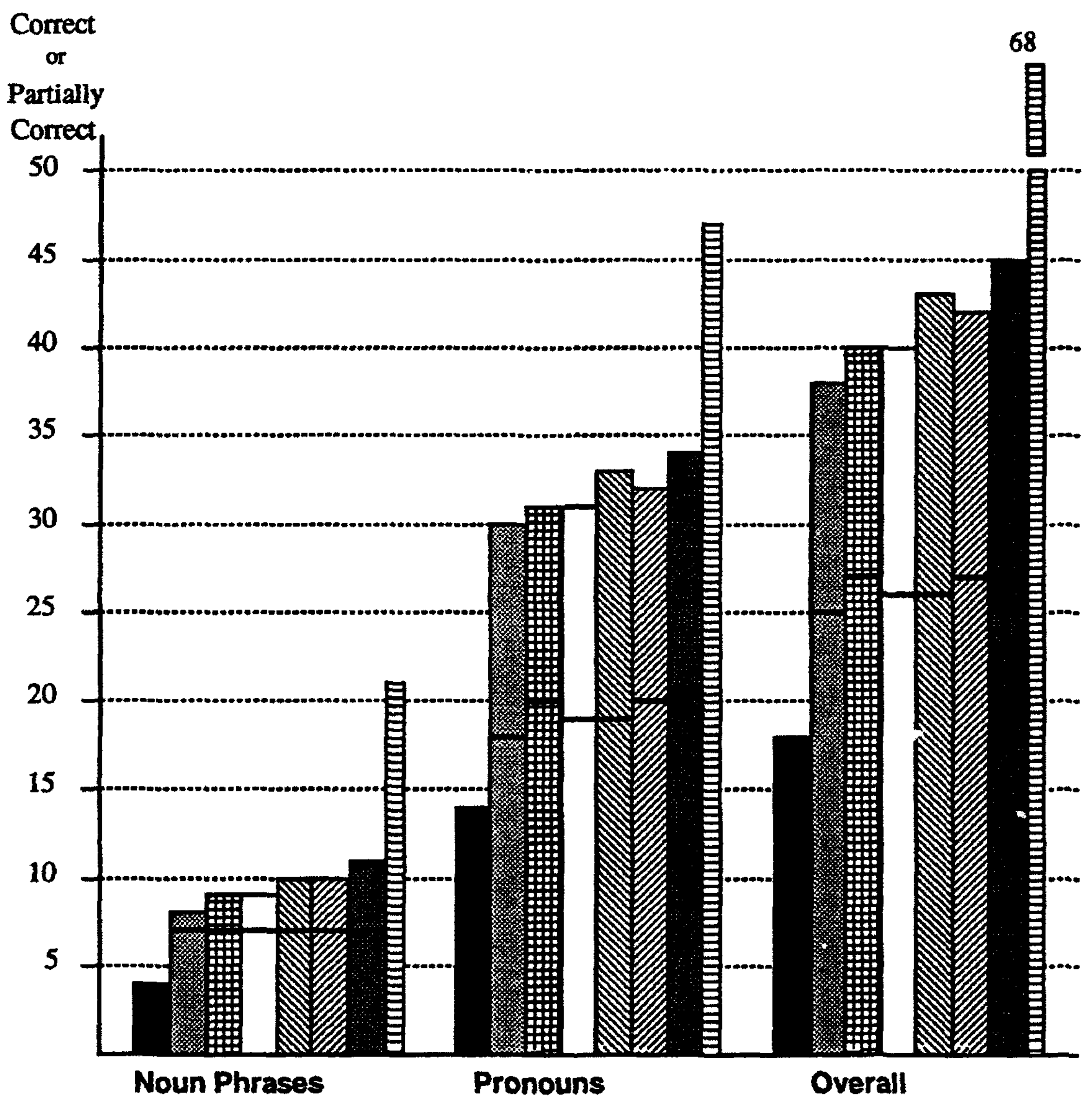
1. **How much does adaptive reweighting improve performance?**
   For the test case in this dissertation, the resolver was able to identify referents for 11 to 18 percent more demonstratives (depending on whether all partially-correct resolutions are counted) when adaptive reweighting was used.

2. **Which reweighting algorithm is optimal?**
   Fixed-percentage and incremental adjustment appear to be similar in efficacy; weight-by-accuracy proved to be detrimental to the system's performance in its original implementation, but performed similarly to the other two methods after modification. For this small test set (producing results which are by no means definitive), incremental adjustment performed best if total resolutions are counted and the corrected weight-by-accuracy strategy performed best if weighted correctness is examined. However, the differences between the three algorithms are negligible and will require much larger test sets to produce a significant difference.

3. **Is it better to reweight strategies consisting of multiple substrategies as a whole or to consider the substrategies as invidually-weightable strategies?**
   For the small test set used here, it is not clear which is better, as the two approaches performed nearly equally.

4. **Does the order in which strategies are applied affect accuracy?**

[10]Note that, unlike earlier sections of this chapter, partially-correct resolutions include all resolutions where the desired referent was one of the returned set of candidates (containing up to nine candidates) rather than just those where the returned set contained at most three candidates. This allows inclusion of resolutions on the less-supported candidate types such as lexical candidates.



Key

| | | |
|---|---|---|
| ■ Naive | ▦ Fixed-Percentage | ▧ No Reweight, Fixed-% Weights |
| ▩ No Reweighting | □ Incremental | ▨ No Reweight, Incr-Change Weights |
| ☰ Total Demonstratives within resolver's scope | | ■ Incremental, Second Run |

**Figure 10-16:** Summary of Correct Resolutions

Not for these test runs, which (in order to perform reweighting) applied all strategies to all candidates. The order can only have an effect if resolution is terminated before complete application of all strategies.

5. **How much does each resolution strategy contribute to the overall performance?**
As "vas shown in the previous chapter, Local Constraints and Reference Type dominate, both in final weight and in accuracy measures, while Recency plays a smaller but still significant role.

6. **Is there a threshold above which candidates can be accepted immediately?**

| | |
|---|---|
| Proximity: | 12 |
| Salience: | 21 |
| Both together: | 10 |
| Proximity: | 12 |
| Case-Role Persistence: | 16 |
| Both together: | 1 |
| Case-Role Persistence: | 16 |
| Salience: | 21 |
| Both together: | 1 |

**Figure 10-17:** Upward Reweightings on Various Strategies

This does not appear to be the case, although it seems likely that there is a *lower* bound below which a candidate can be completely discarded. Lack of such a bound in the current implementation leads to some resolutions which succeed (with an erroneous result) rather than invoking a recovery strategy to create new candidates.

7. **Are any of the strategies correlated?**
An accurate answer to this question is not available due to the sheer amount of data involved--even for the relatively small test set used, 14710 vote pairs must be analyzed for each pair of strategies for which correlation is to be computed. Some indication of correlation can be obtained, however, by examining the reweighting behavior. In examining Figure 10-17, it is clear that Proximity and Salience tend to be reweighted together, while neither of the other two pairings show such a tendency.

The system was only tested on a single example drawn from a single genre and domain, but there is no inherent dependence on genre or domain. Different genres of text would have different styles which may lead to different strategy weights, but will not require any modifications to the resolver. Similarly, different domains would require an expanded or modified ontology which covers the entities discussed in each new domain, but no modification of MASTER-D proper. Therefore, MASTER-D should be effective across all genres of writing and all domains.

# Chapter 11
# Contributions

The work described in this dissertation makes the following contributions:

1. It shows the applicability of the multistrategy approach to demonstrative references.
2. It produced a working demonstrative resolver, MASTER-D.
3. It shows the relative effectiveness of a variety of resolution and reweighting strategies.
4. The MASTER-D system is a flexible implementation of the multistrategy approach which provides a test-bed for further research.

That the multistrategy approach is applicable to demonstrative references was shown by the results of the MASTER-D tests in Sections 10.2 and 10.5. The multistrategy approach was able to resolve more than twice as many demonstratives as a naive heuristic, and was further improved (by more than 10 percent) with the addition of adaptive reweighting. These results also show that MASTER-D is a working demonstrative resolver, and the relative effectiveness of multiple resolution and reweighting strategies.

As will be discussed more thoroughly in the next chapter, MASTER-D provides a flexible platform on which to base further research. It can fairly easily be extended to provide additional coverage of demonstratives, or modified to cover other linguistic phenomena such as pronominal anaphora. In either case, the flexibility of the system allows for a variety of experiments to be performed, comparing various approaches to determine the best one.

Even without the possible extensions mentioned in the next chapter, MASTER-D is a highly flexible and configurable system. Areas that may be customized at run-time include

1. Ordering of strategy applications
2. Substrategy handling
3. Threshold for selection of best candidates
4. Delay before reweighting begins
5. Reweighting method
6. Accuracy computation
7. Decay functions

# Chapter 12
# Possible Enhancements

This chapter covers some of the enhancements which could be made to the basic system. These enhancements fall into the general classes of broadening coverage (to other linguistic phenomena as well as additional types of demonstratives) and improving accuracy. Coverage could be broadened to intrasentential demonstratives; anaphoric pronouns and noun phrases; or idiomatic, metaphoric, and epithetic demonstratives. Accuracy could be improved by additional strategies, better strategy weighting, and learning algorithms. Finally, speed can be greatly enhanced by parallelizing the application of resolution strategies.

## 12.1. Intrasentential Demonstrative References

The current implementation of MASTER-D only treats intersentential demonstrative references, and does not attempt to determine whether the *actual referent* lies in the same sentence as the demonstrative. Handling intrasentential references would require changes to several parts of the system; at a minimum, the candidate determiner and accessibility constraint strategy must be updated to deal with the additional candidates available in the current sentence. The latter might use something similar to Reinhart's c-command relation [37] to determine whether a candidate in the current sentence could be referenced by the demonstrative. The distance computation used by the Recency Preference strategy would also need to be adjusted to allow for candidates to be in the same sentence as the demonstrative.

## 12.2. Anaphoric Pronouns and NPs

Since the basic mechanism and a number of the strategies will be applicable to anaphoric pronouns/NPs as well as demonstratives, it should be fairly straightforward to enhance the system to resolve anaphoric references as well as demonstrative references. This would allow comparing/contrasting demonstrative and anaphoric references. There is also the exciting prospect that the addition of anaphoric pronoun resolution will have a synergistic effect by providing more information about candidate referents for demonstratives and vice versa.

## 12.3. Additional Strategies

Due to the modular nature of the resolver, it will be possible to add new strategies by simply adding another entry to the list of strategies to invoke (see Section 8.5 for details on adding strategies). The strategies are assumed to be mutually independent; new strategies which satisfy this assumption will not affect the operation of existing strategies other than by changing the relative weights of strategies.

Examples of such additional strategies include the treatment of idiomatic, metaphoric, and epithetic demonstratives. Idiomatic demonstratives would require a database of idioms which include demonstratives, and how the antecedent (if any) relates to the demonstrative. Metaphoric demonstratives would require a metaphor-handling module. Epithetic demonstratives such as *that bastard* could be handled with some form of user model showing the speaker's attitudes and beliefs about the various entities in the discourse, coupled with a knowledge base of the connotations of commonly-used epithets.

Better accuracy on demonstratives already covered by the resolver might be achieved by adapting strategies such as Stock of Shared Knowledge [23] to demonstrative pronouns and NPs. Because the Stock of Shared Knowledge approach uses multiple factors which must be individually weighted, it is possible that the system's reweighting mechanism could be used to optimize those weights as well as the strategy weights it already optimizes. One of the factors, Linear Distance, could be omitted when adding SSK to MASTER-D, since distance is already present as the Recency Preference strategy.

## 12.4. Other Reweighting Methods

The reweighting methods described earlier are all fairly simplistic. A more sophisticated reweighting method may lead to better or faster convergence in the strategy weights. Better optimization might also be obtained by permitting multiple weights per strategy, with the appropriate weight selected by some criterion.

For instance, one may globally reweight the strategies, rather than reweighting strategies individually. The global reweighting can take into account the change in a strategy's weight as a result of the rescaling operation since it determines the new weights of all the other strategies at the same time.

Another more sophisticated reweighting method acts not only on the strategy weights, but also on the decay functions and/or distance functions. For example, a particular strategy may be more effective than expected for candidates which are lexically distant from the demonstrative, in which case it would be desirable to reduce the effect of the decay function. This might be accomplished by reducing the "speed" parameter to the decay function or even switching to another decay function. It might also be accomplished by switching to a different distance function which reports a smaller distance, thus reducing the effects of textual separation between the reference and candidate antecedents.

Multiple weights per strategy might be useful if the optimum weights differ for different types of references or different positions of the demonstrative within a parse. For instance, subject demonstratives might require different weights than object demonstratives. While it does not seem likely that there will be a large variation in weights under different circumstances, even the small differences one would expect can have a significant impact on overall performance, making this a worthwhile course of investigation.

## 12.5. Learning

MASTER-D can already be said to contain a rudimentary form of machine learning in its automatic reweighter, which implements a simple "learning by parameter optimization" technique. It seems likely that adding a more powerful learning algorithm will imprcve the system's performance by "remembering" the patterns in the input. Learning might take the form of chunking, as in the SOAR system, or memoization.

Chunking should be able to improve performance by retrieving the relationship of the antecedent to the demonstrative which was determined the previous time that an equivalent construction was encountered. Memoization would be able to do the same thing provided that the memos are generalized prior to storage; without the generalization step, memoization would only be effective when the identical input is encountered again.

## 12.6. Parallelization

As was mentioned at the beginning of Chapter 3, the multistrategy approach offers easy parallelization. Resolutions can be greatly speeded by processing candidates in parallel, in which case the bottleneck becomes the speed at which the best score is selected. With sufficient processors (for example, on a Connection Machine), it would be possible to run not only candidates in parallel, but run the resolution strategies for each candidate in parallel as well.

The current serial implementation already contains some pseudo-parallelism in the form of the agenda handler, which dispatches instances of the strategy applier one at a time. With multiple processors, the agenda handler could activate multiple instances at once, such as in Figure 12-1. One processor executes the agenda handler and demonstrative resolver (which primes the agenda and selects the best score on completion of the agenda), while the remaining processors execute copies of the strategy applier. Parallelizing the strategy applications for a single candidate would require additional but relatively minor modifications to the strategy applier; one approach would be to add all of the strategy applications to the agenda as individual items when a particular candidate initially reaches the head of the agenda. Alternatively, if sufficient processors are available (for example, on a Connection Machine), the candidates could be parceled out to a set of processors which in turn assign the individual strategy applications to other processors and tally the results (see Figure 12-2).

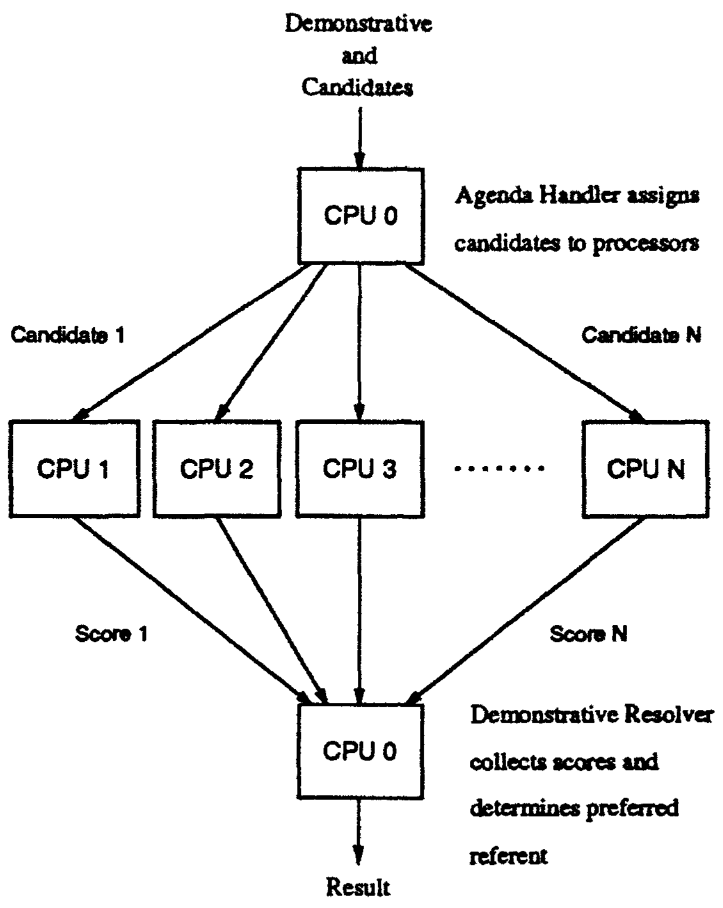


**Figure 12-1: A Parallelized Implementation**

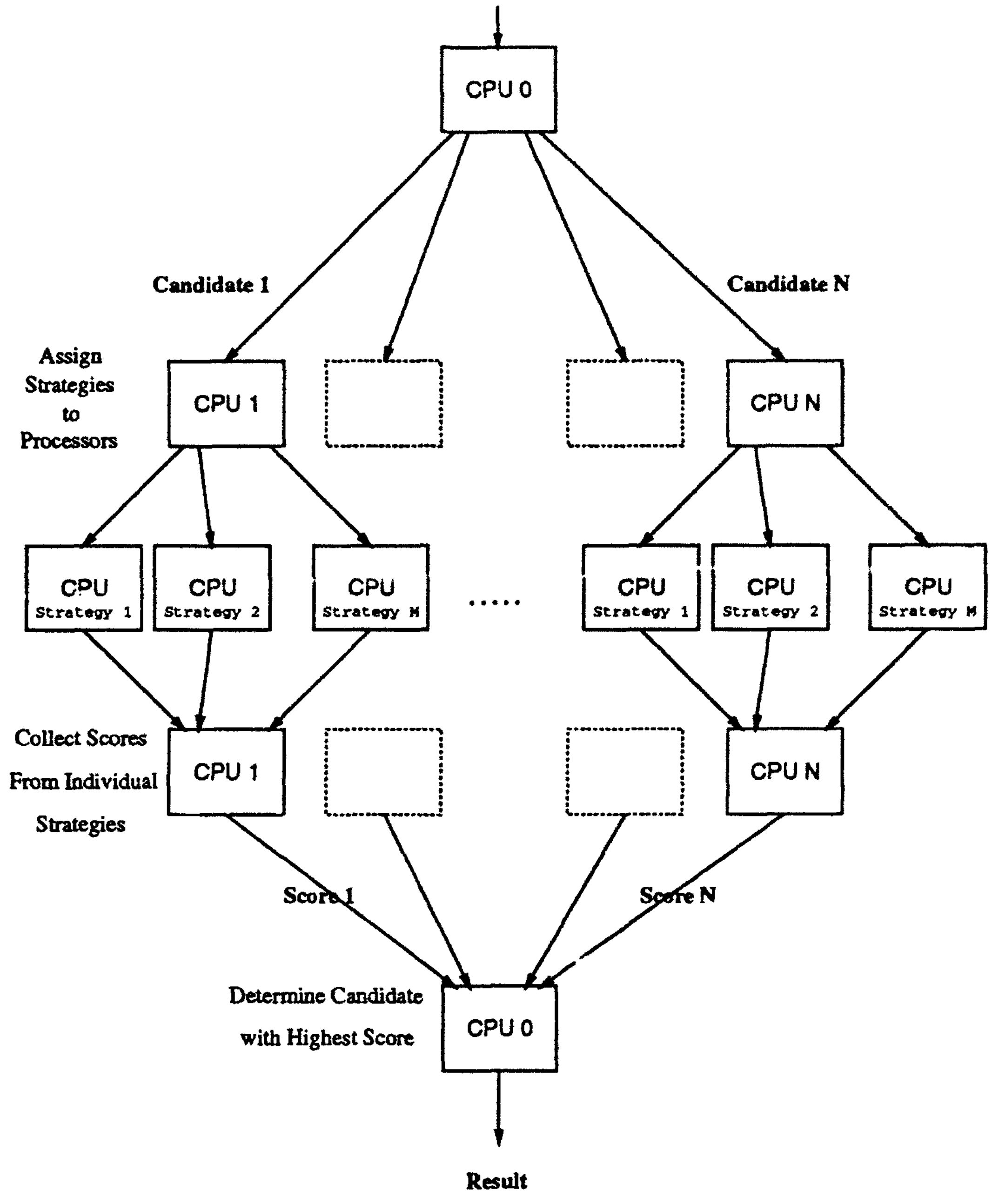


**Figure 12-2:** A Massively Parallel Implementation

# Appendix A
# Annotated Trace

The resolution process is illustrated through a sequence of screen dumps showing the displays seen by the user and an annotated version of the trace file produced during the resolution. Since a trace can easily amount to several thousand lines per sentence when tracing is set to maximum verbosity, the trace file shown here was created at level 7 of 30, and certain repetitive sections have been elided. In addition, portions of the trace have been reformatted to fit on the page and to improve the readability of the displayed parses.

Figure A-5 shows the general layout of the screen. Menus and dialogs requesting input from the user are shown in the upper left corner; in this figure, MASTER-D has detected that the file which is to contain the resolved parses already exists and gives the user the option of overwriting the file or appending the new output. The three windows in the center of the display are the optional monitoring windows. On the left, the current accuracies and weights of the resolution strategies are displayed and updated after every resolution. The upper right window displays the current item on the agenda, while the lower right window displays the five candidates with the highest scores after each strategy application. Finally, the bottom line of the display (not seen in Figure A-5 because it is blank, but shown in Figures A-2 and A-6, among others) shows the system's current processing state. This status line is actually in reverse video, which could not be reproduced here.

The trace and screen dumps in this appendix show the processing of a file containing parses for five sentences (listed in Appendix D) containing a total of three demonstratives. The screen dumps illustrate various points of interest from the loading of the system through completion of processing on the file.

The first screen dump (Figure A-1) shows MASTER-D being loaded. No tracing output is generated until after MASTER-D loads its configuration (Figure A-2) and has been set up by the user (Figure A-3).

The trace begins with MASTER-D announcing that it is starting, and showing the initialization of various components (which, if any, are reported in the trace file depends on the tracing level). As part of its initialization for this example, it opens an output file to contain the resolved parses. Since the file already exists, the user is prompted whether the file should be overwritten or have new data appended (Figure A-5).

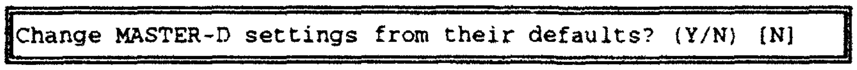
```
========================================
MASTER-D started at 7:10:35 on 5-07-1993
Trace Level 7

(init-correctness)
(init-statistics)
```

```
; Loading stuff from #<Stream for file "/usr1/ralf/src/md/master-d.lisp">.

;;;
;;;  The FrameKit Knowledge Representation System
;;;
;;; FrameKit 2.0 (01-Feb-88) Copyright (c) 1985, 1988
;;; Carnegie Mellon University - All Rights Reserved
;;;
;;; Stripped-down version for MASTER-D by Ralf Brown, 1993..


;;;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;;+                                                      +
;;;+                     MASTER-D                         +
;;;+    Multiple Adaptively-weighted STratEgies for       +
;;;+              Resolving Demonstratives                +
;;;+              (version 1.03, 4/29/93)                 +
;;;+                                                      +
;;;+                   by Ralf Brown                      +
;;;+                                                      +
;;;+  Copyright (c) 1993 Ralf Brown.  All Rights Reserved +
;;;+                                                      +
;;;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;; [loading from /usr/ralf/src/md/]
;;; ....
```

**Figure A-1: Loading MASTER-D**

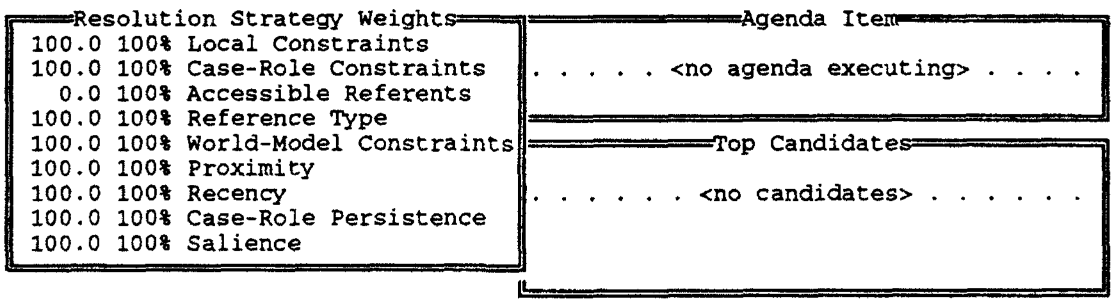
```
[loading settings from demo.cfg]
```

**Figure A-2: MASTER-D loading its configuration**



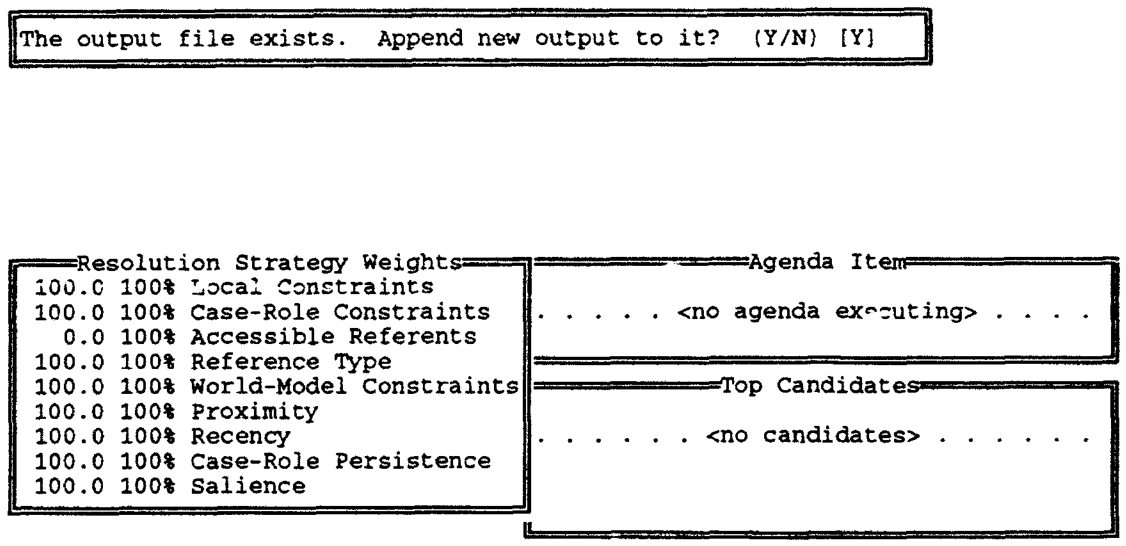
```
Change MASTER-D settings from their defaults? (Y/N) [N]
```

