Information Seeking Spoken Dialogue Systems—Part I: Semantics and Pragmatics

Egbert Ammicht, Eric Fosler-Lussier, *Senior Member, IEEE*, and Alexandros Potamianos, *Member, IEEE*

***Abstract*—In this paper, the semantic and pragmatic modules of a spoken dialogue system development platform are presented and evaluated. The main goal of this research is to create spoken dialogue system modules that are portable across applications domains and interaction modalities. We propose a hierarchical semantic representation that encodes all information supplied by the user over multiple dialogue turns and can efficiently represent and be used to argue with ambiguous or conflicting information. Implicit in this semantic representation is a pragmatic module, consisting of context tracking, pragmatic analysis and pragmatic scoring submodules, which computes pragmatic confidence scores for all system beliefs. These pragmatic scores are obtained by combining semantic and pragmatic evidence from the various submodules (taking into account the modality of input) and are used to rank-order attribute-value pairs in the semantic representation, as well as identifying and resolving ambiguities. These modules were implemented and evaluated within a travel reservation dialogue system under the auspices of the DARPA Communicator project, as well as for a movie information application. Formal evaluation of the semantic and pragmatic modules has shown that by incorporating pragmatic analysis and scoring, the quality of the system improves for over 20% of the dialogue fragments examined.**

***Index Terms*—Multimedia communication, Natural language interfaces, speech communication.**

I. Introduction

DESPITE THE significant progress that has been made in the areas of speech recognition and spoken language processing, building a successful dialogue system still requires large amounts of development time and human expertise. In addition, spoken dialogue systems algorithms often have little generalization power and are not portable across application domains. Our main goal in this paper is to reduce prototyping time and effort by creating application-independent tools and algorithms to automate the design process of the semantic and pragmatic modules for use by non-expert application developers.

Manuscript received November 16, 2005; revised August 10, 2006. This work was performed in part while A. Potamianos and E. Fosler-Lussier were with Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 USA. This work was supported in part by DARPA under the auspices of the Communicator Project. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ryoichi Komiya.

E. Ammicht is with Bell Labs, Lucent Technologies, Whippany, NJ 07981 USA (e-mail: eammicht@lucent.com).

E. Fosler-Lussier is with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: fosler@cse.ohio-state.edu; homepage: http://www.cse.ohio-state.edu/~fosler/).

A. Potamianos is with the Department of Electronics and Computer Engineering, Technical University of Crete, Chania 73100, Greece (e-mail: potam@telecom.tuc.gr; homepage: http://www.telecom.tuc.gr/~potam).

Digital Object Identifier 10.1109/TMM.2006.888011

In state-of-the-art spoken dialogue systems, the system must be able to cope with semantic ambiguity and dynamically changing task definitions. Ambiguity might arise from system misrecognitions or inherent ambiguity in the user utterances. The user can also change his/her mind and attempt to modify the semantic state of the system by implicit or explicit requests. To cope with such user and system behavior we introduce the concept of *persistence* in the semantic representation of our system. The system collects and argues with *all* available information that the user supplies *in the course of the dialogue*. In addition, the semantic representation is augmented with a dynamic parameter that determines the evolution of an attribute-value pair over time, i.e., the birth and death of semantic values. This parameter, referred to as the *pragmatic confidence score*, determines the system's confidence in a specific attribute-value based on *all available information* up to the current dialogue turn. The pragmatic confidence of a semantic value is determined from acoustic, semantic, pragmatic and task-specific information sources.

The proposed algorithms build on recent efforts in dialogue system design [7], [6], [35], [24], [19], [27], [23]. The system uses a hierarchical semantic representation that is dynamically constructed based on user input (see also [10], [24]), as well as an agenda – an ordered list of tasks to be accomplished [24], [19]. The pragmatic module presented in this paper builds on prior work on semantic confidence scores [26], [14], pragmatic/dialogue modeling [9], [36], [18] and extends context tracking algorithms such as the Question Under Discussion paradigm [17] to handle hierarchical task relationships. Additional features of the dialogue system include the clear separation of application-specific (e.g., the artificial intelligence module) and application-independent components, a formal representation of semantic ambiguity and application-independent dialogue management algorithms [20]. For recent work on rapid prototyping of dialogue systems and application-independent system design, see also [8].

The main contributions of this paper are the introduction of: 1) a domain-independent semantic representation that can efficiently represent ambiguity and compound attribute-values of multimodal input; 2) a context tracking algorithm that is application-independent, uses a semantic taxonomy and can handle/produce confidence scores; and 3) a pragmatic analysis and scoring algorithm that is application-independent, parsimonious and easy to train; the algorithm takes into account all available acoustic, semantic and pragmatic information and produces confidence scores that are attached to system beliefs. The proposed algorithms are simple, easy to implement, yet general and powerful enough to be applicable to numerous applications of spoken dialogue system design.

The organization of this paper is as follows. We first present an overview of the system architecture in Section II. In Section III, the semantic domain ontology and semantic structures are reviewed. The pragmatic module is presented in two parts, context tracking in Section IV and pragmatic analysis/scoring in Section V. The proposed algorithms are evaluated in Section VI and extensions to the algorithms are proposed in Section V-D.

II. Architecture and System Overview

The semantics and pragmatic modules were implemented as part of the Bell Labs Communicator multimodal dialogue system. The main application domain for the system is travel reservation, i.e., flight booking, car rental and hotel booking. The system was also ported to a movie information application to verify portability and domain-independence claims; however, most of the examples presented in this paper are from the flight reservation domain.

The system, designed and built under the auspices of the DARPA Communicator project, utilizes a star hub architecture: the MIT Galaxy hub provides the basic message routing, logging and state maintenance capabilities for the various application servers [27]. The multimodal dialogue system (MDS) server is connected directly to the hub and comprises of the semantic module, pragmatic module, multimodal dialogue module and the MDS subcontroller. The audio platform (telephony control, audio input/output, speech recognizer, text-to-speech synthesis) [37] and the database back-end are also connected directly to the hub [35].

There are four major components to the multimodal dialogue system.

- **Semantic Module**: Converts speech or graphical input into a common internal semantic representation used by later components. Included are the parsers and interpreters for both the spoken language and visual (particularly textual) input, as well as a module that interprets the intended dialogue action of the user.
- **Pragmatic Module**: Adds dialogue context to input semantics, merges raw data into candidate beliefs, computes the confidence for each candidate and detects semantic ambiguities. These tasks are accomplished by the context tracking, pragmatic analysis, pragmatic scoring, domain knowledge and inference modules.
- **Multimodal Dialogue Module**: Performs internal actions, e.g., database queries via the task manager. The sequence of system actions and goals are stored in a dynamically updated agenda; the corresponding task definition is stored in electronic forms (e-forms). In addition, this module determines the system communicative goals and implements them for the spoken and visual output modalities (in the natural language generation (NLG) and graphical user interface (GUI), respectively).
- **Controller Module**: Handles communication with the hub, audio platform, database server, multimodal input and output devices using a multithreaded architecture.

The activation sequence of various MDS modules and submodules in a typical multimodal interaction (dialogue) turn between the user and the system are shown in Fig. 1. A typical interaction turn proceeds as follows. 1) User input is first analyzed in the semantic module; the parser/interpreter extracts a set of attribute-value pairs from the user utterance and the action interpreter identifies the user's communication goals. Parsing and interpreting occurs separately for each input modality (allowing for late integration). 2) Next, the context tracker augments the semantic representation with dialogue context information. All information supplied by the user via multiple interaction turns and various input modalities is taken into account during pragmatic analysis, permitting identification of semantic ambiguities. The results are extended by the domain knowledge and inference submodule and confidence scores are attached to the system beliefs. 3) Finally, the task manager identifies internal system actions that have to be taken and decides on the system's communicative goals. These are specified in the agenda and can be dynamically updated in the course of an interaction; the specific type of information that will be requested from the user is determined by the electronic form submodule. The realization of the system communication goals is determined by the multimodal interface submodule. The surface realization is then determined by the natural language generation and the graphical user interface submodules for the speech and graphical interfaces, respectively. Finally, the system output information is communicated to the devices and the audio platform by the controller.

III. Semantic Representation

Hierarchical semantic representations have been extensively documented in the artificial intelligence literature (ontologies) [13], [25], as well as in the natural language and spoken dialogue research areas [5], [9], [23]. Advanced dialogue systems typically use a tree representation to hold the current state of the system's belief. Each node has a *type* that represents a *semantic class*; each type has a set of predefined *features* that define a *semantic class structure*. Such semantic representations are sometimes referred to as *typed feature structures* [5]. There has been substantial work in underspecified typed feature structures in the artificial intelligence literature [9], [1] and some work in spoken dialogue system design [9]. However, the bulk of these efforts ignore important problems that appear in spoken dialogue and multimodal systems, namely 1) how to represent and update semantics in interactive systems, where semantics and system beliefs are *dynamically* updated and time plays an important role and 2) how to combine or merge values from various information sources or with different confidences.

Our goal is to create domain independent data structures that hold all information supplied to the system by the user or by the database back-end from the beginning of the dialogue up to the current dialogue turn. Whenever new information becomes available, the system can argue using the complete history information to produce the best interpretation, i.e., the optimal system belief. We are looking for a data representation that is simple but at the same time can represent user input in a persistent manner. To achieve domain independence, the data structures and the semantic algorithms should apply across application domains. We believe that a hierarchical semantic data struc-

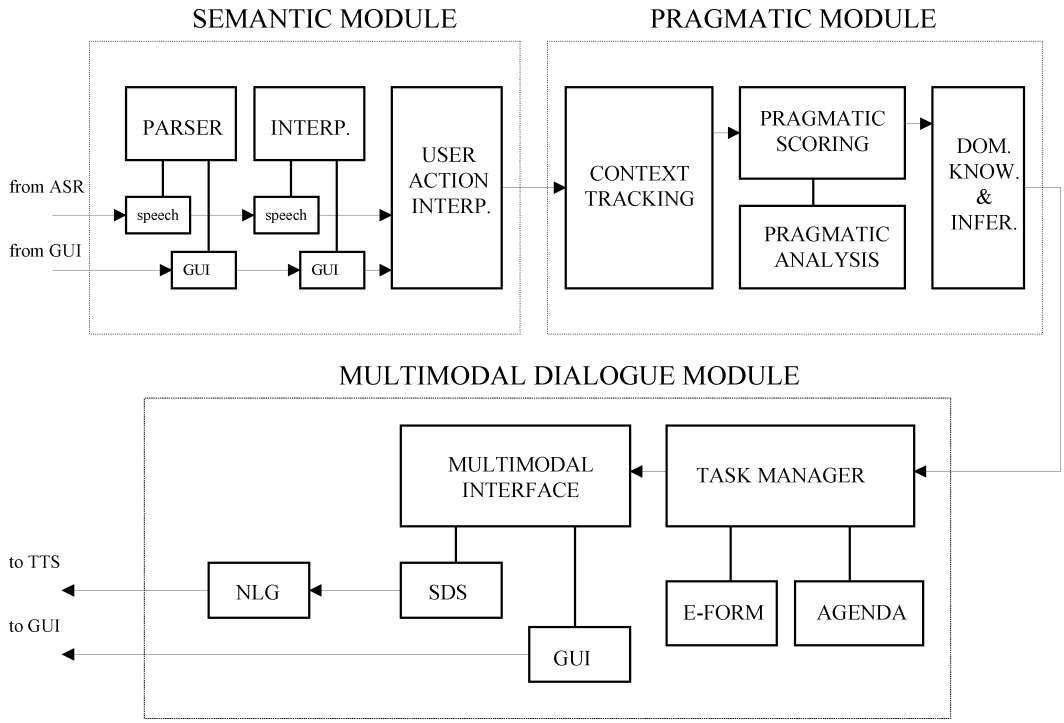


Fig. 1. Architecture of the multimodal dialogue system. Arrows denote the sequence of modules activated in a typical interaction turn.

ture provides advantages for spoken dialogue systems both in terms of readability and improved pragmatic and semantic analysis.[1]

Domain semantics are captured in a **prototype tree** representing the domain ontology [2]. This ontological representation enables our algorithms to operate in terms of the tree structure, ignoring the particular labels on the tree; this is the key point in ensuring domain independence. We also, for the same reasons, maintain all data derived from user utterances and from database queries in tree structures that mirror the prototype tree. In particular, these are 1) the **raw data tree** that holds all values elicited or directly inferred from user input, 2) the **candidate data tree** that holds the candidate values derived from the raw data, and 3) database trees that hold results established from one or more database queries. These are described in depth below.

