Dialog Parsing in the TRAINS System

Mark G. Core and Lenhart K. Schubert

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 612

March 1996

Abstract

Currently, the TRAINS dialog system uses a more or less standard chart parser as the interface between the text of the dialog and the rest of the dialog processing system. However, traditional chart parsers are not well equipped to handle dialogs because dialog constituents can be discontinuous, with interspersed acknowledgments, editing terms, repairs, etc. This paper proposes some modifications of the current TRAINS parser enabling it to handle discontinuous dialog structure. The representation of a dialog is still superficially hierarchical (rather than consisting of interleaved structures). This is made possible by two devices: one is to accommodate repairs (*e.g., to uh .. to Corning*) through explicit grammar rules; the other is to accommodate mid-sentence acknowledgments (*e.g., okay*), editing terms (*e.g., uh*), *etc.* as "trailers" attached to lexical items. We show how this works on a simple sample dialog. Because allowing for repairs and interruptions introduces much ambiguity, we also discuss some initial disambiguation techniques.

This work was supported in part by National Science Foundation grant IRI-95033312.

A: Bring the oranges ...
B: Okay
A: and the bananas .. to , uh .. to ..
B: to Corning?
A: right, to Corning.

Figure 1: Possible dialog from a train scheduling domain

1 Introduction

Written texts usually consist of sequences of sentences, and these in turn form topically coherent clusters (such as paragraphs). By contrast, spoken dialogs show a more haphazard (or at least more complex) organization. Topics may abruptly shift as speakers take turns, and the speakers frequently fragment their own utterances with turn-keeping sounds, editing terms and repairs, and interject acknowledgments, corrections, questions, helpful continuations, etc., into each other's utterances in mid-sentence. In addition, there is a higher-level dialog/subdialog structure in which groups of utterances are organized into question-and-answer pairs, requests and responses, proposals and acceptances (or rejections, amendments, etc.), descriptions, narratives, elaborations, digressions, and so on, and these can be nested to multiple levels.

Our eventual goal is to integrate all levels of structural analysis, from the dialog/subdialog level to the sentential, phrasal and speech prosody level, in a robust, chart-based dialog parser. The parser should be applicable to the kinds of dialogs that are held or could be held between a user and the TRAINS interactive transportation planning system [All94, AFMR95]. However, in this initial work our focus is on allowing for the kinds of intertwined, discontinuous constituents mentioned above. We take for granted a written transcription of a spoken dialog (such as are available for the 1991-3 TRAINS dialogs [GT93, HA95], and we largely neglect the higher-level organization of dialogs.

To keep the necessary modifications to the existing parser to a minimum, it will be convenient to regard a dialog as a sequence of utterances (sentences or phrases), with three kinds of rather arbitrarily inserted segments: *lulls*, *interpolated* segments and *backtrack* segments. Lulls include pauses (transcribed as dashes or dots), commas, and editing terms such as *uh, umm, let's see, I mean, etc.* For reasons explained later, changes of turn are also treated as lulls, dominating the "word" *c-of-t*. Interpolated segments are typically acknowledgments (*right, okay, etc.*) but could also be self-interruptions (*e.g., no* in *use engine E1,* no*, E2; I guess; wait a sec*) and subdialogs (such as interjected question-answer pairs, or error-correction subdialogs). Backtrack segments are complete or incomplete phrases that are "left behind" when a phrase or sentence is restarted or repaired.

A brief sample dialog is shown in figure 1 , containing several lulls (in addition to changes of turn) and the interpolated segments *okay* and *right*.[1] The backtrack segments in the sample dialog are two incomplete prepositional phrases starting with *to*. Note that both interpolated segments begin and end at lulls, and both backtrack segments end at lulls. We assume that this holds generally for interpolated and backtrack segments.[2]

Note that while lulls, interpolated segments and backtrack segments can all be viewed as "extra baggage" within a sentence, they differ in that backtrack segments extend the preceding text (in terms of ordinary phrase structure), while lulls and interpolated segments do not. Also note that we

[1] Though the dialog is constructed (for simplicity), it reflects the kinds of discontinuities observed in actual TRAINS 91-3 dialogs.

[2] We will need to refine the notion of a lull eventually to take better account of prosody.

do not in general require changes of turn to occur at utterance or even phrase boundaries, though more often than not, they do. Thus we allow for "collaborative sentence production", where both speakers supply parts of a sentence. The last contribution by speaker B in the sample dialog can be viewed as tentatively completing A's imperative sentence.

The goal of the TRAINS project is to build a computerized planning assistant that can interact conversationally with its user, helping to produce plans for the production and delivery of goods within a set of cities linked by a railroad network. The current version of this planning assistant, described in [AFMR95], uses a chart parser which is not designed to handle lulls and interpolated or backtrack segments. In the following we propose a modified version of the current parser, equipped with special actions and grammar rules to handle such segments. This modified parser has reached the point where it can produce structural analyses of some dialogs; these are not fully disambiguated, higher-level dialog structure is ignored, and no semantic interpretations are provided as yet.

Before describing the parser, we briefly review some previous work. In section 3 we describe how the proposed dialog parser handles lulls, interpolated segments and backtrack segments, and in section 4 we illustrate its operation in some detail, also pointing out the ambiguities that arise. This leads to a discussion of incremental disambiguation in section 5. In sections 6 and 7 we deal with some complications concerning VP backtrack segments (caused for example by the presence of gaps and passive verb forms). We conclude with a discussion of future work in section 8.

2 Previous Work

The Gemini dialog system [Dow93] has an utterance grammar which it applies after it has built all the traditional syntactic structures. The utterance grammar is designed for single sentences, sentence fragments, and run-on sentences. The system does not license interpolated segments or backtrack segments as part of the regular dialog structure, but instead uses special mechanisms to skip over some kinds of repairs and interpolations (see also [LD93]). Though this is reasonable as a rough-and-ready expedient, it neglects the important role such segments can play in the dialog structure. For example, acknowledgments often signal "grounding" – the achievement of mutual understanding or mutual acceptance of a plan; and backtrack segments can contain referents that are needed to interpret subsequent text (*e.g., Take the oranges to Elmira, uh, I mean, take* them *to Corning*).

Biermann, et al. in [Bie93] present a dialog system that initiates subdialogs with the user to achieve its goals. In addition, the user is allowed to initiate a new subdialog. The dialog system uses plan recognition to see if the new subdialog is part of an appropriate plan. If so the system will continue the new subdialog; otherwise it will try to continue the old dialog.