**Figure A-3: Opportunity to change setttings**



```
Resolution Strategy Weights                Agenda Item
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints       . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints               Top Candidates
 100.0 100% Proximity
 100.0 100% Recency                     . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience
```

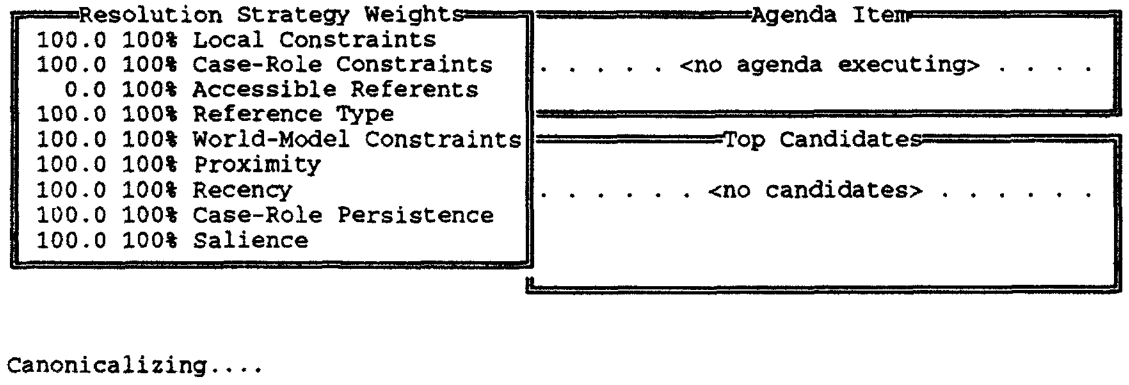
**Figure A-4: MASTER-D initialized and starting**



```
The output file exists.  Append new output to it?  (Y/N) [Y]
```

```
Resolution Strategy Weights                     Agenda Item
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints       . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints                 Top Candidates
 100.0 100% Proximity
 100.0 100% Recency                     . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience
```

**Figure A-5: Option to overwrite existing output file**



```
Resolution Strategy Weights                     Agenda Item
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints       . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints                 Top Candidates
 100.0 100% Proximity
 100.0 100% Recency                     . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience

Canonicalizing....
```

**Figure A-6: Canonicalizing the first parse**

```
==Resolution Strategy Weights==  ================Agenda Item================
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints  . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ==============Top Candidates==============
 100.0 100% Proximity
 100.0 100% Recency                . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience

Processing "The comment is made in this document 'Against UI Copyright':"
```

**Figure A-7: Processing the first sentence**

```
=============================Verification=============================
MASTER-D has selected
   <<<nothing>>>
as the referent for the demonstrative
   Demonstrative <this document>
in the sentence
"The comment is made in this document 'Against UI Copyright':"
Is this correct?
 ==> Yes / No [Yes]

==Resolution Strategy Weights==  ================Agenda Item================
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints  . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ==============Top Candidates==============
 100.0 100% Proximity
 100.0 100% Recency                . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience

Updating internal data....
```

**Figure A-8: Verifying first resolution**

```
(init-agenda)
(init-corpus)
(init-candidate-pool)
(init-ontology)
(load-ontology "md.ont")
(init-world-model)
(init-discourse-model)
(init-reweighting)
(init-generator)
```

*Figure A-5 is displayed at this point*

```
(init-parser)
(open-parse-file "/usr/ralf/src/md/test.parses")
(load-corpus NIL)
```

Next, we begin processing the first sentence by reading the parse from the data file and canonicalizing it into MASTER-D's internal representation:

```
(get-parse-from-file) --> (*MAKE
                            (TENSE PAST)
                            (THEME
                             (*O-COMMENT
                              (NUMBER SINGULAR)
                              (REFERENCE DEFINITE)))
                            (LOCATION
                             (*DOCUMENT
                              (REFERENCE DEMONSTRATIVE)
                              (NUMBER SINGULAR)
                              (DISTANCE NEAR)
                              (NAME "Against UI Copyright")
                              (INPUT "this document")))
                            (SENTENCE
                             "The comment is made in this document
                                 'Against UI Copyright':")
                            ($ATTRIBUTES
                             (PARA 34)
                             (SPEAKER "Frank Ingari")))
(get-parse) --> "The comment is made in this document
                 'Against UI Copyright':" [PARSE-1]
>>> processing >>> "The comment is made in this document
                    'Against UI Copyright':"
(canonicalize (*MAKE
               (TENSE PAST)
               (THEME
                (*O-COMMENT
                 (NUMBER SINGULAR)
                 (REFERENCE DEFINITE)))
               (LOCATION
                (*DOCUMENT
                 (REFERENCE DEMONSTRATIVE)
                 (NUMBER SINGULAR)
                 (DISTANCE NEAR)
                 (NAME "Against UI Copyright")
                 (INPUT "this document")))
               (SENTENCE
                "The comment is made in this document
                   'Against UI Copyright':")
               ($ATTRIBUTES
                (PARA 34)
                (SPEAKER "Frank Ingari"))))
```

*Figure A-6 is displayed at this point*

```
(canonicalize-ILT-embedded)
Canonicalized parse is:
(ROLE36
   (PARENT-SLOT LOCATION)
   (PARENT-FRAME PROPOSITION33)
   ($ID TAG37)
   (INPUT "this document")
   (NAME "Against UI Copyright")
   (DISTANCE NEAR)
   (NUMBER SINGULAR)
   (REFERENCE DEMONSTRATIVE)
   (PROPOSITION PROPOSITION33)
   (IS-A *DOCUMENT)
   (FRAME-MAKER MASTER-D)
)
(ROLE34
   (PARENT-SLOT THEME)
   (PARENT-FRAME PROPOSITION33)
   ($ID TAG35)
   (REFERENCE DEFINITE)
   (NUMBER SINGULAR)
   (PROPOSITION PROPOSITION33)
   (IS-A *O-COMMENT)
   (FRAME-MAKER MASTER-D)
)
(PROPOSITION33
   ($ID TAG38)
   ($ATTRIBUTES (PARA 34) (SPEAKER "Frank Ingari"))
   (SENTENCE
    "The comment is made in this document
     'Against UI Copyright':")
   (LOCATION ROLE36)
   (THEME ROLE34)
   (TENSE PAST)
   (PROPOSITION PROPOSITION33)
   (IS-A *MAKE)
   (FRAME-MAKER MASTER-D)
)
(canonicalize ...)
    --> PROPOSITION33
```

After canonicalization, MASTER-D scans the parse for demonstratives and finds ROLE36. It passes this to the demonstrative resolver, which on this first sentence has no candidates (the global candidate for the speaker Frank Ingari is not added until after the sentence is processed because this information was not available before reading the first parse). Since there are no candidates, there is no suitable candidate antecedent, and thus MASTER-D initiates the recovery strategies.

```
(master-d-resolver-main PARSE-1), 1 unresolved
*** resolving PARSE-1/ROLE36 ***
(resolve-demonstrative ROLE36 NIL)
(run-entire-agenda), 0 items on agenda
```

***Figure A-7 is displayed at this point***

```
(best-candidates (RESOL-2 ROLE36 0 0)) --> NIL
(initiate-recovery0 RESOL-2)
  applying recovery strategy "Extralinguistic Reference"
    new candidates: NIL
  applying recovery strategy "Metonyms"
    new candidates: NIL
  applying recovery strategy "Relax Constraints"
    new candidates: NIL
(initiate-recovery0 RESOL-2) exit
```

Since there were no previous sentences processed, and no candidates were added by the recovery strategies, MASTER-D selects NIL (i.e. no referent) as the desired antecedent. The next step is to ask the user to verify that the selection is correct (Figure A-8).

```
(verify-resolution "The comment is made in this document 'Against UI
                           Copyright':" PROPOSITION33 ROLE36 ...)
(verify-resolution ... ROLE36) --> T
```

After each resolution is complete and verified, the strategies are reweighted, and then rescaled so that the total of the weight's absolute values remains constant. Once the strategies have been normalized, any outdated information about how strategies voted is deleted. When the strategy weight window is enabled, as it is for this example, the display is also updated at this time (Figure A-9). In this case, the strategy weights and accuracies remain unchanged because the first resolution effectively did not occur--there were no candidates and the correct candidate was not in the candidate pool. Due to an interaction between this situation and adjustments which are made to avoid penalizing strategies for demonstratives which are beyond the scope of the system, the strategy accuracies will appear to be greater than 100 percent later in this example.

Since there were no candidates on which to vote, the strategies were never applied and will thus not be reweighted for this demonstrative. Thus, reweighting, rescaling, and voting-history adjustments are skipped this time.

The final phase of processing a parse is to update the candidate pool. This occurs in two steps: removal of candidates which have become too old to be considered potential antecedents, followed by addition of candidates made available by the parse which was just processed. In this case, no candidates are purged since there are none in the pool, and five new candidates (ROLE34, GLOBAL39, LEXICAL40, LEXICAL41, DISCOURSE42) are added.

```
(resolve-demonstrative ROLE36 ...) --> (NIL)
*** completing resolution of PARSE-1 ***
Referents found: ((ROLE36 . NIL))
(update-candidate-pool PROPOSITION33 ...)
(purge-candidate-pool) --> 0 candidates remaining
(nouns-in-parse PROPOSITION33) --> (ROLE34)
(propositions-in-parse PROPOSITION33) --> NIL
(properties-in-parse PROPOSITION33) --> NIL
(lexical-candidates-in-parse PROPOSITION33) --> (LEXICAL41 LEXICAL40)
(discourse-candidates-in-parse PROPOSITION33) --> (DISCOURSE42)
(update-candidate-pool PROPOSITION33 ...) done, 5 candidates
```

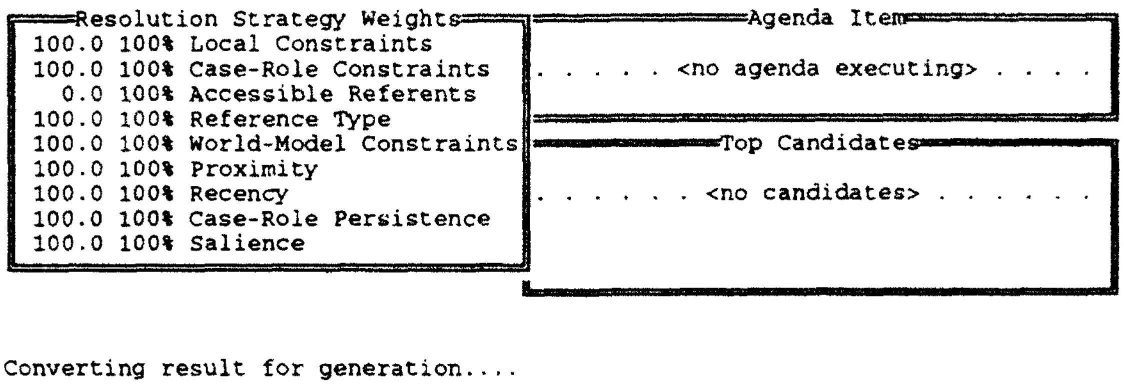
When creating a trace file, performance statistics are generated after this final step of processing. These statistics show how long the resolution took and the updated weights of the resolution strategies. In addition to the strategy weights, a summary of reweighting and the computed accuracy for each strategy are shown. The reweighting summary indicates how many times the strategy's weight was increased or decreased. For this sentence, the computed accuracy is -99.99 percent because the strategies have never been applied; in later statistics, some of the strategies remain at -99.99 percent because they have never given an opinion on any candidates, and thus their correctness scores are 0 points out of a possible 0 points. The rescaling factor shown at the end of the statistics indicates how much the weights had to be rescaled to keep the total constant, and is the product of the individual rescaling factors applied. An individual rescaling factor is the amount by which the strategy weights were multiplied to return the sum of weights to its original value.

```
<<< processing complete <<<
CPU time used:      2.48 seconds
Elapsed time:       7.87 seconds

Performance Summary
===================
Sentence Number 1
1 demonstrative
   ROLE36 (Demonstrative <this document>)
       --> NIL (<<<nothing>>>)
                                  Initial   Current   Reweights
      Strategy                    Weight    Weight      Up/Down  Accuracy
---------------------------------------------------------------
Local Constraints                 100.000   100.000    0/   0    -99.99%
Case-Role Constraints             100.000   100.000    0/   0    -99.99%
Accessible Referents                0.000     0.000    0/   0    -99.99%
Reference Type                    100.000   100.000    0/   0    -99.99%
World-Model Constraints           100.000   100.000    0/   0    -99.99%
Proximity                         100.000   100.000    0/   0    -99.99%
Recency                           100.000   100.000    0/   0    -99.99%
Case-Role Persistence             100.000   100.000    0/   0    -99.99%
Salience                          100.000   100.000    0/   0    -99.99%
---------------------------------------------------------------
Total Rescaling Factor: 1.00
===================
```

MASTER-D can now feed the processed parse to the generator or output file. Note the additional slot REFERENT under LOCATION in the processed parse, indicating that there was no referent in the prior text.

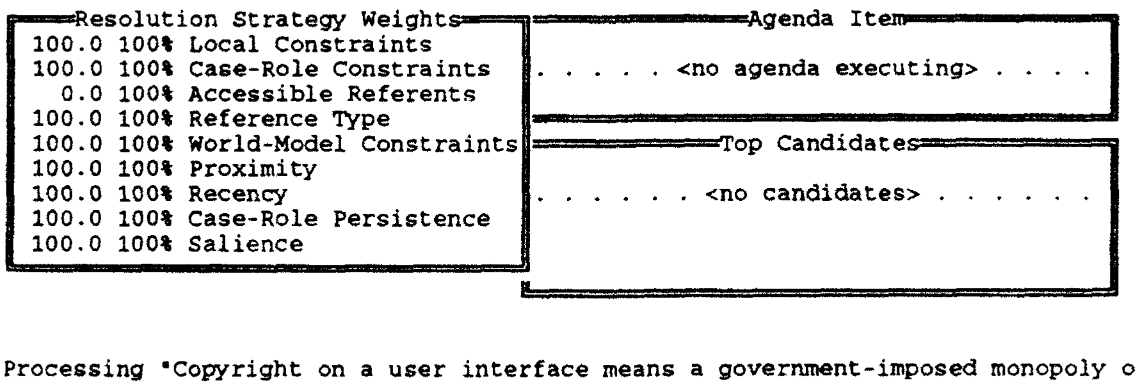
```
(generate (*MAKE
           (TENSE PAST)
           (THEME
            (*O-COMMENT
             (NUMBER SINGULAR)
             (REFERENCE DEFINITE)
             ($ID TAG35)))
           (LOCATION
            (*DOCUMENT
             (REFERENCE DEMONSTRATIVE)
             (NUMBER SINGULAR)
             (DISTANCE NEAR)
             (NAME "Against UI Copyright")
             (INPUT "this document")
             ($ID TAG37)
             (REFERENT NIL)))
           (SENTENCE
            "The comment is made in this document 'Against UI Copyright':")
           ($ID TAG38)))
[generating to file #<Stream for file "/usr1/ralf/src/md/md.output">]
```

The second sentence is processed in much the same way, but does not trigger resolution as it contains no demonstratives (Figure A-10). Therefore, the processing steps are retrieving the parse from the data file, canonicalizing it, updating the candidate pool, and storing the parse in the output file.



```
==Resolution Strategy Weights==  ==========Agenda Item==========
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints   . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ==========Top Candidates==========
 100.0 100% Proximity
 100.0 100% Recency                 . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience

Converting result for generation....
```

Figure A-9: Strategies Reweighted



```
==Resolution Strategy Weights==  ==========Agenda Item==========
 100.0 100% Local Constraints
 100.0 100% Case-Role Constraints   . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ==========Top Candidates==========
 100.0 100% Proximity
 100.0 100% Recency                 . . . . . . <no candidates> . . . . . .
 100.0 100% Case-Role Persistence
 100.0 100% Salience

Processing "Copyright on a user interface means a government-imposed monopoly o
```

Figure A-10: Processing second sentence

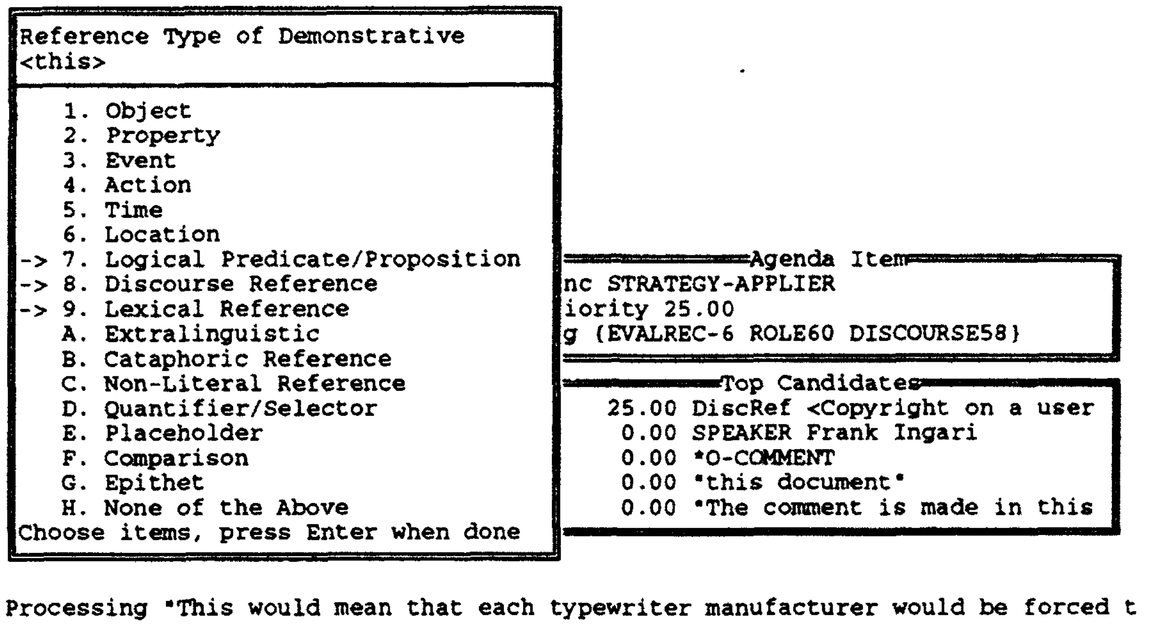
```
(get-parse-from-file) --> (*MEAN
                            (TENSE PRESENT)
                            (NUMBER SINGULAR)
                            (THEME
                             (*O-COPYRIGHT
                              (NUMBER SINGULAR)
                              (REFERENCE DEFINITE)
                              (OBJECT
                               (*USER-INTERFACE
                                (REFERENCE INDEFINITE
                                (NUMBER SINGULAR)))
                              (INPUT
                               "copyright on a user interface")))
                            (OBJECT
                             (*MONOPOLY
                              (NUMBER SINGULAR)
                              (REFERENCE INDEFINITE)
                              (CREATOR *GOVERNMENT)
                              (OBJECT
                               (*USE-OF-THING
                                (REFERENCE POSSESSIVE)
                                (PERSON 3)
                                (NUMBER SINGULAR)
                                (ROOT "its use")))
                              (INPUT
                               "a government-imposed monopoly on its use")))
                            (SENTENCE
                             "Copyright on a user interface means a
                              government-imposed monopoly on its use.")
                            ($ATTRIBUTES (PARA 35)))
(get-parse) --> "Copyright on a user interface means a government-imposed
                 monopoly on its use." [PARSE-3]
>>> processing >>> "Copyright on a user interface means a government-imposed
                    monopoly on its use."
Canonicalized parse is:
(ROLE49
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME ROLE48)
   ($ID TAG50)
   (ROOT "its use")
   (NUMBER SINGULAR)
   (PERSON 3)
   (REFERENCE POSSESSIVE)
   (PROPOSITION PROPOSITION43)
   (IS-A *USE-OF-THING)
   (FRAME-MAKER MASTER-D)
)
(ROLE48
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME PROPOSITION43)
   ($ID TAG51)
   (INPUT "a government-imposed monopoly on its use")
   (OBJECT ROLE49)
   (CREATOR *GOVERNMENT)
   (REFERENCE INDEFINITE)
   (NUMBER SINGULAR)
   (PROPOSITION PROPOSITION43)
   (IS-A *MONOPOLY)
   (FRAME-MAKER MASTER-D)
)
```

```
(ROLE45
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME ROLE44)
   ($ID TAG46)
   (NUMBER SINGULAR)
   (REFERENCE INDEFINITE)
   (PROPOSITION PROPOSITION43)
   (IS-A *USER-INTERFACE)
   (FRAME-MAKER MASTER-D)
)
(ROLE44
   (PARENT-SLOT THEME)
   (PARENT-FRAME PROPOSITION43)
   ($ID TAG47)
   (INPUT "copyright on a user interface")
   (OBJECT ROLE45)
   (REFERENCE DEFINITE)
   (NUMBER SINGULAR)
   (PROPOSITION PROPOSITION43)
   (IS-A *O-COPYRIGHT)
   (FRAME-MAKER MASTER-D)
)
(PROPOSITION43
   ($ID TAG52)
   ($ATTRIBUTES (PARA 35))
   (SENTENCE
    "Copyright on a user interface means a government-imposed monopoly on
     its use.")
   (OBJECT ROLE48)
   (THEME ROLE44)
   (NUMBER SINGULAR)
   (TENSE PRESENT)
   (PROPOSITION PROPOSITION43)
   (IS-A *MEAN)
   (FRAME-MAKER MASTER-D)
)
(canonicalize ...)
    --> PROPOSITION43
(master-d-resolver-main PARSE-3), 0 unresolved
*** completing resolution of PARSE-3 ***
(update-candidate-pool PROPOSITION43 ...)
(purge-candidate-pool) --> 5 candidates remaining
(nouns-in-parse PROPOSITION43) --> (ROLE44 ROLE48 ROLE45 ROLE49)
(propositions-in-parse PROPOSITION43) --> (PROPOS53)
(properties-in-parse PROPOSITION43) --> NIL
(lexical-candidates-in-parse PROPOSITION43) --> (LEXICAL57
                                                 LEXICAL56
                                                 LEXICAL55
                                                 LEXICAL54)
(discourse-candidates-in-parse PROPOSITION43) --> (DISCOURSE58)
(update-candidate-pool PROPOSITION43 ...) done, 15 candidates
<<< processing complete <<<
CPU time used:     1.81 seconds
Elapsed time:      1.95 seconds

Performance Summary
===================
Sentence Number 2
0 demonstratives
```

```
                                        Initial    Current    Reweights
     Strategy                           Weight     Weight       Up/Down    Accuracy
-------------------------------------------------------------------------------
Local Constraints                        100.000    100.000       0/   0     -99.99%
Case-Role Constraints                    100.000    100.000       0/   0     -99.99%
Accessible Referents                       0.000      0.000       0/   0     -99.99%
Reference Type                           100.000    100.000       0/   0     -99.99%
World-Model Constraints                  100.000    100.000       0/   0     -99.99%
Proximity                                100.000    100.000       0/   0     -99.99%
Recency                                  100.000    100.000       0/   0     -99.99%
Case-Role Persistence                    100.000    100.000       0/   0     -99.99%
Salience                                 100.000    100.000       0/   0     -99.99%
-------------------------------------------------------------------------------
Total Rescaling Factor: 1.00
==================

(generate (*MEAN
           (TENSE PRESENT)
           (NUMBER SINGULAR)
           (THEME
            (*O-COPYRIGHT
             (NUMBER SINGULAR)
             (REFERENCE DEFINITE)
             (OBJECT
              (*USER-INTERFACE
               (REFERENCE INDEFINITE)
               (NUMBER SINGULAR)
               ($ID TAG46)))
             (INPUT "copyright on a user interface")
             ($ID TAG47)))
           (OBJECT
            (*MONOPOLY
             (NUMBER SINGULAR)
             (REFERENCE INDEFINITE)
             (CREATOR *GOVERNMENT)
             (OBJECT
              (*USE-OF-THING
               (REFERENCE POSSESSIVE)
               (PERSON 3)
               (NUMBER SINGULAR)
               (ROOT "its use")
               ($ID TAG50)))
             (INPUT "a government-imposed monopoly on its use")
             ($ID TAG51)))
           (SENTENCE
            "Copyright on a user interface means a government-imposed
             monopoly on its use.")
           ($ID TAG52)))
[generating to file #<Stream for file "/usr1/ralf/src/md/md.output">]
```

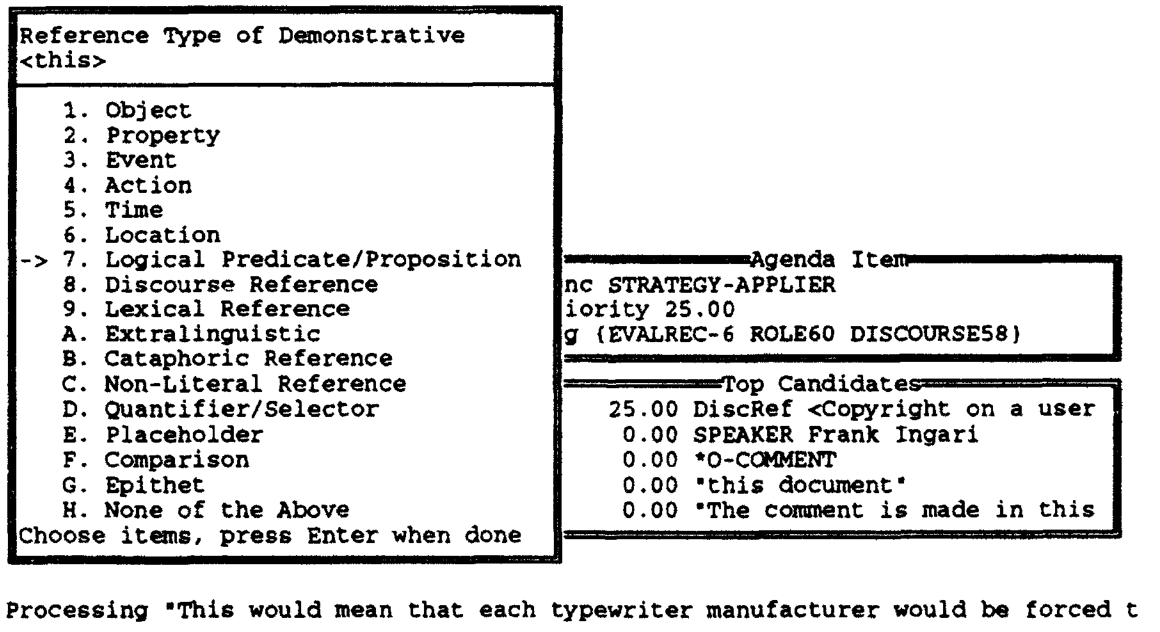
The third sentence becomes interesting, as there are now several candidates in the candidate pool and a demonstrative to be resolved. Again, processing begins by reading the parse and canonicalizing it.

```
>>> processing >>> "This would mean that each typewriter manufacturer
                    would be forced to arrange the keys differently."
Canonicalized parse is:
(ROLE68
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME ROLE67)
   ($ID TAG69)
   (INPUT "keys")
   (REFERENCE DEFINITE)
   (NUMBER PLURAL)
   (PROPOSITION PROPOSITION59)
   (IS-A *KEY)
   (FRAME-MAKER MASTER-D)
)
```