The **prototype tree** is the basic data structure for organizing the semantics. It is constructed from the ontology of the domain: nodes encode concepts and edges encode the *is-a* or *has-a* relationships in the ontology. A trip, for example, consists of flights, hotel stays and car rentals. A flight, in turn, consists of legs, with departure and arrival dates, times and airports. Data are defined in terms of the path from the tree root to a leaf (the attribute), and the associated value. These paths can be uniquely expressed as a string consisting of a sequence of concatenated node names starting from the root, with a suitable separator such as a period. The attribute for the departure city *"Atlanta"* shown in Fig. 2, for example, is given by the "trip.flight.leg1.departure.city" attribute. Thus, data values are associated with the leaves of the prototype tree and are characterized as attribute-value (AV) pairs.

[1]The necessity of a hierarchical representation to perform complex context tracking tasks is demonstrated in Section IV.

During a dialogue, the system extracts data values from user utterances and places them in a **raw data tree**, a tree structure that mirrors the prototype tree. Values extracted from user input usually contain incomplete references to attributes, e.g., "I want to leave from Atlanta" yields the (".departure.city", "Atlanta") pair. The partial attribute ".departure.city" needs to be combined with dialogue context, e.g., "trip.flight.leg1", to form a complete attribute "trip.flight.leg1.departure.city". For details on the domain independent context tracking algorithm see Section IV. An example of the raw data tree (after context tracking) is shown in Fig. 2. Examples of value and position ambiguity are also shown in this figure (and discussed in detail in the next section).

Frequently, the system has to operate on the raw data by merging or conditioning on values, e.g., in Fig. 2, the two "Atlanta" entries under "leg1.departure.city" should be merged into a single candidate. The formulation of these raw data manipulation algorithms is greatly simplified by maintaining candidates for individual attributes in a **candidate data tree** structure that is separate from the raw data extracted from individual user utterances. As the dialogue proceeds, candidate values are added, removed and otherwise modified, based on user inputs and system rules. Candidate values for specific attributes are scored based on all available evidence, including the raw data history as described in Section V. The candidate data tree is a tree data structure with nodes that contain ordered (by score) lists of values.

Finally, the **database tree** is a semantic structure that holds the database query results. The database tree uses the same domain ontology as the raw data and candidate trees, but the database tree ontology is usually a superset of the candidate tree ontology because database queries often return data values and attributes that are not specified in the query proper (e.g., flight numbers). Storing the database query results in the database tree

structure allows us to use the existing semantic and pragmatic modules for processing user requests that refer to database results, e.g., "What time is flight 314 leaving Atlanta?".

The raw data, candidate data and database trees are data structures similar to *frame representations* or *feature structures*. However, the proposed data structures respect the hierarchical nature of the domain semantics and efficiently represent ambiguity.

A. Semantic Ambiguity

While interacting with the system, data values are extracted from user utterances, interpreted and added to the raw data tree. When the system attempts to derive candidates from these raw data values, two cases of ambiguity arise.

1) A **value ambiguity** occurs when the system creates multiple candidates for a particular attribute from the user input. The cause may be system errors, (e.g., recognizer and parsing errors), or attempts by a user to modify previous (possibly erroneous) input. In Fig. 2, "leg1.departure.city" could be either "New York" or "Atlanta".
2) A **position ambiguity** occurs when the context does not uniquely identify the attribute associated with a given value (Recognizer: "I want $\langle \mathrm{garbage} \rangle$ New York"). Most position ambiguities are automatically resolved by the context tracking, pragmatic analysis and pragmatic scoring algorithms. An example of a position ambiguity is shown at Fig. 2: "New York" could be the arrival or the departure city for the first flight leg. Ambiguities that cannot be resolved by the system are passed on to the dialogue manager. For more details on ambiguity resolution see Section V-A1 and [22].

B. Values and Candidates

The notion that a given attribute is associated with a given value must be broadened in the case of an information seeking spoken dialogue system. When constructing queries, a user does not necessarily specify an exact value, but rather constraints on possibly compound values. This observation leads to the generalization of a **value** to a **value expression**, with syntax and semantics that are domain dependent and specific to each attribute. Consider the following example concepts from the travel reservation application.

- CITY: When reduced to canonical form (e.g., the airport code JFK), a city should have a single, unique value. However, users often specify a constraint, e.g., the city "New York" in the utterance "I want to leave from New York" signifies a constraint on nearby airports "$.\mathrm{departure.city} = (\mathrm{JFK\ or\ LGA\ or\ EWR})$".
- DATE: Date is a compound data structures that typically consists of day, month and year. Dates are usually associated with a single value, but date constraints or intervals are also possible, e.g., "before June first", "any day next week".
- TIME: Time is a compound data structure (hours and minutes) that is usually constrained to belong to a time interval, as for example in the utterance: "I want to leave after 5 p.m.".

Note that concepts such as DATE and TIME appear in other contexts and in many other applications as well, but may have different associated value expressions. Further, each type of value expression associated with a specific attribute has an associated calculus, i.e., different manipulation and combination rules as explained in the following paragraphs.

In addition to being able to represent constraints and compound values, value expressions can also represent ambiguous interpretations of a user utterance, e.g., "next Friday", may often be interpreted as one of two possible dates. Thus, value expressions have the following uses: 1) compound values, 2) constraints on values or compound values, and 3) a representation of ambiguous interpretations.

In the course of a dialogue, users may add and modify specific AV pairs over a number of utterances. Users may implicitly or explicitly specify how values are to be modified or combined, e.g., "change the departure time to 7 p.m." Refinements of existing constraints can also be specified by the user, e.g., reducing or expanding a time interval. Establishing new candidate values, or merging raw values with previously defined candidates is usually predefined and attribute specific. For example, our travel reservation system provides a predefined default operation for time interval merging: when a user specifies a new time or time interval, it is compared to the existing candidates (for that attribute). If the two intervals overlap, they are merged to form a new, longer time interval.

In general, the modification of candidate values by combining raw data values is an attribute specific problem. For sets of values (enumerations) and for intervals, operations such as unions, intersections and differences are readily conceivable. The specific calculus will however depend on the actual attribute, even for similar concepts. A detailed description of how to create new candidate values from raw data, which we have expressed as a calculus for value types, is beyond the scope of this paper. The candidate value creation problem remains an important research area that has been largely ignored in the natural language and artificial intelligence literature. In our experience, creating the appropriate abstraction that enables a general and domain independent solution is a hard problem.

C. Semantic Parser and Interpreters

In order to fill the raw data tree, user input (either from natural language input or spoken input) is parsed using a recursive finite-state parser [21] that acts as a semantic island parser. The parser iteratively builds up semantic phrase structures by transducing string patterns in the input into their semantic concepts (e.g., "Atlanta" is rewritten as $\langle \mathrm{CITY} \rangle$; "arriving in $\langle \mathrm{CITY} \rangle$" is subsequently transformed into $\langle \mathrm{TOCITY} \rangle$). Parser concepts are mapped to application concepts that correspond to branches of the prototype tree, e.g., $\langle \mathrm{TOCITY} \rangle$ is transformed to "arrival.city". Thus, the output of the parser is a set of tree branches (islands) with their corresponding values, e.g., ("arrival.city","Atlanta"), ("time", "five pm"). The values are then transformed by a set of application-dependent routines in an interpreter, yielding a canonical form that can be used by the raw data tree. For example, in the travel domain, cities are changed into airport codes, date strings (e.g., "Tuesday") are

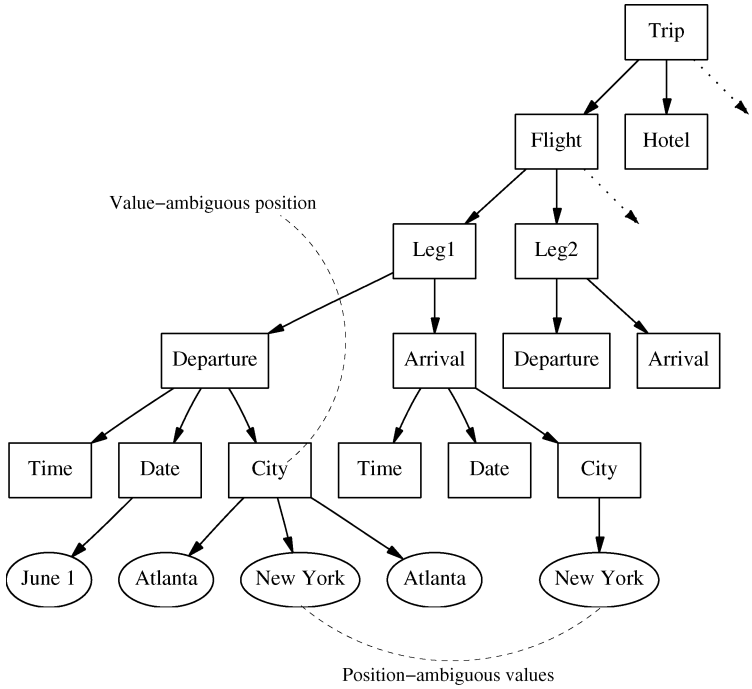


Fig. 2. Raw data tree illustrating value ambiguity (departure city Atlanta or New York) and position ambiguity (New York).

converted to the relevant date, and phrases like "first class" are modified into corresponding fare codes.

D. User Action Interpreter

The allowed range of user requests is not limited to specifying attribute-value pairs: it goes beyond the basic system functionality (query formulation and database results navigation) and offers the user the ability to dynamically update application semantics, navigate through the application and correct erroneous system beliefs.

User requests are categorized based on the result they have on the interaction and semantic state. They are classified by the user action interpreter into one or more of the following domain acts based on cues extracted from the parser.

- **Fill requests** (the default actions), e.g., "I want to leave from Atlanta" to create candidate values;
- **Information requests**, e.g., "what is the departure city?", to ascertain the value for a specific attribute;
- **Clear requests**, e.g., "clear the departure city", to force the removal of all candidate values for a given attribute;
- **Freeze requests**, e.g., "freeze the departure city", to inhibit the system from further changing the value of a particular attribute;
- **Change requests**, e.g., "change Atlanta to New York", "change the departure city to New York", or "not Atlanta, New York!", to change the value of a specific attribute;
- **Focus Change requests**, e.g, "let's first make a car reservation" to go to the car reservation application;
- **Tree/Agenda transformation requests**, e.g, "let's add a side-trip to Chicago here", resulting in the insertion of a new subtree in the raw/candidate data trees and a new agenda item;
- **Database result navigation requests**, e.g, "next flight" or "get me an earlier flight";
- **User Interface requests**, e.g, "help" or "repeat".

As the above examples show, the user is allowed to refer to attributes directly, modify or replace attribute-value pairs, refer to the tree structure, request specific information or modify the structure itself. Given the user's ability to specify attributes, the parser output was extended to include **focus** information providing additional context or context modification parameters. A focus consists of additional partial paths and/or values and may further contain **iterators** that formalize position modification in an ordered set. For example, consider the utterance "When are we leaving New York?": the parser outputs an *information request* action with the *focus* ".time", as well as a partial path and value that constrain the query (".departure.city","New York"). A more complex example is provided by the user utterance "What is the departure time in the first leg?" where an *information request* has two *focus* directives: one with path ".departure.time" and a second with path ".leg" and associated iterator[2] "begin".

Overall, the semantic representation is domain-independent and uses only domain specific data structures (e.g., prototype tree) and attribute-specific interpreters to represent and argue with task semantics. The typical output of the semantic module consists of the following information: 1) the type of action the user is requesting, 2) (optional) focus information, 3) a list of

[2]To illustrate the concept of an iterator, consider the concept *leg* for the travel application. There are one or more flight legs in a trip, forming an ordered set numbered $1, \ldots, N$. Users typically refer to individual legs by their absolute or relative position (e.g., "the last leg", or "the next leg"). The parser extracts a partial path ".leg" and an *iterator* with a position parameter as part of the focus information. Iterators have the standard semantics *begin*, *current* and *end* with offsets as appropriate: "the next leg" refers to a partial path matching ".leg", with an associated iterator "current+1" used to specify the actual flight leg desired.

