An Expectation-Driven Response Understanding Paradigm

Dong-Guk Shin, Member, IEEE

Abstract—This paper describes a model that can account for ad hoc user-responses to argument interrogative type of system-initiated questions. Successful implementation of the model can provide an alternative solution that is more effective than the menu-driven approach that has been proposed as a meager solution to enable the system to ask a question to the user. The proposed model assumes that when the system asks a question, it maintains an expectation of the potential answers. The system then uses the expectation as the focus to perform the most likely interpretation of the user’s response. Without using such a focus the interpretation process could be unbounded. The interpretation process is mapped into a heuristic search problem. The interpretation process results in identifying a particular expectation-response relationship type, which the system can use to tailor its response strategy with respect to the given user-response. A prototype has been constructed to demonstrate the soundness of the proposed model.

Index Terms—Expectation-driven approach, mixed-initiative system, man-machine dialogue, two-way communication, natural language interface

I. INTRODUCTION

MODELING a man–machine dialogue involves making the system handle ad hoc user questions as well as ad hoc user responses to the system’s own questions. Despite this double-sided nature of the problem, many previous studies of man–machine dialogue have addressed only the first half of the problem: to design the system to handle ad hoc user questions. Some typical examples of these studies include how a database interface can answer ad hoc natural language (NL) queries (e.g., [17], [19]); how an expert system can respond to a user’s question, which requires some problem solving (e.g., [29]); and how a natural language dialogue system can produce useful extended responses during interactive question-answering sessions (e.g., [24]). In existing dialogue systems, the second half of the problem, to design the system to process ad hoc user responses to its own question, is not addressed (or more or less bypassed by resorting to a quick and easy solution, the menu-driven approach). For example, a NL interface to database systems may generate a clarification question when an ill-formed query is entered, but the system’s question is usually followed by a set of menu options, and the user answers by simply choosing one of the available options (e.g., [19], [23]).

The menu-driven approach is cumbersome and unnatural. For example, when we ask a friend, “Where did you see the movie Rocky V?,” we neither impose on the friend the possible choices of movie theaters nor provide a catchall category, such as “Others,” to indicate to the friend that an additional round of question-answer exchange would be needed if his answer did not fit any of the given choices. A more natural response-and-understanding scenario would be the following: If the friend answered with the name of a known movie theater, we would continue to the next subject. If he answered with a name that we did not know, but probably referred to a movie theater, we might clarify by asking, “Where is it?” If he answered with a city name, such as “Hartford,” we might be reminded of some movie theaters in Hartford and ask, “Which one in Hartford?” If he answered, “Oh, I rented the movie,” we might be puzzled, because we would know the movie was just released, and that it takes a few months for a new movie to be released for home video.

From the viewpoint of the Gricean principle [16], the menu-driven approach also violates both maxims of Quantity and Manner. For example, when asking a question, particularly an argument interrogative, the questioner in general does not furnish the potential answers. Specifically, when a toll-free information operator is asked, “What is the number for Sprint?,” he may, and in fact he did, simply respond, “What type of company is Sprint?”; but he never enumerates (at least at this stage of the conversation) all of the possible company types. The operator who provides all of the possible company types would be verbose, a violation of Manner, and would make his contribution more informative than is required, a violation of Quantity. If the caller has no idea what is meant by type of company, then at this time, the operator might exemplify some of the company types. For example, in order to aid the caller’s understanding of company type, the operator might use a verbal cue, such as “Is this a long distance company?,” which would eventually help the caller to understand that “US Sprint” should have been mentioned in lieu of “Sprint.”

Why the existing dialogue systems resort to the menu-driven approach is clear: This approach provides a quick and easy solution to a difficult problem. The design philosophy behind these dialogue systems is to take a short-cut, which means to precompute the set of possible answers, regardless of what the user may respond, and force the user to choose one of the options as his answer. An inevitable consequence of this approach is the frustration of the user who does not know what to do when no menu-option fits his intended answer, or who
sometimes cannot even figure out what each option means. A better approach, as opposed to the short-cut approach, would not precompute the set of potential answers and impose them on the user, but rather would compute on the fly how the user response relates to the system’s question a posteriori to the user response. This seems to be the way that we understand someone else’s response to our questions.