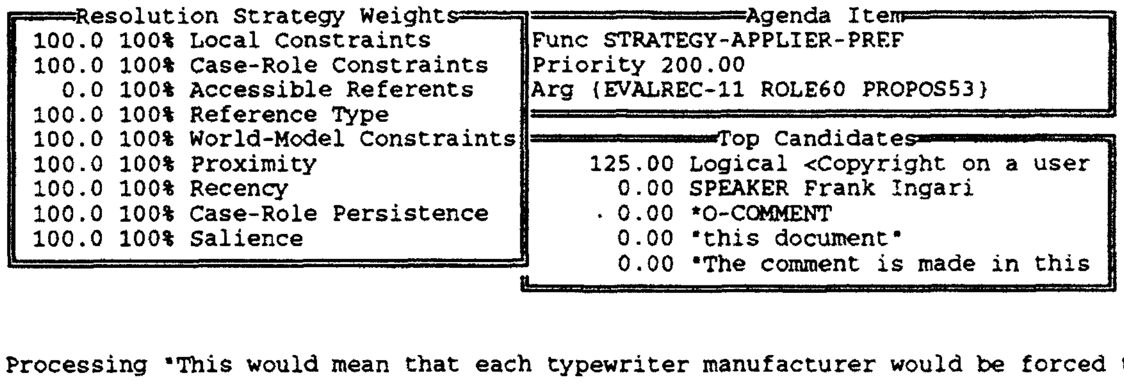
```
Reference Type of Demonstrative
<this>

   1. Object
   2. Property
   3. Event
   4. Action
   5. Time
   6. Location
-> 7. Logical Predicate/Proposition        ═══════════Agenda Item═══════════
-> 8. Discourse Reference                  nc STRATEGY-APPLIER
-> 9. Lexical Reference                    iority 25.00
   A. Extralinguistic                      g (EVALREC-6 ROLE60 DISCOURSE58)
   B. Cataphoric Reference
   C. Non-Literal Reference                ═════════Top Candidates══════════
   D. Quantifier/Selector                     25.00 DiscRef <Copyright on a user
   E. Placeholder                              0.00 SPEAKER Frank Ingari
   F. Comparison                               0.00 *O-COMMENT
   G. Epithet                                  0.00 'this document'
   H. None of the Above                        0.00 'The comment is made in this
Choose items, press Enter when done

Processing 'This would mean that each typewriter manufacturer would be forced t
```

**Figure A-11: Asking user for reference type in third sentence**



```
Reference Type of Demonstrative
<this>

   1. Object
   2. Property
   3. Event
   4. Action
   5. Time
   6. Location
-> 7. Logical Predicate/Proposition        ═══════════Agenda Item═══════════
   8. Discourse Reference                  nc STRATEGY-APPLIER
   9. Lexical Reference                    iority 25.00
   A. Extralinguistic                      g (EVALREC-6 ROLE60 DISCOURSE58)
   B. Cataphoric Reference
   C. Non-Literal Reference                ═════════Top Candidates══════════
   D. Quantifier/Selector                     25.00 DiscRef <Copyright on a user
   E. Placeholder                              0.00 SPEAKER Frank Ingari
   F. Comparison                               0.00 *O-COMMENT
   G. Epithet                                  0.00 'this document'
   H. None of the Above                        0.00 'The comment is made in this
Choose items, press Enter when done

Processing 'This would mean that each typewriter manufacturer would be forced t
```

**Figure A-12: User selected reference type**



```
══Resolution Strategy Weights══  ════════Agenda Item════════
 100.0 100% Local Constraints     Func STRATEGY-APPLIER-PREF
 100.0 100% Case-Role Constraints Priority 200.00
   0.0 100% Accessible Referents  Arg (EVALREC-11 ROLE60 PROPOS53)
 100.0 100% Reference Type
 100.0 100% World-Model Constraints ══════Top Candidates══════
 100.0 100% Proximity                 125.00 Logical <Copyright on a user
 100.0 100% Recency                     0.00 SPEAKER Frank Ingari
 100.0 100% Case-Role Persistence     · 0.00 *O-COMMENT
 100.0 100% Salience                    0.00 "this document"
                                        0.00 "The comment is made in this

Processing "This would mean that each typewriter manufacturer would be forced t
```

**Figure A-13: Processing third sentence**

```
(ROLE67
   (PARENT-SLOT ACTION)
   (PARENT-FRAME ROLE62)
   ($ID TAG70)
   (INPUT "arrange the keys differently")
   (MANNER *DIFFERENTLY)
   (OBJECT ROLE68)
   (PROPOSITION PROPOSITION59)
   (IS-A *ARRANGE)
   (FRAME-MAKER MASTER-D)
)
(ROLE64
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME ROLE63)
   ($ID TAG65)
   (INPUT "typewriter")
   (NUMBER SINGULAR)
   (REFERENCE INDEFINITE)
   (PROPOSITION PROPOSITION59)
   (IS-A *TYPEWRITER)
   (FRAME-MAKER MASTER-D)
)
```

```
(ROLE63
   (PARENT-SLOT THEME)
   (PARENT-FRAME ROLE62)
   ($ID TAG66)
   (INPUT "each typewriter manufacturer")
   (OBJECT ROLE64)
   (REFERENCE INDEFINITE)
   (NUMBER SINGULAR)
   (PROPOSITION PROPOSITION59)
   (IS-A *MANUFACTURER)
   (FRAME-MAKER MASTER-D)
)
(ROLE62
   (PARENT-SLOT OBJECT)
   (PARENT-FRAME PROPOSITION59)
   ($ID TAG71)
   (INPUT
    "each typewriter manufacturer would be forced to arrange
     the keys differently")
   (ACTION ROLE67)
   (THEME ROLE63)
   (MODALITY WOULD)
   (TENSE PRESENT)
   (PROPOSITION PROPOSITION59)
   (IS-A *FORCE-ACTION)
   (FRAME-MAKER MASTER-D)
)
(ROLE60
   (PARENT-SLOT THEME)
   (PARENT-FRAME PROPOSITION59)
   ($ID TAG61)
   (ROOT "this")
   (DISTANCE NEAR)
   (NUMBER SINGULAR)
   (PROPOSITION PROPOSITION59)
   (IS-A *DEMONSTRATIVE*)
   (FRAME-MAKER MASTER-D)
)
(PROPOSITION59
   ($ID TAG72)
   ($ATTRIBUTES (PARA 35))
   (SENTENCE
    "This would mean that each typewriter manufacturer would be forced
     to arrange the keys differently.")
   (OBJECT ROLE62)
   (THEME ROLE60)
   (MODALITY WOULD)
   (TENSE PRESENT)
   (PROPOSITION PROPOSITION59)
   (IS-A *MEAN)
   (FRAME-MAKER MASTER-D)
)
```

Having determined that ROLE160 is a demonstrative, MASTER-D invokes the demonstrative resolver with this demonstrative and a list of the candidate referents.

```
(master-d-resolver-main PARSE-4), 1 unresolved
*** resolving PARSE-4/ROLE60 ***
(resolve-demonstrative ROLE60 (DISCOURSE58    LEXICAL54
                               LEXICAL55      LEXICAL56
                               LEXICAL57      PROPOS53
                               ROLE49         ROLE45
                               ROLE48         ROLE44
                               DISCOURSE42    LEXICAL40
                               LEXICAL41      ROLE34
                               GLOBAL39))
```

The demonstrative resolver, in turn, queues an instance of the strategy applier for each candidate referent, and then invokes the agenda handler to process the strategy applier instances. For this example, the agenda handler ordered execution by current total score; as a result of the positive score by the first strategy, the first candidate to be processed continues processing until its score becomes negative. The two values shown as the return values of the `apply-strategy` function are the raw score returned by the strategy and the adjusted score (including any decay function).

```
(run-entire-agenda), 15 items on agenda
(strategy-applier (EVALREC-6 ROLE60 DISCOURSE58)) start
(apply-substrategies ROLE60 DISCOURSE58) --> 0.25
(apply-strategy "Local Constraints" (EVALREC-6 ROLE60 DISCOURSE58))
                    --> 0.25 [0.25]
(strategy-applier EVALREC-6) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-6 ROLE60 DISCOURSE58)) start
(apply-strategy "Case-Role Constraints" (EVALREC-6 ROLE60 DISCOURSE58))
                    --> 0.0 [0.0]
(strategy-applier EVALREC-6) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-6 ROLE60 DISCOURSE58)) start
(apply-strategy "Accessible Referents" (EVALREC-6 ROLE60 DISCOURSE58))
                    --> 0.0 [0.0]
(strategy-applier EVALREC-6) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-6 ROLE60 DISCOURSE58)) start
(apply-strategy "Reference Type" (EVALREC-6 ROLE60 DISCOURSE58))
                    --> INVALID [-1000000]
(strategy-applier EVALREC-6) --> (:REQUEUE -999975.0)
```

Now that the score is negative, `DISCOURSE58` is no longer the candidate with the highest priority, so the next candidate is processed. Again, the initial positive score keeps the candidate at the head of the queue until its score becomes negative.

```
(strategy-applier (EVALREC-7 ROLE60 LEXICAL54)) start
(apply-substrategies ROLE60 LEXICAL54) --> 0.25
(apply-strategy "Local Constraints" (EVALREC-7 ROLE60 LEXICAL54))
                    --> 0.25 [0.25]
(strategy-applier EVALREC-7) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-7 ROLE60 LEXICAL54)) start
(apply-strategy "Case-Role Constraints" (EVALREC-7 ROLE60 LEXICAL54))
                    --> 0.0 [0.0]
(strategy-applier EVALREC-7) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-7 ROLE60 LEXICAL54)) start
(apply-strategy "Accessible Referents" (EVALREC-7 ROLE60 LEXICAL54))
                    --> 0.0 [0.0]
(strategy-applier EVALREC-7) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-7 ROLE60 LEXICAL54)) start
(apply-strategy "Reference Type" (EVALREC-7 ROLE60 LEXICAL54))
                    --> INVALID [-1000000]
(strategy-applier EVALREC-7) --> (:REQUEUE -999975.0)
```

Now the third candidate referent moves to the head of the agenda, and processing continues similarly for another 40 lines of the trace, until a candidate passes the Reference Type constraint. This candidate, `PROPOS53`, remains at the head of the agenda until it has been completely processed.

```
(strategy-applier (EVALREC-11 ROLE60 PROPOS53)) start
(apply-substrategies ROLE60 PROPOS53) --> 0.25
(apply-strategy "Local Constraints" (EVALREC-11 ROLE60 PROPOS53))
                    --> 0.25 [0.25]
(strategy-applier EVALREC-11) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-11 ROLE60 PROPOS53)) start
(apply-strategy "Case-Role Constraints" (EVALREC-11 ROLE60 PROPOS53))
                    --> 0.0 [0.0]
(strategy-applier EVALREC-11) --> (:REQUEUE 25.0)
```

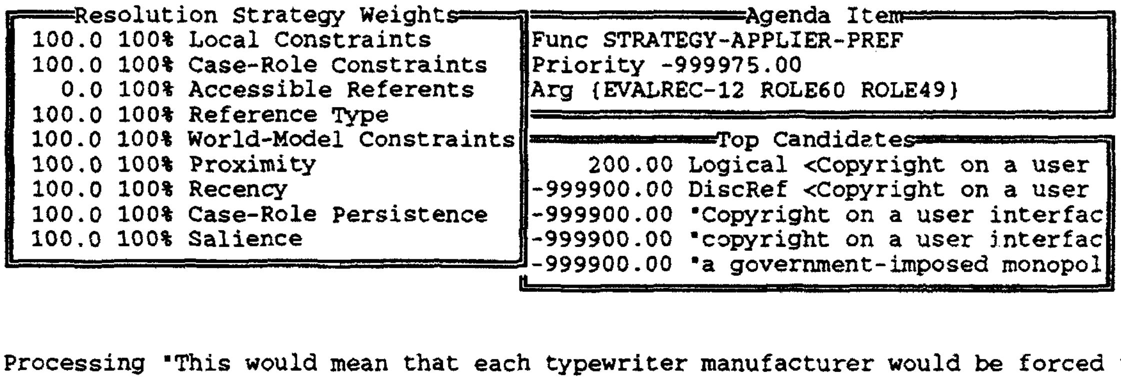
```
(strategy-applier (EVALREC-11 ROLE60 PROPOS53}) start
(apply-strategy 'Accessible Referents' (EVALREC-11 ROLE60 PROPOS53})
                   --> 0.0 [0.0]
(strategy-applier EVALREC-11) --> (:REQUEUE 25.0)
(strategy-applier (EVALREC-11 ROLE60 PROPOS53}) start
(apply-strategy 'Reference Type' (EVALREC-11 ROLE60 PROPOS53})
                   --> 1.0 [1.0]
(strategy-applier EVALREC-11) --> (:REQUEUE 125.0)
(strategy-applier (EVALREC-11 ROLE60 PROPOS53}) start
(apply-strategy 'World-Model Constraints' (EVALREC-11 ROLE60 PROPOS53})
                   --> 0.0 [0.0]
(strategy-applier EVALREC-11) --> (:REQUEUE 125.0)
(strategy-applier (EVALREC-11 ROLE60 PROPOS53}) start
(apply-strategy 'Proximity' (EVALREC-11 ROLE60 PROPOS53})
                   --> 0.0 [0.0]
(strategy-applier EVALREC-11) --> PENDING
(apply-strategy 'Recency' (EVALREC-11 ROLE60 PROPOS53}) --> 1.0 [0.75]
(apply-strategy 'Case-Role Persistence' (EVALREC-11 ROLE60 PROPOS53})
                   --> 0.0 [0.0]
(apply-substrategies ROLE60 PROPOS53) --> 0.0
(apply-strategy 'Salience' (EVALREC-11 ROLE60 PROPOS53}) --> 0.0 [0.0]
```

Processing continues similarly for all the other candidates, producing an additional 258 lines of trace data. After the candidates have been processed, the strategy applier determines the highest score and the threshold value above which candidates will be accepted (for this trace, the threshold is 98% of the highest score or 195.9998). Only a single candidate (PROPOS53) is accepted for this sentence, and it is returned as the referent of the demonstrative.

```
(best-candidates), hiscore = 195.9998
   {EVALREC-6 ROLE60 DISCOURSE58}, score = -999900.0
   {EVALREC-7 ROLE60 LEXICAL54}, score = -999900.0
   {EVALREC-8 ROLE60 LEXICAL55}, score = -999900.0
   {EVALREC-9 ROLE60 LEXICAL56}, score = -999900.0
   {EVALREC-10 ROLE60 LEXICAL57}, score = -999900.0
   {EVALREC-11 ROLE60 PROPOS53}, score = 200.0
   {EVALREC-12 ROLE60 ROLE49}, score = -999900.0
   {EVALREC-13 ROLE60 ROLE45}, score = -999900.0
   {EVALREC-14 ROLE60 ROLE48}, score = -999900.0
   {EVALREC-15 ROLE60 ROLE44}, score = -999825.0
   {EVALREC-16 ROLE60 DISCOURSE42}, score = -1000050.0
   {EVALREC-17 ROLE60 LEXICAL40}, score = -1000050.0
   {EVALREC-18 ROLE60 LEXICAL41}, score = -1000050.0
   {EVALREC-19 ROLE60 ROLE34}, score = -1000050.0
   {EVALREC-20 ROLE60 GLOBAL39}, score = -999950.0
(best-candidates (RESOL-5 ROLE60 15 15}) --> ({EVALREC-11 ROLE60 PROPOS53})
```

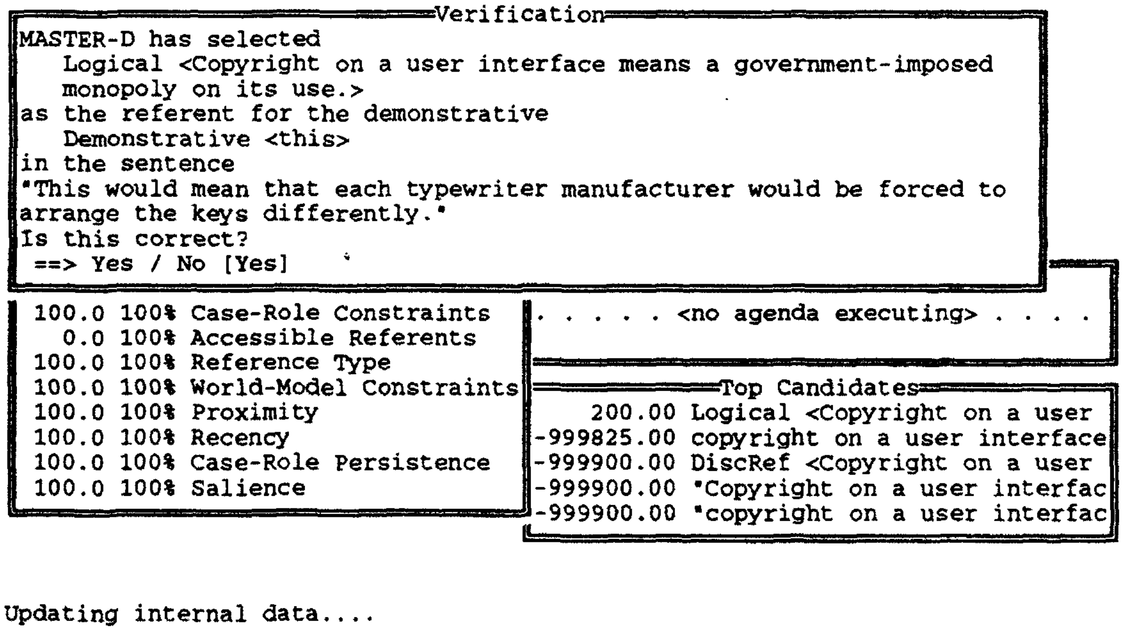
The system now performs its verification and update steps. First, the user is queried whether the selected referent is in fact the correct antecedent. The strategies are then reweighted and normalized, outdated votes are removed from the voting history, and the candidate pool is updated.

```
(verify-resolution 'This would mean that each typewriter manufacturer would
                     be forced to arrange the keys differently.'
                    PROPOSITION59 ROLE60 ...)
(verify-resolution ... ROLE60) --> T
(strategy-reweighter (RESOL-5 ROLE60 15 15})
(rescale-strategies) --> scaled by factor of 0.914286
(purge-voting-history) done
(purge-voting-history) done
(strategy-reweighter (RESOL-5 ROLE60 15 15}) done
(resolve-demonstrative ROLE60 ...) --> ({EVALREC-11 ROLE60 PROPOS53})
*** completing resolution of PARSE-4 ***
Referents found: ((ROLE60 . PROPOS53))
(update-candidate-pool PROPOSITION59 ...)
```



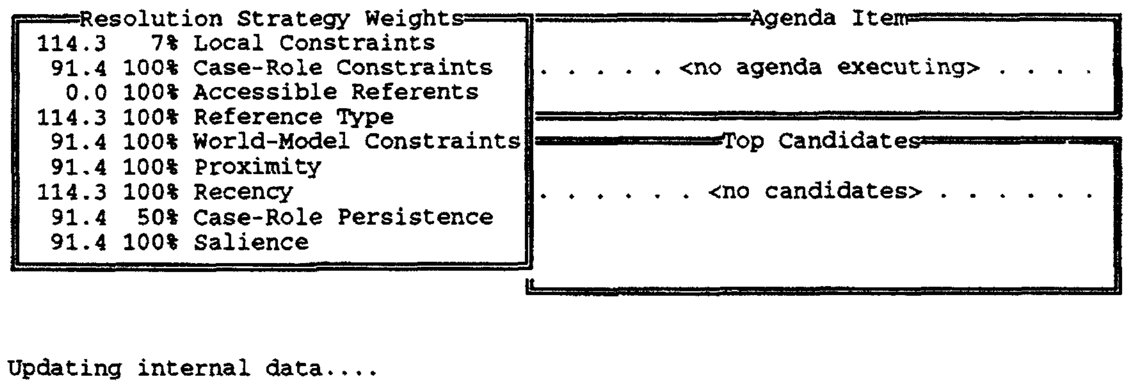
```
==Resolution Strategy Weights==     ==========Agenda Item==========
 100.0 100% Local Constraints       Func STRATEGY-APPLIER-PREF
 100.0 100% Case-Role Constraints   Priority -999975.00
   0.0 100% Accessible Referents    Arg (EVALREC-12 ROLE60 ROLE49)
 100.0 100% Reference Type
 100.0 100% World-Model Constraints =========Top Candidates=========
 100.0 100% Proximity                    200.00 Logical <Copyright on a user
 100.0 100% Recency                 -999900.00 DiscRef <Copyright on a user
 100.0 100% Case-Role Persistence   -999900.00 "Copyright on a user interfac
 100.0 100% Salience                -999900.00 "copyright on a user interfac
                                    -999900.00 "a government-imposed monopol

Processing "This would mean that each typewriter manufacturer would be forced t
```

**Figure A-14: Processing nearing completion**



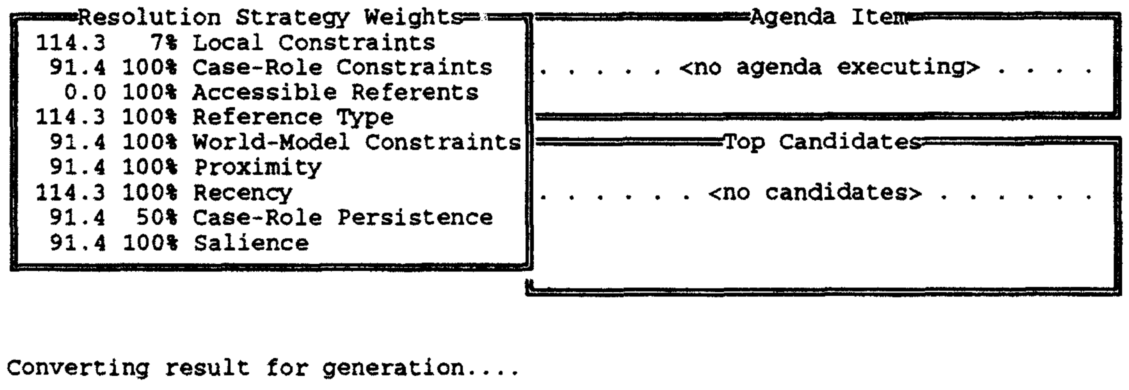
```
=======================Verification=======================
MASTER-D has selected
   Logical <Copyright on a user interface means a government-imposed
   monopoly on its use.>
as the referent for the demonstrative
   Demonstrative <this>
in the sentence
"This would mean that each typewriter manufacturer would be forced to
arrange the keys differently."
Is this correct?
 ==> Yes / No [Yes]
 100.0 100% Case-Role Constraints   . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 100.0 100% Reference Type
 100.0 100% World-Model Constraints =========Top Candidates=========
 100.0 100% Proximity                    200.00 Logical <Copyright on a user
 100.0 100% Recency                 -999825.00 copyright on a user interface
 100.0 100% Case-Role Persistence   -999900.00 DiscRef <Copyright on a user
 100.0 100% Salience                -999900.00 "Copyright on a user interfac
                                    -999900.00 "copyright on a user interfac

Updating internal data....
```

**Figure A-15: Verifying second resolution**



```
==Resolution Strategy Weights==    ================Agenda Item================
 114.3   7% Local Constraints
  91.4 100% Case-Role Constraints    . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 114.3 100% Reference Type
  91.4 100% World-Model Constraints  ==============Top Candidates==============
  91.4 100% Proximity
 114.3 100% Recency                  . . . . . . <no candidates> . . . . . .
  91.4  50% Case-Role Persistence
  91.4 100% Salience

Updating internal data....
```

**Figure A-16: Reweighting strategies**



```
==Resolution Strategy Weights==    ================Agenda Item================
 114.3   7% Local Constraints
  91.4 100% Case-Role Constraints    . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 114.3 100% Reference Type
  91.4 100% World-Model Constraints  ==============Top Candidates==============
  91.4 100% Proximity
 114.3 100% Recency                  . . . . . . <no candidates> . . . . . .
  91.4  50% Case-Role Persistence
  91.4 100% Salience

Converting result for generation....
```

**Figure A-17: Storing resolved parse**

```
(purge-candidate-pool)
(purge-candidate-pool) --> 15 candidates remaining
(nouns-in-parse PROPOSITION59) --> (PROPOS53$73
                                    ROLE62
                                    ROLE63
                                    ROLE64
                                    ROLE68)
(propositions-in-parse PROPOSITION59) --> (PROPOS75 PROPOS74)
(properties-in-parse PROPOSITION59) --> NIL
(lexical-candidates-in-parse PROPOSITION59) --> (LEXICAL82
                                                 LEXICAL81
                                                 LEXICAL80
                                                 LEXICAL79
                                                 LEXICAL78
                                                 LEXICAL77
                                                 LEXICAL76)
(discourse-candidates-in-parse PROPOSITION59) --> (DISCOURSE83)
(update-candidate-pool PROPOSITION59 ...) done, 29 candidates
```

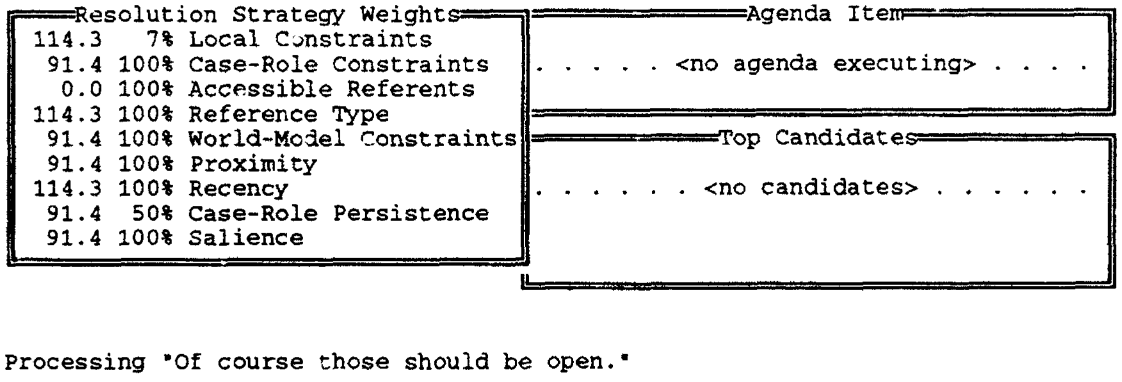
As with the first two sentences, a statistics display is generated and the final parse is transformed into the format required by the generator and stored in the output file. Note the :REFERENCES slot under THEME. This shows that the frame now filling the THEME slot is the unification of the named frames, ROLE60 and PROPOS53.

```
<<< processing complete <<<
CPU time used:     48.11 seconds
Elapsed time:      88.94 seconds

Performance Summary
===================
Sentence Number 3
1 demonstrative
   ROLE60 (Demonstrative <this>)
       --> PROPOS53 (Logical <Copyright on a user interface means a
                              government-imposed monopoly on its use.>)
                                  Initial   Current   Reweights
      Strategy                    Weight    Weight     Up/Down   Accuracy
---------------------------------------------------------------
Local Constraints                 100.000   114.286    1/   0      6.67%
Case-Role Constraints             100.000    91.428    0/   0    -99.99%
Accessible Referents                0.000     0.000    0/   0    -99.99%
Reference Type                    100.000   114.286    1/   0    100.00%
World-Model Constraints           100.000    91.428    0/   0    -99.99%
Proximity                         100.000    91.428    0/   0    -99.99%
Recency                           100.000   114.286    1/   0    100.00%
Case-Role Persistence             100.000    91.428    0/   0     50.00%
Salience                          100.000    91.428    0/   0    -99.99%
---------------------------------------------------------------
Total Rescaling Factor: 0.91
===================

(generate (*MEAN
           (TENSE PRESENT)
           (MODALITY WOULD)
           (THEME
            (*PROPOSITION*
             (LOGICAL-PROP PROPOSITION43)
             ($ID TAG61)
             (ROOT "this")
             (DISTANCE NEAR)
             (NUMBER SINGULAR)
             (:REFERENCES
              (PROPOSITION59 THEME ROLE60)
              (NIL NIL PROPOS53)))))
              (OBJECT
               (*FORCE-ACTION
                (TENSE PRESENT)
                (MODALITY WOULD)
                (THEME
                 (*MANUFACTURER
                  (NUMBER SINGULAR)
                  (REFERENCE INDEFINITE)
                  (OBJECT
                   (*TYPEWRITER
                    (REFERENCE INDEFINITE)
                    (NUMBER SINGULAR)
                    (INPUT "typewriter")
                    ($ID TAG65)))
                  (INPUT "each typewriter manufacturer")
                  ($ID TAG66)))
                (ACTION
                 (*ARRANGE
                  (OBJECT
                   (*KEY
                    (NUMBER PLURAL)
                    (REFERENCE DEFINITE)
                    (INPUT "keys")
                    ($ID TAG69)))
                  (MANNER *DIFFERENTLY)
                  (INPUT "arrange the keys differently")
                  ($ID TAG70)))
                (INPUT
                 "each typewriter manufacturer would be forced to arrange
                  the keys differently")
                ($ID TAG71)))
              (SENTENCE
               "This would mean that each typewriter manufacturer would be
                forced to arrange the keys differently.")
              ($ID TAG72)))
    [generating to file #<Stream for file "/usr1/ralf/src/md/md.output">]
```

The final two sentences are processed similarly, producing approximately 1100 lines of tracing information. Sentence 4 does not contain a demonstrative, while Sentence 5 does. Figures A-18 through A-24 show various stages in processing the fifth sentence. In this case, MASTER-D cannot determine a unique best candidate, so it asks the user to select among the two best candidates in Figure A-24. Since the fifth sentence is the last parse in the data file, MASTER-D allows the user to choose what action to take next (Figure A-25), which may include invoking the main command menu (Figure A-26).