TABLE I
CONTEXT TRACKING EXAMPLES FOR DIFFERENT MATCH TYPES. POSITION AMBIGUITY IS RESOLVED USING THE DISAMBIGUATION RULES IN TABLE II FOR "INTERPOLATED MATCH" AND THE "USER OVERRIDE" RULE FOR "OVERLAPPING MATCH"

Match Type	Context r (from system)	Partial Attribute l (from user)	Candidate Full Attr. a	Selected Full Attr.	Conf.
exact	trip.flight.leg1	.departure.city	trip.flight.leg1.departure.city	trip.flight.leg1.departure.city	1
interpolated	trip.flight	.departure.city	trip.flight.leg1.departure.city trip.flight.leg2.departure.city trip.flight.leg3.departure.city	trip.flight.leg1.departure.city	0.9
overlapping	trip.flight.leg1.arrival	.departure.city	trip.flight.leg1.departure.city trip.flight.leg1.arrival.city	trip.flight.leg1.departure.city	1

values, and 4) a list of partial attributes, e.g., "I want to fly to New York" results in the action *"fill request"*, the value *"New York"* and the partial attribute *".departure.city"*. The raw attribute-value pairs are then passed on to the pragmatic module for context tracking, pragmatic scoring and merging of processed attribute-value pairs onto the candidate data tree. The details of these operations are presented in the next sections.

IV. CONTEXT TRACKING

The pragmatic module mainly consists of the context tracking, pragmatic analysis and pragmatic scoring submodules. The function of the context tracking algorithm is to incorporate the dialogue context information into the parser/interpreter output to obtain complete attributes, and where appropriate, to infer attribute-value pairs. For example, suppose that the parser returns the pair (".departure.city", "Atlanta") and the dialogue context (provided by the dialogue manager) is "trip.flight.leg1", i.e., the user is providing information about the first leg of a flight. In this example, the attribute provided by the parser ".departure.city" is partially (or ambiguously) defined; by adding the dialogue context "trip.flight.leg1" the fully defined attribute-value pair is obtained ("trip.flight.leg1.departure.city", "Atlanta"). Note that the resulting attribute corresponds to a full path from the root to a leaf of the prototype tree (see Fig. 2).

The context tracking algorithm is specific to the request type. For *fill requests*, the context tracking algorithm finds the appropriate attribute for a value extracted from a user utterance, given a single associated partial attribute. For *information requests* and *change requests*, where user utterances often contain explicit references to values, (e.g., "When are we leaving New York") context tracking is a harder problem. The matching algorithms must construct full attributes from a number of constraints on the set of possible attributes. These constraints arise from user focus information, from the dialogue context and from partial attributes and values that the parser is able to extract from a user utterance, and may further involve iterators. In general, context tracking is formulated as a set of successive (string matching) filters applied to a set of possible attributes. The context tracking algorithm is detailed next.

A. Attribute for a Given Value

First, consider the simple case of establishing an attribute for a given data value for *fill requests*. The system maintains an expected dialogue context for every dialogue turn. This context is expressed as a path r from the root to some node of the prototype tree, e.g., "trip.flight.leg1". Values extracted from a user utterance by the semantic parser/interpreter are associated with a partial attribute, i.e., a path l from some prototype tree node to a leaf. Derivation of the full attribute for a given value requires combining the dialogue context r and the partial attribute l to form a complete path a from the root of the tree to the leaf. In the following discussion, we use the operator $\cdot$ to express concatenation.

A succinct formulation of the context tracking algorithm is to specify a set of possible (fully defined) attributes a, i.e., paths from the root to a leaf of the prototype tree and to consider a set of filter constraints of the form $*.l$ and $r.*$, where $*$ denotes any sequence of concepts from the prototype tree. In general, we need to consider three types of matches.

1) **Exact match:** the context and partial attribute can be directly concatenated to form a full path, i.e., $a = r \cdot l$ exists in the prototype tree. For example, given the context r "trip.flight.leg1" and a datum with partial attribute l ".departure.city", the complete path "trip.flight.leg1.departure.city" is seen to exist in the tree in Fig. 2.
2) **Interpolated match:** the context and/or partial attribute need to be extended to form a full path, i.e., for some paths m, the attribute $a = r \cdot m \cdot l$ exists in the prototype tree. For example, a context "trip.flight" and a datum with partial attribute ".departure.city" may be completed with the choice $m =$ ".leg1", $m =$ ".leg2" or $m =$ ".leg3".
3) **Overlapping match:** the context and partial attribute overlap and need to be truncated to form a full path, i.e., there exists a path m such that $r = r' \cdot m$ for which $a = r' \cdot l$ exists in the prototype tree. In this latter case, the general strategy chosen is to minimize the length of the path m, i.e., to maximize the overlap. The overlapping substrings may not necessarily agree. For example, the context "trip.flight.leg1.departure" may have to be shortened to "trip.flight.leg1" so as to combine with ".arrival.city" to form a possible attribute. By shortening the dialogue context, we essentially allow the user to override the system's context.

In Table I, context tracking examples are shown; note the position ambiguity for the cases of "interpolated match" and "overlapping match". The list of candidate attributes a is shown for each case. The "Selected Full Attribute" and "Confidence" columns refer to the context disambiguation algorithms discussed next.

To resolve (or reduce) the context ambiguity in the case of "interpolated match", we introduce a set of prioritized rules. Each rule maps a partial attribute l to an extended partial attribute l'. The rules are in the form of tuples (r, l, l', γ) where

r is the dialogue context, l is the partial attribute, $l' = m \cdot l$ is the extended partial attribute and γ in [1] is the normalized confidence (probability) attached to this specific rule. Examples of context tracker disambiguation rules are shown in Table II.

Rules fire only in the case of "interpolated match", with dialogue context r or, in general, $r' = r \cdot m$, and partial attribute l. Consider the "interpolated match" context tracking example where $r =$ "trip.flight"; and $l =$ ".city". Using the rules in Table II for $l =$ ".city" in the context of $r =$ "trip.flight", l is extended to either ".departure.city" or ".arrival.city" each with probability[3] 0.5. Next the set of rules for $l =$ ".departure.city" and $l =$ ".arrival.city" fire. The result of the context tracking algorithm is a set of two full attributes $a =$ "trip.flight.leg1.arrival.city" and $a =$ "trip.flight.leg1.departure.city" each with probability 0.45 (other options also exist but with very low probability).[4] In the example above, position ambiguity (while not fully resolved), is significantly reduced to two possible attributes. In Table I, a similar example is shown where the "interpolated match" case ambiguity is fully resolved using the rules in Table II. The confidence attached to this context disambiguation decision is 0.9.

The proposed mechanism for context resolution allows the exclusion of undesirable paths and the reduction of the probability of others. In cases where position ambiguity persists after the application of the context resolution rules, multiple attributes are kept in the raw and candidate data trees. An example of position ambiguity representation in the raw data tree is shown in Fig. 2. Position ambiguity affects the confidence that the pragmatic module attaches to attribute-value pairs. The context tracking confidence score γ (normalized in [0, 1]) is used as a weight in the pragmatic scoring algorithm described in Section V.

To resolve ambiguity in the "overlapping match" case, priority is given to the partial path obtained from user input, i.e., in case of disagreement the user overrides the system context. In practice, we have found this to be a good choice for our spoken dialogue system that uses a mixed initiative dialogue manager. More research is needed to determine the best way to resolve context conflicts between the system expectation and the user input.[5]

A further refinement of the context-tracking system is to allow for context changes while analyzing data from a given user utterance. These changes can be made unconditionally, or may be pushed on a stack, with previous contexts searched if the current context should fail within the current utterance. For example, the user can provide information for both flight and car reservation in the same sentence. In this case, the context switch is detected; the "trip.flight" context is used to analyze the first part of the sentence and the "trip.car" context is used for the second part.

[3]Note that the ".stopover.city" extension is excluded as undesirable by setting its confidence γ to 0.

[4]Note that when two rules fire in succession the confidences of the rules are multiplied.

[5]In any case, it is easy to modify the context tracking algorithm by distributing the probability between the two options: one that favors the system's expectations (i.e., dialogue context wins) and one that favors the user's input (i.e., partial attribute wins).

TABLE II
APPLICATION DEPENDENT EXAMPLE RULES USED FOR CONTEXT DISAMBIGUATION IN THE CONTEXT TRACKER FOR THE CASE OF "INTERPOLATED MATCH"

Context r	Partial Attr. l	Extended Attr. l'	Conf. γ
trip.flight	.city	.departure.city	0.5
trip.flight	.city	.arrival.city	0.5
trip.flight	.city	.stopover.city	0
trip.flight	.departure.city	.leg1.departure.city	0.9
trip.flight	.departure.city	.leg2.departure.city	0.09
trip.flight	.departure.city	.leg3.departure.city	0.01
trip.flight	.arrival.city	.leg1.arrival.city	0.9
trip.flight	.arrival.city	.leg2.arrival.city	0.09
trip.flight	.arrival.city	.leg3.arrival.city	0.01

B. Attribute With Additional Constraints

Allowing users to explicitly refer to attributes gives rise to more complex attribute matching problems than those considered above. The data values, partial paths, and focus information extracted from a user utterance each serve to constrain the possible set of attributes. The context tracking algorithms are generalized as a set of constraints that are applied to an initial set of possible attributes or attribute-value pairs, thereby filtering out a subset of attributes that meets the constraints.

To motivate the algorithmic formulation, consider the following example for the *information request* query "When are we leaving New York?" and its solution. The parser returns a focus that consists of a partial path ".time" and an associated partial attribute-value pair (".departure.city", "New York"). The context tracking problem in this case consists of two subproblems that have to be solved in order: 1) determine the context for the attribute-value pair(s) that constrain the query, i.e., ".departure.city," and 2) determine the context for the focus (or subject) of the query, i.e., ".time".

To solve problem 1) we search in the candidate (or database) tree for full paths that end in ".departure.city" for which the leaf node ".city" has the associated candidate value "New York". Suppose that the candidate tree contains two ".city" leaf nodes with specific values "New York" ("trip.flight.leg1.arrival.city", "New York") and ("trip.flight.leg2.departure.city", "New York"). Given that "New York" is a ".departure.city" the correct context is "trip.flight.leg2.departure.city", i.e., the user is referring to the second leg of the flight.

Note that the context tracking algorithm for *information requests* is very different compared to *fill requests* (see Section IV-A). Dialog context is not used here[6]; the context is found by comparing the partial attribute-value pair with (full) attribute-value pairs found in the candidate or database trees, that is, instead of looking for attribute matches in the prototype tree we are looking for value matches in the candidate tree or database tree.

Continuing the example, the focus context tracking problem 2) is solved by applying two filters sequentially. First we find the set of attributes that end in ".time" (the target concept of the *information request*) from the prototype tree, i.e., "trip.flight.leg1.departure.time" … "trip.flight.leg3.arrival.time". Next, we select the subset of attributes ending in ".time" that share the longest possible path (starting from

[6]In general, the dialogue context plays a secondary role in context tracking for *information requests*.

the root) with the attribute(s) that constrain(s) the query, in our example "trip.flight.leg2.departure.city", resulting in "trip.flight.leg2.departure.time", i.e., the user is asking about the departure time of the second flight leg.[7]

In general, the context tracking algorithms are implemented using the following filters:

- $Attribute\ Match(a)$: find all attributes in the prototype tree that contain the partial path a, e.g.,

$$Attribute\ Match(\text{`.arrival.city'}) = \{\text{`trip.flight.leg1.arrival.city'} \ldots \text{`trip.flight.leg3.arrival.city'}\}.$$

- $Attribute\,Constrain(L, a)$: select from a list of attributes L those sharing the *longest possible path* with the given partial path a, e.g.,

$$Attribute\ Constrain(Attribute\ Match(\text{`.time'}), \text{`.arrival.city'}) = \{\text{`trip.flight.leg1.arrival.time'} \ldots \text{`trip.flight.leg3.arrival.time'}\}$$

  selects all the "time" attributes containing "arrival".
- $Value\ Constrain(L, v)$: select from a list of attributes L those that contain the specified leaf value v in the candidate or database trees, e.g., $Value\ Constrain($ $Attribute\ Match$".city"), "Chicago") returns the subset of "city" attributes that have been instantiated with the value "Chicago".
- $Attribute\ Truncate(L, a)$: same as Attribute Constrain only here the longest overlapping path between members in L and a is returned,[8] e.g.,

$$Attribute\ Truncate(\{\text{`.leg1.departure.city'}, \text{`.leg1.arrival.city'}\}, \text{`.leg1.departure.time'}) = \{\text{`.leg1.departure'}\}$$

The context tracking algorithm applies these filters in sequence to obtain a set of possible full attributes. The initial set chosen critically depends on the user request type. The starting set of attributes is drawn from the following.