Our focus in the present work is different. We are trying to deal with the fragmentation of utterances by acknowledgments, disfluencies, repairs, etc., rather than trying to deal with the higher-level aspects of dialog structure that reflect the problem-solving process. Eventually we will extend the parser to recognize these higher-level structures. They are to some extent indicated by superficial clues (such as turn-taking, cue words, and coreference relations), but cannot be reliably recognized independently of the problem-solving process. The approach to higher-level dialog processing used in the TRAINS 93 system is described in [All94]. Since the dialogs involve interactive development of a plan, the state of the discourse can be thought of as the current mutually accepted plan (along with proposed but not yet accepted plans). Roughly speaking, each new utterance is interpreted as extending this plan or starting a subplan.

1) TEXT <- UTT
2) TEXT <- UTT TEXT

Figure 2: Main dialog parsing rules

```
0:Bring the oranges                    and the bananas
1:                  ...                                ..
2:                     c-of-t
3:                           okay
4:                               c-of-t


0: [to   ..  [to  c-of-t [to Corning c-of-t       to Corning]]]
1:    ,         ..                          right
2:     uh                                       ,
```

Figure 3: Representation of sample dialog

3 TRAINS Dialog Parser

The TRAINS dialog parser we have been developing is a chart parser with special actions and grammar rules to handle lulls, interpolated segments, backtrack segments, and multiple utterances. The two rules in figure 2 allow for "texts" (dialogs) consisting of multiple utterances. (In the TRAINS 95 grammar, an utterance constituent can be formed from an NP, PP, S, VP, ADVBL (adverbial), PATH (a complex adverbial describing a path) or a specific phrase such as "okay" or "I'm done."). These simplistic text rules will be replaced with more subtle dialog rules when we begin to address dialog/subdialog structure more seriously.

Interpolated segments are "hidden" as trailers attached to lexical items, and backtrack segments are admitted through special grammar rules (discussed later); lulls are used as final constituents of backtrack segments, and those not absorbed in this way become lexical trailers. By these means, the parser is able to skip over interruptions and to produce a hierarchical grammatical structure as usual.

Interpolated segments and backtrack segments may themselves contain lulls, interpolated segments and backtrack segments, so that hidden items can themselves contain hidden items. In fact, a succession of lulls or interpolated items (such as *... c-of-t okay c-of-t*) leads to multiple levels of "hiding", since each new item "hides" in the last lexical item of its predecessor. Figure 3 is a schematic representation of the layers of hidden items in the sample dialog of 1.

The top (unhidden) layer corresponds to the main utterance, "Bring the oranges and the bananas to Corning"; backtrack segments and the lulls that terminate them are also shown at this level, since they are accommodated through phrase structure rules, rather than as lexical trailers. The interpolated segments *okay* and *right* are hidden at levels 3 and 1 respectively, as a result of the number of lulls (2 and 1 respectively) that happen to precede them and the fact that *right* is preceded by a backtrack segment. We have also indicated the (preferred) grammatical structure we would assign to the backtrack segments and their replacements, using square brackets. Each pair of brackets contains an incomplete or complete PP ending at a lull, followed by its replacement-PP (which may again be such a pair).

3.1 Rules and Parser Actions for Lulls

As can be inferred from our previous discussion, lulls are treated as a lexical category, according to the following lexical rules: [3]

3) LULL -> .. | ... | - | – | , | uh | um | let's see | *etc.*
4) (LULL (turn +)) -> c-of-t

One reason for treating changes of turn as lulls is that the input to the parser is a word stream, and we have no easy way of marking a change of turn except by inserting an item – a special word – into the word stream. But since this is not a syntactic constituent, we then need a way of skipping over it; making *c-of-t* a lull allows this. Of course, we could just restart the parser at each change of turn (as in all previous TRAINS parsers). But this would thwart our goal of forming constituents that bridge over acknowledgements, subdialogs, etc., or allowing for collaborative sentence production, or the formation of higher-level dialog structure.[4] Another reason for treating *c-of-t* as a lull is that this makes our assumptions about the role of lulls in delimiting INTER and backtrack segments much more tenable. For instance, acknowledgements can be interjected into an utterance rather "seamlessly", with only the changes of turn marking their boundaries. Similarly, a repair of a phrase may be supplied by another speaker, and the change of turn may be the only word-level clue to its presence.

When a lull is removed from the agenda, it is added as a trailer to all lexical items ending at the lull. In effect, this corresponds to use of "lexical transformation" rules of form

5) (X (bkpt +)) -> X LULL,

where X is any lexical category. The "breakpoint" feature *bkpt +* marks X as a constituent ending at a lull. For this feature to serve its purpose, it also needs to be propagated "upward"; i.e., if a constituent has a *bkpt +* feature on its final subconstituent, it also receives that feature value. Thus any constituent whose final lexical item has a lull as trailer carries the feature value *bkpt +*.

One might ask, why not attach lulls to other categories (which would be equally effective in allowing the parser to skip over them)? The answer is just that we pick a particular, fixed level for "hiding" lulls, namely the lexical level, to avoid gratuitous ambiguity. For instance, if we allowed lulls to extend NPs and VPs, then the sequence V NP LULL would be a (VP (bkpt +)) made up either of a V and an (NP (bkpt +)), or of a VP and a LULL.

Besides attaching lulls to preceding lexical items, the parser also uses them as final constituents of backtrack segments – see the further discussion below.

3.2 Rules and Parser Actions for Interpolated Segments

The rules we use for INTER are

6) INTER -> (UTT (bkpt +)) | (PATH (bkpt +)).

In other words, an interpolated segment can be any utterance or PATH constituent. Some sample rules for PATH (complex adverbials describing paths) will be seen later. Note that the *bkpt +*

[3] A *turn* feature value of + differentiates changes of turn from other lulls.

[4] Concerning dialog structure, note that rule 2 allows a new utterance to be started by either dialog participant.

feature ensures that an INTER terminates at a lull. However, given our assumption that interpolated segments begin and end at lulls, the parser also checks whether an INTER of the above type immediately follows a lull, before forming a corresponding agenda item. When an INTER is taken from the agenda, it is treated much like a lull; i.e., it is used to extend all immediately preceding lexical items, in effect implementing the rule

7) (X (bkpt +)) -> X INTER.