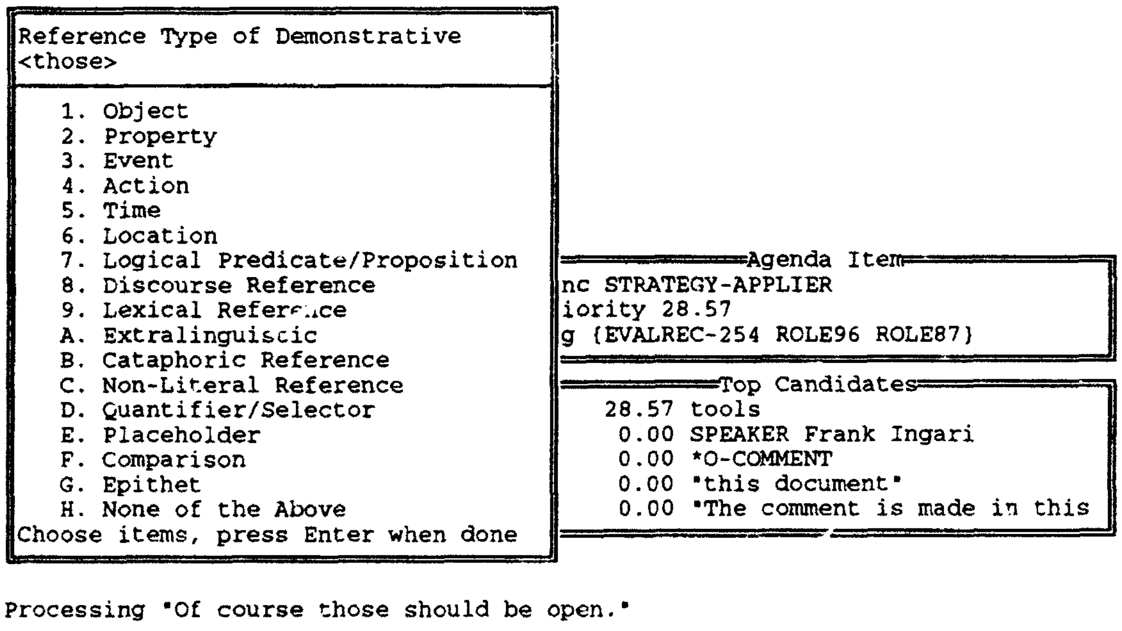
When MASTER-D is shut down (Figure A-27), it outputs final statistics to the trace file. For the example shown in this appendix, the following summary is written to the trace file. Execution times were gathered using the Lisp functions `get-internal-run-time` and `get-internal-real-time`, which have a granularity of 0.01 seconds on the system used. As a result, there can be minor discrepancies such as real time being slightly less than CPU time on the more-quickly executing functions. The total time is computed using a separate timer rather than summing the different phases, and thus does not exactly equal the sum of the other values.

Following the display of execution times, the final strategy weights are shown, along with a summary of reweighting actions and the computed accuracy.



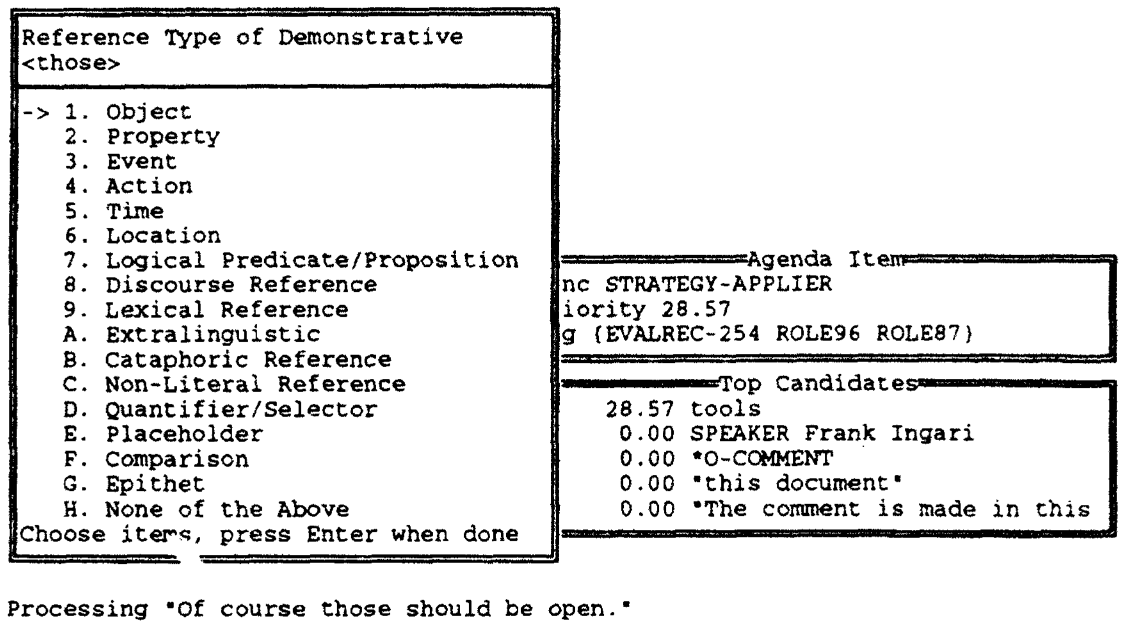
```
══Resolution Strategy Weights══   ══════════════Agenda Item═══════════
 114.3    7% Local Constraints
  91.4 100% Case-Role Constraints   . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 114.3 100% Reference Type
  91.4 100% World-Model Constraints ═════════════Top Candidates═══════════
  91.4 100% Proximity
 114.3 100% Recency                 . . . . . . <no candidates> . . . . . .
  91.4  50% Case-Role Persistence
  91.4 100% Salience

Processing "Of course those should be open."
```

Figure A-18: Beginning processing of fifth sentence



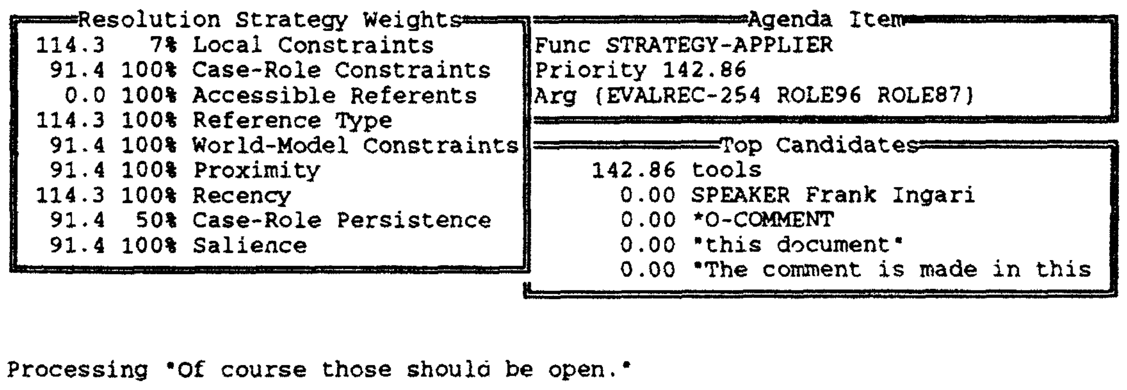
```
Reference Type of Demonstrative
<those>

    1. Object
    2. Property
    3. Event
    4. Action
    5. Time
    6. Location
    7. Logical Predicate/Proposition ═════════════Agenda Item═══════════
    8. Discourse Reference           nc STRATEGY-APPLIER
    9. Lexical Reference             iority 28.57
    A. Extralinguistic               g (EVALREC-254 ROLE96 ROLE87)
    B. Cataphoric Reference
    C. Non-Literal Reference         ═══════════Top Candidates═════════════
    D. Quantifier/Selector             28.57 tools
    E. Placeholder                      0.00 SPEAKER Frank Ingari
    F. Comparison                       0.00 *O-COMMENT
    G. Epithet                          0.00 "this document"
    H. None of the Above                0.00 "The comment is made in this
Choose items, press Enter when done

Processing "Of course those should be open."
```

Figure A-19: Asking user for reference type



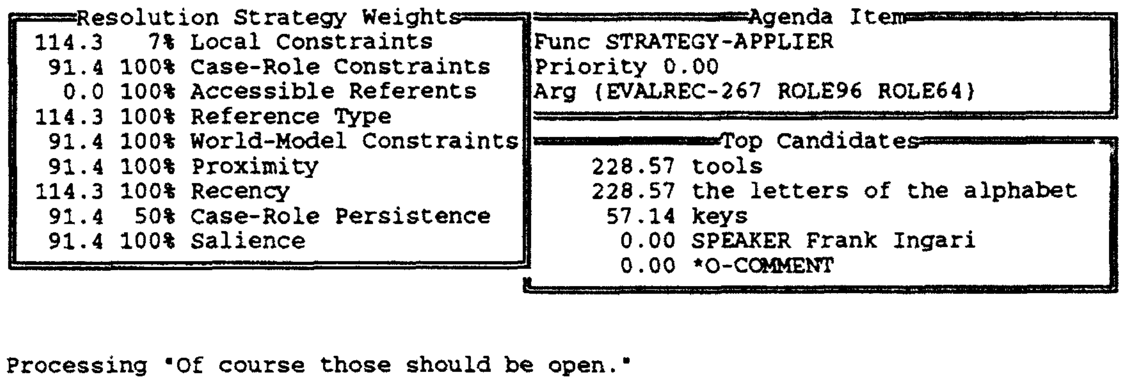
```
Reference Type of Demonstrative
<those>

-> 1. Object
   2. Property
   3. Event
   4. Action
   5. Time
   6. Location
   7. Logical Predicate/Proposition
   8. Discourse Reference
   9. Lexical Reference
   A. Extralinguistic
   B. Cataphoric Reference
   C. Non-Literal Reference
   D. Quantifier/Selector
   E. Placeholder
   F. Comparison
   G. Epithet
   H. None of the Above
Choose iters, press Enter when done

Agenda Item
nc STRATEGY-APPLIER
iority 28.57
g (EVALREC-254 ROLE96 ROLE87)

Top Candidates
  28.57 tools
   0.00 SPEAKER Frank Ingari
   0.00 *O-COMMENT
   0.00 "this document"
   0.00 "The comment is made in this

Processing "Of course those should be open."
```

Figure A-20: User selected reference type



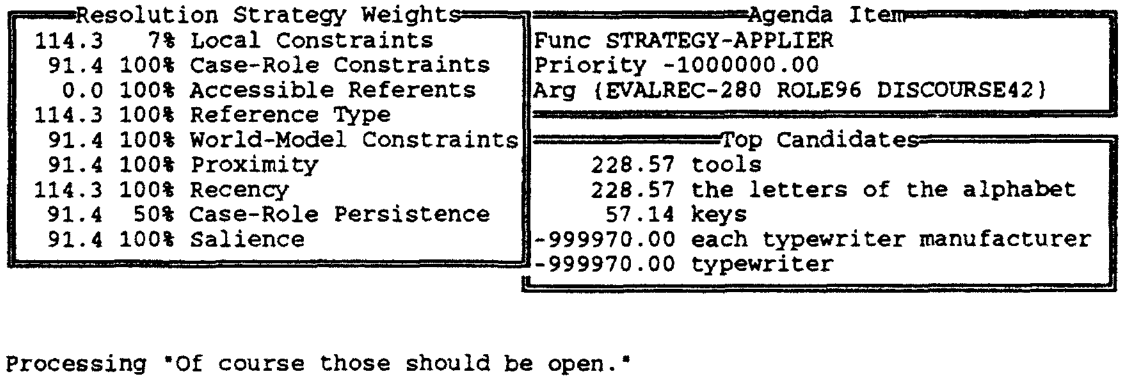
```
Resolution Strategy Weights
 114.3    7% Local Constraints
  91.4 100% Case-Role Constraints
   0.0 100% Accessible Referents
 114.3 100% Reference Type
  91.4 100% World-Model Constraints
  91.4 100% Proximity
 114.3 100% Recency
  91.4  50% Case-Role Persistence
  91.4 100% Salience

Agenda Item
Func STRATEGY-APPLIER
Priority 142.86
Arg (EVALREC-254 ROLE96 ROLE87)

Top Candidates
  142.86 tools
    0.00 SPEAKER Frank Ingari
    0.00 *O-COMMENT
    0.00 "this document"
    0.00 "The comment is made in this

Processing "Of course those should be open."
```

Figure A-21: Applying strategies



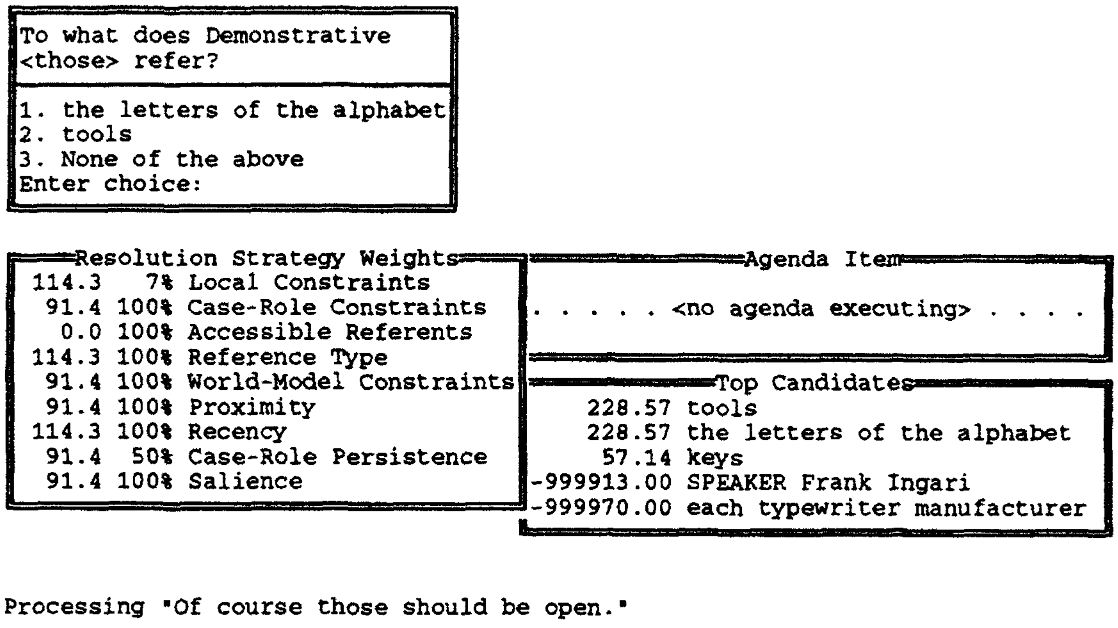
```
Resolution Strategy Weights                 Agenda Item
 114.3   7% Local Constraints            Func STRATEGY-APPLIER
  91.4 100% Case-Role Constraints        Priority 0.00
   0.0 100% Accessible Referents         Arg (EVALREC-267 ROLE96 ROLE64)
 114.3 100% Reference Type
  91.4 100% World-Model Constraints                Top Candidates
  91.4 100% Proximity                         228.57 tools
 114.3 100% Recency                           228.57 the letters of the alphabet
  91.4  50% Case-Role Persistence              57.14 keys
  91.4 100% Salience                            0.00 SPEAKER Frank Ingari
                                                0.00 *O-COMMENT

Processing "Of course those should be open."
```

**Figure A-22: Processing continues**



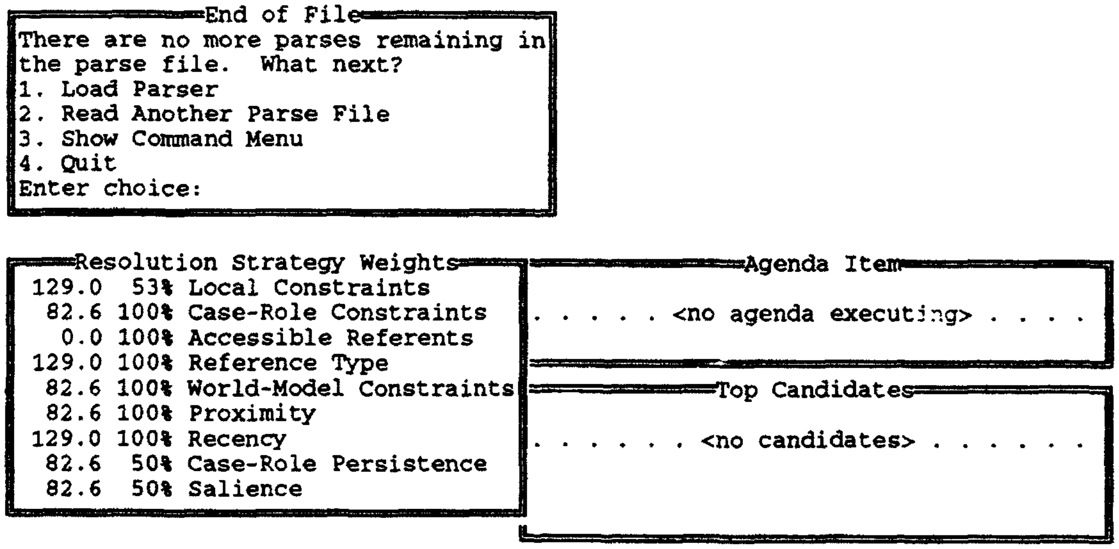
```
Resolution Strategy Weights                 Agenda Item
 114.3   7% Local Constraints            Func STRATEGY-APPLIER
  91.4 100% Case-Role Constraints        Priority -1000000.00
   0.0 100% Accessible Referents         Arg (EVALREC-280 ROLE96 DISCOURSE42)
 114.3 100% Reference Type
  91.4 100% World-Model Constraints                Top Candidates
  91.4 100% Proximity                         228.57 tools
 114.3 100% Recency                           228.57 the letters of the alphabet
  91.4  50% Case-Role Persistence              57.14 keys
  91.4 100% Salience                      -999970.00 each typewriter manufacturer
                                          -999970.00 typewriter

Processing "Of course those should be open."
```

**Figure A-23: Processing nearing completion**



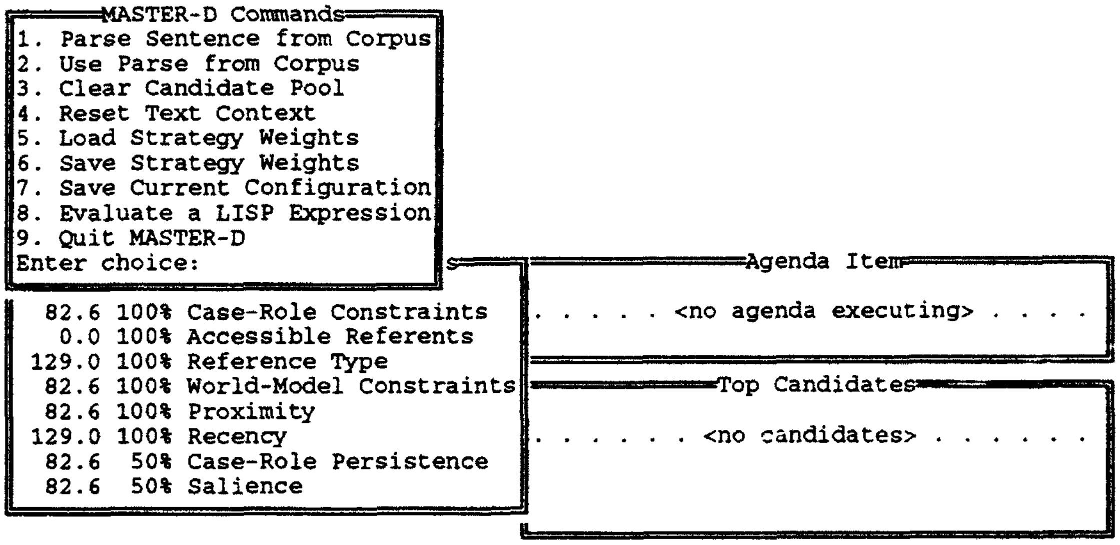
```
To what does Demonstrative
<those> refer?

1. the letters of the alphabet
2. tools
3. None of the above
Enter choice:

Resolution Strategy Weights                       Agenda Item
 114.3    7% Local Constraints
  91.4 100% Case-Role Constraints        . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 114.3 100% Reference Type
  91.4 100% World-Model Constraints                Top Candidates
  91.4 100% Proximity                       228.57 tools
 114.3 100% Recency                         228.57 the letters of the alphabet
  91.4  50% Case-Role Persistence            57.14 keys
  91.4 100% Salience                    -999913.00 SPEAKER Frank Ingari
                                        -999970.00 each typewriter manufacturer

Processing "Of course those should be open."
```

**Figure A-24: Asking user to disambiguate**



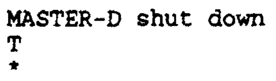
```
End of File
There are no more parses remaining in
the parse file.  What next?
1. Load Parser
2. Read Another Parse File
3. Show Command Menu
4. Quit
Enter choice:

Resolution Strategy Weights                       Agenda Item
 129.0  53% Local Constraints
  82.6 100% Case-Role Constraints        . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 129.0 100% Reference Type
  82.6 100% World-Model Constraints                Top Candidates
  82.6 100% Proximity
 129.0 100% Recency                      . . . . . . <no candidates> . . . . . .
  82.6  50% Case-Role Persistence
  82.6  50% Salience
```

**Figure A-25: End of parse file reached**



```
MASTER-D Commands
1. Parse Sentence from Corpus
2. Use Parse from Corpus
3. Clear Candidate Pool
4. Reset Text Context
5. Load Strategy Weights
6. Save Strategy Weights
7. Save Current Configuration
8. Evaluate a LISP Expression
9. Quit MASTER-D
Enter choice:                 s
                                          Agenda Item
  82.6 100% Case-Role Constraints   . . . . . <no agenda executing> . . . .
   0.0 100% Accessible Referents
 129.0 100% Reference Type
  82.6 100% World-Model Constraints       Top Candidates
  82.6 100% Proximity
 129.0 100% Recency                 . . . . . . <no candidates> . . . . . .
  82.6  50% Case-Role Persistence
  82.6  50% Salience
```

Figure A-26: MASTER-D Command Menu



```
MASTER-D shut down
T
*
```

Figure A-27: MASTER-D shut down

```
(get-parse-from-file) --> :QUIT
(get-parse) --> QUIT
(shutdown) invoked
(shutdown-parser)
(shutdown-generator)
(shutdown-reweighting)
(shutdown-agenda)
(shutdown-world-model)
(shutdown-ontology)
(shutdown-discourse-model)
(shutdown-statistics)

Final Summary of Performance
============================
Sentences Processed:  5
Demonstratives: 3 (1 nonreferential)
Candidates Processed: 49 (avg 16.33 per demonstrative)

Correctness            Occurrences
---------+----------------------------------------------
       1 |** (2)
     1/2 |* (1)
---------+----------------------------------------------

      Phase                Real Time        CPU time     Calls
----------------------------------------------------------------
Parser Interface             2.65 sec        2.36 sec       14
Generator Interface          5.63 sec        5.03 sec        7
Canonicalization             5.03 sec        5.04 sec       12
Reweighting                  0.68 sec        0.52 sec        4
Agenda/Cand Display         60.72 sec       60.29 sec     1489
Strategy Application        17.78 sec       18.33 sec      459
Recovery Strategies          0.15 sec        0.11 sec        2
Resolution Overhead         15.57 sec       15.09 sec        3
World Model                  0.72 sec        0.35 sec       59
Ontology                     8.38 sec        8.17 sec       32
Candidate Management         1.34 sec        1.28 sec       12
Corpus Access                0.03 sec        0.02 sec        1
Garbage Collection           0.00 sec        0.00 sec        0
User Interface             198.37 sec       56.86 sec     1023
Other                        7.75 sec        3.90 sec        1
----------------------------------------------------------------
        Total              325.02 sec      177.42 sec
                                 Initial    Final    Reweights
      Strategy                   Weight     Weight     Up/Down   Accuracy
----------------------------------------------------------------
Local Constraints               100.000    129.032    2/   0     53.33%
Case-Role Constraints           100.000     82.581    0/   0    -99.99%
Accessible Referents              0.000      0.000    0/   0    -99.99%
Reference Type                  100.000    129.032    2/   0    100.00%
World-Model Constraints         100.000     82.581    0/   0    -99.99%
Proximity                       100.000     82.581    0/   0    100.00%
Recency                         100.000    129.032    2/   0    100.00%
Case-Role Persistence           100.000     82.581    0/   0     50.00%
Salience                        100.000     82.581    0/   0     50.00%
----------------------------------------------------------------
Total Rescaling Factor: 0.83
============================

(shutdown-correctness)
(shutdown-tracing)
MASTER-D stopped at 7:15:49 on 5-07-1993
==========================================
```

Not surprisingly, displaying the current agenda item and top five candidates after every strategy application uses considerable amounts of CPU time--the 60.29 seconds listed as "Agenda/Cand Display" as well as most of the 56.86 seconds listed for "User Interface". Creating the execution trace uses a majority of the remaining CPU time, especially in strategy application and "Resolution Overhead" (which includes agenda handling, determining the best candidate, and updating strategy voting information).