- *Prototype tree*: in the case of *fill requests, focus change and tree/agenda transformation requests* (using the *Attribute Match*, *Attribute Constrain*, and *Attribute Truncate* filters).
- *Candidate data tree*: in the case of *information requests, clear, fill and change requests* (using the *Value Constrain* filter).
- *Database tree*: in the case of the *database result navigation request* (using the *Value Constrain* filter).

As shown in the additional examples below, the exact sequence of application of these filters further depend upon the actual data supplied.

The context tracking example for the user query "When are we leaving New York?" shown in Table III, is formulated in terms of the context filters. For attribute-value context tracking the *Attribute Match* and *Value Constrain* filters are applied in sequence. For focus context tracking the *Attribute Match* and *Attribute Constrain* filters are applied. Note that *Attribute Constrain* filter takes as input the output of the *Value Constrain* filter. Context tracking for other types of *information requests* is simpler, e.g., for the user query "What is the departure time?", the context tracker applies the filter

$$Attribute\ Constrain(Attribute\ Match(r), \text{`.departure.time'})$$

where r is the dialogue context. Note that this is the same filter used in Section IV-A.

TABLE III
CONTEXT TRACKING EXAMPLE FOR THE INFORMATION REQUEST "WHEN ARE WE LEAVING NEW YORK?"

Context Tracker Input
action: information request
focus: '.time'
AV pair: ('.departure.city', 'New York')
Attribute Value Context Tracking
Attribute Match('.departure.city')=
{'trip.flight.leg1.departure.city',
'trip.flight.leg2.departure.city',
'trip.flight.leg3.departure.city'}
Value Constrain(*Attribute Match*(
'.departure.city'), 'New York')=
{'trip.flight.leg2.departure.city'}
Focus Context Tracking
Attribute Match('.time') =
{ 'trip.flight.leg1.departure.time',
'trip.flight.leg1.arrival.time',
'trip.flight.leg2.departure.city',
'trip.flight.leg2.arrival.time',
'trip.flight.leg3.departure.city',
'trip.flight.leg3.arrival.time'}
Attribute Constrain(*Attribute Match*('.time'),
'trip.flight.leg2.departure.city')=
{'trip.flight.leg2.departure.time'}

The most complex set of filters in our system occurs for the *change request*, due to its many variations. Consider two examples "Change the time to 3 p.m." and "Change 3 p.m. to 4 p.m.". In the first case, the context tracker runs the filter $Attribute\ Constrain(Attribute\ Match(r), \text{`.time'})$, where $r$ is the dialogue context. In the second case, the filter $Value\ Constrain(Attribute\ Match(\text{`.time'}), \text{`3 p.m.'})$ is used. If that fails to produce a unique result the filter

$$Attribute\,Constrain(Attribute\,Match(r),\ Value\ Constrain(Attribute\ Match(\text{`.time'}), \text{`3 p.m.'}))$$

is used in sequence.

Overall, the context tracking algorithm proposed here can efficiently handle a variety of user requests including complex requests where users refer to or constrain attributes and values directly, e.g., for *information* and *change requests*. In addition, the proposed context tracking algorithm can use sentence focus and dialogue context information to resolve position ambiguity in user requests. Most importantly, the proposed context tracking

[7]Note that we were able to find the overlapping context "trip.flight.leg2.departure" thanks to the hierarchical semantic data structures used, i.e., the hierarchy in semantics makes it possible to solve such complex context tracking problems.

[8]Note that '.leg1' (i.e., the intersection of '.leg1.arrival.city' and '.leg1.departure.time') is not returned since it is shorter than '.leg1.departure'.

algorithm is application-independent; only the domain-dependent context resolution rules need to be specified by the system developer.

V. Pragmatic Analysis and Scoring

In spoken dialogue systems, it is often the case that multiple candidate values exist for a particular attribute. These (often conflicting) candidate values may be supplied by the user over multiple dialogue turns or can be due to misrecognitions or misinterpretations of user input. In this work, we propose a general framework for dealing with the semantic ambiguity in spoken dialogue systems. The proposed *pragmatic analysis and scoring algorithm* combines all the information supplied by the user at the acoustic, linguistic, semantic and pragmatic levels, and produces a ranked order list of the multiple candidate values for each attribute along with a confidence score attached to each value. The proposed algorithm takes into consideration all the relevant dialogue history, i.e., all user-system interaction up to that point in the dialogue. The proposed framework extends work on acoustic confidence scores and (more recent work on) semantic confidence scores [14], [26], to produce *pragmatic confidence scores*.

The main steps of pragmatic analysis and scoring are the following: 1) confidence scores are attached to each candidate value by the speech understanding system as described in [26];[9] 2) scores are updated using the application independent pragmatic analysis rules described in Section V-B2, e.g., if the user answers "yes" to an explicit confirmation question the pragmatic score of the candidate in question increases, 3) scores of all the candidate values for a specific attribute are updated based on the similarity between the recently supplied candidate value and the preexisting candidate value(s), e.g., if the user gives two conflicting values for an attribute then the pragmatic scores for both candidates are reduced, and 4) if two candidates have the same value the two candidates are merged and their combined score is increased accordingly.

To better demonstrate the inner workings of the scoring algorithm consider the three examples shown in Table VI. Pragmatic scores are normalized between -1 (certain to be incorrect) and 1 (certain to be correct). In the first dialogue fragment, the user gives the departure city information twice: "Boston" (due to a misrecognition) and "Austin". The original confidence 0.5 is reduced to 0.44 for both candidates due to their conflicting values as detailed in the next section. In the second dialogue fragment, we have three pieces of information for the departure city: "Boston", "not Boston", "Austin". "not Boston" is interpreted as negative evidence for "Boston" resulting in 0 confidence for "Boston" and 0.5 confidence for "Austin". Finally, in the third dialogue fragment, the user again supplies three pieces of information for the departure city "Austin", "Austin, Texas" and "Boston". The first two correspond to the same value "Austin" and the score update formula gives 0.72 for "Austin" and 0.38 for "Boston". The quantitative inner workings of the pragmatic analysis and score update algorithms are presented next. (Note that the system uses the airport codes "BOS" for "Boston" and "AUS" for "Austin").

[9]If no semantic confidence scores are available, the acoustic confidence scores can be used instead. If neither the speech recognizer nor the speech understanding modules are capable of producing confidence scores, a generic (constant) confidence value can be used to initialize all candidate values.

TABLE IV
Pragmatic Scoring Example for the Attribute "trip.flight.leg1.departure.city" When Three Pieces of Evidence are Available: "BOS", "AUS" and "AUS"; Successive Updates of the Score s for Each of the Two Possible Candidates "BOS" and "AUS" are Obtained Using the MYCIN Formula ($p = 0.5$)

Candidate	Evidence	c	γ/n	p	e	s
BOS						0
	BOS	1	1	0.5	0.5	0.5
	AUS	0	1	0.5	-0.1	0.44
	AUS	0	1	0.5	-0.1	**0.38**
AUS						0
	AUS	1	1	0.5	0.5	0.5
	AUS	1	1	0.5	0.5	0.75
	BOS	0	1	0.5	-0.1	**0.72**

A. Score Update Algorithm

Given multiple candidates for a particular attribute, we require a scoring mechanism that is sufficiently parameterized to allow training. At the dialogue management level the scoring algorithm will be used to determine if confirmation or disambiguation is needed for a particular candidate value. Some desirable characteristics of the scoring algorithm are that 1) the algorithm should allow the use of any kind of evidence for or against a particular candidate, 2) the score should be independent of the order of application of each type of evidence,[10] and 3) the range of scores should be normalized to a fixed range to allow for comparisons.

Based on these observations, we chose MYCIN style confidence factors [28], [11] that are normalized in the range $[-1, 1]$, where 1 denotes certainty for the value correctness, 0 denotes ignorance about the true value and -1 denotes certainty for the value incorrectness. Candidate scores s are updated using evidence e for or against the candidate based on the MYCIN formula

$$S_v(s,e) = \begin{cases} s + (1-s)e, & s \geq 0, e \geq 0 \\ s + (1+s)e, & s < 0, e < 0 \\ \frac{s+e}{1-\min(|s|,|e|)}, & \text{otherwise} \end{cases} \quad (1)$$

where e denotes evidence for (positive) or against (negative) an attribute-value v, s is the score before evidence e is considered and $S_v(s,e)$ is the updated score when evidence e is also taken into account. For each candidate value the formula is applied repeatedly, once for each piece of evidence. As described in the Section V-B we use two types of evidence: raw candidates and pragmatic analysis rules.

1) Ambiguity Detection and Resolution Using Pragmatic Scores: Pragmatic confidence scores can be used to determine whether the value of an attribute should be considered unknown, ambiguously or unambiguously known. Based on the confidence score values (and differences between scores) the dialogue manager can detect and resolve semantic ambiguity.

For this purpose, we define the threshold parameters $\sigma_1 > 0$ and $\sigma_2 > 0$ to govern ambiguity detection as follows: 1) for

[10]Although, in general, this is a desirable property of the scoring mechanism, there are dialogue system applications where it is more appropriate to introduce a time forgetting factor for past candidate values supplied by the user.

a unique candidate with score s to be considered established, its score must be positive and sufficiently large, i.e., $s \geq \sigma_1$ and 2) given multiple candidates with scores $s_1, s_2, \ldots$, where $s_1 \geq s_2 \geq \cdots \geq s_n$, we require $s_1 \geq \sigma_1$ and $s_1 - s_2 \geq \sigma_2$ to consider candidate 1 sufficiently well established (unambiguously known). We currently trained the system to use the settings $\sigma_1 = .2$, $\sigma_2 = .2$.

Once ambiguity is detected, it is up to the dialogue manager to decide when and how to best resolve ambiguity. Typically the dialogue manager will attempt to resolve ambiguity as early as possible, either using implicit confirmation, explicit confirmation or direct disambiguation subdialogues. The selected disambiguation method depends on the "degree" and "importance" of the existing ambiguity, i.e., how close are the pragmatic scores and score differences to the thresholds σ_1 and σ_2 and how important are the attribute-values in question for the successful completion of the task. For more details see [22].

B. Types of Evidence

Evidence for or against candidate values in our system comes from two main sources: 1) evidence derived from the raw data and associated scores derived from the user utterance and 2) evidence derived from pragmatic considerations by analyzing the dialogue. Details about each type of evidence follow.

1) Raw Data: Raw data values typically have confidence scores attached, e.g., an acoustic confidence or semantic confidence score.[11] To use such scores in (1), they are first combined to derive a single individual score p with range [0, 1] based on the score type and the particular attribute of the datum. The strength of the evidence e for or against a particular candidate s is obtained by setting

$$e = \begin{cases} \alpha c p, & \text{if } \mathbf{p} \text{ and } \mathbf{s} \text{ are consistent} \\ \beta p, & \text{otherwise} \end{cases} \tag{2}$$

where c is the degree of consistency between the raw value and the candidate value (e.g., the percent overlap in the case of two ranges), and where $\alpha \in (0, 1]$ and $\beta \in [-1, 0]$ are constants. We currently use $\alpha = \gamma/n$, $\beta = -0.2\gamma/n$, where n is the number of position ambiguous attributes found for the raw datum and γ is a weight obtained from the context tracking algorithm (see Section IV).

For example, consider the case in the third subdialogue presented in Table VI where there are three pieces of information for the attribute "…departure.city": "BOS" (Boston) and "AUS" (Austin) occurring twice, resulting in two candidates, "BOS" and "AUS" (note that there is no position ambiguity and for all candidates $\gamma = 1$ and $n = 1$). In this example, $\alpha = 1$, $\beta = -0.2$, $c = 1$ and $p = 0.5$.