Thus, just as in the case of lulls, arcs can be extended over the interpolated segment even if they were not expecting it.

3.3 Rules and Parser Actions for Backtrack Segments

A backtrack segment is a partial or complete phrase that is followed by a corrected or reformulated phrase of the same type. (Only lulls and interpolated segments may intervene.) To describe this structure, we might assume a feature *back* with value + for backtrack segments, and "repair rules" of form X -> (X (back +)) X, for X, any category. However, introducing such a *back* feature would licence many inappropriate uses of incomplete phrases. For example, the words

`let's leave the .. uh leave engine E3 at Elmira`

could then be analyzed as containing a sentence starting at *the*, based on parsing *the .. uh* as a subject NP with feature (*back +*). To avoid this problem, we use category names different from "regular" category names for backtrack constituents. For example, an NP backtrack segment would be of category, NP-back, which is of course not permissible as an NP subject. Thus the form of repair rules is X -> X-back X; *e.g.,*

8) PP -> PP-back PP

In effect such rules allow arcs in the chart to be extended over backtrack segments, forming a "regular" constituent.

The formation of X-back constituents is initiated by the parser. When a lull is removed from the agenda, all arcs ending at this lull and containing at least one constituent are used to form backtrack segments. Such backtrack segments corresponding to incomplete constituents are given the feature *incomplete* with value + (for reasons that will become apparent shortly). Similarly, completed constituents ending at the lull are made into backtrack segments.[5] This can be viewed as equivalent to the use of rules such as

9) (PP-back (incomplete +)) -> P LULL
10) PP-back -> PP LULL

In addition, an X-back constituent can be used as the final subconstituent of another phrase, which should then be categorized as a backtrack phrase as well. This suggests rules like PP -> P NP-back. However, we do not want a *completed* X-back constituent to participate in this construction, since we would end up building complete backtrack segments twice (*i.e.*, at least once in this indirect way, as well as by direct attachment of a lull). So at this point, we make use of the feature value *incomplete +*, introducing rules such as

[5] Such a mechanism may not be the most efficient; if there is a sequence of lulls, each lull causes its own of set of backtrack segments to be created. To avoid this redundant effort, sequences of lulls could be combined into one lull in a preprocessing step.

11) PP-back -> P (NP-back (incomplete +))

The creation of rules like 11 is automated, so that the details are mostly hidden from the grammar builder. This rule would be generated corresponding to the original rule PP -> P NP. Pairs of rules like 8 and 11 were created correponding to all TRAINS grammar rules having two or more elements on the right-hand side.

There are some complications for backtrack versions of VP rules, for instance because a particle complementing a verb cannot be repaired, and because of the possible presence of gaps. However, we postpone a discussion of these issues to section 6, turning now to a detailed example.

4 Processing a Sample Dialog

Below is an example to illustrate the basic functioning of the TRAINS dialog parser:

A: bring the engine
B: to Corning
A: okay, to Corning

The example initially resulted in 1,089 different parses. The TRAINS grammar, although not probabilistic, has penalties assigned to rules the grammar writers thought should only be used as a last resort. These penalties are combined through multiplication just as probabilities would be. Almost all of the parses are eliminated by examining only parses with no associated penalties. However, there are still 13 parses to deal with.

The structure of the first and second parse trees is shown in figure 4 (the top level TEXT and UTT constituents are not shown in the parse trees). The lulls are given the category LULL and appended to lexical items such as *engine* and *okay*. In the TRAINS grammar, utterances can be formed by a simple response such as *okay*. PATHs can form utterances through rules 12 and 13. When the lull is encountered after the first occurrence of *to Corning* (a PATH constituent), the incomplete arcs corresponding to rules 12 and 13 form two UTT-back constituents which combine with the UTT *okay* to form two UTT constituents. These UTT constituents end in a lull and occur before a lull which allows them to be interpreted as interpolated segments by rule 6. These interpolated segments are attached to the preceding lexical item, a change of turn, forming complex lulls. These lulls are attached to the lexical item *engine* forming two constituents of the form: *engine c-of-t to Corning c-of-t okay comma*.

12) UTT -> PATH PUNC
13) UTT -> PATH "instead of" NP|PATH|VP

Each UTT-back constituent is composed of a PATH constituent as well as the lull that triggered the creation of the UTT-back constituent.

Another point to notice is that the parser does not check to see if the backtrack segment and its replacement were formed from the same rule or contain any similar lexical items; only the resultant category needs to be the same. In this case, such a permissive policy leads to the unlikely conclusion that the speaker corrected the phrase *to Corning* (regarded as an utterance) with the phrase *okay*.

In the third parse tree, since the UTT constituent, *okay*, appears between lulls, it is treated as an interpolated segment attaching to the preceding lexical item, a change of turn. Thus, *to Corning* is between a change of turn and the complex lull, *okay comma*. So the PATH, *to Corning*, can form an interpolated segment attaching to the change of turn before it.