For comparison, the following timing report is generated for the same set of sentences when tracing output is minimized and the debugging windows are disabled. This is the performance one would normally experience in a production environment rather than the testing/debugging environment used in this appendix. Further performance gains of five to ten percent can be achieved by entirely removing the debugging and timing code. Note that most of the time under Ontology was used in loading the knowledge base; a smaller knowledge base containing only the items required for these five parses would have significantly reduced the overall processing time.

```
        Phase                   Real Time         CPU time     Calls
---------------------------------------------------------------------
Parser Interface                0.56 sec          0.54 sec        14
Generator Interface             2.96 sec          2.90 sec         7
Canonicalization                0.20 sec          0.24 sec        12
Reweighting                     0.03 sec          0.03 sec         4
Strategy Application            2.96 sec          2.97 sec       459
Recovery Strategies             0.03 sec          0.03 sec         2
Resolution Overhead             1.28 sec          0.85 sec         3
World Model                     0.30 sec          0.27 sec        59
Ontology                        7.84 sec          7.75 sec        32
Candidate Management            0.83 sec          0.77 sec        12
Corpus Access                   0.00 sec          0.00 sec         1
Garbage Collection              0.00 sec          0.00 sec         0
User Interface                 20.82 sec          6.18 sec        38
Other                           5.72 sec          3.49 sec         1
---------------------------------------------------------------------
        Total                  43.55 sec         26.05 sec
```

# Appendix B
# Selected Source Code

This appendix lists three of the core source modules of the MASTER-D system (amounting to approximately one-ninth of the complete source code) in their entirety. The full source code is available from the CMU Center for Machine Translation and by anonymous FTP to FTP.CS.CMU.EDU in directory /afs/cs/project/cmt/master-d/pub.

The first source module shown here is str-appl.lisp, the actual strategy application mechanism. The second source module is reweight.lisp, which contains the strategy reweighting functions. The last module is strategies.lisp, which contains the top-level definitions of the resolution strategies.

## B.1. Strategy Applier

```
;;*************************************************************************
;;*                                                                       *
;;*                              MASTER-D                                 *
;;*   (Multiple Adaptively-weighted STratEgies for Resolving Demonstratives) *
;;*                                                                       *
;;*  File: str-appl.lisp        resolution-strategy applier               *
;;*  Last Edit: 27 Apr 93                                                 *
;;*                                                                       *
;;*  Copyright (c) 1993 Ralf Brown.  All Rights Reserved.                 *
;;*  Permission granted for educational and non-commercial research uses.  *
;;*  Other uses require prior permission by Ralf Brown or the CMU Center   *
;;*  for Machine Translation.                                             *
;;*                                                                       *
;;*************************************************************************


#|
Note: constraint and preference strategies return two values.  The first is
the base score for the demonstrative/candidate pair, regardless of the distance
between them, and may be either 'invalid or a number between -1.0 and +1.0,
inclusive.  The second value is the distance attenuation, and is a list
whose first element is a function of two arguments, the first of which is the
base score and the second of which is the second element of the returned list,
normally an attenuation factor.

Recovery strategies return a list of the candidates added (recovery0) or the
candidates still remaining (recovery2).
|#

;;-----------------------------------------------------------------
;;  define constants used by this module
```

```
;;
(defconstant *empty-agenda-window*
            '(#\Newline ". . . . . <no agenda executing> . . . . "
              #\Newline " "))
(defconstant *empty-topcand-window*
            '(#\Newline ". . . . . . <no candidates> . . . . . . "
              #\Newline #\Newline #\Newline
              " "
             ))

;;-----------------------------------------------------------------
;;  declare globals defined in other modules
;;
(defvar *default-constraint-strategies*)
(defvar *default-preference-strategies*)
(defvar *default-recovery0-strategies*)
(defvar *default-recovery2-strategies*)
(defvar *constraint-strategies*)
(defvar *preference-strategies*)
(defvar *recovery0-strategies*)
(defvar *recovery2-strategies*)

;;-----------------------------------------------------------------
;; declare global variables used by this module
;;
(defvar *strat-timer* nil)
(defvar *resol-timer* nil)
(defvar *recov-timer* nil)
(defvar *agenda-timer* nil)
(defvar *agenda-window* nil)
(defvar *top-cand-window* nil)
(defvar *current-candidates* nil)

;;-----------------------------------------------------------------
;; Determine the relative priority to be given to the specified evaluation
;; record, which controls the order in which strategies are applied on
;; demonstrative-candidate pairs.
;;
(defun application-priority (eval-rec)
   (case *prioritize-resolutions*
      (nil      0.0)
      (:strat   (+ (length (evalrec-constraints eval-rec))
                   (length (evalrec-preferences eval-rec)))
                )
      (:score   (evalrec-score eval-rec))
      (t        (evalrec-score eval-rec))
   )
)

;;-----------------------------------------------------------------
;; Determine the highest score among the given list of evaluation records
;;
(defun highest-score (cands &aux hiscore)
   (setf hiscore -infinity)
   (dolist (ev cands)
      (if (> (evalrec-score ev) hiscore)
         (setf hiscore (evalrec-score ev))
      )
   )
   hiscore
)
```

```
;;---------------------------------------------------------------------
;; Determine the best candidates for the specified resolution, which are
;; the candidate with the highest score and any others within a specified
;; threshold amount of that highest score.
;;
(defun best-candidates (res &aux hiscore valid result)
   (when (/= (resol-valid-candidates res) 0)   ; leave result==NIL otherwise
      (dolist (evalrec (resol-eval-recs res))
         (if (evalrec-valid evalrec)    ; collect all the valid candidates
            (push evalrec valid)
         )
      )
      ;
      (setf hiscore (highest-score valid))
      (if (> hiscore 0)
         (setf hiscore (* hiscore *high-score-threshold*))
      ;else
         (incf hiscore (* hiscore (- 1 *high-score-threshold*)))
      )
      #-nodebug (mdtrace 7 "(best-candidates), hiscore = ~S" hiscore)
      (dolist (ev valid)
         #-nodebug (mdtrace 4 "   ~S, score = ~S" ev (evalrec-score ev))
         (if (>= (evalrec-score ev) hiscore)
            (push ev result)
         )
      )
   )
   #-nodebug (mdtrace 5 "(best-candidates ~S) --> ~S" res result)
   result
)

;;---------------------------------------------------------------------
;; Display the candidates which currently have the highest scores in
;; a separate window on the screen.
;;
(defun display-top-candidates (&aux count body)
   #-notimer (start-split-timer *agenda-timer*)
   (setf *current-candidates* (sort *current-candidates* #'>
                                                          :key #'evalrec-score))
   (setf count 0)
   (dolist (cand *current-candidates*)
      (push (format nil "~10,2,,'-F ~29A" (evalrec-score cand)
                        (left-string (readable-name
                                          (evalrec-candidate-ref cand)) 29)
                        )
            body)
      (incf count)
      (if (>= count *max-top-candidates*)
         (return)
      )
   )
   (if body
      ; force each item in the list to appear on a separate line
      (setf body (intersperse (nreverse body) #\Newline))
   ;else
      (setf body *empty-topcand-window*)
   )
   (redisplay-window *top-cand-window* body)
   #-notimer (stop-split-timer)
)

;;---------------------------------------------------------------------
```

```
;; reset the list of votes associated with each strategy for the current
;; resolution
;;
(defun clear-strategy-votes (&rest strat-lists)
   (dolist (strategies strat-lists)
      (dolist (strat strategies)
         (setf (strategy-votes strat) nil) ; reset current-resol vote list
      )
   )
   t
)

;;-----------------------------------------------------------------------
;; add the votes for the current resolution to the overall voting history
;; for each strategy
;;
(defun update-voting-history (&rest strat-lists)
   (dolist (strategies strat-lists)
      (dolist (strat strategies)
         (when (strategy-votes strat)
            (push (strategy-votes strat) (strategy-voting-history strat))
            (incf (strategy-voting-hist-len strat)
                  (length (strategy-votes strat)))
         )
      )
   )
   t
)

;;-----------------------------------------------------------------------
;;  Process a strategy which consists of multiple strategies.  For each
;;  of the substrategies, invoke the resolution function, and add up the
;;  returned scores from all of the substrategies.  Also add up the weights
;;  of all the strategies in order to scale the sum back down into the
;;  proper range.
;;
(defun apply-substrategies (demonstrative candidate strat
                             &aux strats valid sum weight total-weight
                                  score s vote)
   #+debug (mdtrace 7 "(apply-substrategies ~S ~S ~S)" demonstrative candidate
                                                  (strategy-name strat))
   (setf strats (strategy-substrats strat))
   (setf valid t sum 0.0 total-weight 0.0)
   (dolist (strat strats)
      (let (score)
         (setf score (funcall (strategy-func strat) demonstrative candidate))
         (case score
            (invalid    (setf valid nil)
                        (setf s *score-invalid*)
                        (if *short-circuit-eval*
                           (return)
                        ))
            (t          (setf weight (strategy-weight strat))
                        (setf s score)
                        (incf sum (* score weight))
                        (incf total-weight weight)
                        )
         )
         ; record the substrategy's vote
         (setf vote (make-vote
                        :strategy strat
                        :dem demonstrative
```

```
                                  :candidate candidate
                                  :dist-func (strategy-dist-func strat)
                                  :distance -1.0    ; we don't know
                                  :decay-func (strategy-decay-func strat)
                                  :decay-speed (strategy-decay-speed strat)
                                  :decays-to (strategy-decays-to strat)
                                  :raw-score score
                                  :score s
                                  ))
              #+debug (intern-struct vote)
              (push vote (strategy-votes strat))
           )
       )
       (if valid
          (setf score (/ sum total-weight))
       ;else
          (setf score 'invalid)
       )
       #-nodebug (mdtrace 7 "(apply-substrategies ~S ~S) --> ~S" demonstrative
                                                               candidate score)
       (values score total-weight)
   )

   ;;------------------------------------------------------------------------
   ;; Process a single strategy.
   ;;
   (defun apply-strategy (strat eval-rec
                               &aux dem candidate score s weight dist decay func
                                    tot-weight vote)
       #-notimer (start-split-timer *strat-timer*)
       #+debug (mdtrace 7 "(apply-strategy ~S ~S)" (strategy-name strat) eval-rec)
       (setf dem (evalrec-demonstrative eval-rec))
       (setf candidate (evalrec-candidate-ref eval-rec))
       (if *display-top-candidates*
          (display-top-candidates)
       )
       (if (strategy-substrats strat)
          (multiple-value-setq (score tot-weight)
                               (apply-substrategies dem candidate strat)
                               )
       ;else
          (setf score (funcall (strategy-func strat) dem candidate))
       )
       (push (list strat score) (evalrec-scores eval-rec))
       (setf weight (strategy-weight strat))
       (if (and (= weight 0) tot-weight)
          (setf weight tot-weight)
       )
       ;
       ; determine the distance between demonstrative and candidate, and the
       ; value of the specified decay function for that distance
       ;
       (setf dist (funcall (strategy-dist-func strat) dem candidate))
       (setf decay (funcall (strategy-decay-func strat)
                            dist (strategy-decay-speed strat)))
       ;
       ; adjust the raw score into a full score which is then added to the total
       ;
       (cond
          ((equal score 'invalid)
                     (setf s *score-invalid*)
                     (incf (evalrec-score eval-rec) s)
```

```
                )
    (t          (setf func (first decay))
                (if (consp func)
                   (setf func (eval func))
                )
                (setf s (funcall func score (second decay)))
                ; did the decay go beyond our preset limit?
                (if (/= (signum (- score (strategy-decays-to strat)))
                        (signum (- s (strategy-decays-to strat)))
                    )
                   (setf s (strategy-decays-to strat))
                )
                (incf (evalrec-score eval-rec) (* s weight))
                )
   )
   (when (and (equal score 'invalid) (equal (evalrec-valid eval-rec) t))
      (decf (resol-valid-candidates (evalrec-resol eval-rec)))
      (setf (evalrec-valid eval-rec) 'invalid)
   )
   ;
   ; record the strategy's vote
   ;
   (setf vote (make-vote
                :strategy strat
                :dem dem
                :candidate candidate
                :dist-func (strategy-dist-func strat)
                :distance dist
                :decay-func (strategy-decay-func strat)
                :decay-speed (strategy-decay-speed strat)
                :decays-to (strategy-decays-to strat)
                :raw-score score
                :score s
                ))
   #+debug (intern-struct vote)
   (push vote (strategy-votes strat))
   (push vote (evalrec-votes eval-rec))
   #+debug (mdtrace 25 "recorded vote ~S" vote)
   ;
   ; and finally return the resulting score
   ;
   #-nodebug (mdtrace 6 "(apply-strategy ~S ~S) --> ~S [~S]"
                                   (strategy-name strat) eval-rec score s)
   #-notimer (stop-split-timer)
   score
)

;;-------------------------------------------------------------------
;; Loop through the constraint strategies for a particular
;; demonstrative-candidate pair.
;;
(defun strategy-applier-const (eval-rec res &aux exitcode)
   (declare (ignore res))
   #+debug (mdtrace 8 "(strategy-applier-const EVALREC--D)"
                                                   (evalrec-id eval-rec))
   (do ((constraint (evalrec-constraints eval-rec)
                    (evalrec-constraints eval-rec)))
       ((or (not constraint) exitcode)) ; quit if no more constraints to apply
      (setf constraint (first constraint))
      (if (and *handle-substrats-individually* (strategy-substrats constraint))
         (dolist (substrat (strategy-substrats constraint))
            (apply-strategy substrat eval-rec)
            )
         ;else
            (apply-strategy constraint eval-rec)
         )
         (pop (evalrec-constraints eval-rec))
         (when (and *short-circuit-eval*
                    (equal (evalrec-valid eval-rec) 'invalid))
            #-nodebug (mdtrace 6 "(strategy-applier EVALREC--D) short-circuited"
                                 (evalrec-id eval-rec))
            (setf exitcode 'done)
         )
         (if *interleave-resolutions*
            (setf exitcode (list :requeue (application-priority eval-rec)))
         )
      )
      exitcode
   )

   ;;-----------------------------------------------------------------
   ;; Loop through the preference strategies for a particular
   ;; demonstrative-candidate pair.
   ;;
   (defun strategy-applier-pref (eval-rec res &aux exitcode)
      #+debug (mdtrace 8 "(strategy-applier-pref EVALREC--D)"
                         (evalrec-id eval-rec))
      (do ((pref (evalrec-preferences eval-rec) (evalrec-preferences eval-rec)))
          ((or (not pref) exitcode))       ; quit if no more preferences to apply
         (setf pref (first pref))
         (if (and *handle-substrats-individually* (strategy-substrats pref))
            (dolist (substrat (strategy-substrats pref))
               (apply-strategy substrat eval-rec)
            )
         ;else
            (apply-strategy pref eval-rec)
         )
         (pop (evalrec-preferences eval-rec))
         (when *interleave-resolutions*    ; go to sleep and let others run
            (queue-agenda-item (application-priority eval-rec)
                               'strategy-applier-pref eval-rec res)
            (setf exitcode 'pending)
         )
      )
      (if exitcode
         exitcode
      ;else
         'done
      )
   )

   ;;-----------------------------------------------------------------
   ;; The actual application mechanism; an instance of this is added to the
   ;; main agenda for each demonstrative/candidate pair.
   ;;
   ;;  We loop through the list of constraints, applying each one in turn and
   ;;  recording the results.  After that, we do the same for preferences
   ;;  (if not thrown out, or told to do so by a global flag).  When we
   ;;  finish, we decrement a counter;  once all instances for a given
   ;;  demonstrative are done, a demon will fire and add a cleanup strategy
   ;;  which will invoke the recovery strategies if necessary.
   ;;
   ;;  Unless told otherwise by a global flag, we go to sleep after each
   ;;  strategy is applied so that all candidates are processed roughly
```

```
;;  in parallel.
;;
(defun strategy-applier (eval-rec &aux exitcode res)
   #-nodebug (mdtrace 6 "(strategy-applier ~S) start" eval-rec)
   (setf res (evalrec-resol eval-rec))
   (setf exitcode (strategy-applier-const eval-rec res))
   (if (not exitcode)
      (setf exitcode (strategy-applier-pref eval-rec res))
   )
   (when (eq exitcode 'done)
      (decf (resol-pending-evals res))
      (push eval-rec *completely-processed-evals*)
   )
   #-nodebug (mdtrace 6 "(strategy-applier EVALREC--D) --> ~S"
                                              (evalrec-id eval-rec) exitcode)
   exitcode
)

;;---------------------------------------------------------------------
;; Recover the original frame if the specified frame is a metonym; returns
;; the specified frame if it is not a metonym
;;
(defun un-metonym (item)
   (if (evalrec-p item)
      (or (first (get-values (evalrec-candidate-ref item) 'metonym-for))
          (evalrec-candidate-ref item))
   ;else
      (if (frame-p item)
         (or (first (get-values item 'metonym-for)) item)
      ;else
         item
      )
   )
)

;;---------------------------------------------------------------------
;; The main demonstrative resolver.
;;
;; What we want to do for each demonstrative:
;;  For each candidate referent, apply constraints, preferences, and (if
;;  needed) recovery strategies, keeping track of how each strategy
;;  voted.  Once all candidates have been processed, pick out the most
;;  preferred as the referent, find out whether that choice was correct,
;;  and reweight the strategies accordingly.
;; Note: the &optional is a workaround for a compiler bug which causes
;;   a runtime error if the desired &key is used
;;
(defun resolve-demonstrative (parse-rec dem candidates
                         &optional (constraints *default-constraint-strategies*)
                               (preferences *default-preference-strategies*)
                               (recovery0 *default-recovery0-strategies*)
                               (recovery2 *default-recovery2-strategies*)
                               (recursive nil)
                         &aux res eval-record num-candidates refs
                         )
   (declare (ignore recursive))
   #-notimer (start-split-timer *resol-timer*)
   (mdtrace 2 "(resolve-demonstrative ~S ~S)" dem candidates)
   (setf *constraint-strategies* constraints
         *preference-strategies* preferences
         *recovery0-strategies* recovery0
         *recovery2-strategies* recovery2)
```

```
(setf num-candidates (length candidates))
(setf res (make-resol :demonstrative dem
                      :candidates candidates
                      :add-candidates nil
                      :pending-evals num-candidates
                      :valid-candidates num-candidates
                      :eval-recs nil
                      :constraints constraints
                      :preferences preferences
                      :recovery-0 recovery0
                      :recovery-2 recove.y2
                  ))
#+debug (intern-struct res)
(dolist (cand candidates)
   (setf eval-record (make-evalrec :resol res
                                   :demonstrative dem
                                   :candidate-ref cand
                                   :constraints constraints
                                   :preferences preferences
                                   :scores nil
                                   :score 0.0
                                   :valid t
                                   ))
   (push eval-record (resol-eval-recs res))
   #+debug (intern-struct eval-record)
   (queue-agenda-item (application-priority eval-record)
                      'strategy-applier eval-record)
)
(if *display-top-candidates*
   (setf *current-candidates* (copy-list (resol-eval-recs res)))
)
(run-entire-agenda)                   ; fire off evaluations until done
;
; after all the candidates have been processed, we want to check whether
; a unique referent is left.  If not, initiate recovery actions.  Also
; initiate recovery if even the best candidate was rules out by the
; constraints
;
(setf (resol-best-cand res) (best-candidates res))
(cond
   ((or (null (resol-best-cand res))
        (< (highest-score (resol-best-cand res)) *score-invalid*))
             (initiate-recovery0 res)
             )
   ((rest (resol-best-cand res))    ; multiple candidates remaining?
             (initiate-recovery2 res)
             )
)
;
; now that we have invoked the recovery strategies (if applicable), also
; trigger the reweighting mechanism
;
(show-status "Updating internal data....")
(setf refs (resol-best-cand res))
(setf (resol-referent res) (if (consp refs)
                               (un-metonym (first refs))
                               (un-metonym refs)
                               ))
(update-voting-history constraints preferences)
(determine-resolution-correctness parse-rec res dem)
(clear-strategy-votes constraints preferences)
(if candidates              ; did we run any normal strategies?
      (strategy-reweighter res)
   )
   (when *display-top-candidates*
      (setf *current-candidates* nil)
      (display-top-candidates)
   )
   (mdtrace 2 "(resolve-demonstrative ~S ...) --> ~S" dem refs)
   #-notimer (stop-split-timer)
   res
)

;;------------------------------------------------------------------
;; Invoke the priming function (if present) of each strategy so that it
;; can update the strategy's view of the world before each parse is
;; processed.
;;
(defun prime-strategy-data-aux (strategies parse-rec)
   (dolist (strategy strategies)
      (if (strategy-priming-func strategy)
         (funcall (strategy-priming-func strategy) strategy parse-rec)
      )
      (dolist (substrat (strategy-substrats strategy))
         (if (strategy-priming-func substrat)
            (funcall (strategy-priming-func substrat) substrat parse-rec)
         )
      )
   )
   t
)

(defun prime-strategy-data (parse-rec)
   (prime-strategy-data-aux *constraint-strategies* parse-rec)
   (prime-strategy-data-aux *preference-strategies* parse-rec)
   (prime-strategy-data-aux *recovery0-strategies* parse-rec)
   (prime-strategy-data-aux *recovery2-strategies* parse-rec)
   t
)

;;------------------------------------------------------------------
;; Invoke the update function (if present) of each strategy so that it
;; can update the strategy's view of the world after each parse has been
;; processed.
;;
(defun update-strategy-data-aux (strategies parse-rec)
   (dolist (strategy strategies)
      (if (strategy-update-func strategy)
         (funcall (strategy-update-func strategy) strategy parse-rec)
      )
      (dolist (substrat (strategy-substrats strategy))
         (if (strategy-update-func substrat)
            (funcall (strategy-update-func substrat) substrat parse-rec)
         )
      )
   )
   t
)

(defun update-strategy-data (parse-rec)
   (update-strategy-data-aux *constraint-strategies* parse-rec)
   (update-strategy-data-aux *preference-strategies* parse-rec)
   (update-strategy-data-aux *recovery0-strategies* parse-rec)
   (update-strategy-data-aux *recovery2-strategies* parse-rec)
   t
)

;;--------------------------------------------------------------------------
;;  fire up the strategies with which to expand the space of candidate
;;  referents.  In most cases, the strategies will recursively invoke
;;  the standard resolution mechanism to apply the normal constraints
;;  and preferences.
;;
(defun initiate-recovery0 (res &aux new-cands result)
   #-notimer (start-split-timer *recov-timer*)
   #-nodebug (mdtrace 3 "(initiate-recovery0 RESOL--D)" (resol-id res))
   (dolist (strat (resol-recovery-0 res))
      #-nodebug (mdtrace 4 "  applying recovery strategy ~S"
                          (strategy-name strat))
      (if *display-top-candidates*
         (display-top-candidates)
      )
      (setf new-cands (funcall (strategy-func strat) res))
      #-nodebug (mdtrace 5 "    new candidates: ~S" new-cands)
      (dolist (cand new-cands)
         (push cand (resol-add-candidates res))
         (push cand (resol-candidates res))
      )


   )
   (if (not (consp result))
      (setf result (list result))
   )
   (setf (resol-best-cand res) result)
   #-nodebug (mdtrace 3 "(initiate-recovery0 RESOL--D) exit" (resol-id res))
   #-notimer (stop-split-timer)
   result
)

;;--------------------------------------------------------------------------
;;  fire up the strategies with which to further restrict the set of
;;  candidate referents.  Stop as soon as a single candidate remains.
;;
(defun initiate-recovery2 (res &aux remaining)
   #-notimer (start-split-timer *recov-timer*)
   #-nodebug (mdtrace 3 "(initiate-recovery2 RESOL-D)" (resol-id res))
   (dolist (strat (resol-recovery-2 res))
      #-nodebug (mdtrace 4 "  applying recovery strategy ~S"
                          (strategy-name strat))
      (if *display-top-candidates*
         (display-top-candidates)
      )
      (setf remaining (funcall (strategy-func strat) res))
      (if (not (consp remaining))
         (setf remaining (list remaining))
      )
      #-nodebug (mdtrace 5 "    remaining: ~S" remaining)
      (setf (resol-best-cand res) remaining)
      (if (= (length remaining) 1)
         (return)
      )
   )
   #-nodebug (mdtrace 3 "(initiate-recovery2 RESOL-D) exit" (resol-id res))
   #-notimer (stop-split-timer)
)
```

```
;;-----------------------------------------------------------------------
;; This function is called by the agenda handler just before it invokes
;; an agenda item's function.  The strategy applier uses this hook to
;; display each strategy application as it occurs.
;;
(defun agenda-pre-hook (priority func args)
   #-notimer (start-split-timer *agenda-timer*)
   (redisplay-window *agenda-window*
                     (list (format nil "Func ~35A" func)
                           #\Newline
                           (format nil "Priority ~,2F" priority)
                           #\Newline
                           (left-string (format nil "Arg ~A" (first args)) 40)
                           ))
   #-notimer (stop-split-timer)
   t
)

;;-----------------------------------------------------------------------
;; This function is called by the agenda handler just after it invokes
;; an agenda item's function.  The strategy applier uses this hook to
;; erase the agenda display window once all strategy applications have
;; been completed.
;;
(defun agenda-post-hook (func args retval)
   (declare (ignore args retval))
   #-notimer (start-split-timer *agenda-timer*)
   (if (null func)
      (redisplay-window *agenda-window* *empty-agenda-window*)
   )
   #-notimer (stop-split-timer)
   t
)

;;-----------------------------------------------------------------------
;; Perform all initializations needed to prepare the strategy applier
;; for use.
;;
(defun init-strategy-applier ()
   #-nodebug (mdtrace 9 "(init-strategy-applier)")
   #-notimer (setf *resol-timer* (create-split-timer "Resolution Overhead"
                                                     nil))
   #-notimer (setf *recov-timer* (create-split-timer "Recovery Strategies"
                                                     nil))
   #-notimer (setf *strat-timer* (create-split-timer "Strategy Application"
                                                     nil))
   #-notimer
   (if (or *display-agenda* *display-top-candidates*)
      (setf *agenda-timer* (create-split-timer "Agenda/Cand Display" nil))
   )
   ;
   ; initialize the agenda display if the user requested it
   ;
   (when *display-agenda*
      (set-agenda-hooks #'agenda-pre-hook #'agenda-post-hook)
      (setf *agenda-window*
            (display-help-window *agenda-window-pos* *empty-agenda-window*
                                 "Agenda Item"
                                 ))
   )
   ;
   ; initialize the display of the top candidates if the user requested it
   ;
   (if *display-top-candidates*
        (setf *top-cand-window* (display-help-window *top-cand-window-pos*
                                                     *empty-topcand-window*
                                                     "Top Candidates"))
   )

   t
)

;;-------------------------------------------------------------------------
;; Perform all cleanup required by the strategy applier before termination.
;;
(defun shutdown-strategy-applier ()
   #-nodebug (mdtrace 9 "(shutdown-strategy-applier)")
   (remove-help-window *agenda-window*)
   (remove-help-window *top-cand-window*)

   t
)

;;***********************************************************************
;; end of file str-appl.lisp
;;***********************************************************************
```