Given an initial value of 0 for the score of the AV pair candidate ("…departure.city","BOS"), the MYCIN formula with evidence $e_1 = 0.5$ for "BOS" yields a new score $s_1 = 0.5$. Next, the evidence for "AUS" is computed as $e_2 = -0.2 * 0.5 = -0.1$ (the evidence is negative since the value "AUS" is inconsistent with "BOS"). This evidence value is then used in the MYCIN formula to obtain the updated score $s_2 = (s_1 + e_2)/(1 - \min(|s_1|, |e_2|)) = 0.44$. The process is repeated for the third piece of evidence "AUS" to get $e_3 = -0.1$ and a final score $s_3 = (s_2 + e_3)/(1 - \min(|s_2|, |e_3|)) = 0.38$ for "BOS". Note that we would have gotten the same results independent of the order in which the three values appear in user input. The score for the AV pair ("…departure.city", "AUS") is similarly obtained by computing the three evidence scores for the candidate "AUS" $e_1 = 0.5$, $e_2 = 0.5$ and $e_3 = -0.1$ due to the respective occurrences of "AUS", "AUS" and "BOS" in the user input, and using the MYCIN formula to update the score for the candidate value, successively obtaining new scores $s_1 = 0.5$, $s_2 = 0.75$ and $s_3 = 0.72$. For details see Table IV.

Given the ambiguity thresholds provided in Section V-A1 the pragmatic module decides that in this case the value for the attribute "…departure.city" is unambiguously established to be "AUS" because $s_{AUS} = 0.72 > \sigma_1$ and $s_{AUS} - s_{BOS} = 0.72 - 0.38 > \sigma_2$.

In general, the degree of consistency c between values is a function of the value type.[12]

At each dialogue turn, the scoring operation proceeds as follows: 1) as each datum is added to the raw data tree, its score is computed; 2) the evidence that the new datum introduces is computed using (2); and 3) the scores of all known candidates are updated based on the new evidence using the score update formula (1). Note that the score computation algorithm is simple, yet powerful; only two parameters α and β need to be trained to quantify the contribution of positive and negative evidence. More complex scoring algorithms based on Bayesian networks are discussed in Section V-D.

2) Pragmatic Analysis: The spoken dialogue system uses a dialogue strategy of frequent implicit confirmation. For example, the system prompt "Flying from Atlanta to New York. When do you want to leave Atlanta?", attempts to obtain implicit confirmation of the departure and arrival city, before asking the user for the departure date. Given the nature of the system prompts, we can classify the information extracted from the resulting user response as follows.

1) **Expected requested response (RR)**: the user input contains values, attributes and action requests that are a direct response to the system prompt, i.e., the user provides the information that the prompt was designed to elicit. For example, the user responds with "Friday July 13th" (providing departure date information) to the example prompt given above.
2) **Expected error response (ER)**: the user input contains attributes and values that are not a direct response to the prompt, but instead relate to the implicit confirmation part of the prompt, i.e., the user attempts to correct (or verify) the value of a specific attribute that is being implicitly confirmed by the current prompt. For example, the user responds with "I am flying to Newark New Jersey" to the prompt given above, attempting to

[11] In the absence of such scores, we use a modality-dependent default score: .5 for speech input and .9 for graphical input.

[12] A more complex scoring example where the degree of consistency is not $c = 1$, is provided by the candidates extracted from user inputs "New York" and "Newark", which are partially consistent since Newark is one of the three airports around New York. By defining the consistency between two candidates consisting of a set of discrete values as the ratio of the cardinality of their intersection to the cardinality of their union, we obtain $c = 0.33$.

correct the AV pair ("…arrival.city", "New York") given in the first part of the prompt.

3) **unrelated response (UR)**: The user response contains attributes and action requests that are neither a direct response to the prompt nor mentioned in the implicit confirmation. For example, the user provides a departure date for the return flight when asked for the departure date of the first leg, as in "I want to leave on the tenth and *return on the fourteenth*".
4) **"Yes"/"no" confirmation response (Confirm)**: the user responds with a "yes"/"no" (or synonyms) to an explicit/implicit confirmation question or disambiguation prompt, e.g., to the disambiguation prompt "Is the departure New York or Boston?" the user responds with " *Neither*, I am leaving from Austin". A simpler example is the explicit confirmation prompt "Are you leaving from Atlanta on July fifth?" and the user reply "Yes, that's correct".

Note that user input often contains more than one type of response, e.g., the user reply "I want to leave on the tenth and return on the fourteenth" (to the "…When do you want to leave…?" prompt) contains both an RR (departure date for current leg) and a UR (departure date for return flight leg). The classification algorithm of user input semantics to RR, ER and UR is automatic and application independent.[13] The confirmation response type is identified by the semantic module.

During pragmatic analysis, the user input is characterized and its relation to the make-up of the corresponding system prompt is examined. Specifically, the following prompt types are identified:[14]

1) **Explicit single attribute confirmation (ESAC)**: a single attribute-value pair is being explicitly confirmed, e.g., the ("…departure.city","Paris") AV is being confirmed by the prompt "Are you leaving from Paris?".
2) **Explicit multiple attribute confirmation (EMAC)**: in this case, explicit confirmation of more than one AV pair is being requested, e.g., "Are you leaving from Paris on January 15?".
3) **Implicit single attribute confirmation (ISAC)**: same as ESAC but in this case the confirmation is implicit, e.g., "You are leaving from Paris …" or "At what time are you leaving from Paris?".
4) **Implicit multiple attribute confirmation (IMAC)**: same as EMAC but implicit, e.g., "You are leaving from Paris on January 15…".
5) **Value disambiguation (VD)**: a request to select one of several (typically two) values for an attribute, e.g., "Are you leaving from Boston or New York?".
6) **Attribute disambiguation (AD)**: a request to select one of several attributes for a given value, e.g., "Is Boston your arrival or your departure city?".

[13]The implementation of the user input classification algorithm is very simple in our system, thanks to the common semantic representation (prototype tree) for both system output and user input: the attributes and actions in the implicit confirmation and information request parts of the system prompt are compared to the attributes and actions in user input to classify input semantics into RR, ER or UR.

[14]The proposed tagging scheme has similarities with the dialogue act tagging scheme proposed [31] for dialogue evaluation purposes. For tagging of error-related pragmatic information in dialogues, see also [3].

The precise definition of the pragmatic analysis rules for the evidence derived depends on the system capabilities as well as the aggressiveness with which the system confirms or abandons hypotheses about the appropriate candidate values. In Table V, the rules that are used in the current system are shown. The rules are fired for specific types of user input (RR, ER, UR, Confirm) and system prompts (EMAC, ESAC, IMAC, ISAC, VD, AD); each rule updates the pragmatic scores of candidates. The firing of each pragmatic analysis rule is interpreted as evidence for or against the associated candidates as specified in the table. Other rules may readily be derived in addition to those in Table V.

Most of the rules in Table V are self-explanatory. For example, rule 1 increases the scores of implicitly confirmed attribute-values if the user provided a requested response (RR). Rules 5, 6, 7, 9 increase the score(s) of the AV pair(s) being confirmed if the user replies affirmatively (with a "yes" or synonyms). Rules 2, 8, 10, decrease the score of the AV pair being confirmed if the user replies with a "no" (note that there is no such rule for multiple AV pair confirmation here, since a "no" response could correspond to any of the AV pairs being confirmed). Finally, the value disambiguation rules 3 and 11, cover the requested response case (where one candidate is selected by the user) and the rejection of all candidates case respectively.

A pragmatic analysis example can be found in the first subdialogue of Table VI. Specifically, the value disambiguation rule 3 is applied at the third dialogue turn of the subdialogue with system prompt "Is the departure city Boston or Austin?" and user reply "Yes, Austin Texas". According to pragmatic analysis rule 3, and (additional) positive evidence of strength 0.3 is incorporated into the candidate "AUS" using the MYCIN formula in (1), resulting in a pragmatic score of 0.8 for "AUS", while the score of the candidate "BOS" (that was not selected by the user) goes to 0. Note that in the absence of the pragmatic analysis rules the pragmatic scores for the two candidates would have been 0.38 and 0.72 for "BOS" and "AUS" respectively, just as is the case for third subdialogue of Table VI.

C. Training of Scoring Parameters

In this section, we discuss the training of the evidence computation parameters of the scoring algorithm, i.e., the parameters α and β in (2) and the evidence contribution values from the pragmatic analysis rules 1–11 in Table V. For simplicity we have encoded all the evidence rules in Table V into a single parameter γ, where γ is the evidence for rules {1, 3–7, 9}, $-\gamma$ is the evidence for rules {8, 10} and $-2\ \gamma$ is the evidence for rule 2. Note that in Table V we have set $\gamma = 0.3$.

The scoring algorithm parameters $\theta = \{\alpha, \beta, \gamma\}$ can be optimized using the following supervised training algorithm. Dialogues are manually labeled; at each dialogue turn each candidate value is assigned a (pragmatic) confidence score by the labeler. The scores are normalized between -1 and 1 (as in (1)), and quantized (for example into five levels (-1, -0.5,0,0.5,1)). The optimal value of the parameters $\hat{\theta}$ are then computed so as to minimize the mean absolute distance between the automatically computed pragmatic scores and the (quantized) manually

TABLE V
PRAGMATIC EVIDENCE RULES USED IN THE TRAVEL RESERVATION SYSTEM. USER INPUT IS CLASSIFIED IN THE FOLLOWING CATEGORIES: REQUESTED RESPONSE (RR), EXPECTED ERROR RESPONSE (ER), UNRELATED RESPONSE (UR) AND "YES"/"NO" CONFIRMATION. THE SYSTEM PROMPT CATEGORIES ARE: EXPLICIT/IMPLICIT MULTIPLE/SINGLE ATTRIBUTE CONFIRMATION (EMAC/ESAC/IMAC/ISAC) AND VALUE/ATTRIBUTE DISAMBIGUATION (VD/AD). THE PRAGMATIC SCORING DECISION FOR EACH RULE IS SHOWN QUALITATIVELY AND QUANTITATIVELY AS POSITIVE OR NEGATIVE EVIDENCE e FOR CANDIDATE VALUES. NOTE THAT THE "REMOVE CANDIDATE" OPERATION SETS THE PRAGMATIC SCORE OF THAT CANDIDATE TO 0

Rule No	RR	ER	UR	Confirm	System Prompt Elements	System scoring action	Evid. e
1	Y	-	-	-	(no query) IMAC or ISAC	up all implicit candidate scores	0.3
2	Y	-	-	"no"	ESAC	drastically lower explicit candidate score	-0.6
3	Y	-	-	-	VD	set selected candidate score to confirmed remove the rest of the candidate ($s = 0$)	0.3 –
4	-	-	Y	"yes"	any	up all implicit candidate scores	0.3
5	-	Y	-	"yes"	EMAC or ESAC	up all explicit candidate scores	0.3
6	Y	Y	Y	"yes"	EMAC or ESAC	up all explicit candidate scores	0.3
7	-	-	-	"yes"	(no query) IMAC or ISAC	up all implicit candidate scores	0.3
8	-	-	-	"no"	(no query) ISAC	lower implicit candidate score	-0.3
9	-	-	-	"yes"	EMAC or ESAC	up all explicit candidate scores	0.3
10	-	-	-	"no"	ESAC	lower explicit candidate score	-0.3
11	-	-	-	"no"	VD	remove both candidates ($s = 0$)	–

transcribed pragmatic scores over all dialogue turns and all candidates, i.e.,

$$\hat{\theta} = \arg\min_{\theta} \sum_{d} \sum_{t} \sum_{AV} |s^a(AV,t,d) - s^m(AV,t,d)| \quad (3)$$

where d is the dialogue index, t is the dialogue turn index, *AV* is the candidate index (attribute-value pair), s^a is the score computed automatically from (1) and s^m is the manually transcribed score. The minimization can be implemented using gradient descent.

The supervised training procedure presented above requires a significant number of dialogues to be manually transcribed. Alternatively, an unsupervised training algorithm can be used that requires no manual labor. Users are asked to complete the same task for different values of the parameters θ. The optimal parameters are selected so as to minimize an objective criterion, e.g., time-to-completion (average number of dialogue turns it takes to complete the task). Note that the unsupervised training algorithm requires far more training data (typically a few hundred dialogues) than the supervised algorithm. In practice, system developers can use more than one objective criterion to select the optimal parameters θ and focus their attention mainly on the error correction subdialogues to speed up the training process. For our travel reservation systems we found that the values $\hat{\alpha} = 1$, $\hat{\beta} = -0.2$ and $\hat{\gamma} = 0.3$ were close to optimal. However, more research is needed to investigate how well these parameter values generalize across different applications and to what extent these parameters are attribute-dependent.

Overall, the proposed pragmatic scoring algorithm has the advantages outlined in Section V-A, namely: the use of all available sources of information to determine the validity of an attribute-value pair, a parsimonious representation with few trainable parameters and a normalized score that is easy to use in the dialogue manager to disambiguate among AV pairs.