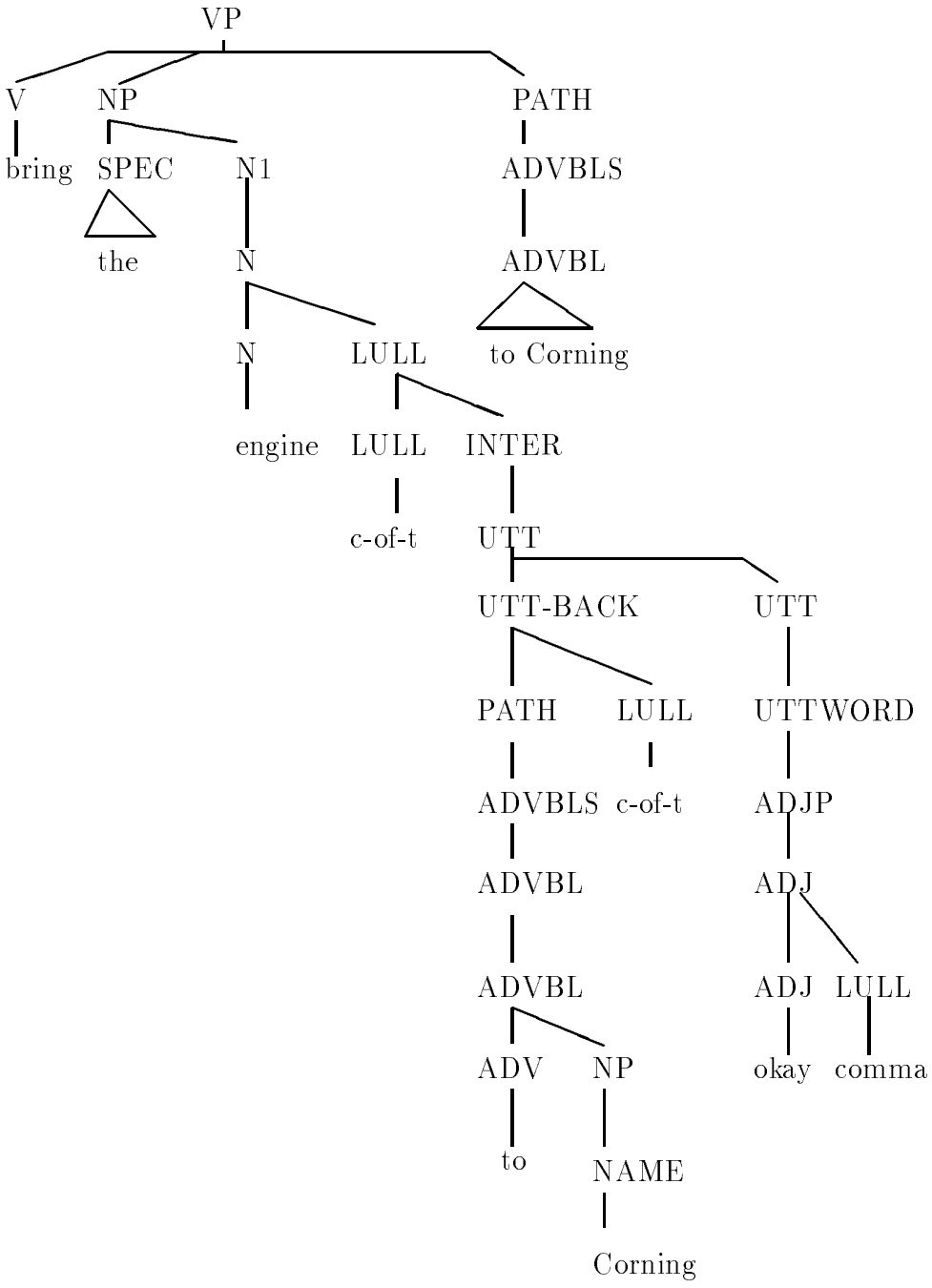

Figure 4: Structure of parse trees 1 and 2 of the example dialog

The fourth parse tree is shown in figure 5. It contains a complex lull, *c-of-t okay comma*, causing the incomplete arc associated with rule 14 to form a backtrack segment. This backtrack segment contains the NP, *Corning* as well as the complex lull; this NP-back forms an ADVBL-back constituent along with the adverb preceding it. This ADVBL-back constituent is allowed since it is followed by the ADVBL, *to Corning*. Parses 5 through 7 are identical to this parse except that the NP-back segment is formed from incomplete arcs taken from rules 15, 16, and 17. (Note that rules 18 and 19 are used to form NPSEQs). Recall that complete constituents before the lull can form backtrack segments but cannot be part of a complex backtrack segment such as these ADVBL-back constituents. Thus, the complete NP-back, *Corning* cannot form an ADVBL-back.

14) NP -> NP (PUNC (LF COMMA))
15) NP -> name (S (relative-clause))
16) NP -> NPSEQ "and" NP
17) NP -> NPSEQ "or" NP
18) NPSEQ -> NP
19) NPSEQ -> NPSEQ NP

An ADVBL-back can be formed from a complete ADVBL created by rule 20 as shown in the eighth parse depicted in figure 6.[6] The ninth parse is structurally the same as the eighth except that the ADVBL-back is taken from the incomplete arc formed from rule 21. The constituent ADVBLS shown in both parse trees can be composed of one or more ADVBL constituents. Thus, the tenth parse treats the two instances of *to Corning* as a sequence of ADVBLs forming a PATH.

20) ADVBL -> ADV NP
21) ADVBL -> ADVBL ADV

PATH-back constituents can be formed as shown in figure 7. The complex lull causes the complete PATH constituent to form a PATH backtrack segment which combines with the PATH constituent following it. In addition, there are two incomplete arcs involving this PATH constituent and rules 22 and 23. These are made into PATH-back constituents and appear in the twelfth and thirteenth parses.

22) PATH -> ADVBLS "and/but" ADVBLS
23) PATH -> ADVBLS "or" ADVBLS

5 Incremental Disambiguation

It is fortunate in the above example that the penalties assigned by the grammar eliminate most of the interpretations. However, this example is a very short dialog involving one simple sentence with an interruption and a backtrack segment. A realistic dialog will create too many parse trees if disambiguation does not take place as the parse progresses. Also, the parser should not be considering alternatives that are not valid such as replacing the backtrack segment *to Corning* with *okay*. Another problem lies in left-recursive rules such as: NP -> NP PP. In the case of left-recursive

[6] Here we should mention that the current TRAINS grammar handles prepositions and PPs in a slightly unconventional way. Prepositions and PPs are assumed to occur only as subcategorized complements, as in *attach the car* **to the engine**. In these cases the preposition is semantically vacuous. PPs functioning as adverbials are treated as consisting of an ADV (subcategorizing for an NP – in effect a preposition) and an NP, as shown in rule 20.