## B.2. Strategy Reweighting Mechanism

```
;;**************************************************************************
;;*                                                                        *
;;*                              MASTER-D                                  *
;;*   (Multiple Adaptively-weighted STratEgies for Resolving Demonstratives) *
;;*                                                                        *
;;*  File: reweight.lisp       resolution-strategy reweighting             *
;;*  Last Edit:  6 May 93                                                  *
;;*                                                                        *
;;*  Copyright (c) 1993 Ralf Brown.  All Rights Reserved.                  *
;;*  Permission granted for educational and non-commercial research uses.  *
;;*  Other uses require prior permission by Ralf Brown or the CMU Center   *
;;*  for Machine Translation.                                              *
;;*                                                                        *
;;**************************************************************************


;; Note: The reweighting code assumes a number of things about the decay
;; functions in decay.lisp:
;;       1. the functions are monotonically nonincreasing with increasing
;;          distance (for a fixed decay speed).
;;       2. the functions are monotonically nonincreasing with increasing
;;          decay speed (for a fixed distance).
;;       3. a decay speed of 0 results in a constant function (i.e.
;;          weight is independent of distance)

;;-------------------------------------------------------------------------
;; Declare the global variables for this module
;;
(defvar *reweight-timer* nil)
(defvar *strat-weight-window* nil)

;;-------------------------------------------------------------------------
;; Include experimental reweighting code
;;
(load "md:reweighx")

;;-------------------------------------------------------------------------
;; Return a list containing the items to be displayed in the strategy
;; weight window on the screen.
;;
(defun build-strat-weight-display (&aux result)
   (dolist (c *constraint-strategies*)
      (push (format nil "~6,1,,'*F ~3D% ~A" (strategy-weight c)
                                            (round (strategy-accuracy c))
                                            (strategy-name c))
            result)
   )
   (dolist (p *preference-strategies*)
      (push (format nil "~6,1,,'*F ~3D% ~A" (strategy-weight p)
                                            (round (strategy-accuracy p))
                                            (strategy-name p))
            result)
   )
   (intersperse (nreverse result) #\Newline)
)

;;-------------------------------------------------------------------------
;; Determine the two highest scores among the listed votes
;;
```

```
(defun highest-scores (votes &aux hiscore second)
   (setf hiscore -infinity)
   (setf second -infinity)
   (dolist (vote votes)
      (if (> (vote-score vote) hiscore)
         (setf hiscore (vote-score vote))
      ;else
         (if (> (vote-score vote) second)
            (setf second (vote-score vote))
         )
      )
   )
   (values hiscore second)
)

;;----------------------------------------------------------------------
;; Given the list of votes a resolution strategy made for a particular
;; demonstrative, determine whether it voted correctly
;;
(defun correct-vote (votes &aux score hiscore second correct-vote correct-p)
   (multiple-value-setq (hiscore second) (highest-scores votes))
   ;
   ; store the highest score in all vote records
   ;
   (dolist (vote votes)
      (setf (vote-best-weight vote) hiscore)
      (setf (vote-second-best vote) second)
   )
   ;
   ; determine whether the candidate deemed correct got the highest score
   ;
   (setf correct-vote (vote-correct-vote (first votes)))
   (if correct-vote
      (setf score (vote-score correct-vote))
   ;else
      (setf score hiscore)                        ; assume this is correct vote
   )
   (if (eq score 'invalid)
       (setf score most-negative-fixnum)
   )
   (setf correct-p (eql score hiscore))
   #+debug (mdtrace 12 "  (correct-vote ~S) --> ~S" (vote-dem (first votes))
                                                   correct-p)
   correct-p
)

;;----------------------------------------------------------------------
;;  determine the percentage of the time that the strategy voted correctly
;;
(defun hit-rate (strat &aux (correct 0) rate)
   (dolist (dem (strategy-voting-history strat))
      (if (or (null dem) (correct-vote (rest dem)))
         (incf correct)
      )
   )
   (setf rate (percentage correct (length (strategy-voting-history strat))
                          100.0))
   #+debug (mdtrace 11 "(hit-rate ~S) --> ~S" (strategy-name strat) rate)
   rate
)

;;----------------------------------------------------------------------
```

```
;; The simplest reweighting strategy: assign a new weight based on the
;; percentage of times that the strategy has voted accurately in the past.
;;
(defun weight-by-accuracy (strat &aux new-weight)
   ;
   ; return a weight between minus one-half the default weight and the
   ; full default weight
   ;
   (setf new-weight (* *default-strategy-weight*
                       (/ (- (strategy-accuracy strat) 20) 80)))
   ; (accuracy - 20) / 80 --> -0.25..1.0
   #+debug (mdtrace 10 "(weight-by-accuracy ~S) --> ~S (delta ~S)"
                         (strategy-name strat) new-weight
                         (- new-weight (strategy-weight strat)))
   new-weight
)

;;----------------------------------------------------------------------
;; Another simple reweighting strategy: make an incremental adjustment
;; depending on whether the strategy voted correctly for the current
;; resolution.  The incremental adjustment decreases as the number of
;; completed resolutions increases.  The rescaling which is done after each
;; reweighting ensures that the weights of highly-accurate strategies do not
;; grow without bound.
;;
;; The first time the strategy is reweighted, its weight is adjusted up or
;; down by *initial-incremental-change* percent; thereafter, that initial
;; increment is divided by the number of previous reweightings performed on
;; the strategy.
;;
(defun adjust-weight-incrementally (strat &aux vote direction increment new-weig
   (setf increment (/ *initial-incremental-change*
                      (max 1.0 (/ (strategy-reweights strat)
                                  *incremental-change-decay-ratio*))
                      ))
   ;
   ; reweight upward if strategy voted for the correct candidate, down if
   ; it voted against, and leave unchanged if strategy offered no opinion
   ;
   (setf vote (or (first (strategy-votes strat))
                  (caar (strategy-voting-history strat))))
   (if (and vote (vote-correct-vote vote))
      (setf direction (signum (vote-score (vote-correct-vote vote))))
   ;else
      (setf direction 0)
   )
   (setf increment (* increment direction))
   ;
   ; now adjust the weight by the specified increment
   ;
   (setf new-weight (* (+ 1.00 (/ increment 100.0)) (strategy-weight strat)))
   #+debug (mdtrace 10 "(adjust-weight-incrementally ~S) --> ~S (incr ~S)"
                          (strategy-name strat) new-weight increment)
   new-weight
)

;;----------------------------------------------------------------------
;; Yet another simple reweighting strategy: make a fixed-percentage
;; incremental adjustment each time depending on whether the strategy voted
;; correctly or not.  The rescaling which is done after each reweighting
;; ensures that the weights of highly-accurate strategies do not grow without
;; bound.
```

```
;;
(defun fixed-percentage-change (strat &aux vote direction new-weight increment)
   ;
   ; reweight upward if strategy voted for the correct candidate, down if
   ; it voted against, and leave unchanged if strategy offered no opinion
   ;
   (setf vote (or (first (strategy-votes strat))
                  (caar (strategy-voting-history strat))))
   (if (and vote (vote-correct-vote vote))
      (setf direction (signum (vote-score (vote-correct-vote vote))))
   ;else
      (setf direction 0)
   )
   (setf increment (* direction *fixed-change-increment*))
   ;
   ; now adjust the weight by the specified increment
   ;
   (setf new-weight (* (+ 1.00 (/ increment 100.0)) (strategy-weight strat)))
   #+debug (mdtrace 10 "(fixed-percentage-change ~S) --> ~S (~,2F%)"
                     (strategy-name strat) new-weight increment)
   new-weight
)

;;-------------------------------------------------------------------
;; multiply the weights of all of the specified strategies by a scaling factor
;; such that the resulting weights have the given average
;; Note: the &optional is a workaround for a compiler bug which causes
;;   a runtime error if the desired &key is used
;;
(defun rescale-strategies (&optional (avg-weight *default-strategy-weight*)
                                     (constraints *constraint-strategies*)
                                     (preferences *preference-strategies*)
                           &aux (sum 0)
                                (total 0)
                                (count 0) factor
                          )
   ;
   ; add up all the reweightable constraints
   ;
   (dolist (const constraints)
      (if (and (strategy-substrats const) *handle-substrats-individually*)
         (dolist (c (strategy-substrats const))
            (when (/= (strategy-init-weight c) 0.0)
               (incf sum (abs (strategy-weight c)))
               (incf total (strategy-init-weight c))
               (incf count)
            )
         )
      ;else
         (when (/= (strategy-init-weight const) 0.0)
            (incf sum (abs (strategy-weight const)))
            (incf total (strategy-init-weight const))
            (incf count)
         )
      )
   )
   ;
   ; add up all the reweightable preferences (there shouldn't be any unweighted
   ; preferences, but we'll play it safe
   ;
   (dolist (pref preferences)
      (if (and (strategy-substrats pref) *handle-substrats-individually*)
         (dolist (p (strategy-substrats pref))
            (incf sum (abs (strategy-weight p)))
            (incf total (abs (strategy-init-weight p)))
            (if (/= (strategy-init-weight p) 0.0)
               (incf count)
            )
         )
      ;else
         (progn
            (incf sum (abs (strategy-weight pref)))
            (incf total (strategy-init-weight pref))
            (if (/= (strategy-init-weight pref) 0.0)
               (incf count)
            )
         )
      )
   )
   (if (/= avg-weight *default-strategy-weight*)
      (setf total (* avg-weight count))
   )
   (if (= sum 0)
      (setf factor 1.000)              ; no further change if weights total 0
   ;else
      (setf factor (/ total sum))      ; multiplication factor to adjust
   )
   (dolist (const constraints)
      (if (and (strategy-substrats const) *handle-substrats-individually*)
         (dolist (c (strategy-substrats const))
            (if (/= (strategy-init-weight c) 0.0)
               (setf (strategy-weight c) (* (strategy-weight c) factor))
            )
         )
      ;else
         (if (/= (strategy-init-weight const) 0.0)
            (setf (strategy-weight const) (* (strategy-weight const) factor))
         )
      )
   )
   (dolist (pref preferences)
      (if (and (strategy-substrats pref) *handle-substrats-individually*)
         (dolist (p (strategy-substrats pref))
            (setf (strategy-weight p) (* (strategy-weight p) factor))
         )
      ;else
         (setf (strategy-weight pref) (* (strategy-weight pref) factor))
      )
   )
   (setf *total-rescale-factor* (* factor *total-rescale-factor*))
   #-nodebug (mdtrace 5 "(rescale-strategies) --> scaled by factor of ~S"
                                                                  factor)
   t
)

;;------------------------------------------------------------------------
;; Determine whether the specified vote is so old that it is no longer of
;; interest in determining the behavior of the reweighting method.
;;
(defun vote-outdated (vote-list)
   (<= (- *sentence-number* (vote-sentence-number (first vote-list)))
       *max-vote-age*))
)
```

```
;;-----------------------------------------------------------------------
;; Remove the oldest votes for the specified strategy
;;
(defun purge-votes (strategy)
   #+debug (mdtrace 15 "(purge-votes ~S), ~D vote~:P"
                       (strategy-name strategy)
                       (length (strategy-voting-history strategy)))
   #-debug
   (setf (strategy-voting-history strategy)
         (delete-if #'vote-outdated (strategy-voting-history strategy)))
   #+debug
   (let (new-hist)
      (dolist (vote-list (strategy-voting-history strategy))
         (if (not (vote-outdated vote-list))
            (push vote-list new-hist)
         ;else
            (progn
               (unintern-struct vote-list)
               (mdtrace 25 "purged vote ~S" vote-list)
            )
         )
      )
      (setf (strategy-voting-history strategy) (nreverse new-hist))
   )
   (setf (strategy-voting-hist-len strategy) 0)
   (dolist (votes (strategy-voting-history strategy))
      (incf (strategy-voting-hist-len strategy) (length votes))
   )
   #-nodebug (mdtrace 8 "(purge-votes ~S) --> ~D voting record~:P left"
                         (strategy-name strategy)
                         (strategy-voting-hist-len strategy))
   t
)

;;-----------------------------------------------------------------------
;; Remove the oldest votes (those which are no longer of interest) from
;; the voting history, so that we don't waste time examining them while
;; reweighting strategies.
;;
(defun purge-voting-history (strategies)
   #-nodebug (mdtrace 10 "(purge-voting-history) start")
   (dolist (strategy strategies)
      (if *handle-substrats-individually*
         (dolist (strat (strategy-substrats strategy))
            (purge-votes strat)
         )
      )
      (purge-votes strategy)
   )
   #-nodebug (mdtrace 7 "(purge-voting-history) done")
)

;;-----------------------------------------------------------------------
;; Dispatch to the desired weight-computation function, or return the current
;; weight if no function is defined
;;
(defun new-strategy-weight (strat)
   (if (and *strategy-reweighter* (/= (strategy-init-weight strat) 0.0))
      (funcall *strategy-reweighter* strat)
   ;else
      (strategy-weight strat)           ; return unchanged weight if no func
   )                                    ; or strategy not weighted
```

```
)

;;----------------------------------------------------------------------
;; Apply the desired reweighting method to the specified resolution strategy
;; Also maintain statistics on the reweighting for the strategy.
;;
(defun reweight-strategy (strat &aux new-weight)
   #+debug (mdtrace 9 "(reweight-strategy ~S)" (strategy-name strat))
   ;
   ; first, determine how accurate the strategy has been
   ;
;   (setf (strategy-accuracy strat) (hit-rate strat))
   (setf (strategy-accuracy strat) (percentage (strategy-correct-votes strat)
                                               (strategy-total-votes strat)
                                               100.0))
   ;
   ; now invoke the reweighting method on the strategy
   ;
   (setf new-weight (new-strategy-weight strat))
#|
   (if (and (/= (strategy-weight strat) 0.0) ; prevent strategy from being left
            (= new-weight 0.0))              ; out of rescaling
      (setf new-weight *least-positive-weight*)
   )
|#
   ;
   ; see whether we are increasing or decreasing the weight, and adjust the
   ; statistics accordingly
   ;
   (cond
      ((> new-weight (strategy-weight strat))
          (incf (strategy-reweights-inc strat))
          (incf (strategy-reweights-up strat)
                (- new-weight (strategy-weight strat)))
          )
      ((< new-weight (strategy-weight strat))
          (incf (strategy-reweights-dec strat))
          (incf (strategy-reweights-down strat)
                (- (strategy-weight strat) new-weight))
          )
   )
   (incf (strategy-reweights strat))
   ;
   ; finally, update the strategy's weight
   ;
   (setf (strategy-weight strat) new-weight)
   #-nodebug (mdtrace 9 "(reweight-strategy ~S) --> ~S" (strategy-name strat)
                                                         new-weight)
   new-weight
)

;;----------------------------------------------------------------------
;; The actual reweighting mechanism.  Call with a completed resolution.
;; It performs the following steps:
;;      apply the desired reweighting method to each constraint and preference
;;      rescale the strategies to keep the total of all weights constant
;;      remove outdated voting records
;;      optionally display the strategy weights to the user
;;
(defun strategy-reweighter (resolution)
   #-notimer (start-split-timer *reweight-timer*)
   #-nodebug (mdtrace 2 "(strategy-reweighter ~S)" resolution)
   (when (> *sentence-number* *reweight-delay*)
      (if *global-reweighting*
         ;perform multivariate optimization on all strategies and store results
         ;(future enhancement)
      )
      (dolist (const *constraint-strategies*)
         (if (and (strategy-substrats const) *handle-substrats-individually*)
            (dolist (c (strategy-substrats const))
               (reweight-strategy c)
            )
         ;else
            (reweight-strategy const)
         )
      )
      (dolist (pref *preference-strategies*)
         (if (and (strategy-substrats pref) *handle-substrats-individually*)
            (dolist (p (strategy-substrats pref))
               (reweight-strategy p)
            )
         ;else
            (reweight-strategy pref)
         )
      )
      (rescale-strategies)
      (if *display-strat-weights*
         (redisplay-window *strat-weight-window* (build-strat-weight-display))
      )
   )
   (purge-voting-history *constraint-strategies*)
   (purge-voting-history *preference-strategies*)
   (push resolution *completely-processed-evals*)
   #-nodebug (mdtrace 2 "(strategy-reweighter ~S) done" resolution)
   #-notimer (stop-split-timer)
   t
)

;;--------------------------------------------------------------------
;; Perform all initializations needed to prepare the reweighter for use.
;;
(defun init-reweighting ()
   #-notimer (setf *reweight-timer* (create-split-timer "Reweighting"))
   #-nodebug (mdtrace 4 "(init-reweighting)")
   (setf *total-rescale-factor* 1.0)
   (if *autoload-strat-weights*
      (load-strategy-weights *auto-strat-weights-filename*)
   )
   ;
   ; initialize the display of current strategy weights if the user requested
   ; that display
   ;
   (if *display-strat-weights*
      (setf *strat-weight-window*
            (display-help-window *strat-weights-window-pos*
                                 (build-strat-weight-display)
                                 "Resolution Strategy Weights"
                                 ))
   )

   #-notimer (stop-split-timer)
   t
)
```

```
;;----------------------------------------------------------------------------
;; Perform all cleanup required of the reweighter before terminating.
;;
(defun shutdown-reweighting ()
   #-notimer (start-split-timer *reweight-timer*)
   #-nodebug (mdtrace 4 "(shutdown-reweighting)")
   (remove-help-window *strat-weight-window*)
   (if *autosave-strat-weights*
      (save-strategy-weights *auto-strat-weights-filename*)
   )
   #-notimer (stop-split-timer)
   t
)

;;****************************************************************************
;; end of file reweight.lisp
;;****************************************************************************
```