The objective of this work is to describe a response-understanding model in which the user can present an ad hoc response to an argument interrogative type of system-initiated question. In this model, the system uses the expectation, which accompanies its own question as the focus to perform the most likely interpretation of the user’s response. This interpretation process is said to compute the conceptual distance between the expectation and the response. This computation process involves inferencing with the domain-dependent and domain-independent facts of the given subject.

The rest of the paper is organized in the following way. Section II describes the general background of the proposed work. Section III provides an overview of the proposed model. Section IV formally introduces the notion of the conceptual distance between expectation and response, and then discusses in detail how the conceptual distance computation can be modeled as a heuristic search problem. Section V describes the prototyping and summarizes the results of the computational experimentation with the prototype. Section VI provides the conclusion.

II. BACKGROUND

Current strategies for designing a dialogue-capable interface fall into two broad categories: strictly one-way and limited mixed-initiative. Menu-driven dialogue offers one-way communication, meaning that the system holds the initiative once the dialogue is initiated by the user. Some recent natural language interfaces for database systems support a more sophisticated form of menu-driven interaction by using knowledge and inferencing mechanisms (e.g., [19], [23]). The nature of one-way communication remains the same, however, and the high level of user frustration caused by these systems would be comparable to the frustration caused by the early menu-driven interface RENDEZVOUS [12].

Mixed-initiative, though limited, has been used in some interfaces for computer-assisted instruction systems and expert systems. Two common techniques are constrained turn-taking (e.g., [8]) and command-driven (e.g., [15]). The constrained turn-taking approach is artificial in the sense that the user may ask a question while the system’s question is pending, but the user’s question bears no logical relationship to the system’s pending question. Similarly, in the command-driven approach, users may be allowed to take the initiative, but they are generally restricted to a set of commands prearranged between system and user; therefore, very little if any contextual information is used to allow the user to take the initiative.

Use of contextual information has been studied in various dialogue modeling work. Frames have been used to maintain the context and to control the general flow of the man-machine dialogue (e.g., [4]). Domain-independent discourse processing rules have been suggested to model utterance relationships (e.g., [35]). But the use of plans has become most popular in recent dialogue modeling work. Allen and Perrault [1] modeled cooperative response generation into a plan recognition problem. Subsequently, Litman and Allen [25] discuss how a plan recognition model can also account for a variety of subdialogues, such as clarifications, corrections, and topic changes. Pollack [33] suggests that the modeling of the questioner’s plan should include invalid plans as well as the sources of the invalidity. Carberry [6], [7] argues that the questioner’s task-related plan should be inferred dynamically from an ongoing dialogue, and that the resulting context information should be used to produce cooperative responses to pragmatically ill-formed questions. Moore and Swartout [30] discuss modeling the system’s discourse goals and plans in order to allow the user and the system to engage in an explanation dialogue. Finally, Lochbaum et al. [26] model multiple agent collaboration as the augmentation of the participating agents’ beliefs about their actions and intentions.

Another class of question-answering work focuses on the problems of dealing with a user who holds incorrect beliefs or misconceptions about the domain. Kaplan [21] and McCoy [27] study particular types of problematic queries whose ill-formedness is caused by the user’s false presupposition or misconception about the queried object. Quilici [34] deals with correcting user’s misconceptions about actions in order to generate explanatory responses to advice-seeking requests. Weischedel and Sondheimer [39] propose to use metarules as a uniform framework for handling lexically, syntactically, semantically, and pragmatically ill-formed user inputs. Yet another genre of related work addresses the problem of text generation (e.g., [28]) and response tailoring according to the user’s focus of attention and level of expertise (e.g., [11], [32]).