D. Bayesian Formulation of Pragmatic Scores

Instead of using the MYCIN update formula in (1) one may use a probabilistic framework to compute pragmatic scores. In this case, pragmatic scores represent the probability that an attribute has a specific value given all evidence available (for or against) that value. In [18], a Bayesian probabilistic framework is proposed for combining evidence from various sources to compute the probability of an AV. However, the proposed framework is hard to use in practice since all possible values have to be taken into account, i.e., even if there is no evidence for a value the probability of this value has to be computed. An interesting alternative approach to statistical modeling and semantic/pragmatic evidence combination can be found in recent work [4], but this approach requires sufficient training data; a more practical approach on how to combine evidence from user input for or against an AV pair is presented below.

Each candidate with value a and confidence 0.5 is represented as a Bayesian network node with a probability table $P(a) = 0.5$ and $P(\bar{a}) = 0.5$ (the latter is the probability of any value but a). We refer to these binary valued nodes as "evidence" nodes; there are typically multiple evidence nodes per attribute. Each attribute is modeled with an "attribute" node *AV* that holds all possible values for that attribute; all evidence nodes are connected (point to) to their corresponding attributes. The possible values for an attribute node are the values of the evidence nodes connected to it and the negation of these values. For example, if two evidence nodes E_1 and E_2 with values a and b exist and are connected to an attribute node *AV* the possible values for this attribute node are a, b and $\bar{a}\bar{b}$. Continuing with this example the probability table for node *AV* is $P(AV = a|E_1 = a, E_2 = b)$, $P(AV = a|E_1 = \bar{a}, E_2 = b)$, $P(AV = a|E_1 = a, E_2 = \bar{b})$, $P(AV = a|E_1 = \bar{a}, E_2 = \bar{b})P(AV = b|E_1 = a, E_2 = b), \ldots, P(AV = \bar{a}\bar{b}|E_1 = a, E_2 = b), \ldots$ a 3×4 table. As the number of evidence nodes becomes larger the size of the probability tables becomes unmanageable and the training requires a large amount of data. In practice, one can make reasonable assumptions for the attribute node probability table $P(AV|E_1E_2)$, e.g.,

	E_1E_2			
AV	ab	$\bar{a}b$	$a\bar{b}$	$\bar{a}\bar{b}$
a	0.5	0	1	0
b	0.5	1	0	0
$\bar{a}\bar{b}$	0	0	0	1

Assuming that for nodes E_1 and E_2, $P(E_1 = a) = 0.5$ and $P(E_2 = b) = 0.5$, it is easy to compute $P(AV = a) = 0.375$ using the table above. Similarly for evidence nodes $P(E_1 = a) = 0.5$ and $P(E_2 = a) = 0.5$, we obtain $P(AV = a) = 0.5$,

TABLE VI
EXAMPLES OF IMPLICIT AND EXPLICIT CORRECTION SUB-DIALOGUES INITIATED BY THE USER. IN ALL CASES THE USER IS TRYING TO CHANGE THE DEPARTURE CITY FROM BOSTON TO AUSTIN. THE ACOUSTIC/SEMANTIC CONFIDENCE SCORES ARE ASSUMED TO BE $p = 0.5$ FOR ALL AV PAIRS. IN THE FIRST CASE, THE IMPLICIT CORRECTION LEADS TO VALUE AMBIGUITY WHICH IS THEN DISAMBIGUATED BY THE DIALOGUE MANAGER. IN THE SECOND CASE, THE USER PERFORMS AN IMPLICIT CHANGE REQUEST. IN THE THIRD CASE, THE USER PROVIDES EVIDENCE FOR AUSTIN; THE EVIDENCE IS WEIGHTED AGAINST THE EVIDENCE FOR BOSTON AND THE SYSTEM DECIDES THAT AUSTIN (WITH SCORE 0.72 VERSUS 0.38) PREVAILS (UNAMBIGUOUSLY)

Sub-Dialogue 1	Attribute	Value	Score
S: *How can I help you?*	trip.flight.leg1.departure.city	BOS	0.5
U: I want to fly from Boston to New York.	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I've got you leaving Boston and arriving to New York.*	.departure.city	BOS	0.44
S: *Leaving Boston on what date?*	.departure.city	AUS	0.44
U: Leaving from Austin on July 6th	.departure.date	Jul 6, 2004	0.5
	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I understand you are leaving on Tuesday, July 6th.*	.departure.city	BOS	0
S: *I heard you say that the departure city was Austin.*	.departure.city	AUS	**0.8**
S: *Was the departure city Boston or Austin?*	.departure.date	Jul 6, 2004	0.5
U: Yes, Austin Texas.	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I understand you are leaving Austin.*			
S: *Leaving Austin at what time?*		...	
U: ...			

Sub-Dialogue 2	Attribute	Value	Score
S: *How can I help you?*	trip.flight.leg1.departure.city	BOS	0.5
U: I want to fly from Boston to New York.	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I've got you leaving Boston and arriving to New York.*	.departure.city	BOS	0
S: *Leaving Boston on what date?*	.departure.city	AUS	**0.5**
U: Not Boston, Austin.	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I am changing the departure city to Austin.*			
S: *Leaving Austin on what date?*		...	
U: ...			

Sub-Dialogue 3	Attribute	Value	Score
S: *How can I help you?*	trip.flight.leg1.departure.city	BOS	0.5
U: I want to fly from Boston to New York.	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *I've got you leaving Boston and arriving to New York.*	.departure.city	BOS	0.38
S: *Leaving Boston on what date?*	.departure.city	AUS	**0.72**
U: From Austin, Austin Texas you stupid!	.arrival.city	NYC/JFK_LGA_EWR	0.5
S: *You are leaving from Austin*			
S: *Leaving Austin on what date?*		...	
U: ...			

given the assumption that $P(AV = a|E_1 = \bar{a}, E_2 = a) = 0.5$. Compare the probabilities 0.375 and 0.5 with the scores 0.44 and 0.72 respectively that the MYCIN formulas produce in the same situation; clearly the proposed Bayesian formulation produces lower confidence scores compared to MYCIN. More research is needed to better understand how the Bayesian formulation can be used for pragmatic confidence scoring and how to parameterize and train the attribute node probabilities table efficiently.

E. Combining Inputs From Multiple Modalities

The proposed pragmatic analysis and scoring mechanisms can be also used for combining input from different modalities. Consider for example a multimodal system that can handle speech and graphical input sequentially.[15] Each piece of evidence (raw data, pragmatic analysis) is independently evaluated for each input mode. A default evidence score p is assigned to each input modality depending on its reliability, e.g., $p = 0.5$ for speech input and $p = 0.9$ for graphical input. The score for each piece of evidence is then computed using (2) and the pragmatic analysis rules. Finally evidence from various input modalities is combined using (1). Note that the evidence combination algorithm operates independent of modality; the mode of input only affects the evidence score e.

For example, consider two pieces of input for the attribute "…departure.city": speech input "leaving from Boston" and graphical input (via the departure city field in the graphical user interface) "NYC". Based on the default confidence scores for the speech and graphical input modalities the two pieces of evidence "BOS" and "NYC" have scores $e_1 = 0.5$ and $e_2 = 0.9$ respectively. Assuming an initial value of 0 for all candidates, the evidence is computed and combined using the formulas in Section V-A. as follows:[16]

Candidate	Evidence	c	γ/n	p	e	s
BOS						0
	BOS	1	1	0.5	0.5	0.5
	NYC	0	1	0.9	-0.18	0.39
NYC						0
	NYC	1	1	0.9	0.9	0.9
	BOS	0	1	0.5	-0.1	**0.89**

The final scores for "BOS" and "NYC" are $s_{BOS} = 0.39$ and $s_{NYC} = 0.89$ respectively. Given the ambiguity thresholds provided in Section V-A1 the attribute "…departure.city" is unambiguously established as "NYC" because $s_{NYC} = 0.89 > \sigma_1$ and $s_{NYC} - s_{BOS} = 0.5 > \sigma_2$.

Overall, the proposed pragmatic module can be used as an simple and efficient way of combining inputs from multiple

[15]For a discussion on how the semantic and pragmatic modules can be extended to handle concurrent multimodal input, see Section VII and [15].

[16]Note that both the positive and negative evidence is affected by the input mode reliability score p.

modalities: in essence, merging input from multiple modalities is formulated and solved as an evidence combination problem. In [22], a multimodal spoken dialogue system that can handle both speech and graphical input is implemented using the pragmatic module presented here.

VI. Evaluation

In this section, evaluation results for the proposed semantic and pragmatic modules are presented. The proposed semantic and pragmatic algorithms were implemented as part of the 2001 Bell Labs DARPA Communicator travel reservation system. We present the 2000 [33] and 2001 [34] DARPA Communicator evaluation campaign results for the Lucent Bell Labs system (evaluation organized by NIST/DARPA). The main differences between the 2000 and 2001 Bell Labs systems are outlined as follows.

1) The 2001 system employed the semantic and pragmatic representation algorithms described in this paper: pragmatic scoring, pragmatic analysis, complex error correction mechanisms, complex context tracking, improved parsing. The 2000 system used simple parsing and context tracking rules and no pragmatic scoring.
2) The automatic speech recognizers (ASR) employed by both the 2000 and 2001 systems were identical[17]; however, the acoustic and language models used in the 2001 system were trained on a bigger corpus and in-domain (travel reservation) data was used for model adaptation.
3) The Bell Labs text-to-speech synthesizer (TTS) system [37] was used in the 2000 system, while for the 2001 system concatenated pre-recorded prompts were used.
4) The functionality (and evaluation scenarios) of the 2001 system were augmented compared to the 2000 system: more travel destination were added to the system, as well as car and hotel reservation capabilities.

The 2000 system evaluation was performed over a period of three weeks; during this period 75 calls were made to the system by paid subjects. Callers were also asked to judge whether the task was successfully completed and to answer a set of five user satisfaction survey questions on the system usability based on the NIST-derived Likert paradigm [16], [29]. The 2001 evaluation was performed over a period of six months; during this period 215 calls were made by 28 paid subjects. Among the 215 dialogues collected, we present results for 139 dialogues for which user survey data exist. In Table VII, the objective and subjective evaluation metrics are presented for the 2000 and 2001 systems. Perceived task completion (PTC), task duration statistics and word/sentence accuracy are shown for the two systems for completed and non-completed tasks. The task completion for the 2001 system was 83%, significantly higher than the 2000 system (53%). Task duration statistics are significantly lower for the 2000 system due to the simpler evaluation scenarios (no car/hotel reservations in 2000). Finally, the relative word error rate reduction in the 2001 system (compared with 2000) is 40%.

In the second part of Table VII, the average Likert scores in the user survey are shown as a function of perceived task completion for the two systems. The user survey consisted of the following five questions which were rated from 1 to 5 (5 being strong agreement):

1) "In this conversation, it was easy to get the information that I wanted" (Easy to Get Info).
2) "I found the system easy to understand in this conversation" (Easy to Understand).
3) "In this conversation, I knew what I could say or do at each point of the dialogue" (Know What to Say/Do).
4) "The system worked the way I expected it to in this conversation" (Know What to Expect).
5) "Based on my experience in this conversation using this system to get travel information, I would like to use this system again" (Future Use).

TABLE VII
Objective and Subjective Dialogue Metrics for the 2000 and 2001 Bell Labs Communicator Systems

	2000 System		2001 System	
Objective Metric	Comp.	No Comp.	Comp.	No Comp.
(Perceived) Task Compl.	53 %		83%	
Dialogue Duration (secs)	363	392	547	429
# of User Turns	22.9	22.1	33.6	31.2
# of User Words	50.5	66.3	85.6	94.3
Word Error Rate (%)	39.9	44.3	23.5	24.9
Sentence Error Rate (%)	38.2	46.5	28.8	34.3

	2000 System		2001 System	
Subjective Metric	Comp.	No Comp.	Comp.	No Comp.
Easy to Get Info	3.03	1.88	3.61	1.62
Easy to Understand	3.49	3.06	4.02	3.10
Know What to Say/Do	3.15	2.62	3.56	2.97
Know What to Expect	3.05	1.68	3.55	1.98
Future Use	2.36	1.66	3.31	1.69
Overall	3.02	2.16	3.61	2.23

The overall subjective score improvement for the 2001 vs 2000 system is 0.6 in the Likert scale for completed tasks. This is a statistically significant result; the 95% confidence interval for the overall score difference is $[-0.24, 0.24]$. Note, that the difference in subjective scores between the two systems for non-completed tasks is not significant. See [32] for a formal evaluation across all Communicator systems using the PARADISE framework [30].