## B.3. Resolution Strategies

```
;;***********************************************************************
;;*                                                                     *
;;*                              MASTER-D                               *
;;*   (Multiple Adaptively-weighted STratEgies for Resolving Demonstratives) *
;;*                                                                     *
;;*  File: strategies.lisp      resolution strategies for MASTER-D      *
;;*  Last Edit: 17 Apr 93                                               *
;;*                                                                     *
;;*  Copyright (c) 1993 Ralf Brown.  All Rights Reserved.               *
;;*  Permission granted for educational and non-commercial research uses. *
;;*  Other uses require prior permission by Ralf Brown or the CMU Center *
;;*  for Machine Translation.                                           *
;;*                                                                     *
;;***********************************************************************

;;---------------------------------------------------------------------
;;  define the substrategies for the local-constraints strategy
;;
(defvar *local-const-strats*)
(setf *local-const-strats*
        (list (make-strategy
                 :name "Gender"
                 :func 'gender-constraint
              )
              (make-strategy
                 :name "Number"
                 :func 'number-constraint
              )
              (make-strategy
                 :name "Animacy"
                 :func 'animacy-constraint
              )
              (make-strategy
                 :name "Humanness"
                 :func 'humanness-constraint
              )
        ))

;;---------------------------------------------------------------------
;;  define the substrategies for the salience strategy
;;
(defvar *salience-strats*)
(setf *salience-strats*
        (list (make-strategy
                 :name "Syntactic Topicalization"
                 :func 'syntactic-topicalization
              )
              (make-strategy
                 :name "Focus Preference"
                 :func 'focus-preference
              )
        ))

;;---------------------------------------------------------------------
;;  define the constraint strategies to be used by the demonstrative
;;  resolver.  The strategies will be invoked in the order in which they
;;  are listed here.
;;
(defvar *default-constraint-strategies*)
```

```
(setf *default-constraint-strategies*
        (list (make-strategy
                  :name "Local Constraints"
                  :func 'local-constraints
                  :substrats *local-const-strats*
                  :decay-func #'no-decay
              )
              (make-strategy
                  :name "Case-Role Constraints"
                  :func 'case-role-constraints
              )
              (make-strategy
                  :name "Accessible Referents"
                  :func 'accessible-referents
                  :init-weight 0.0  ; only returns 0 and invalid
                  :weight 0.0       ; must be same as init-weight
                  :priming-func 'prime-accessibility
                  :update-func 'update-accessibility
              )
              (make-strategy
                  :name "Reference Type"
                  :func 'reference-type
              )
              (make-strategy
                  :name "World-Model Constraints"
                  :func 'world-model-constraints
                  :update-func 'update-world-model
              )
        ))
(defvar *constraint-strategies* *default-constraint-strategies*)

;;-------------------------------------------------------------------
;;  define the preference strategies used by the demonstrative resolver.
;;  They will be invoked in the order in which they are listed here.
;;
(defvar *default-preference-strategies*)
(setf *default-preference-strategies*
        (list (make-strategy
                  :name "Proximity"
                  :func 'proximity
              )
              (make-strategy
                  :name "Recency"
                  :func 'recency
                  :optimize-decay t
                  :decay-func #'linear-decay
                  :decay-speed 0.25
                  :decays-to -1.0
              )
              (make-strategy
                  :name "Case-Role Persistence"
                  :func 'case-role-persistence
                  :optimize-decay t
                  :decay-func #'linear-decay
                  :decay-speed 0.25
                  :decays-to 0.0
              )
              (make-strategy
                  :name "Salience"
                  :func 'salience
                  :substrats *salience-strats*
              )
        ))
(defvar *preference-strategies* *default-preference-strategies*)

;;----------------------------------------------------------------------
;;  define the strategies to be invoked when recovering from a resolution
;;  which resulted in zero candidates remaining.  They will be applied in
;;  the order in which they are listed here.
;;
(defvar *default-recovery0-strategies*)
(setf *default-recovery0-strategies*
        (list (make-strategy
                 :name "Extralinguistic Reference"
                 :func 'extralinguistic-reference
              )
              (make-strategy
                 :name "Metonyms"
                 :func 'metonyms
              )
              (make-strategy
                 :name "Relax Constraints"
                 :func 'relax-constraints
              )
        ))
(defvar *recovery0-strategies* *default-recovery0-strategies*)

;;----------------------------------------------------------------------
;;  define the strategies to be invoked when a resolution results in
;;  multiple remaining candidates.  They will be invoked in the order
;;  in which they are listed here.
;;
(defvar *default-recovery2-strategies*)
(setf *default-recovery2-strategies*
        (list (make-strategy
                 :name "Ask the User"
                 :func 'ask-the-user
              )
        ))
(defvar *recovery2-strategies* *default-recovery2-strategies*)

;;**********************************************************************

;;----------------------------------------------------------------------
;; Given a strategy's name and a list of strategies, return the strategy
;; or NIL if it is not among the given list of strategies
;;
(defun find-strategy-by-name-aux (name strategies &aux strategy)
   (dolist (strat strategies)
      (cond
         ((string-equal name (strategy-name strat))
                (setf strategy strat)
                (return)                ; terminate loop on match
                )
         ((strategy-substrats strat)
                (dolist (substrat (strategy-substrats strat))
                   (when (string-equal name (strategy-name substrat))
                      (setf strategy substrat)
                      (return)
                   )
                )
                (if strategy
                   (return)             ; terminate loop on match
                ))
      )
   )
   strategy
)

;;-----------------------------------------------------------------------
;; Given a strategy's name, return the strategy or NIL if it does not
;; exist
;;
(defun find-strategy-by-name (name &aux strategy)
   ;
   ; first, scan the constraint strategies and all their substrategies for a
   ; match with the given name.  If no match is found, repeat the process for
   ; preferences and recovery strategies until a match is found or we run out
   ; of strategies to check
   ;
   (cond
      ((setf strategy (find-strategy-by-name-aux name *constraint-strategies*))
               )
      ((setf strategy (find-strategy-by-name-aux name *preference-strategies*))
               )
      ((setf strategy (find-strategy-by-name-aux name *recovery0-strategies*))
               )
      ((setf strategy (find-strategy-by-name-aux name *recovery2-strategies*))
               )
   )
   #+debug (mdtrace 20 "(find-strategy-by-name ~S) --> ~S" name strategy)
   strategy
)

;;-----------------------------------------------------------------------
;; Determine whether the candidate referent has a compatible gender, and
;; provide a preference value if so.
;;
(defun gender-constraint (demonstrative candidate
                          &aux g1 g2 score)
   (setf g1 (gender-of demonstrative))
   (setf g2 (gender-of candidate))
   (cond
      ((or (equal g1 'indet) (equal g2 'indet))
               ; indeterminate gender matches anything
               (setf score 0.0))
      ((equal g1 g2)
               ; full weight if identical genders
               (setf score 1.0))
      ((subsetp g2 g1)
               ; half weight if gender possibilities for demonstrative are
               ; a superset of the candidate's possibilities
               (setf score 0.5))
      ((overlap g1 g2)
               ; quarter weight if some overlap in possibilities
               (setf score 0.25))
      (t       (setf score 'invalid))
   )
   #-nodebug (mdtrace 8 "(gender-constraint ~S ~S) --> ~S" demonstrative
                                                           candidate score)
   score
)

;;-----------------------------------------------------------------------
;; Determine whether the candidate referent has a compatible syntactic
;; number, and provide a preference value if so.
```

```
;;
(defun number-constraint (demonstrative candidate
                          &aux n1 n2 score)
   (setf n1 (number-of demonstrative))
   (setf n2 (number-of candidate))
   (cond
      ((or (equal n1 'indet) (equal n2 'indet))
                ; indeterminate number matches anything
                (setf score 0.0))
      ((equal n1 n2)
                ; assign full weight if identical numbers
                (setf score 1.0))
      ((subsetp n2 n1)
                ; half weight if number possibilities for demonstrative are
                ; a superset of the candidate's possibilities
                (setf score 0.5))
      ((overlap n1 n2)
                ; quarter weight if some overlap in possibilities
                (setf score 0.25))
      (t        (setf score 'invalid))
   )
   #-nodebug (mdtrace 8 "(number-constraint ~S ~S) --> ~S" demonstrative
                                                          candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the candidate referent has a compatible animacy
;; attribute, and provide a preference value if so.
;;
(defun animacy-constraint (demonstrative candidate
                          &aux a1 a2 score)
   (setf a1 (animacy-of demonstrative))
   (setf a2 (animacy-of candidate))
   (cond
      ((or (equal a1 'indet) (equal a2 'indet))
                ; indeterminate animacy matches anything
                (setf score 0.0))
      ((equal a1 a2)
                ; assign full weight if identical animacies
                (setf score 1.0))
      ((subsetp a2 a1)
                ; half weight if animacy possibilities for demonstrative are
                ; a superset of the candidate's possibilities
                (setf score 0.5))
      ((overlap a1 a2)
                ; quarter weight if some overlap in possibilities
                (setf score 0.25))
      (t        (setf score 'invalid))
   )
   #-nodebug (mdtrace 8 "(animacy-constraint ~S ~S) --> ~S" demonstrative
                                                           candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the candidate referent has a compatible humanness
;; attribute, and provide a preference value if so.
;;
(defun humanness-constraint (demonstrative candidate
                          &aux h1 h2 score)
   (setf h1 (humanness demonstrative))
   (setf h2 (humanness candidate))
   (cond
      ((or (equal h1 'indet) (equal h2 'indet))
                ; indeterminate humanness matches anything
                (setf score 0.0))
      ((equal h1 h2)
                ; assign full weight if identical
                (setf score 1.0))
      ((subsetp h2 h1)
                ; half weight if humanness possibilities for demonstrative are
                ; a superset of the candidate's possibilities
                (setf score 0.5))
      ((overlap h1 h2)
                ; quarter weight if some overlap in possibilities
                (setf score 0.25))
      (t        (setf score 'invalid))
   )
   #-nodebug (mdtrace 8 "(humanness-constraint ~S ~S) --> ~S" demonstrative
                                                              candidate score)
   score
)

;;---------------------------------------------------------------------
;; Apply the local constraint strategies if substrategies are not applied
;; individually.
;;
(defun local-constraints (demonstrative candidate)
   #-nodebug (mdtrace 8 "(local-constraints ~S ~S)" demonstrative candidate)
   (apply-substrategies demonstrative candidate *local-const-strats*)
)

;;---------------------------------------------------------------------
;; Determine whether the candidate referent would violate any case-role
;; constraints imposed by the utterance containing the demonstrative.
;;
(defun case-role-constraints (demonstrative candidate
                              &aux proposition case score)
   (setf proposition (proposition-of demonstrative))
   (setf case (caserole-of demonstrative proposition))
   (if (compatible-caserole-filler proposition case candidate)
      (setf score 0.0)   ; no preference, for now
   ;else
      (setf score 'invalid)
   )
   #-nodebug (mdtrace 8 "(case-role-constraints ~S ~S) --> ~S" demonstrative
                                                                candidate score)
   score
)

;;---------------------------------------------------------------------
;; Determine whether the candidate referent is deemed accessible by the
;; current state of the discourse model.
;;
(defun accessible-referents (demonstrative candidate &aux access score)
   (setf access (or (global-candidate-p candidate)
                    (referent-accessible-p demonstrative candidate)
                    ))
   (if access
      (if (numberp access)
         (setf score access)
      ;else
         (setf score 0.0)
         )
      ;else
         (setf score 'invalid)
      )
      #-nodebug (mdtrace 8 "(accessible-referents ~S ~S) --> ~S" demonstrative
                                                                  candidate score)
      score
   )

   ;;-----------------------------------------------------------------------
   ;; Determine what type of reference is being made; returns a list of the
   ;; possible types or 'indet if the possibilities cannot be determined
   ;; Note: the set of types must be the same as that used by object-type-of
   ;;      below
   ;;
   (defun reference-type-of (item &aux types result)
      (setf result (get-reftypes item))
      (when (not result)
         ;
         ; if reftypes was previously set, use its current value;
         ; else, try to figure out the appropriate types and then
         ;   ask the user to verify/correct the selected types
         ;   and finally store the result so we don't have to ask
         ;   again
         ;
         (setf types (determine-reference-type item))
         (if (= (length types) 1)
            (setf result types)
         ;else
            (setf result (ask-user-for-reference-type item types))
         )
         (set-reftypes item result)
      )
      #+debug (mdtrace 13 "(reference-type-of ~S) --> ~S" item result)
      result
   )

   ;;-----------------------------------------------------------------------
   ;; Determine what type of object the given item is; returns a list of the
   ;; possible types or 'indet if the possibilities cannot be determined
   ;; Note: the set of types must be the same as that used by
   ;;      reference-type-of above
   ;;
   (defun object-type-of (item &aux types result)
      (setf result (or (get-objtypes item) (get-reftypes item)))
      (when (not result)
         ;
         ; if objtypes/reftypes was previously set, use the current value;
         ; else, try to figure out the appropriate types and then
         ;   ask the user to verify/correct the selected types
         ;   and finally store the result so we don't have to ask
         ;   again
         ;
         (setf types (determine-object-type item))
         (if (= (length types) 1)
            (setf result types)
         ;else
            (setf result (ask-user-for-entity-type item types))
         )
         (set-objtypes item result)
      )
      #+debug (mdtrace 13 "(object-type-of ~S) --> ~S" item result)
   result
)

;;-------------------------------------------------------------------
;;
(defun reference-type (demonstrative candidate &aux r1 r2 score)
   (setf r1 (reference-type-of demonstrative))
   (setf r2 (object-type-of candidate))
   (cond
      ((or (equal r1 'indet) (equal r2 'indet))
                 ; indeterminate ref-type matches anything, so no preference
                 (setf score 0.0))
      ((equal r1 r2)
                 ; assign full weight if identical types
                 (setf score 1.0))
      ((overlap r1 r2)
                 ; assign quarter weight if some overlap in possibilities
                 (setf score 0.25))
      (t         (setf score 'invalid))
   )
   #-nodebug (mdtrace 8 "(reference-type ~S ~S) --> ~S" demonstrative
                                                  candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the candidate referent would violate any constraints
;; imposed by the current state of the world model, and provide a preference
;; value if no constraints are violated.
;;
(defun world-model-constraints (demonstrative candidate &aux score relation)
   (setf relation (world-model-relation demonstrative candidate))
   (if relation
      (setf score 0.5)                ; some preference if a relation exists
   ;else
      (setf score 0.0)                ; no preference if no relationship
   )
   ;
   ; determine whether the requirements of the dem&cand violate either the
   ; relationship posited by the world model or any other relationships in
   ; the world model
   ;

   #-nodebug (mdtrace 8 "(world-model-constraints ~S ~S) --> ~S" demonstrative
                                                       candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the demonstrative and candidate referent have the
;; same conceptual distance ('near' vs. 'far').
;;
(defun proximity (demonstrative candidate &aux p1 p2 score score2)
   (setf p1 (perceived-spatial-distance-of demonstrative t))
   (setf p2 (perceived-spatial-distance-of candidate nil))
   (cond
      ((or (equal p1 'indet) (equal p2 'indet))
                 ; indeterminate distance matches anything
                 (setf score 0.0))
      ((equal p1 p2)
                 ; assign full weight if identical distances
                 (setf score 1.0))
      ((subsetp p2 p1)
                ; half weight if distance possibilities for demonstrative are
                ; a superset of the candidate's possibilities
                (setf score 0.5))
      ((overlap p1 p2)
                ; assign quarter weight if some overlap in possibilities
                (setf score 0.25))
      (t        (setf score 'invalid))
   )
   (setf p1 (perceived-temporal-distance-of demonstrative t))
   (setf p2 (perceived-temporal-distance-of candidate nil))
   (cond
      ((or (equal p1 'indet) (equal p2 'indet))
                ; indeterminate distance matches anything
                (setf score2 0.0))
      ((equal p1 p2)
                ; full weight if identical distances
                (setf score2 1.0))
      ((subsetp p2 p1)
                ; half weight if distance possibilities for demonstrative are
                ; a superset of the candidate's possibilities
                (setf score2 0.5))
      ((overlap p1 p2)
                ; assign quarter weight if some overlap in possibilities
                (setf score2 0.25))
      (t        (setf score2 'invalid))
   )
   (if (or (equal score 'invalid) (equal score2 'invalid))
      (setf score 'invalid)
   ;else
      (setf score (/ (+ score score2) 2.0))     ; take average
   )
   #-nodebug (mdtrace 8 "(proximity ~S ~S) --> ~S" demonstrative candidate
                                                    score)
   score
)

;;-----------------------------------------------------------------------
;; Determine the distance between the demonstrative and the specified
;; candidate referent.
;;
(defun recency (demonstrative candidate)
   (declare (ignore demonstrative candidate))
   #-nodebug (mdtrace 8 "(recency ~S ~S) --> 1.0" demonstrative candidate)
   1.0  ; always return full weight, decay function handles all changes in
        ; weight with increasing distance
)

;;-----------------------------------------------------------------------
;; Determine the strength of the case-role persistence effect for the
;; given candidate referent.
;;
(defun case-role-persistence (demonstrative candidate
                              &aux case-dem case-cand score)
   (declare (ignore demonstrative candidate))
   (setf case-dem (caserole-of demonstrative (proposition-of demonstrative)))
   (setf case-cand (caserole-of candidate (proposition-of candidate)))
   (if (equal case-dem case-cand)
      (setf score 1.0)
   ;else
      (setf score 0.0)
   )
   ;
   ; we will always return either the full weight or zero, and allow the
   ; decay function to handle the diminishing weight with increasing distance
   ;
   #-nodebug (mdtrace 8 "(case-role-persistence ~S ~S) --> ~S" demonstrative
                                                              candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the given candidate is syntactically topicalized.
;;
(defun syntactic-topicalization (demonstrative candidate
                                 &aux score t1 t2)
   (setf t1 (topicalized-p demonstrative))
   (setf t2 (topicalized-p candidate))
   (cond
      ((and t1 t2)      (setf score 1.0) ; full preference if both topicalized
                        )
      (t2               (setf score 0.5) ; half preference if cand. topicalized
                        )
      (t                (setf score 0.0) ; no preference otherwise
                        )
   )
   #-nodebug (mdtrace 8 "(syntactic-topicalization ~S ~S) --> ~S"
                                                 demonstrative candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine whether the given candidate is in the focus of its sentence.
;;
(defun focus-preference (demonstrative candidate
                         &aux score score1 score2 f1 f2)
   (setf f1 (focused-p demonstrative))
   (setf f2 (focused-p candidate))
   (cond
      ((and f1 f2)      (setf score1 1.0) ; full preference if both in focus
                        )
      (f2               (setf score1 0.5) ; half preference if candid. in focus
                        )
      (t                (setf score1 0.0) ; no preference otherwise
                        )
   )
   (setf f1 (number-of-references candidate))
   (if (> f1 0)
      (setf score2 (min 1.0 (log f1)))
   ;else
      (setf score2 0.0)
   )
   (setf score (/ (+ score1 score2) 2)) ; result is average of partial scores
   #-nodebug (mdtrace 8 "(focus-preference ~S S) --> ~S"
                                                demonstrative candidate score)
   score
)

;;-------------------------------------------------------------------
;; Determine how salient the given candidate referent is.
;;
(defun salience (demonstrative candidate)
   #-nodebug (mdtrace 8 "(salience ~S ~S)" demonstrative candidate)
   (apply-substrategies demonstrative candidate *salience-strats*)
```

```
)

;;---------------------------------------------------------------------
;; (returns a list of the new candidates added during the recovery action)
;;
(defun extralinguistic-reference (res)
   (declare (ignore res))
   #-nodebug (mdtrace 8 "(extralinguistic-reference)")

   nil ;for now
)

;;---------------------------------------------------------------------
;; (returns a list of the new candidates added during the recovery action)
;;
(defun metonyms (res &aux added)
   #-nodebug (mdtrace 8 "(metonyms)")
   (dolist (cand (resol-candidates res))
      (dolist (add (ontological-metonyms cand))
         (pushnew add added)
      )
   )

   added
)

;;---------------------------------------------------------------------
;; (returns a list of the new candidates added during the recovery action)
;;
(defun relax-constraints (res)
   (declare (ignore res))
   #-nodebug (mdtrace 8 "(relax-constraints)")

   nil ;for now
)

;;---------------------------------------------------------------------
;; (returns the list of candidates remaining after the recovery action)
;;
(defun ask-the-user (res &aux cands result)
   #-nodebug (mdtrace 8 "(ask-the-user) start")
   (setf cands (mapcar #'evalrec-candidate-ref (resol-best-cand res)))
   (setf result (ask-user-for-referent res cands))
   #-nodebug (mdtrace 8 "(ask-the-user) --> ~S" result)
   result
)

;;*********************************************************************
;; end of file strategies.lisp
;;*********************************************************************
```

# Appendix C
# Knowledge Base

This appendix contains a portion of the ontology to illustrate its format and the knowledge contained within it. The extract shown here consists generally of the top of the heirarchy.

Each node in the ontological heirarchy is represented as a FrameKit frame, and stored in the ontology file as a `make-frame` or `make-frame-old` form, which would allow the ontology to be read into the system with a simple `load` statement. For security, MASTER-D reads the file one form at a time and only evaluates `make-frame` and `make-frame-old` forms.

All nodes inherit from their parents via the usual FrameKit inheritance mechanism, allowing many nodes to consist of nothing more than an `is-a` link to the parent node.

In addition to the *is-a* slot for inheritance and slots for number, gender, animacy, and similar information, MASTER-D makes use of several special slots for the world modeler, case-role constraints strategy, and reference type strategy. These slots are

- `:CR$CONSTRAINTS` Specifies the types of fillers permitted for various slots of the frame.
- `:MD$REFTYPES` Specifies the type of reference which may be made by a demonstrative in particular circumstances.
- `:WM$PRECOND` Specifies the preconditions on an action.
- `:WM$EFFECT` Specifies the effects on the world model of a particular action.

The first special slot, `:CR$CONSTRAINTS`, specifies for one or more slots in the frame what values are permitted to fill each of those slots. The format is a list of lists, with the first element of each sublist naming the slot and the remaining elements specifying the allowable types. In order for a value to be a valid filler for a slot, it must be a subclass of at least one of the classes named in the list for that slot. For example, the knowledge base entry

```
(make-frame *make
        (is-a *action)
        (:cr$constraints (agent *agent))
        )
```

specifies that the filler of the `agent` slot for any parse frame which `is-a` `*MAKE` must be `is-a` either `*AGENT` or a subclass of `*AGENT`. There are no restrictions on any of the other slots of such a frame. In addition, inheritance rules ensure that the constraint applies to any frame which `is-a` any subclass of `*MAKE` which does not override the `:CR$CONSTRAINTS` slot.

The :MD$REFTYPES slot in ontology frames specifies the taxonomic classes of fillers for a specified slot, and thus the allowable reference types for a demonstrative in that position. The format of this slot is a list of specifications, each consisting of a slot name, a list of allowable reference types, and one or more slot-value lists. Given the following entry in the ontology,

```
(make-frame *mean
        (is-a *proposition*)
        (:md$reftypes
             (theme (lexical-ref discourse-ref proposition)
                    (number singular) (distance near))
             )
        )
```

the system may infer that the reference type of a demonstrative in the theme slot of a parse frame which is-a *MEAN is one of lexical, discourse, or propositional reference if the demonstrative is a singular near demonstrative (i.e. *this*).

The :WM$PRECOND and :WM$EFFECT slots together provide the information needed by the world modeler to update its world model. As with the previous special slots, the format is a list of specifications; for :WM$PRECOND, the specifications are world model relationships, and for :WM$EFFECT the specifications are commands to add/delete relations or objects or to set variable bindings. The following example entry from the knowledge base shows how to express the idea that theft is the act of taking possession of something one does not own:

```
(make-frame-old *theft
        (is-a *action)
        (:wm$precond (bind agent :s agent)
                     (bind theme :s theme)
                     (not own :v agent :v theme)
                     (not possess :v agent :v theme))
        (:wm$effect
             (assert ownership possess :v agent :v theme))
        )
```

The above frame also shows the use of substitutions. Before actually evaluating the preconditions or effects, all substitutions are made by replacing occurrences of :S and the following element with the current value of that slot in the parse frame being processed, and occurrences of :V and the following element with the current binding of the named variable. The BIND commands shown at the beginning of the :WM$PRECOND slot set the variables which are later referenced. For the simple substitutions shown here, use of a direct slot substitution would be about as fast in execution as the variable binding and substitution shown here, but a slot specification can be more complicated. When multiple levels of indirection through frames are involved, it is more efficient to bind the result to a variable if it will be used multiple times. Not shown here are the :O and :R substitutions, which are replaced with the last object(s) and relation(s), respectively, to be added or deleted by the :WM$EFFECT statements.

For this example, the preconditions specify that there no own or possess relationships between the agent and theme exist in the current world model. Provided that there is a match (i.e. none of the preconditions are violated), the specifications in the :WM$EFFECT slot are applied one by one. In this case, the single specification is to add a new relation to the world model--an ownership relation of type *possess*[11] between the agent and theme of the action.

[11]The world model distinguishes between having an item in one's possession and owning the item.

## Knowledge Base

```
;LastEdit: 19 Apr 93

(make-frame *all
        (animate -)
        (human -)
        (number singular)
        (gender indet)
        (:cr$constraints (agent *agent)
                         (manner *manner)
                         (intensity *intensity)
                         (time *time-spec)
                         (size *size)
                         )
        )

(make-frame *lexical-reference*
        (is-a *all)
        (gender neuter)
        (:conceptual_class lexical-ref)
        )

(make-frame *discourse-reference*
        (is-a *all)
        (gender neuter)
        (:conceptual_class discourse-ref)
        )

(make-frame *object
        (is-a *all)
        (animate indet)
        (:conceptual_class object)
        )

(make-frame *property*
        (is-a *all)
        (:conceptual_class property)
        )

(make-frame *quantity*
        (is-a *property*)
        )

(make-frame *action
        (is-a *all)
        (part-of-speech noun)
        (:conceptual_class action)
        )

(make-frame *event
        (is-a *all)
        (:conceptual_class event)
        )

(make-frame *proposition*
        (is-a *all)
        (:conceptual_class proposition)
        )

(make-frame *time-spec
        (is-a *all)
        (:conceptual_class temporal)
        )

(make-frame *location
        (is-a *all)
        (:conceptual_class locative)
        (:conceptual_objtype location)
        )

(make-frame *manner
        (is-a *property*)
        )

(make-frame *demonstrative*
        (reference demonstrative)
        )

(make-frame *this
        (is-a *demonstrative*)
        (number singular)
        (distance near)
        (reference demonstrative)
        (root "this")
        (human -)
        )

(make-frame *that
        (is-a *demonstrative*)
        (number singular)
        (distance far)
        (reference demonstrative)
        (root "that")
        (human -)
        )

(make-frame *these
        (is-a *demonstrative*)
        (number plural)
        (distance near)
        (reference demonstrative)
        (root "these")
        (human) ; override *all
        )

(make-frame *those
        (is-a *demonstrative*)
        (number plural)
        (distance far)
        (reference demonstrative)
        (root "those")
        (human) ; override *all
        )

(make-frame *here
        (is-a *demonstrative*)
        (number singular)
        (distance near)
        (reference demonstrative)
        ($reftypes$ locative)
        (root "here")
        )

(make-frame *there
        (is-a *demonstrative*)
        (distance far)
        (reference demonstrative)
        ($reftypes locative)
        (root "there")
        )

(make-frame *now
        (is-a *demonstrative*)
        (distance near)
        (reference demonstrative)
        ($reftypes$ temporal)
        (root "now")
        )

(make-frame *text-string*
        (is-a *lexical-reference*)
        )

(make-frame *predication*
        (is-a *proposition*)
        )

(make-frame *physical-object
        (is-a *object)
        (:conceptual_objtype physical-object)
        )

(make-frame *mental-object
        (is-a *object)
        (:conceptual_objtype mental-object)
        )

(make-frame *animate-object
        (is-a *object)
        (animate +)
        )

(make-frame *inanimate-object
        (is-a *object)
        (animate -)
        )

(make-frame *agent
        (is-a *animate-object)
        )

(make-frame *mankind
        (is-a *animate-object)
        (instances *person)
        )

(make-frame *person
        (is-a *agent)
        (instance-of *mankind)
        (human +)
        (gender male female)
        )

(make-frame *professional-person
        (is-a *person)
        )

(make-frame *everyone
        (is-a *person)
        (number plural)
        )

(make-frame speaker
        (is-a *person)
        )

(make-frame *lawyer
        (is-a *professional-person)
        (profession *law)
        )

(make-frame *musician
        (is-a *professional-person)
        (profession *music)
        (similar-to *composer)
        )

(make-frame *composer
        (is-a *professional-person)
        (profession *music)
        (similar-tc *musician)
        )

(make-frame *economist
        (is-a *professional-person)
        (profession *economy)
        )

(make-frame *lobbyist
        (is-a *professional-person)
        (profession *lobbying)
        )

(make-frame *manager
        (is-a *professional-person)
        (profession *manage)
        )

(make-frame *place
        (is-a *location)
        )

(make-frame *council
        (is-a *object)
        (has-parts *person)
        )

(make-frame *text
        (is-a *inanimate-object)
        (has-parts *phrase *word)
        )

(make-frame *word
        (is-a *text)
        (part-of *text)
        )

(make-frame *letter-of-alphabet
        (is-a *text)
        (part-of *phrase)
        )

(make-frame *phrase
        (is-a *text)
        (part-of *text)
        (has-parts *letter-of-alphabet)
        )

(make-frame *belief
        (is-a *mental-object)
        (:conceptual_objtype belief)
        )

(make-frame *user-interface
        (is-a *inanimate-object)
        (has-parts *ui-window *ui-pointer *menu-system *ui-element)
        )

(make-frame *display-object
        (is-a *inanimate-object)
        )

(make-frame *ui-element
        (is-a *display-object)
        (part-of *user-interface)
        )

(make-frame *ui-window
        (is-a *display-object)
        (part-of *user-interface)
        )

(make-frame *ui-pointer
        (is-a *display-object)
        (part-of *user-interface)
        )

(make-frame *menu-system
        (is-a *display-object)
        (part-of *user-interface)
        (has-parts *menu)
        )

(make-frame *menu
        (is-a *display-object)
        (part-of *menu-system)
        (similar-to *menu-bar)
        )

(make-frame *pull-down-menu
        (is-a *menu)
        )

(make-frame *menu-bar
        (is-a *display-object)
        (similar-to *menu)
        )

(make-frame *economy
        (is-a *inanimate-object)
        (professional *economist)
        )

(make-frame *law
        (is-a *object)
        (professional *lawyer)
        (similar-to *o-rule)
        )

(make-frame *music
        (is-a *inanimate-object)
        (professional *musician *composer)
        )

(make-frame *musical-composition
        (is-a *music)
        (has-parts *musical-note)
        )

(make-frame *symphony
        (is-a *musical-composition)
        (has-parts *symphonic-movement)
        )

(make-frame *musical-note
        (is-a *inanimate-object)
        (part-of *musical-composition)
        )

(make-frame *symphonic-movement
        (is-a *musical-composition)
        (part-of *symphony)
        )

(make-frame *software
        (is-a *object)
        )

(make-frame *Lotus-1-2-3
        (is-a *software)
        (similar-to *Lotus-Corporation)
        )

(m ke-frame *question
        (is-a *object)
        (:cr$constraints (theme *proposition*))
        )

(make-frame *lobbying
        (is-a *object)
        (professional *lobbyist)
        )

(make-frame *creator
        (is-a *agent)
        )

(make-frame *company
        (is-a *agent)
        )

(make-frame *accept
        (is-a *action)
        (:md$reftypes (object (proposition)
                              (theme *boolean-value))
                      )
        )

(make-frame *a-reward
        (is-a *action)
        (:cr$constraints (agent *agent) (patient *agent)
                         (theme *object *action *event *proposition*))
        )

(make-frame *take
        (is-a *action)
        (:cr$constraints (agent *agent) (object *object)
                         (location *location))
        (:wm$precond (location :any :s theme :s goal))
        (:wm$effect (bind obj :s obj)
                    (add nil :v obj)       ; make sure obj is in w-model
                    (retract location :any :v obj) ; obj no longer there
                    (assert location nil :v obj :s goal)
                    (assert ownership possess :s agent :v obj)
                    )
        )
```

```
(make-frame *throw
        (is-a *action)
        (:cr$constraints (agent *agent) (object *object)
                         (goal *location))
        )

(make-frame *support
        (is-a *action)
        (:cr$constraints (theme *object) (intensity *intensity))
        )

(make-frame *look
        (is-a *action)
        (root "look")
        )

(make-frame *look-at
        (is-a *action)
        (root "look at")
        (:md$reftypes (theme (object locative))
                              )
        )

(make-frame *speak
        (is-a *action)
        (root "speak")
        (:cr$constraints (agent *agent) (manner *manner))
        (:md$reftypes (theme (proposition discourse-ref lexical-ref))
                       )
        )

(make-frame *to-duplicate
        (is-a *action)
        (:cr$constraints (agent *agent) (theme *object *action))
        )

(make-frame *move
        (is-a *action)
        (:cr$constraints (agent *agent) (theme *object)
                         (goal *location))
        (:wm$precond (not location :any :s theme :s goal))
        (:wm$effect (bind obj :s theme)  ; obj is filler of theme slot
                    (add nil :v obj)     ; make sure obj is in w-model
                    (retract location :any :v obj) ; obj no longer there
                    (assert location nil :v obj :s goal)
                    )
        )

(make-frame *a-give
        (is-a *action)
        (:cr$constraints (agent *agent) (theme *object) (patient *agent)
                         (goal *location))
        (:wm$precond (not location :any :s theme :s goal))
        (:wm$effect (bind obj :s theme)  ; obj is filler of theme slot
                    (add nil :v obj)     ; make sure obj is in w-model
                    (retract location :any :v obj) ; obj no longer there
                    (assert location nil :v obj :s goal)
                    )
          )

(make-frame *theft
          (is-a *action)
          (:wm$precond (bind agent :s agent)
                       (bind theme :s theme)
                       (not own :v agent :v theme)
                       (not possess :v agent :v theme)
                       )
          (:wm$effect (assert ownership possess :v agent :v theme))
          )

(make-frame *believe
          (is-a *action)
          (:cr$constraints (agent *agent)
                           (theme *belief *proposition*)
                           (time *time-spec)
                           (location *location)
                           )
          (:md$reftypes (theme (object proposition))
                         )
          (:wm$effect (add belief :s theme)
                      (assert believe nil :s agent :o)
                      )
          )

(make-frame *go-to
          (is-a *action)
          (:wm$precond (location :any :s agent :s location))
          (:wm$effect (add goal :s goal)           ; ensure goal in wm
                      (retract location :any :s agent)
                      (assert location nil :s agent :s goal)
                      )
          )

(make-frame *put
          (is-a *action)
          (:wm$precond (ownership possess :s agent :s object))
          (:wm$effect (retract ownership possess :s agent :s object)
                      (assert location nil :s object :s goal)
                      )
          )

(make-frame *know
          (is-a *action)
          (:wm$effect (add know :s theme)
                      (assert believe know :s agent :o)
                      )
          )

(make-frame *let
          (is-a *action)
          (similar-to *allow)
          )

(make-frame *allow
        (is-a *action)
        (similar-to *let)
        )

(make-frame *manage
        (is-a *action)
        (professional *manager)
        )

(make-frame *is
        (is-a *predication*)
        (:md$reftypes (object (lexical-ref)
                              (theme *text))
                      (object (proposition)
                              (theme *boolean-value))
                      )
        )

(make-frame *located-in
        (is-a *predication*)
        (:md$reftypes (object (object))
                      (location (locative))
                      )
        )

(make-frame *boolean-value
        (is-a *predication*)
        )

(make-frame *true
        (is-a *boolean-value)
        )

(make-frame *false
        (is-a *boolean-value)
        )

(make-frame *mean
        (is-a *proposition*)
        (:md$reftypes (theme (lexical-ref discourse-ref proposition)
                             (number singular) (distance near))
                      )
        )

(make-frame *discussion
        (is-a *event)
        )

(make-frame *present
        (is-a *time-spec)
        )

(make-frame *today
        (is-a *time-spec)
        )

(make-frame *direction
        (is-a *location)
        (:conceptual_objtype direction)
        )

(make-frame *country
        (is-a *location)
        )

(make-frame *between
        (is-a *location)
        (:cr$constraints (subject *location *object)
                         (object *location *object))
        (similar-to *among)
        )

(make-frame *among
        (is-a *location)
        (:cr$constraints (subject *location *object)
                         (object *location *object))
        (similar-to *between)
        )

(make-frame *USA
        (is-a *country)
        (has-parts *state-of-USA)
        )

(make-frame *state-of-USA
        (is-a *inanimate-object)
        (part-of *USA)
        )

(make-frame *p-fundamental
        (is-a *property*)
        )

(make-frame *color
        (is-a *property*)
        )

(make-frame *size
        (is-a *property*)
        )

(make-frame *quality*
        (is-a *property*)
        )
```