The issues addressed in this work differ from the aforementioned related work. This work is not concerned with inferring the questioner’s goal, beliefs, intentions, or plans. Nor does it deal with understanding, correcting, or responding to ill-formed questions. Nor does it aim at generating an optimal answer or text using a user model. In short, this work is not concerned with designing a system that can answer intelligently a question posed by a user. Instead, this work addresses the issues of designing a system that is capable of handling a user’s answer to a question posed by the system. None of the above-mentioned models addresses this issue. These models assume either that the system does not have to counteract (one-shot process à la Moore and Swartout [30]), or that the system would always be able to process the user’s response to the system’s clarification question. For example, in the real-life dialogue example used by Carberry in [7], the information provider (i.e., the system’s role), in order to give advice, solicits additional information: “Did you work outside of the government last year?” Then the information seeker (i.e., the user’s role) responds, “Yes, I did.” The information provider then goes to the next step. Notice that there are many different ways of responding to the question (e.g., “I worked for Aetna,” or, “Do you mean federal government?”). It is not realistic to assume that the user will always respond to the system only in one way or another. Spitz’s recent
empirical analysis of user compliance supports this view [38]. She reports that when the information assistant’s question, “What city, please?,” only 15% of the callers answer with isolated city names. Even when the same question is asked by a machine-imitated information assistant, as many as 23% of the callers responded without including a city name, and 14% – 23% of the callers simply did not respond.

III. THE MODEL

A. Sources of Expectations

The model proposed herein outlines that when the system asks a question, it maintains the expectation of the potential answers and uses the expectation as the focus to perform the most likely interpretation of the user’s response. This view resembles what we often do when we pose a question: we hold some expectation about the potential answers and use it to discern meaningful responses. When the questioner is considered to be a machine, the following question needs to be asked: Where would the machine’s expectation come from?

Expectations can be thought of as something that can be predesigned and stored in a program. A few techniques of prestorage of expectations have been used in a variety of natural language understanding programs. In script-based story understanding programs (e.g., [14]), a script contains a sequence of highly specific typical expectations. In frame-based discourse processing programs (e.g., [4]), expectations are viewed as slots to be filled in task-oriented frames. In plan-based story understanding programs (e.g., [40]), expectations about the forthcoming event are constructed case-by-case from a set of primitive sources called plans, and the identified plans become expectations to predict future inputs. Similarly, in plan-based discourse processing work (e.g., [1], [25]), the user utterance is related to a step in a plan, and the related plan is used to anticipate the subsequent topics of the dialogue.

A script, a frame, or a plan can be used to satisfy the requirement to prestore expectations in the proposed model. The dialogue described below shows how expectations encoded in task-oriented plans can account for a real-life experience of handling an anticipated response to a question. The dialogue originates from a corpus of transcripts recorded while conversing with a Connecticut information operator who can be reached by dialing 411 within Connecticut.

1. Information: What city?
2. Caller: Amherst.
3. Information: Amherst, Massachusetts?
5. Information: You must dial 1 413 555 1212.

Some plans that the information operator might have employed are sketched in Fig. 1. The first question (1) can be seen as a straightforward execution of Action 1 of Plan A. Unfortunately, the response (2) fails to fulfill the expectation that has been specified in Action 1 and that can be considered to be accompanying the question (1). Meanwhile, the caller’s response (2) is not totally irrelevant to what one may ask an information operator, and indeed, the rest of the dialogue shows how the information operator interprets the caller’s intention and helps him accordingly. The operator’s response (3) indicates his intention to switch the current plan and the intention could have been derived from the following reasons. He knows that Amherst is not one of the 169 cities or towns in Connecticut that he originally anticipated he would hear; he knows that for an out-of-state request, he should inform the caller of the corresponding state information number, as specified in Plan B; he knows that Amherst is located in Massachusetts, and possibly in other states (e.g., Amherst is also in New Hampshire and New York). The operator’s response (3) is a subdialogue to clarify the correct state for Amherst. In (4), the operator gathers the right city and state names. In (5), he provides the right information number that the caller should use, which can be seen as him completing successful execution of Actions 2 and 3 of Plan B.