Overall, the 2001 system compares favorably to the 2000 system both in terms of objective (task completion) and subjective metrics (user satisfaction). Due to the numerous differences between the two systems and the different evaluation scenarios it is hard to judge the impact of the improved semantic and pragmatic modules in the overall system performance. The main difference between the 2000 and 2001 systems (apart from the semantic and pragmatic modules) is the improved speech recognition word accuracy in the 2001 system.

The PARADISE evaluation model [30] gives the ability to estimate contributions of each system component to overall user satisfaction, but unfortunately we are unable to run this style of evaluation for our system. However, we do note that Walker *et al.*'s PARADISE analysis of all Communicator systems (including ours) indicates that the critical predictive factors of user satisfaction (in decreasing order) were Task Duration, System Words Per Turn, Task Completion, Sentence Error Rate, Number of Overlaps, and User Words Per Turn [33], with the first three items demonstrating 2 to 2.5 times as much influence as the sentence error rate. This suggests that while ASR

[17]Note that the ASR engine did not produce word level confidence scores; default acoustic confidence values of 0.5 was assumed for all attribute values.

TABLE VIII
A DIALOGUE FRAGMENT FROM THE JUNE 2000 DARPA COMMUNICATOR EVALUATION. S = system utterance, U = recognized user utterance. AT EACH DIALOGUE TURN THE INNER WORKINGS OF THE PRAGMATIC MODULE ARE SHOWN: PROMPT AND USER INPUT CATEGORIZATION, APPLICABLE PRAGMATIC ANALYSIS RULES AND THE AV PAIRS WITH THEIR CORRESPONDING PRAGMATIC SCORES s (THE ACOUSTIC CONFIDENCE SCORE IS $p = 0.5$ FOR ALL CANDIDATES). CONTEXT TRACKING AND DIALOGUE MANAGEMENT DECISIONS ARE ALSO SHOWN WHERE APPROPRIATE (THE COMMON CONTEXT PREFIX "TRIP.FLIGHT" IS OMITTED)

Dialogue Fragment	System inferences
S: How can I help you?	
U: I am leaving Baltimore	AV pair: ('leg1.departure.city', 'Baltimore', $s = 0.5$)
S: Leaving Baltimore. What is the arrival city?	System prompt type: ISAC ('Baltimore')
U: fine Portland *(ASR error)*	User input type: RR, confirm="yes" AV pair: ('leg1.arrival.city', 'Portland', $s = 0.5$) Pragmatic Analysis Rule 7 applies AV pair: ('leg1.departure.city', 'Baltimore', $s = 0.65$)
S: Arriving in Portland-Oregon. On what date do you want to depart Baltimore?	System prompt type: IMAC ('Portland', 'Baltimore')
U: about no not Portland-Oregon Frankfurt-Germany	User input type: ER, confirm = "no" Change request: change city=Portland to city=Frankfurt AV pair: ('leg1.arrival.city', 'Portland', $s = 0.0$) AV pair: ('leg1.arrival.city', 'Frankfurt', $s = 0.5$)
S: Arriving in Frankfurt. On what date do you want to depart Baltimore?	System prompt type: IMAC ('Frankfurt', 'Baltimore')
U: like Fort-Wayne *(ASR error)*	Context tracker: context change from departure.date → departure.city AV pair: ('leg1.departure.city', 'Baltimore', $s = 0.61$) AV pair: ('leg1.departure.city', 'Fort-Wayne', $s = 0.44$) Dialogue Manager: value ambiguity for 'leg1.departure.city'
S: Was the departure city Baltimore or Fort Wayne?	System prompt type: VD
U: ...	

improvements could theoretically account for the improvement in user satisfaction, it is more likely that the improved semantic and pragmatic modules, which would have improved the first three metrics, would likely have been a significant contributor in the improvements seen in our 2001 system. However, further research is needed to understand the exact relationship between utilizing the improved semantic/pragmatic modules and user satisfaction.

A different way of estimating the impact of our proposed semantic and pragmatic algorithms on system performance is to remove the effect of ASR performance by taking the output of the 2000 ASR system as a given for a subset the 2000 evaluation corpus, and then analyze the performance of the 2000 and 2001 Semantic/Pragmatic modules on these constant ASR strings. This does not have the advantage of being tied to user satisfaction ratings, but provides us with an alternative method of evaluating the correctness of our algorithm. In the next section, we perform a detailed turn-by-turn analysis of the 2000 Communicator corpus.

A. Turn by Turn Dialogue System Evaluation

We analyzed 35 dialogues collected during the third week of the June 2000 DARPA Communicator evaluation by the Bell Labs system. Every dialogue was run through both the 2000 and 2001 systems turn by turn until the output prompts of the two systems were different, i.e., the dialogues "diverged."[18] At turns where the two systems diverged a labeler manually characterized the system behavior as "better", "worse" or "no difference". The labeler made such judgments using full knowledge of the system prompts, transcribed user inputs, speech recognizer outputs, as well as information about the internal system state. The labeler did not listen to the synthesized prompts, thus eliminating any TTS performance bias.

The analyzed dialogues contained multiple transactions, frequently covering more than one leg of a flight. For some of the dialogues evaluated, the two systems diverged (e.g., in the first leg) and then latter re-converged (e.g., in the second leg). As a result multiple dialogue fragments per transaction were analyzed and evaluated: a total of 49 fragments. Given these 49 examples where the two systems diverged, we found that the 2001 system improved in 25 cases, compared to 3 cases where the 2000 system was superior. The improvements were due to better parsing/context tracking in 10 cases, the introduction of scoring and pragmatic analysis in 10 cases and the interaction of the semantic and pragmatic modules in 3 cases. In 21 cases, the dialogue diverged in ways that did not allow such a value judgment to be made ("no difference"). The results are summarized in Table IX. A sample interaction where the 2001 system performed "better" is shown in Table VIII. Note the application of pragmatic analysis rule 7 in the third dialogue turn and the disambiguation subdialogue that the system enters at the fifth dialogue turn due to the value ambiguity for the departure city. Overall, the introduction of pragmatic analysis and scoring improved the quality of the 2001 system in about 20% of the dialogue fragments examined, a significant improvement.

B. Multimodal Dialogue System Evaluation

To evaluate the ability of the pragmatic module to operate with visual or multiple input modalities a multimodal version of the travel reservation system was created in [22]. For the multimodal version of the system a GUI parser and interpreter had to be written, e.g., to be able to parse "3/9" as "March 9th". The semantic representations used for the unimodal (speech-only, GUI-only) and multimodal systems were identical. No porting

[18]Note that the (text) input to both systems was identical, thus eliminating any ASR performance bias.

TABLE IX
COMPARISON OF THE 2000 AND THE 2001 VERSION OF THE SPOKEN DIALOGUE SYSTEM

Evaluator judgment	No of Dlg. Fragm.	Percent Dlg. Fragm.
2001 system was better	25	51%
due to better parsing	10	20.5%
due to pragm. scoring	10	20.5%
due to better parsing and pragm. scoring	3	6%
due to other reasons	2	4%
2000 system was better	3	6%
No (appreciable) difference	21	43%
Total Dialogue Fragments	49	100%

TABLE X
OBJECTIVE AND SUBJECTIVE METRICS FOR UNIMODAL AND MULTIMODAL SYSTEMS

Objective Metric	speech-only	GUI-only	multimodal I (click-to-talk)	multimodal II (open-mike)
Task Success (%)	62	100	98	96
Speech Turns (%)	100	0	56	65
Avg. session duration (sec)	193.4	86.2	112.5	115.0
Subjective Metric	**speech-only**	**GUI-only**	**multimodal I**	**multimodal II**
Easy to Get Info	3.57	4.54	3.85	4.06
Easy to Understand	3.50	4.48	3.74	4.06
Know What to Say/Do	3.61	4.62	3.93	3.85
Know What to Expect	3.57	4.54	3.78	3.66
Future Use	3.54	4.24	3.74	3.72
Overall	3.56	4.48	3.81	3.87

and *no modification to the pragmatic module were necessary* for the GUI and multimodal systems. Speech and GUI input is parsed and interpreted and candidates are added to the raw data tree.[19] The context tracker, pragmatic analysis and pragmatic scoring algorithms operate uniformly on the raw data (independent of the mode of input). Input from different modalities is seamlessly merged using the candidate tree and pragmatic scoring machinery as discussed in Section V-E.

The unimodal (speech-only, GUI-only) and two versions of the multimodal travel reservation systems were evaluated by ten non-native English-speaking users. A total of 50 interactions (dialogues) were collected for each system. Selected results from the subjective and objective evaluation are shown in Table X. Note that in terms of task completion, time to completion, and user satisfaction the GUI and multimodal systems significantly outperform the speech-only system. For more details on the development of the multimodal (and GUI) systems, as well as the evaluation procedure see [22].

VII. DISCUSSION

In this section, we discuss our experience from using the semantic and pragmatic algorithms proposed here for a different application domain, namely movie information. Also generalizations of the proposed algorithms to handle simultaneous multimodal input (concurrent multimodality) and input from different modalities (e.g., gestures) are discussed.

The list of implementation steps for porting the system [22] to a new application domain, e.g., movie information system, were the following: 1) design the prototype tree, 2) compose the parser rules (context sensitive grammar), 3) map parser concepts to prototype tree concepts, 4) write/update the interpreters, i.e., define mapping from spoken/written form to internal value representation for each attribute in prototype tree, 5) update application-dependent user requests in "User Action Interpreter", 6) define application-dependent context ambiguity resolution rules, and 7) implement "Domain Knowledge and Inference" rules. Note that steps 1)–5) refer to semantic module and steps 6)–7) refer to the pragmatic module. Steps 2) and 4) were the most time consuming. Steps 5) and 7) were not needed for the simple movie information system implemented but are listed here for completeness.

Most of the difficulties during porting were encountered at the dialogue manager level; no modifications (other than the ones listed above) were necessary to the semantic and pragmatic modules. The context tracking and pragmatic scoring algorithms were used *as is* in the new application domain. For a full description of the movie information system porting experience see [22].

To incorporate new modalities into the system the following steps have to be followed: 1) construct the "parser" for the new modality, 2) construct the "interpreters" for the new modality, 3) define the modality-dependent confidence score for evidence combination, and 4) add modality-dependent pragmatic rules (optional). Note that steps 1), 2) affect the semantic module and steps 3), 4) affect the pragmatic module. The notions of "parser" and "interpreter" are generalized here to mean mapping from user input to attributes and values. Examples of other modalities that could be integrated into the system include gestures and eye-gaze information.

Our discussion has focused on sequential multimodality — when the user is required to utilize only one modality per turn. Concurrent multimodality, where speech and GUI inputs work in concert, will require extensions such as three-tape FSM parsers [15]. In [22] we discuss how such technologies may be integrated with our proposed architecture.

[19]Note that raw data have different a-priori confidences depending on the input modality, $p = 0.5$ for speech and $p = 0.9$ for GUI input.

VIII. Conclusions

In this paper, a new application-independent framework for semantic representation, context tracking and pragmatic analysis in information seeking spoken dialogue systems was developed. A pragmatic scoring mechanism was introduced that combines information from acoustic, semantic and pragmatic sources. The pragmatic analysis and pragmatic scoring algorithms were shown to improve on a previous simpler spoken dialogue system, thanks to the ability of the pragmatic module to directly represent and identify ambiguities introduced either by system errors or by user input. The semantic representation, context tracking, pragmatic analysis and pragmatic scoring algorithms were shown (see also [22]) to be application-independent by porting the system to a new application domain (movie information) and to be modality independent by extending the system to multiple modalities (combination of speech and graphical input). Evaluation results provided strong evidence that the proposed semantic and pragmatic algorithms can improve system performance; further research is required to better understand the impact of this work on the user experience.

In future work, we plan to investigate how to further tune the scoring algorithms through training of the parameters and how to better integrate confidence measures from the semantic module into the pragmatic scoring algorithm. Overall, this work is a first step towards creating state-of-the art algorithms and modules for spoken dialogue systems that are general enough to be applicable to a wide range of application domains and interaction modalities.