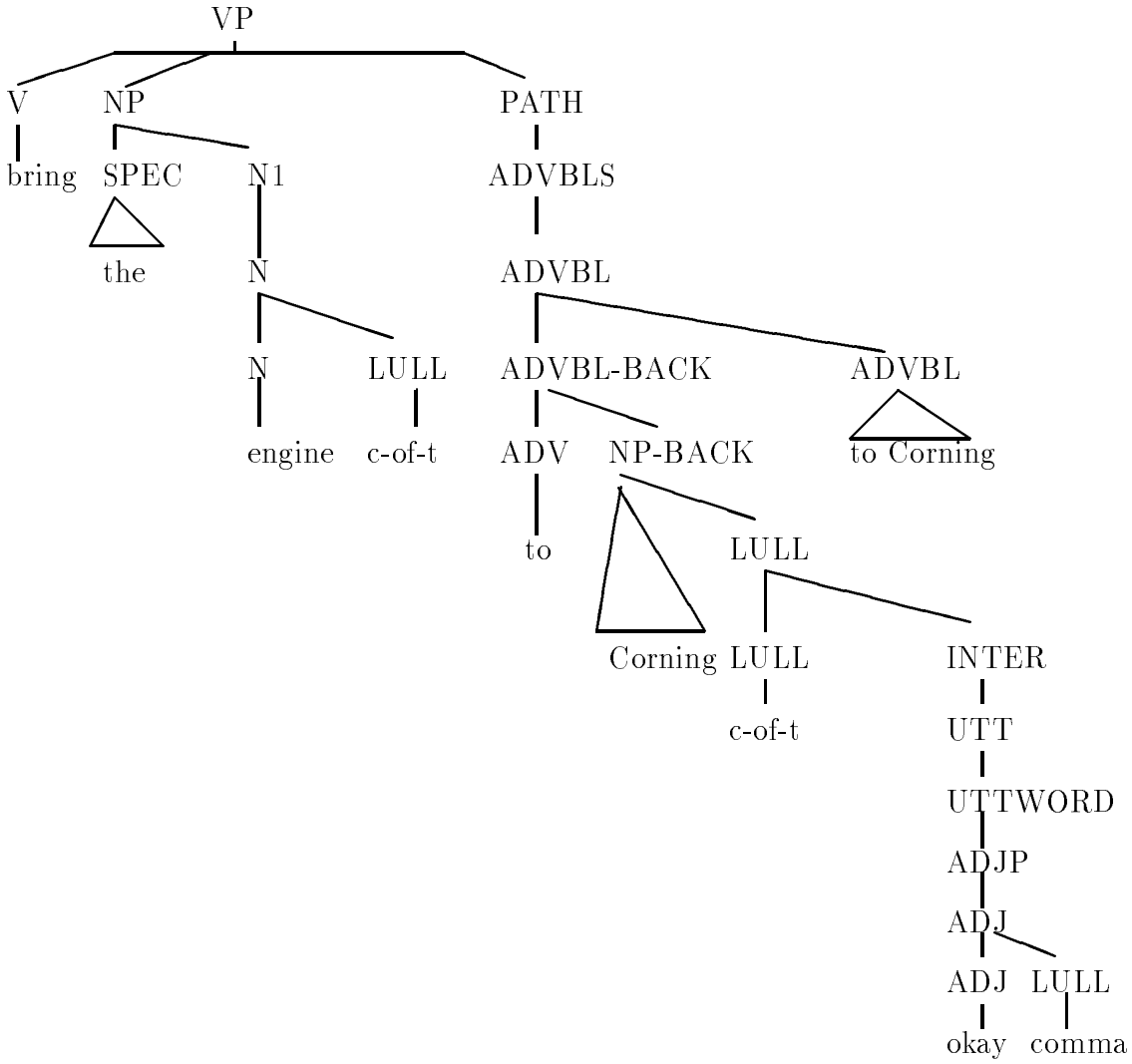

Figure 5: Parse tree 4 for the example dialog

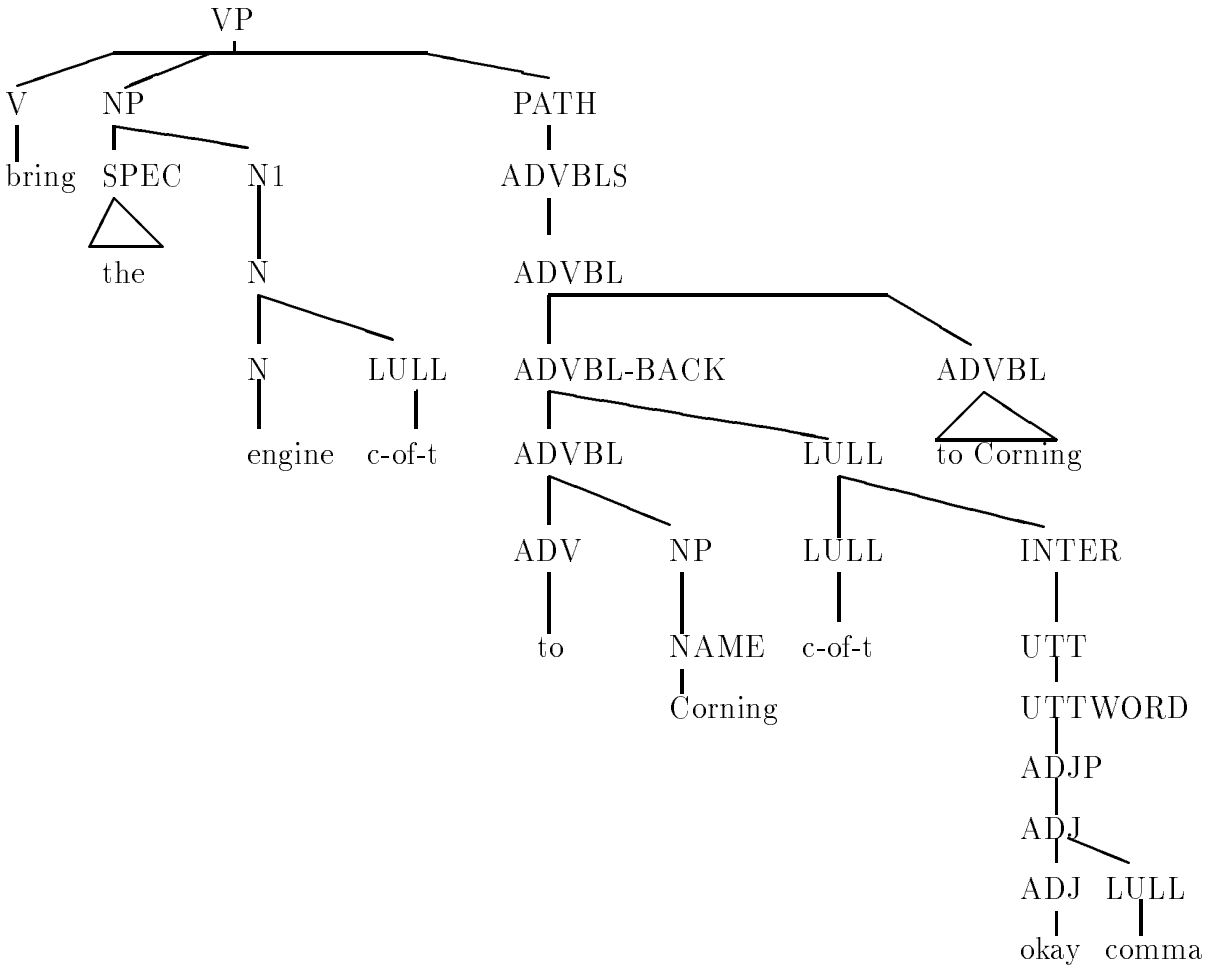

Figure 6: Parse tree 8 for the example dialog

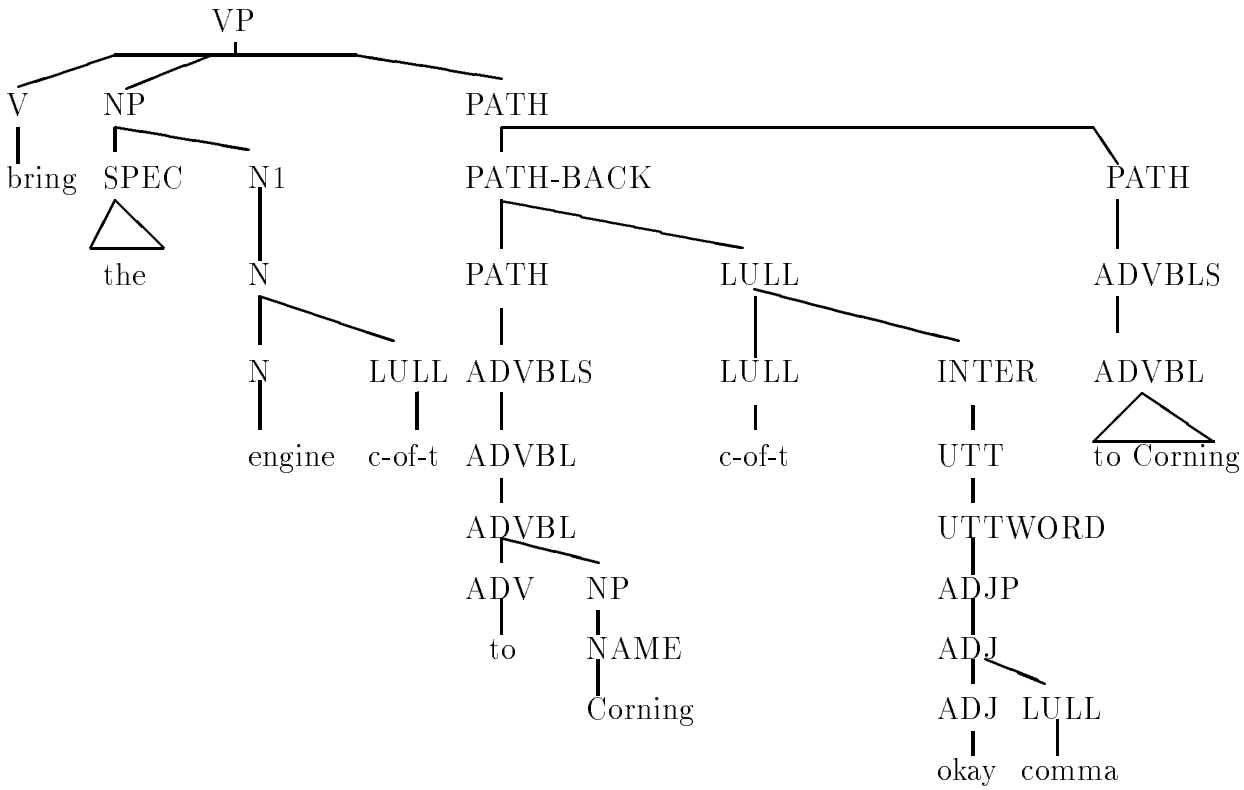

Figure 7: Parse tree 11 for the example dialog

NP rules, if an NP precedes a lull then the complete NP will be a backtrack segment, and so will all the arcs associated with left-recursive NP rules.

All of these alternatives encode the same information, so ultimately left-recursive rules will not be permitted to create backtrack segments. Currently, changes were made to the TRAINS dialog parser to prevent multiple incomplete backtrack segments (of the same category and with the same subconstituents) from being created from the same lull. Thus, two backtrack segments are still produced in the case of left recursive rules. Steps were also taken to prevent unrelated constituents from forming backtrack restarted phrase pairs. Most constituents in the TRAINS grammar have an atomic SEM feature (a head feature) capturing the general meaning category of the constituent. This feature is used to restrict rules in the grammar to apply only to certain semantic categories. As an initial attempt to restrict the backtrack restarted phrase pairs, we require the two constituents to have the same SEM feature. Currently these feature values (ACTION, AGENT, etc.) are vague enough to allow for the usual sorts of repairs. For example, location names and verbs (of the same type: ACTION, MENTAL ACTION, SPEECH ACTION, etc.) can be changed. If the backtrack segment is missing its head, its SEM feature is unspecified and will match any SEM feature of the restarted segment.[7]