B. Expectation Resolution Modes

Expectation resolution is a means of determining how much—or how little—the questioner’s anticipation about an answer is satisfied by a response. Although it would be difficult to quantify the varying degree of satisfaction caused by a response, the degree of satisfaction or dissatisfaction would be measurable in qualifiable terms, provided that various expectation-response relationship types were identified and enumerated. Each relationship type can be used to signify
different ways an expectation is fulfilled, unfulfilled, or otherwise related to the response. Unfortunately, an attempt to identify all of the relations possible between a response and a general class of expectation would be doomed to fail, because many expectations are sometimes quite abstract and difficult to model. For this reason, the expectation resolution modes discussed here are restricted to a particular class of questions called argument interrogatives, for which the accompanied expectations generally can be modeled in terms of a set of nominal references to real-world objects.

In general, understanding a response is not a binary decision; it is not simply judging whether the question has been answered. Responses may indirectly answer the question, or they themselves might constitute a counterquestion. Further characterization of the nature of a response is possible if we discover the different ways in which the expectation is resolved by a response. For example, suppose a waiter asks a customer, "What would you like to have for dessert?" If the customer responds with "What do you have for dessert?," then the response is an attempt to ascertain the kinds of dessert the customer intends to serve. The customer’s response, indeed, is tantamount to requiring the waiter to reveal the contents of his expectation, i.e., to enumerate the set of values that the questioner anticipates to hear as right answers. Alternatively, suppose the response was, "I’d like to have a piece of lemon meringue pie." If lemon meringue pie was not one of the questioner’s planned items for dessert, then the response is an unfortunate mismatch, because lemon meringue pie happens not to concur with any prepared (and thus expected) dessert names. It must be pointed out, however, that the waiter would not be surprised to hear the response, because lemon meringue pie would be one of the general dessert items. On the other hand, imagine what would happen if someone responded, "I’d like to have broiled flounder with herb sauce." Overall, identifying the specific way that a response relates to the expectation is an important metric for discerning how much the expectation is satisfied by an answer, or, if the response is in the form of a question, for determining what aspects of the question need to be further clarified.

A collection of expectation resolution modes have been devised in order to identify a general class of expectation-response relationship types. Before going into the details, the following scenario is assumed. A system presents an argument interrogative to a user in an attempt, say, to initiate the system’s intended task, clarify a user-provided question, or request additional information for a problem-solving need. The system already has formed an expectation, denoted by $E$, regarding the potential answers. The user responds to the system’s question. Depending on the system’s resolution of the user-response, the system may counterrespond to the user-response or proceed to the next task. The expectation $E$ is a set of nominal references to real-world objects. For example, $E$ could have been {cheesecake, apple pie, chocolate mousse}, as a robot waiter anticipates to hear when asking, "What would you like to have for dessert?"

Below, 13 expectation resolution modes are discussed briefly that would account for representative response cases. Each resolution mode is illustrated by annotating with a real-life response to a Connecticut information operator’s question, "What city?," and the operator’s typical subsequent counterresponse.

**Direct Match:** The response directly matches an item among the anticipated values. Specifically, the answer provides a definitive nominal reference $r$ to some real-world object (or simply $r$ from here on) such that $r \in E$. The system’s expectation is completely satisfied, and the system may proceed toward its next step. (e.g., "Storrs," followed by, "Yes?")

**Indirect Match:** The response indirectly corresponds to one of the anticipated values. Given $r$, for some set $K$ of nominal references, where $K \neq E$, it holds that $r \in K$. Some inferencing allows $r$ to be corresponded to some $r' \in E$, where $r' \neq r$ (e.g., "Electric Boat, please," followed by, "Which division of Electric Boat?" [The operator knows that Electric Boat is in Groton, CT]).

**Indirect Ambiguous Match:** The response indirectly corresponds to multiple items among the anticipated values. Given $r$, for some set $K$ of nominal references where $K \neq E$, it holds that $r \in K$. Some inferencing allows $r$ to be corresponded to some subset $S$ of $E$. The system needs to launch a disambiguation subdialogue, unless the ambiguity is resolved otherwise, for example, from the context. (e.g., "North East Utilities," followed by, "Which city of North East Utilities?")