Acknowledgment

The data collection was designed and organized by NIST. The user survey and scenarios were created by the DARPA Communicator evaluation committee. The authors would like to express their sincere appreciation to Dr. S. Lee, Dr. J. Kuo, Dr. A. Pargellis, Prof. C.-H. Lee, and Prof. J. Olive for their many contributions to the Bell Labs DARPA Communicator program; to A. Saad and Dr. Q. Zhou for building the Communicator-compliant Bell Labs audio platform; to T. Ren, R. Barkan, and M. Einbinder for their help with system implementation and data collection; to Dr. J. Chu-Carroll, Dr. M. Tsangaris, and Prof. S. Narayanan for many helpful discussions; and to the Colorado University Communicator team (and especially Dr. B. Pellom) for providing and supporting the travel information database back-end.

References

[1] M. K. Abrahams, D. L. McGuinness, R. Thomason, L. A. Resnick, P. F. Patel-Schneider, V. Cavalli-Sforza, and C. Conati, *NeoClassic Tutorial: Version 1.0*. Whippany, NJ: Artificial Intelligence Principles Research Department, AT&T Bell Labs, 1996.

[2] E. Ammicht, A. Potamianos, and E. Fosler-Lussier, "Ambiguity representation and resolution in spoken dialogue systems," in *Proc. Eur. Conf. Speech Communication and Technology*, Aalborg, Denmark, Sep. 2001.

[3] D. Bohus and A. Rudnicky, "Sorry, I didn't catch that! – An investigation of non-understanding errors and recovery strategies," in *Proc. SIGdial Workshop on Discourse and Dialogue*, Lisbon, Portugal, Sep. 2005.

[4] ——, "Constructing accurate beliefs in spoken dialog systems," in *Proc. Workshop on Automatic Speech Recognition and Understanding*, Cancun, Mexico, Nov. 2005.

[5] B. Carpenter, *Type-Logical Semantics*. Cambridge, MA: MIT Press, 1998.

[6] J. Chu-Carroll, "Formbased reasoning for mixed-initiative dialogue management in information-query systems," in *Proc. Eur. Conf. on Speech Communication and Technology*, Budapest, Hungary, Sep. 1999.

[7] ——, "MIMIC: An adaptive mixed initiative spoken dialogue system for information queries," in *Proc. 6th ACL Conf. Applied Natural Language Processing*, Seattle, WA, May 2000.

[8] M. Denecke, "Rapid prototyping for spoken dialogue systems," in *Proc. Internat. Conf. on Computational Linguistics*, Taipei, Taiwan, R.O.C., Aug. 2002.

[9] M. Denecke and A. Waibel, "Dialogue strategies guiding users to their communicative goals," in *Proc. Eur. Conf. Speech Communication and Technology*, Rhodes, Greece, Sep. 1999.

[10] D. Goddeau, H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai, "A form-based dialogue manager for spoken language applications," in *Proc. Int. Conf. Speech Language Processing*, Philadelphia, PA, Oct. 1996.

[11] D. Heckerman, "Probabilistic interpretations for MYCIN's certainty factors," in *Uncertainty in Artificial Intelligence*, L. Kanal and J. Lemmer, Eds. Amsterdam, The Netherlands: North Holland, 1986, pp. 11–22.

[12] R. Higashinaka, M. Nakano, and K. Aikawa, "Corpus-based discourse understanding in spoken dialogue systems," in *Proc. Annu. Meeting Assoc. Comput. Linguist.*, 2003.

[13] J. Hobbs and R. Moore, *Formal Theories of the Commonsense World*. Norwood, NJ: Ablex, 1985.

[14] K. Komatani and T. Kawahara, "Generating effective confirmation and guidance using two-level confidence measures for dialogue systems," in *Proc. Int. Conf. Speech Language Processing*, Beijing, China, Oct. 2000.

[15] M. Johnston and S. Bangalore, "Finite-state multimodal integration and understanding," *J. Natural Lang. Eng.*, vol. 11, no. 2, pp. 159–187, 2005.

[16] L. B. Larsen, "Combining objective and subjective data in evaluation of spoken dialogues," in *Proc. ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, Kloster Irsee, Germany, Jun. 1999.

[17] S. Larsson and D. Traum, "Information state and dialogue management in the TRINDI dialogue move engine toolkit," *Natural Lang. Eng.*, vol. 6, pp. 323–340, 2000.

[18] O. Lemon, P. Parikh, and S. Peters, "Probabilistic dialogue modeling," in *Proc. 3rd SIGdial Workshop on Discourse and Dialogue*, Philadephia, PA, Jul. 2002.

[19] E. Levin, R. Pieraccini, W. Eckert, G. D. Fabbrizio, and S. Narayanan, "Spoken language dialogue: From theory to practice," in *Proc. Workshop on Automatic Speech Recognition and Understanding*, Keystone, CO, Dec. 1999.

[20] A. Potamianos, H.-K. Kuo, C.-H. Lee, A. Pargellis, A. Saad, and Q. Zhou, "Design principles and tools for multimodal dialog systems," in *Proc. ESCA Workshop Interact. Dialog. Multi-Modal Syst.*, Kloster Irsee, Germany, Jun. 1999.

[21] A. Potamianos and H.-K. Kuo, "Speech understanding using finite state transducers," in *Proc. Int. Conf. Speech Language Processing*, Beijing, China, Oct. 2000.

[22] A. Potamianos, E. Fosler-Lussier, and E. Ammicht, "Information seeking spoken dialogue systems-Part II: Multimodal dialogue," *IEEE Trans. Multimedia*, vol. 9, no. 3, Apr. 2007, to be published.

[23] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh, "Creating natural dialogs in the carnegie mellon communicator system," in *Proc. Eur. Conf. Speech Communication and Technology*, Budapest, Hungary, Sep. 1999.

[24] A. Rudnicky and W. Xu, "An agenda-based dialog management architecture for spoken language systems," in *Proc. Workshop on Automatic Speech Recognition and Understanding*, Keystone, CO, Dec. 1999.

[25] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice-Hall, 1995.

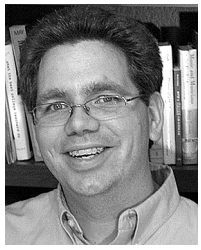
[26] R. SanSegundo, B. Pellom, K. Hacioglu, W. Ward, and J. Pardo, "Confidence measures for dialogue systems," in *Proc. Int. Conf. Speech Language Processing*, Salt Lake City, UT, May 2001.

[27] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "Galaxy-II: A reference architecture for conversational system development," in *Proc. Int. Conf. Speech Language Processing*, Sydney, Australia, Dec. 1998.

[28] E. Shortliffe, *Computer-Based Medical Consultation: MYCIN*. New York: Elsevier, 1976.
[29] M. Walker, L. Hirschman, and J. Aberdeen, "Evaluation for DARPA communicator dialog systems," in *Proc. Int. Conf. Language Resources and Evaluation*, Athens, Greece, Jun. 2000.
[30] M. Walker, C. Kamm, and D. Litman, "Towards developing general models of usability with PARADISE," *Nat. Lang. Eng.: Special Issue on Best Practice in Spoken Dialogue Systems*, 2000.
[31] M. Walker and R. Passonneau, "DATE: A dialogue act tagging scheme for evaluation of spoken dialogue systems," in *Proc. Human Language Technology Conf.*, San Diego, CA, Mar. 2001.
[32] M. A. Walker, R. J. Passonneau, and J. E. Boland, "Quantitative and qualitative evaluation of DARPA communicator spoken dialogue systems," in *Proc. Annu. Meeting of the Association for Computational Linguistics*, 2001.
[33] M. A. Walker, A. I. Rudnicky, R. Prasad, J. Aberdeen, E. O. Bratt, J. S. Garofolo, H. Hastie, A. N. Le, B. Pellom, A. Potamianos, R. Passonneau, S. Roukos, G. A. Sanders, and S. S. Stallard, "DARPA communicator evaluation: Progress from 2000 to 2001," in *Proc. Int. Conf. Speech Language Processing*, Sep. 2002.
[34] M. A. Walker, A. I. Rudnicky, R. Prasad, J. Aberdeen, E. O. Bratt, J. S. Garofolo, H. Hastie, A. N. Le, B. Pellom, A. Potamianos, R. Passonneau, S. Roukos, G. A. Sanders, S. Seneff, and D. Stallard, "DARPA communicator: Cross-system results for the 2001 evaluation," in *Proc. Int. Conf. Speech Language Processing*, Keystone, CO, Sep. 2002.
[35] W. Ward and B. Pellom, "The CU communicator system," in *Proc. Workshop on Automatic Speech Recognition and Understanding*, Keystone, CO, Dec. 1999.
[36] W. Xu, B. Xu, T. Huang, and H. Xia, "Bridging the gap between dialogue management and dialogue models," in *Proc. 3rd SIGdial Workshop on Discourse and Dialogue*, Philadephia, PA, Jul. 2002.
[37] Q. Zhou, A. Saad, and S. Abdou, "An enhanced BLSTIP dialogue research platform," in *Proc. Int. Conf. Speech Language Processing*, Beijing, China, Oct. 2000.

Egbert Ammicht received the B.S. degrees in mathematics and physics from Bogazisi University, Turkey, in 1974 and the Ph.D. degree in applied mathematics from Northwestern University, Evanston, IL, in 1978.

He held post-doctoral and teaching positions at the Courant Institute, New York; the Karlsruhe Nuclear Research Center, Karlsruhe, Germany; Ames Labs, Ames, IA; and the University of Delaware, Newark, prior to joining AT&T Bell Laboratories in 1985. He currently is a Distinguished Member of Technical Staff at Lucent Bell Laboratories. His research interests include signal processing applications in acoustics and E&M, including adaptive beamforming, echo cancellation, noise suppression, as well as image processing and image compression. His most recent areas of work include speech processing applications, natural language understanding algorithms, and wireless MIMO modems. In addition to having publications and seven patents in these areas, he is the main author of a commercialized high performance acoustic echo canceler.



Eric Fosler-Lussier (SM'05) received the B.A.S degree in computer and cognitive studies and the B.A. degree in linguistics from the University of Pennsylvania, Philadelphia, in 1993. He received the Ph.D. degree from the University of California, Berkeley, in 1999; his Ph.D. research was conducted at the International Computer Science Institute, where he was also a Postdoctoral Researcher through August 2000.

From August 2000 to December 2002, he was a Member of Technical Staff in the Multimedia Communications Lab at Bell Labs, Lucent Technologies; from January to July 2003, he was a Visiting Scientist in the Department of Electrical Engineering, Columbia University, New York. Currently, he is an Assistant Professor in the Department of Computer Science and Engineering, The Ohio State University, Columbus, with a courtesy appointment in the Department of Linguistics, where he co-directs the Speech and Language Technologies (SLaTe) Laboratory. His interests include linguistic modeling for automatic speech recognition, spoken dialogue systems, statistical pattern recognition, and natural language processing. He has authored or co-authored over 50 journal and conference papers.

Dr. Fosler-Lussier currently serves on the IEEE Speech and Language Technical Committee.

Alexandros Potamianos (M'92) received the Diploma degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 1990. He received the M.S and Ph.D. degrees in engineering sciences from Harvard University, Cambridge, MA, in 1991 and 1995, respectively.

From 1991 to June 1993, he was a Research Assistant at the Harvard Robotics Lab, Harvard University. From 1993 to 1995, he was a Research Assistant at the Digital Signal Processing Lab, Georgia Tech, Atlanta. From 1995 to 1999, he was a Senior Technical Staff Member at the Speech and Image Processing Lab, AT&T Shannon Labs, Florham Park, NJ. From 1999 to 2002, he was a Technical Staff Member and Technical Supervisor at the Multimedia Communications Lab at Bell Labs, Lucent Technologies, Murray Hill, NJ. From 1999 to 2001, he was an adjunct Assistant Professor at the Department of Electrical Engineering, Columbia University, New York. In the spring of 2003, he joined the Department of Electronics and Computer Engineering at the Technical University of Crete, Chania, Greece, as Associate Professor. His current research interests include speech processing, analysis, synthesis and recognition, dialogue, and multimodal systems, nonlinear signal processing, natural language understanding, artificial intelligence, and multimodal child-computer interaction. He has authored or co-authored over 60 papers in professional journals and conferences. He is the coauthor of the paper "Creating conversational interfaces for children" that received a 2005 IEEE Signal Processing Society Best Paper Award. He holds four patents.

Dr. Potamianos has been a member of the IEEE Signal Processing Society since 1992 and he served as a member of the IEEE Speech Technical Committee from 2000 to 2003.