The combination of these two restrictions allowed the elimination of 4 parses from the example of section 4. *okay* is no longer allowed to replace the UTT-back *to Corning* because of their different SEM features. One of the NP-back constituents is eliminated (one involving an NPSEQ) because it involves the same subconstituents as another NP-back segment. Only one is eliminated because the other two do not involve NPSEQs but instead involve a NAME and an NP constituent. One of the PATH-back constituents is eliminated because it involves the same subconstituents as another PATH-back segment.

6 VP Backtrack Segments

The example in section 4 is simplistic. There is only one verb phrase so VP-back constituents are not used in the final structural interpretations. However, as more complicated examples are tested, a comprehensive account of VP-back constituents will be necessary. Rule 24 is analogous to the PP backtrack rule 11 given earlier.

24) VP-back -> V NP-back

However, the TRAINS VP grammar is more complicated than this rule suggests. The structure of the main VP rule is given in rule 25.[8] The details have been left out; in fact the verb has features *subj*, *iobj*, *dobj*, *part*, and *comp* which may be empty (have value -) or describe a subcategorized constituent. These features are unified with the four elements following the verb in rule 25. For example, consider a lexical entry for the verb, *go*, which only subcategorizes for a subject and a comp element. The comp is required to be a PATH constituent with certain logical form features. The *iobj*, *dobj*, and *part* features will be -, *i.e.,* rule 25 will only allow a PATH constituent with the specified features to follow this particular form of *go*.