**Mismatch:** The response does not match, not even indirectly, any item among the anticipated values; but the nature of the response is identifiable by the system. That is, given $r$, for some set $K$ of nominal references where $K \neq E$, it holds that $r \in K$ (e.g., "Amherst," followed by, "Call 1-413-555-1212").

**No Match:** This is the case when the system cannot recognize the user response at all. That means not only that $r \in E$ holds but also that no set of nominal references is known to include $r$. Perhaps the system’s only counterresponse would be of the type, "I do not understand what you are talking about," or to request the user to explain further what the user-response means. (e.g., "Hollies," followed by, "No such city in Connecticut").

**Definition Request:** The response is the least sophisticated form of clarification question that straightforwardly requests the system to elaborate the anticipated values. The system may counterrespond to this type of response by simply enumerating each member of $E$, or, alternatively, by providing some statement that describes what $E$ is all about. (e.g., "What do you mean by city?" followed by, "The name of the city where the person you are looking for lives").

**Exemplify:** The response is a clarification question that reveals the user’s attempt to guess the system’s anticipating values. In general, the user presents an example item $r$, which is believed to hold that $r \in E$. If $r \in E$ holds, then the system may respond affirmatively. Otherwise, the system should launch a subdialogue to correct the user’s misunderstanding of the system’s anticipated values (e.g., "Do you mean like Storrs?" followed by, "Yes?").

**Verify:** This response is another form of a clarification question. But unlike the definition request case, here the
user provides a definitional description of $E$ to the system to verify whether his guess of $E$ is correct. If the user's description of $E$ is correct, then the system would respond affirmatively; but the system still anticipates that its original question will be answered (e.g., "Do you mean the city where my friend lives?" followed by, "Yes").

**Alternative Test:** This response requests permission to respond with some values other than what the system anticipates to hear. This case would occur if the user intended to answer with a value in some set $K$ of nominal references that he believes is different from $E$. The system may respond affirmatively if the system sees the possibility of a value $k \in K$ leading to an indirect match to $E$ (e.g., "Can I give you an exchange number?" followed by, "Yes, what is it?").

**Refine:** The response attempts to refine the system's expectation by explicitly associating it with additional constraints. In order for this to occur, the user first guesses the system's expectation and then intends to further clarify it by presenting his own more restricted version of $E$. The system may conclude the user's refinement is irrelevant or unnecessary. Otherwise, the system may respond affirmatively to the user's refinement attempt (e.g., "Do you mean a city name within Connecticut?" followed by, "Yes").

**Challenge:** The response is a counterquestion about the system's motivation or the relevancy of its question. The system needs to provide or explain the intention behind forming the expectation $E$ (e.g., "Why are you asking me a city name?" followed by, "A city phone directory needs to be brought up").

**Annul:** The response makes the system's expectation inappropriate or invalid in the current context. If feasible, the system switches its current dialogue plan to another one in which the user-response becomes more appropriate (e.g., "What is the number for toll-free information?" followed by, "Yes").

**Deny:** This is when the user asserts his inability to answer the question or his uncertainty about a possible response. It is presumed that the user has a good understanding of what the system's expectation $E$ is, but he just does not know the right answer. The system's counterresponse would be to find an alternative way of getting the anticipated information. If no such alternative exists, the dialogue would halt (e.g., "I don't know," followed by, "Sorry, I can't help you").

The list of the resolution modes given above is by no means exhaustive. This is provided to set forth a framework useful for analyzing the types of responses to questions.

### C. Knowledge Identification, Organization and Processing

The process of determining a specific expectation-response relationship hinges on using the domain knowledge that underlies the subject of the question-answering. Another real-life dialogue given below demonstrates this point. This is the case where a Connecticut information operator did not know that Amherst is a city in some neighboring states.

(6) **Information:** What city?

(7) **Caller:** Amherst.

(8) **Information:** No such city in Connecticut.