# Appendix D
# Sample Texts

The following example text was used f^r the trace in Appendix A, as well as for testing during most of MASTER-D's development. The numbers to the left indicate the paragraph and sentence numbers in the ori;inal text from which the example was excerpted.

34.2 The comment is made in this document 'Against UI Copyright':

35.1 Copyright on a user interface means a goverment-imposed monopoly on its use.

35.2 This would mean that each typewriter manufacturer would be forced to arrange the keys differently.

36.1 (omitted) I think that's a very simplistic description.

36.2 By my description, the letters of the alphabet would be tools.

36.3 Of course those should be open.

The remainder of this appendix lists the example text used for the main tests described in Chapters 9 and 10. As for the previous text, the numbers at the left indicate paragraph and sentence; however, some of the original sentences have been split into multiple sentences, which remain groups under the original sentence number. Additionally, several sentences and small portions of others have bee[illegible]ded; this is indicated by square brackets.

23.1. I'm going to try to talk specifically about intellectual property protection in software.

23.2. I've been asked to play a role. That role is the advocate of strong protection, which is something like the sacred acolytes at Chichen Itza. Those acolytes were asked to take the role for a year at the end of which they get thrown into the sacred pit.

23.3. That's my feeling here tonight, anyway.

23.4. But I'm a brave if foolish lad, so I'm going to take the challenge heartily.

23.5. I'm going to make two disclaimers:

23.6. While I am very much behind the Lotus position, I am speaking tonight as an individual, not as an officer of Lotus.

23.7. I say that by choice. I wasn't asked to say that.

23.8. Secondly, I'm not a lawyer; I've looked at this from the standpoint of a business person, and I want to say (you can believe me or not) I've thought about these issues as seriously as I know how from the standpoint not only of a large company, but from the standpoint of small companies as well.

23.9. I take my job as a trustee of the Massachusetts Software Council very seriously, a[illegible] 'ook at small and large companies all the time.

23.10. So, right or wrong, I'm trying to think in that mode.

24.1. Significance of this issue:
24.2. This is really a mammoth issue.
24.3. Decisions and feelings and positions taken today will have, I believe, major impact on the world.
24.4. That's a strange phrase for a business person to ever utter.
24.5. We're usually thinking about market shares and points of margin.
24.6. This is really a fundamental, economic, and quality of life issue, I believe.

25.1. Why?
25.2. Well, first of all, I want to start with a statement that some people may dispute, but it's a place that's a sort of bedrock for me.
25.3. I believe property protection is fundamental to the proper functioning of societies.
25.4. Two or three years ago, I might have stood up here and said to the proper functioning of capitalist societies.
25.5. I think right now that statement would be too narrow.
25.6. You can go to almost any country in the world and people would say the management of property is important to how society functions in any form of society.
25.7. If that statement is taken as true the following statment is an important one. [,one that people may not agree with.]
25.8. And that is that if property protection is fundamental to the proper functioning of societies, then intellectual property managment, intellectual property protection, is a critical issue today for everyone.
25.9. And there's a number of reasons why that's true.
25.10. First of all, the notion that property can be only physical, material, corporeal, is an incredibly burdensome and frightening notion to me.
25.11. To say that someone who can manufacture a widget has the right to property protection but that someone who writes a symphony does not seems to me to be a very troubling notion. This would reduce the very creative people in society to the state of having to depend on powerful patrons who own the material world.
25.12. So to me it's critical in order to recognize the higher achievements of man, and also, from a much more narrow perspective, it's important in understanding how countries, states, and individuals will prosper in the world economy to come.
25.13. [To put it very bluntly,] if we don't have intellectual property protection, we are giving up some very, very important positioning for our state, our country, our industry in the world economy, because intellectual property is the best property we've got.
25.14. We are very poorly positioned to compete in the physical world coming down the road; we are very well positioned to compete in the intellectual and spiritual domain, I believe.
25.15. It is an urgent issue today in global competition in the way that companies are battling with each other, positioning with each other, and in the way that countries are positioning with each other.

25.16. This is a fact.

25.17. Taiwan, Singapore, Italy--there are companies there manufacturing blatant copies of works done in this country. {parse actually written as: There are companies in Taiwan,Singapore,and Italy manufacturing blatant copies of works done in this country.}

25.18. There is significant lobbying going on by countries who have an advantage from a cost/production standpoint in the material domain to reduce intellectual property to a commodity that can be freely copied and reverse engineered.

25.19. I believe that's extremely dangerous and important.

25.20. That's why I think this is an issue with great significance.

26.1. Well, if intellectual property protection is important, the question is, how should it be managed to the best ends, what are the boundaries, what is the degree of intensity?

26.2. I think you have to look at the goals.

26.3. What do we really want to do in society in regards to intellectual property protection?

26.4. First, we want to reward the creator of value for the customer [, for the consumer].

26.5. That's where we have to start in our explication of value.

26.6. It's customer value; it's consumer value.

26.7. Related to that, the idea is that you should have law [and implementation of law] that will continue to develop value for customers in the ongoing manner.

26.8. And that says to me that you want to continually incent entrepreneurial initiative.

26.9. The French economist Seay wrote that:

26.10. "The entrepreneur takes resources from an area of lower productivity and moves them to an area of higher productivity."

26.11. That's what defines an entrepreneur.

26.12. Innovation is a specific tool, Drucker says, of the entrepreneur, in which we create new resources or improve the use of resources.

26.13. To me that process of entrepreneurial innovation is essential, and what intellectual property law has to do is to enable startups to flourish on the one hand, to make sure that continual innovation is able to happen with very low barriers to entry. At the same time [(and I think that this is an equivalent good)] it must enable successful companies to flourish.

26.14. Those two things are both important. Any application of law that only makes it possible for brand new companies to start up may sound romantically very correct but I think practically will be disastrous, because it will mean that the software industry is capped effectively at $10 million or $20 million or $30 million companies [, which will shift the burden of power back to those companies which have huge scale; that is to say, the hardware manufacturers]. This is a troubling notion for me.

26.15. The third goal, I believe, is that we want to punish theft.

26.16. That's a hard statement. I made it consciously.

26.17. I thought about softening it up for this group; I thought about making it "prevent theft."

26.18. But then I thought about that; that's how you get into fascism, when you try to prevent bad behavior.

26.19. What we're really saying is that if you do this, then if everyone in society's mechanism decides it's wrong, then you ought to get punished for it.

26.20. Now theft is a very strong term. I'll come back to that term.

27.1. Well, if those are the goals, how do you figure out what to protect?

27.2. I think the question that starts is, Where does customer value come from in software?

27.3. These are simple words, but I think you have to start in simple principles.

27.4. Software enables customers to accomplish something in a particular way[, by specific means].

27.5. I think that the particular way in which software enables someone to do something deserves some kind of protection without getting into how much or what the means are because I think those are very tough questions.

27.6. We'd like them to be simple; they're hard, and this is new stuff.

27.7. The rules for defining what is protectable and what should be protectable need to be developed; they are not developed.

27.8. And there is a debate today between two schools of thought.

27.9. One school of thought says let copyright law sort it out through the application of case history, which looks at all of these suits that are in process now; that is very important for building the accretion of experience that will make a really meaningful law in ten or fifteen years[, which is the way the law works].

27.10. [You look back and you look at music, for example;] it took 20 or 30 years of copyright application before the case law gave us a viable framework [, which I think most musicians today would agree is largely a viable framework].

27.11. People understand it. There's room for innovation, and [like software,] there's room for copying, because the creative musician cannot be afraid of a two-five-one progression.

27.12. You must be allowed to have a two-five-one progression but he's not allowed to have a complete sequence of notes and intervals that duplicate someone else's melody.

27.13. That's the domain-specific kind of information that gets developed [over years of case law, when you figure out who was thieving and what drove them, and how did they behave, and what are the domain-specific concerns of that area of music versus painting versus film.]

27.14. So this debate between "let case law precedence sort it out," and "it's never going to work, copyright is wrong or copyright is bad, we need a new policy and we need it now," which scares a lot of people like me[, more than anything because we're in a very vulnerable period in the world around the subject of intellectual property protection.]

27.15. New laws take a lot of time to draft, and no matter how well we draft the new law around software protection, it's going to take 20 or 30 years of case law in that domain to make it work.

27.16. Whether we like that or not, that's how the law works.

28.1. So let's look at these customer value sources in software.

28.2. I think these are seven that I've come up with[, having been in the software business for ten years or so].

28.3. There are others, I'm sure.

28.4. I think that the customer derives value when using a piece of software [from things as varied as the way that the software designer conceived of the problem solution from the actual function in the product; of course, the ability to add, subtract, multiply and divide, in 1-2-3, for example.]

28.5. A third area of value derivation is that which enables the user to achieve the desired result--the user interface.

28.6. And I'll come back to this in detail.

28.7. Fourth there are what I like to call conceptual data structures which the user is taught or which are illuminated to the user, provided to the user, and the user then uses.

28.8. In 1-2-3, for example, there's the concept of a range to describe a set of values in a cartesian coordinate system; Excel has a similar concept called the array.

28.9. Then there is the actual internal implementation.

28.10. [For example, the algorithms, internal data structures, code detail--the actual writing of the code.]

28.11. This value also comes from bundled data, that comes with the program code[; things like fonts, font outlines].

28.12. [Also, for example, marketing information, or typesetter width tables.]

28.13. And then finally, value comes from usability with other hardware and software.

28.14. There are many others, I'm sure.

28.15. Each one of these is a topic in itself [, and what I've just made by, in our community today, is that we're all fighting about "Is protection needed or is protection not needed; is it good or bad?"]

28.16. And we really should be working on the very hard work of detailing the policy on each of these areas.

28.17. And maybe I'm wrong about what the areas are, but there's a lot of really hard work in answering this challenge.

28.18. I believe the information technology community needs to guide society [--the judges, the legislators, and so on--] on how to provide the proper kind of protection, and the proper degree of protection, for each of those sources of customer value, so that we have the desired result; and that is to both allow successes to flourish (that is an extremely important principle, I believe, [in terms of developing ongoing quality and excellence]), and also, making it such that the barriers to entry are not so high that small companies cannot enter the market.

29.1. Let's talk specifically about an area that I think is subject right now to a massive amount of misinformation and disinformation.

29.2. And I'm very happy to have been given my prop here when I came in, "Against User Interface Copyright," which I think misses exactly some of the points that I would like to focus on.

29.3. And this is not a setup because I had done my foils before[; I promise].

30.1. What is user interface?

30.2. This is obviously matter for debate and discussion, but I would say that the user interface does consist of tools such as pull-down menus, moving cursor menu bars, and so on; it consists of conventions [(the MacUser Interface Style Guide; the many conventions around the use of 1-2-3, such as the moving ring cursor)]; it also consists of inventions, things that didn't exist before (stuff that came out of PARC--everyone has some idea of that).

30.3. But most important, I believe, user interface has to do with the uses of those tools to solve a particular problem.

30.4. [It's not that a pull-down menu doesn't do much for me; it's when I pull it down, what's in there: What words are chosen, what functions are available to me?]

30.5. To me, a way to describe it is a user interface is a creative expression of an idea, of how best to enable a user to achieve something in a particular way.

30.6. And the way that that happens is 1) by using UI tools like the pull-down; 2) by adopting, by flouting, or by proposing new conventions; and 3) occasionally by inventing a UI technique, something that just doesn't exist--it's unique.

30.7. But I think especially, a user interface is the sum total of that which enables access to the program function by linking your usage of those UI elements [(tools and conventions)] and the program resource, the program data structure.

30.8. It's the program function: It's the holistic merging of those things.

30.9. That's not a nice tidy description of what a UI is, but I think in reality that's what it is, and, unfortunately, we're going to have to help develop a case law that deals with that soft and fungible nature of software.

31.1. So what should be protectable in UI technology?

31.2. This is my position, not Lotus Development Corporation (I don't think we would differ much).

31.3. Basic tools, we believe, should be open.

31.4. I think that it has been very convenient to sort of think that Lotus hasn't taken that position; Lotus has never argued that the moving cursor, the menu bar, or the inverted 'L' were Lotus inventions. {Lotus} never argued that at all, and still don't.

31.5. We are supporting the UI's, like the Mac UI, like Windows, like PM. We believe in that stuff.

31.6. Tools should be open.

31.7. Conventions should be voluntary.

31.8. People should put them out.

31.9. You pay the price in the Mac world if you don't adhere to the UI convention.

31.10. True invention, I believe, should be protectable.

31.11. Someone who comes up with a brand new way of doing something should have some kind of protection.

31.12. I myself feel very very nervous about patenting in software.

31.13. I don't know what the right levels are.

31.14. But I think that at some level an inventor ought to be able to be rewarded and have some protection.

31.15. But I do think that the integration of the UI element application--how you use this total sum of tools and conventions in their total effect--should be protectable.

31.16. That is, how the program enables the user to accomplish something in a particular way. [At some level, some way, we have to figure it out,] a person has got to be rewarded for that or the structure's going to break down.

31.17. [And unfortunately, not very nice from a political standpoint], the work, the devil, is in the detail and in the degree; and there's a lot of hard work that's going to get required to sort these problems out.

32.1. Now, to give you an idea of what I mean by this sum total: This is the 1-2-3 release 2.01 menu system. This is the entire menu.

32.2. Now people say, "Oh, you guys are suing those guys because they used the same top level menu bar."

32.3. I've left out one menu here, [which is graph,] That's this page.

32.4. This is about 97 or 98 percent of all the program structure and all the user interface access in 1-2-3.

32.5. Every single node [--every single node--] on these two slides is present in exactly the same place, and exactly the same detail in the two products that we sued.

32.6. This is not a question of "They used a moving cursor menu bar. We don't like that."

32.7. This is a question of taking all the work that was done, [all the engagement with the customers, all the refinement and testing,] and then taking the documentation and using that as your functional spec.

32.8. If this is not within the bounds of protection, then we have a very interesting world ahead.

33.1. Final slide.

33.2. [As I said to someone on the way over here, "If I'm going to wear a target, it's going to be fluorescent red."]

33.3. Suitability of copyright.

33.4. I believe personally (I'm not a lawyer) that (but I'm a musician; that's what I was doing before I came to this business, and it works for me) the concept of idea and expression, [which is fundamental to copyright,] to me is a very good analogy (maybe not perfect; I haven't heard a better one) for functionality implementation; for doing something in a particular way; for doing the same thing in the same way.

33.5. Copyright also has a proven ability to meet industry-specific or domain-specific needs over time via the accretion of case history.

33.6. There is substantial value in the current copyright law and case history to guide decision making. [, and finally,] The application of this law can be influenced by community leaders, and I don't just mean industry.

33.7. I mean academia. I mean users, customers, all the people who care.

33.8. So I think that it's a viable vehicle. I am concerned at this particular moment when the world order is changing so dramatically and intellectual protection laws are about to be signed or agreed to among many nations, that if we in the States who are driving that movement suddenly lose our conviction about intellectual property protection, I know very specifically what that's going to mean to my company and to other companies.

33.9. [It's going to mean major losses to us from people who see a good opportunity not to come to terms with the intellectual property demands of the US and the other Western countries; we'll just go without.]

34.1. [Just a comment--] I won't go through in detail, but there's a statement in here that I think elucidates some of the apparent difference in opinion which I think is not really germane.

34.2. The comment is made in this document "Against UI Copyright":

35.1 Copyright on a user interface means a goverment-imposed monopoly on its use.

35.2 This would mean that each typewriter manufacturer would be forced to arrange the keys differently.

36.1 I think that's a very simplistic description.

36.2 By my description, the letters of the alphabet would be tools.

36.3 Of course those should be open.

# Appendix E

## References

[1] H. Alshawi.
*Memory and Context for Language Interpretation.*
Cambridge University Press, 1987.

[2] F. Boas.
*Handbook of American Indian Languages, Part 1.*
Scholarly Press, 1911, 1976.

[3] P. Bosch.
*Agreement and Anaphora: A Study of the Role of Pronouns in Syntax and Discourse.*
Academic Press, 1983.

[4] B. Bower.
Clues to the brain's knowledge systems.
*Science News* 142, 1992.

[5] R.D. Brown.
*Using Multiple Adaptively-weighted Strategies for the Resolution of Demonstratives.*
PhD thesis, Carnegie Mellon University, 1993.

[6] R.D. Brown.
Augmentation.
*Machine Translation (formerly Computers and Translation)* 4:129-147, 1989.

[7] R.D. Brown and S. Nirenburg.
Human-Computer Interaction for Semantic Disambiguation.
In *Proceedings of the 13th International Conference on Computational Linguistics*, pages 42-47. Helsinki, Finland, 1990.

[8] K. Bühler.
*Sprachtheorie: Die Darstellungsfunktion der Sprache.*
Gustav Fischer Verlag, 1934, 1965.

[9] K. Bühler.
The Deictic Field of Language and Deictic Words.
*Speech, Place, and Action: Studies in Deixis and Related Topics.*
In R.J. Jarvella and W. Klein,
John Wiley and Sons, 1982.

[10] R.N. Campbell.
*Noun Substitutes in Modern Thai.*
Morton, 1969.

[11] J.G. Carbonell and R.D. Brown.
Anaphora Resolution: A Multi-Strategy Approach.
In *Proceedings of the Twelfth International Joint Conference on Computational Linguistics*, pages 96-101. COLING '88, August 1988.

[12] D.M. Carter.
*A Shallow Processing Approach to Anaphor Resolution.*
Technical Report Technical Report No. 88, University of Cambridge Computer Laboratory, 1986.

[13] D. Carter.
*Interpreting Anaphors in Natural Language Texts.*
Ellis Horwood Limited, 1987.

[14] A.C. Clarke and G. Lee.
*The Garden of Rama.*
Bantam Books, 1991.

[15] A.C. Clarke and G. Lee.
*Rama II.*
Bantam Books, '989.

[16] D.A. Dahl.
Focusing and Reference Resolution in PUNDIT.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1083-1088. AAAI-86, 1986.

[17] D.A. Dahl.
personal communication.
November, 1990.

[18] D.A. Dahl.
personal communication.
January, 1991.

[19] M.A. Gernsbacher and S. Shroyer.
Signalling Importance in Spoken Narratives: The Cataphoric Use of the Indefinite *This*.
In *Program of the Tenth Annual Conference of the Cognitive Science Society*, pages 587-593. Cognitive Science Society, 1988.

[20] B.A. Goodman.
Reparing Reference Identification Failures by Relaxation.
In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 204-217. ACL'85, 1985.

[21] B.A. Goodman.
Reference Identification and Reference Identification Failures.
*Computational Linguistics* 12(4):273-305, 1986.

[22] B.J. Grosz and C.L. Sidner.
Attention, Intentions, and the Structure of Discourse.
*Computational Linguistics* 12(3):175-204, 1986.

[23] E. Hajičová, V. Kuboň, and P. Kuboň.
Stock of Shared Knowledge -- A Tool for Solving Pronominal Anaphora.
In *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING-92)*, pages 127-133. 1992.

[24] J. Hart, Jr. and B. Gordon.
Nerual Subsystems for Object Knowledge.
*Nature* 359:60-64, 1992.

[25] J. Hintikka and J. Kulas.
*Anaphora and Definite Descriptions.*
D. Reidel Publishing Company, 1985.

[26] G. Hirst.
*Semantic Interpretation and the Resolution of Ambiguity.*
Cambridge University Press, 1987.

[27] F. Ingari, M. Kapor, J. Landry, T. Lemberg, and R. Davis.
Intellectual Property in Computing: (How) Should Software be Protected? An Industry Perspective.
Transcript of panel discussion.
October, 1990

[28] H. Kamp.
A Theory of Truth and Semantic Representation.
*Formal Methods and the Study of Language, Part 1.*
In J. Groenendijk, T. Janssen, and M. Stokhof,
Mathematisch Centrum, 1981.

[29] H. Kamp.
A Theory of Truth and Semantic Representation.
*Truth, Interpretation, and Information.*
In J. Groenendijk, T. Janssen, and M. Stokhof,
Foris, 1984.

[30] J.R.R. Leavitt.
*The DIBBS User's Guide Version 1.0.*
Technical Memorandum, Carnegie Mellon University Center for Machine Translation, 1990.

[31] S. LuperFoy.
personal communication.
February, 1991.

[32] S. LuperFoy and E. Rich.
A Computational Model for the Resolution of Context Dependent References.
1990.

[33] S.W. McRoy.
Using Multiple Knowledge Sources for Word Sense Discrimination.
*Computational Linguistics* 18(1):1-30, 1992.

[34]
Brain Transplant.
PBS television series *Nova.*
April, 1993

[35] E. Nyberg.
*FrameKit User's Guide.*
Technical Memorandum, Carnegie Mellon University Center for Machine Translation, 1988.

[36] M.S. Palmer, D.A. Dahl, R.J. Schiffman, L. Hirschman, M. Linebarger, and J. Dowding.
Recovering Implicit Information.
In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 10-19. ACL'86, 1986.

[37] T. Reinhart.
*Croom Helm Linguistics Series: Anaphora and Semantic Interpretation.*
Croom Helm, 1983.

[38] E. Rich and S. LuperFoy.
An Architecture for Anaphora Resolution.
In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 18-24. ACL, 1988.

[39] A.L. Samuel.
Some Studies in Machine Learning Using the Game of Checkers.
*IBM Journal of Research and Development* 3(3), 1959.

[40] A.L. Samuel.
Some Studies in Machine Learning Using the Game of Checkers II. Recent Progress.
*IBM Journal of Research and Development* 11(6), 1967.

[41] E. Sapir and H. Hoijes.
*The Phonology and Morphology of the Navaho Language.*
Univerity of California Press Berkeley, 1967.

[42] C.L. Sidner.
*Towards a Computational Theory of Definite Anaphora Comprehension.*
Technical Report TR-537, M.I.T. Artificial Intelligence Laboratory, 1979.

[43] C.L. Sidner.
Focusing in the Comprehension of Definite Anaphora.
*Computational Models of Discourse.*
In Brady and Berwick,
M.I.T. Press, 1983, pages 267-330.

[44] M. Tzschaschel.
Zwei Augen im Gesicht--und viele Augen im Gehirn.
*P.M.: Peter Moosleitner's interessantes Magazin* :58-64, 12/1991.

[45] B.L. Webber.
Discourse Deixis: Reference to Discourse Segments.
In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 113-122. ACL'88, 1988.

[46] Y.A. Wilks.
Preference Semantics.
*The Formal Semantics of Natural Language.*
In Keenan, E.,
Cambridge University Press, 1975, pages 329-348.

[47] Y.A. Wilks.
Making Preferences More Active.
*Artificial Intelligence* 11:197-223, 197?.