Rule 26 is similar to rule 25 except that it allows the particle to come before the direct object if the direct object is not a pronoun.[9] Any or all of these complements may be optional depending on the subcategorization features of the verb. Each verb should be able to form VP-back constituents when the last item on the verb's subcategorization frame is a backtrack segment.

[7] These unspecified SEM features may be restricted by the rule involved with the incomplete arc.

[8] iobj=indirect object. dobj=direct object. part=particle. comp=third complement.

[9] Such constructions usually do not involve an indirect object, which is why it is not allowed in this rule.

25) VP -> V iobj dobj part comp
26) VP -> V part (dobj (pronoun -)) comp

To allow this, new lexical entries were created for each verb with the last subcategorized constituent specified as a backtrack segment.[10] These new lexical entries are marked with a feature value, *back-form active*, preventing their use in the current verb rules.[11] Rules 27 and 28 are special versions of rules 25 and 26 and form backtrack VPs from the new verb entries. Verbs subcategorizing for a particle but not a comp cannot be used in rule 27 because particles cannot be backtrack segments. These verbs are marked with a feature value of *partsub +* in order to conflict with the feature value *partsub* - in rule 27. For example, one lexical entry for *drop* subcategorizes for a direct object and a particle, *off*. This entry must be marked as *partsub +* to prevent an VP-back such as *drop the , off*.

27) VP-back -> (V (back-form active) (partsub -)) iobj dobj part comp
[the types of the complements are in the verb's subcat feature. The last complement will be a backtrack segment]
28) VP-back -> (V (back-form active)) part dobj comp

There are two other major VP rules in the TRAINS grammar: one deals with direct object gaps and the other deals with comp gaps. The direct object gap is required to be an NP. Rule 29 is the version of this rule for backtrack segments which moves an NP out of a VP ending in a backtrack segment.[12] For example, one of the lexical entries for *keep* subcategorizes for both a direct object and a comp. The following sentence involving *keep* should be possible: "Which engine did we keep in , keep in Corning." The grammar should allow a direct object to be extracted as well as allowing a backtrack comp.

Another sentence that should be allowed is "Which message did we tell the , tell the manager." One of lexical entries for *tell* subcategorizes for a direct and indirect object. When the direct object is moved to the front of the sentence then the indirect object can be a backtrack segment. New lexical entries for such verbs were created that specify their indirect objects to be backtrack segments. These verbs were marked with a feature value of *back-form dogap* so that they can only be used in rule 30, a version of rule 29 created specially for these verb entries.

29) VP-back -> (V (back-form active)) iobj part comp
[where the verb subcategorizes for a direct object and the resulting VP-back has an NP gap]
30) VP-back -> (V (back-form dogap)) iobj part comp

A similar situation arises with comp gaps which are allowed in normal verbs by rule 31. If a comp is moved and there is no particle in the subcategorization list then a backtrack verb object should be allowed. For example, one of the entries for *leave* subcategorizes for both a comp and an object but not a particle. So a sentence such as "Where do we keep the , keep the boxcar" should be permitted. New lexical entries were created for verbs like *leave* specifying their object as a backtrack segment. These entries were given a feature value of *backform comp-gap* to limit their use to rule

[10] The ordering assumed here is indirect object, direct object, and the third complement. The particle is not allowed to be a backtrack segment.

[11] This means that all the current rules involving verbs need to specify *back-form* - in order to prevent the use of the new verb entries. Future refinements may include a special lexical category for these verbs or methods of automatically modifying an existing VP grammar.

[12] A backtrack segment will not be moved because it will not be of category NP. The restriction on *partsub* verbs is not necessary here because if a *partsub +* verb has a direct object it will be a backtrack segment.

32, a version of rule 31 created for these entries.

31) VP -> V iobj dobj
[where the verb subcategorizes for a comp and the resulting VP-back has a PP gap]
32) VP-back -> (V (back-form comp-gap)) iobj dobj

7 Backtrack Segments and Lexical Verb Rules

In addition to WH movement, the TRAINS grammar handles passive verb forms and dative shift. The dative shift rule moves an indirect object into an empty comp position as a "to" PP. A version of this rule was added to create a *back-form active* verb with a "to" PP-back constituent. For example, the lexical entry for *give* only subcategorizes for a direct and indirect object, with the indirect object required to be of type person. The backtrack dative shift rule will create a *back-form active* entry for *give* that subcategorizes for a direct object and a "to" PP-back comp having a PP object of type person.

The rules creating passive verb forms can affect the appearance of backtrack segments in the VP. Two of these rules shift the subject into an empty comp position as a "by" PP which could potentially be a backtrack segment. To allow this possibility, versions of these rules were created that formed verbs with *back-form active* feature values and specifying the comp as a backtrack PP; rule 33 shows the structure of these rules. The lexical entry for *calculate* subcategorizes for a direct object of type amount. Furthermore, the entry for *calculate* specifies that its subject must be a person. Applying rule 33 to *calculate* produces a new entry with the subject required to be an amount and having a PP-BACK comp with a PP object of type person. Note, in all the following passivization rules the right hand side is given the feature value, *passive* + and the left hand side is required to be *passive* -.

33) (V (back-form active) (subcat (comp pp-back))) -> (V (backform -))
[not shown here is the movement of an object to the subject position]

If a direct object is moved to subject position and the subject is deleted then an indirect object can be a backtrack segment. Such verbs are already marked with a feature value of *back-form dogap*. Rule 34 converts them into "back-form active" passive verbs. For example, the lexical entry for *tell* subcategorizes for a direct and indirect object where the direct object is a fact and the indirect object is a person. The new entry for *tell* produced by rule 34 has a subject which must be a fact, a backtrack indirect object, and no direct object. Active *back-form active* verbs should be able to be passivized as well. Rules 35 and 36 allow such a change; note, only NPs can be moved to subject position so this rule will not turn a backtrack segment into a subject.

34) (V (back-form active)) -> (V (back-form dogap))
[the movement of the direct object to subject is not shown]
35) (V (back-form active)) -> (V (back-form active) (subcat dobj))

36) (V (back-form active)) -> (V (back-form active) (subcat iobj))

```
1)  M: We better ship a boxcar of oranges to Bath by eight am
2)  S: okay
3)  M: um we need to get a boxcar to Corning where there are oranges
4)     there're oranges at Corning
5)     right
6)  S: right
7)  M: so we need an engine to move the boxcar
8)     right
9)  S: right
10) M: so there's an engine at Avon
11)    right
12) S: right
```

Figure 8: Dialog d91-6.1

8 Future Work

Currently, the TRAINS dialog parser is being tested on a dialog from the TRAINS 91 corpus.[13] A portion of this dialog which has been run through the dialog parser is shown in figure 8.