(9) **Caller:** OK.

Contrasting the above dialogue (6)-(9) with the previous one (1)-(5) illustrates how the domain knowledge is used in expectation processing. In the dialogue (1)-(5), knowing that Amherst is a city in Massachusetts makes the response (2) a mismatch to the current expectation; and at the same time, such knowledge eventually leads the response (2) to become a direct match to some expectation hidden in another task plan. On the other hand, in the dialogue (6)-(9), because the information operator did not know where Amherst is, the user's response (7) becomes a no match to the current expectation, which turns the conversation into a dead end.

In the proposed model, the domain knowledge is organized by using the representation primitives, called concept-terms (or simply $c$-terms). A $c$-term is a framelicke structure of the following form:

$$l[c_1 : t_1, \ldots, c_n : t_n].$$

where $n \geq 0$, $l$ is called the head concept label (or simply a label), and for each $i$, $1 \leq i \leq n$, $c_i$ is called a concept-connector, and $t_i$, being itself a $c$-term, is called the target for the concept-connector $c_i$. Each component $c_i : t_i$ is called a concept-restrictor (or simply a restrictor) of the label $l$, and the component $[c_1 : t_1, \ldots, c_m : t_m]$ is called a restriction of the label $l$. For each $l$, only a specific set of connectors is permitted to be included in its restriction, and that relationship is described as connector-label associativity.

Each constituent of a $c$-term is made out of some pre-designed set of English words and special symbols. Each $c$-term as a whole denotes some particular abstract or concrete concept of the real world, where the denoted concept is constructed by combining the meanings associated with each constituent of the $c$-term in a certain way. To be more specific, each concept-restrictor constrains in a specific way how the concept denoted by its corresponding label is further specialized. For example, the following $c$-term:

$$t = \text{PAY}[\text{agent: HARDWARE-STORE}[\text{location: VERNON}], \text{recipient: ROBBIE}],$$

denotes the action in which a hardware store in Vernon pays Robbie. A formal and more complete description of the syntax and semantics of $c$-terms can be found in [36].

$C$-terms are used to construct two classes of knowledge. The first class of knowledge is made of the $c$-terms beginning with the two special labels, $\in$ and $\subseteq$. Two expressions, $\in[c_1 : t_1, c_2 : t_2]$ and $\subseteq[c_1 : t_1, c_2 : t_2]$, where $c_1$ and $c_2$ are some generic connectors, are intended to describe instance-of and subclass-of relations respectively, and these two expressions are called an instantiation assertion and a classification assertion, respectively. The collection of instantiation and classification assertions constitutes a concept hierarchy knowledge base. Below, a few examples illustrate entries of this class of knowledge, where $\in$ and $\subseteq$ are used in infix notation.

<table>
<thead>
<tr>
<th>$Id$</th>
<th>$C$-Terms</th>
<th>Denoted Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>ROBBIE $\in$ ROBOT</td>
<td>Robbie is a robot.</td>
</tr>
</tbody>
</table>
The concept hierarchy provides a means of recognizing the mentioned object by finding its corresponding position within the hierarchy. Once the position is found, clusters of related concepts are identifiable, and their classification knowledge becomes available for future inferencing. Specifically available from the hierarchy are the following: two related concepts share any common properties, the notion of genus, and how they differ from each other, the notion of differentia.¹

1 A hierarchy construction is an old technique (see Sowa's definition on Aristotelian type hierarchy [37]). In traditional AI programs, concept hierarchies are built by using IS-A links. Instead of using IS-A links, some practitioners use the notions of subtype, supertype, and type-instance. Others suggest a more elaborate scheme of distinguishing subconcept, superconcepts, and concept instances. For example, Brachman [5] attempts in KL-ONE to encode into the hierarchy not only the subconcept or superconcept relationships but also the restriction relationships holding between the roles or properties of the involved concepts.

<table>
<thead>
<tr>
<th>Id</th>
<th>C-Terms</th>
<th>Denoted Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>k₂</td>
<td>Q-SUPPLY ∈ ACME[location:Vernon]</td>
<td>Q. Supply is an Acme store in Vernon.</td>
</tr>
<tr>
<td>k₃</td>
<td>ACME ⊆ HARDWARE-STORE</td>
<td>Acmes are hardware stores.</td>
</tr>
<tr>
<td>k₄</td>
<td>VERNON ∈ CITY[state:CT]</td>
<td>Vernon is a city in Connecticut.</td>
</tr>
</tbody>
</table>

Event or state c-terms make assertions by themselves. Event or state c-terms are included in the system if their denoted concepts are believed to be valid in the world. In comparison, entity c-terms are included in the system if their denoted objects exist in the world.

The second class of knowledge is labeled predication assertions. Each predication assertion establishes direct associations among the concepts denoted by its constituents; for example, k₇ shows that how “Robbie” is related to “the license number 361 EJP of Connecticut.” Indirect association among related concepts can also be derived by properly linking appropriate direct associations through the established concept hierarchy. For example, the fact that “the car with the license number 361 EJP of Connecticut” is owned by “a robot” is derivable by generalizing the label ROBBIE of k₇ with the instantiation assertion k₈.

In order to perform indirect associations and other reasoning steps systematically, eight c-term manipulation operations are defined. (Detailed description of these operations is given in [36].)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>O₁</td>
<td>Upward label substitution</td>
</tr>
<tr>
<td>O₂</td>
<td>Restrictor Release</td>
</tr>
<tr>
<td>O₃</td>
<td>Downward label substitution</td>
</tr>
<tr>
<td>O₄</td>
<td>Restrictor introduction</td>
</tr>
<tr>
<td>O₅</td>
<td>Restrictor inheritance</td>
</tr>
<tr>
<td>O₆</td>
<td>Membership identification</td>
</tr>
<tr>
<td>O₇</td>
<td>Concept-connector identification</td>
</tr>
<tr>
<td>O₈</td>
<td>Label-target rotation</td>
</tr>
</tbody>
</table>

A few examples are given below to illustrate what possible changes might occur when each of the eight transformation operations is applied to a c-term. The application of some operations requires use of some related domain knowledge. Below, the knowledge used is identified as an argument of the operation, and the changes brought about by the transformations are underlined.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>O₁(t, k₁)</td>
<td>PAY[agent:HARDWARE-STORE[location:Vernon],recipient:ROBOT]</td>
</tr>
<tr>
<td>O₂(t)</td>
<td>PAY[agent:HARDWARE-STORE,recipient:ROBBIE]</td>
</tr>
<tr>
<td>O₃(t, k₃)</td>
<td>PAY[agent:ACME[location:Vernon],recipient:ROBBIE]</td>
</tr>
<tr>
<td>O₄(t, k₅)</td>
<td>PAY[agent:HARDWARE-STORE[location:Vernon],recipient:ROBBIE[color:SILVER]]</td>
</tr>
<tr>
<td>O₅(t, k₄)</td>
<td>PAY[agent:HARDWARE-STORE[location:VERNON[state:CT],recipient:ROBOT]</td>
</tr>
<tr>
<td>O₆(k₂)</td>
<td>ACME[val:Q-SUPPLY,location:Vernon]</td>
</tr>
<tr>
<td>O₇(t, k₆)</td>
<td>MONITOR[agent:IRS,object:CAR[license.no:361 EJP[state:CT]]]</td>
</tr>
<tr>
<td>O₈(t)</td>
<td>HARDWARE-STORE[location:VERNON,recipient:PAY[agent:ROBBIE]]</td>
</tr>
</tbody>
</table>

The result produced by O₆(k₂) can be used in subsequent steps. For example, O₄(0₃(t,k₃),O₆(k₂)) can make t be transformed into the following form:

PAY[agent:ACME[val:Q-SUPPLY,location:Vernon],recipient:ROBBIE].