One goal at this point is to make needed adjustments in the TRAINS 95 grammar to accommodate the phrase types in this dialog. More crucially the idea is to track the formation of interpolated and backtrack constituents, to check the adequacy of our approach and to get a sense of the amount of ambiguity generated, and possible ways of curbing it. Testing has already revealed some problems, for instance the fact that backtrack segment formation and interpolated segment formation may incorrectly interfere, through "contention" for a lull. Also, it was necessary to hand-prune the parse at selected points, to prevent the accumulation of a huge number of alternatives. This has suggested various heuristics for automatic pruning, such as dispreferring UTT-backs, TEXT-backs, and sentence backtracks, and preferring interpolated utterances to be short. Considerable work will still be needed to obtain satisfactory performance on this dialog, and on more challenging ones.

Even without the results of this testing, it is clear that much remains to be done. We will mention some technical issues and then some larger issues. First, the VP-back grammar was created by hand but there should be functions to create it automatically. Second, the disambiguation system currently does not completely eliminate the ambiguity caused by left recursive rules. In addition, the SEM feature is a crude method of eliminating backtrack restarted segment pairs that do not match. The first change to this matching technique should be that mismatched pairs are penalized, not rejected. Second, the values of the SEM feature need to be examined closely to determine if they are too restrictive or too permissive. Third, other features should be examined to determine their relevance in determining matches between backtrack segments and restarted phrases.

Another possible technique to aid disambiguation would be to mark potential backtrack segments before they reach the parser. Work in [Dow93] and [HA94] reports success in isolating backtrack segments using word matching techniques. Having likely backtrack segments already marked would help the dialog parser, which has many possible high-level constituents it can assign to an utterance with a restarted phrase. A word matching approach is also used in the PUNDIT system [LD93], but only in cases where a parse cannot be found. The PUNDIT parser skips words until a keyword is seen and then it restarts the parse. It is not clear how keywords could be used in the TRAINS dialog parser since it always seeks a comprehensive syntactic analysis.

[13] These are a set of problem solving dialogs (between humans) used in development of the TRAINS 93 system.

Another subject to address in the dialog parser is local ambiguity. This refers to constituents and arcs that are never used in a complete parse. Efficiency may become an issue as the extra overhead introduced by dialog parsing may limit real-time use. Preliminary investigation into top-down filtering shows that the time used in checking the filters is greater than the extra time resulting from the proposal of constituents and arcs that would be eliminated by the top-down filter. The top-down filter used in these tests contains a hash table in which the parser can look up the permissible constituents following another constituent in the parser. Each time an item is added to the chart, the parser adds to its list of constituents allowed at that point. It seems likely that a more efficient algorithm for top-down filtering could be implemented; so it remains to be seen if top-down filtering will be helpful to dialog parsing. Similarly having look-ahead could further limit the formation of unnecessary constituents and arcs. Again it remains to be seen if the overhead of lookahead would outweigh the cost of the extra constituents and arcs produced without it.

Ultimately the chart of the dialog parser is intended to be used by the TRAINS dialog system for dialog processing. This raises some broader issues. The current representation of dialogs is hierarchical, and contains all the "pieces" that seem to be needed to make sense of the dialog. However, this does not adequately reflect the "give and take" structure of dialogs. For example, the attachment of an acknowledgement to a lexical item does not directly indicate what is being acknowledged. Presumably, the acknowledgement pertains to some partial or complete constituent (expressing at least one predication that is not presupposed) ending at or near the point of acknowledgement. A more adequate dialog structure would reflect this acknowledgement relation. Even more obviously (and as we have mentioned), we are not yet addressing higher-level dialog structure, such as question-answer pairs, or such as the pairs of *right*s seen in the dialog of figure 8, consisting of a confirmation request followed by a confirmation. Thus it is clear that additional structural processing will be needed so that the final analysis obtained will properly reflect dialog structure.

However, this structure is very ambiguous without some conception of the semantics and the pragmatic roles of the dialog constituents, and so it will be important to use semantic and pragmatic considerations in guiding the parser. For example, if a question is asked then the parser should favor an answer as opposed to an acknowledgement. Corpus-based statistics about the locations of interpolations, backtrack segments and other dialog-level constituents could also provide significant help in deciding on the correct analysis.

References

[AFMR95] J.F. Allen, G. Ferguson, B. Miller, and E. Ringger. Spoken dialogue and interactive planning. In *Proc. of the ARPA Spoken Language Technology Workshop*, Austin, TX, 1995.

[All94] L.K.; et al. Allen, J.F.; Schubert. the TRAINS project: a case study in building a conversational planning agent. Technical Report 532, Department of Computer Science, University of Rochester, Rochester, NY 14627-0226, September 1994.

[Bie93] C.I.; et al. Biermann, A. W.; Guinn. Efficient collaborative discourse: A theory and its implementation. In M. Bates, editor, *Human Language Technology*, pages 177–181, San Francisco, March 1993. Advanced Research Projects Agency, Morgan Kaufman Pub. Inc.

[Dow93] J.M.; et al. Dowding, J.; Gawron. Gemini: A natural language system for spoken-language understanding. In *Proc. ACL-93*, pages 54–61, Columbus, Ohio, 1993.

[GT93] J.; Gross, D.; Allen and D. Traum. the TRAINS 91 dialogues. TRAINS Technical Note 92-1, Department of Computer Science, University of Rochester, Rochester, NY 14627-0226, 1993.

[HA94] P. Heeman and J. Allen. Detecting and correcting speech repairs. In *Proc. ACL-94*, pages 295–302, Las Cruces, New Mexico, June 1994.

[HA95] P. Heeman and J. Allen. the TRAINS 93 dialogues. TRAINS Technical Note 94-2, Department of Computer Science, University of Rochester, Rochester, NY 14627-0226, 1995.

[LD93] L. M.; Linebarger, M. C.; Norton and D. A. Dahl. A portable approach to last resort parsing and interpretation. In M. Bates, editor, *Human Language Technology*, pages 31–36, San Francisco, March 1993. Advanced Research Projects Agency, Morgan Kaufman Pub. Inc.