The above transformations illustrate what each operation is introduced for: O₁ is for generalization; O₂ is for abstraction; O₃ and O₄ are for plausible specialization; O₅ is for sound refinement; O₇ is for plausible association; and O₆ and O₈ are for focus change, where the focus of a c-term is determined by its head concept label. O₇ is similar to O₄, but differs in that O₇ produces a transformation whose focus changed. The operations O₁ and O₅ are introduced to perform deduction. In comparison, the operations O₃, O₄, and O₇ are introduced to perform a form of abduction.²

² Intuitively, the form of abduction obtained by the operations O₃, O₄, and O₇ appears to be different from the conventional notion of abduction described in [10], but we do not attempt to clarify this intuition here.
all paying events. Nonetheless, abduction of these types is essential in the proposed model, because the system often has to generate hypotheses to find ways of establishing a logical connection between the user-provided response and the system's expectation. This characterization of abduction can be seen as similar to that of other work on abduction. For example, in story understanding, abduction is used as a way to establish causal chains between an action mentioned in the story and possible explanatory motivations [9]. In medical diagnosis, hypotheses are generated and composited to explain the symptoms of a patient [20]. In natural language understanding, attempting to explain why the sentence was mentioned is seen as a process of interpreting the sentence [18].

IV. CONCEPTUAL DISTANCE COMPUTATION

A. The Search Space

Given the system-expectation and a user-response, the process of determining their specific expectation-response relationship can be modeled into a search problem. This modeling involves a number of issues: What constitutes the search space? What are the starting and ending points of the search? What meaning is associated with the search path that is found? Does it suffice to find some path, or is it necessary to find the optimal path? What heuristics can be employed to reduce search? A discussion of heuristics is lengthy, and is deferred until the next subsection. This subsection is devoted to addressing the general issues regarding the modeling.

The search space that determines an expectation-response relationship is constructed as follows. First, the user response that is expressed in English is translated into a c-term, say, r, by performing a sequence of linguistic analyses.\(^3\) This process identifies the focus of the response that can be compared with the system's expectation. For example, if the user responds, "Can I give you an exchange number?" to the system asking, "What city?", then "an exchange number" needs to be identified comparable to the system's expectation of "city names." The c-term r, which represents the focus of the user's response, becomes the starting point for the search. The search begins by expanding r into a search tree by repeatedly applying, whenever possible, the eight c-term transformation operators introduced in Section III-C. The available domain knowledge is used in this process.

Now, imagine all of the possible c-term expressions that are derivable by repeatedly applying the eight transformation operations to the initial c-term, r. The collection of all of these c-terms constitutes the search space, and it is expected that the search space will contain one or more transformations that are a direct match to the system's expectation, if such things exist. Intuitively, the search space's size will grow immense, though it would be finite because of the way that transformations are performed. Given below is a best-first search procedure, labeled COMPUTE-PATH, in which some heuristic measure is used to compute the merit of each expanded c-term expression, and the c-term with the highest merit among all of the unexplored c-terms is chosen for the next expansion.

Procedure COMPUTE-PATH([E, r])

STEP 1: Create a search tree T containing only the root node (r, nil). Put r in a list called OPEN.

STEP 2: Initially set the variable called LEVEL 0.

STEP 3: Select the first element in OPEN, remove it from OPEN, and call this element r<sub>t</sub>.

STEP 4: If LEVEL = n<sub>max</sub> for some positive integer n<sub>max</sub>, then exit with the report indicating the failure of recognizing r and produce the path obtained by tracing the back-pointers from the r<sub>t</sub>’s node to the r’s node in T.

STEP 5: If r<sub>t</sub> is a direct match to E, then exit with the path obtained by tracing the back-pointers from the r<sub>t</sub>’s node to the r’s node in T.

STEP 6: Generate the set S of ordered-pairs such that if (r<sub>k</sub>, o) e S then r<sub>k</sub> is a transformation which is obtained by applying the transformation operator o to r<sub>k</sub>. Eliminate s from S if s contains a transformation which appears in an ancestor of the r<sub>t</sub>’s node. Install each member of S as a successor of the r<sub>t</sub>’s node in T.

STEP 7: Establish a back-pointer from each member of S to the r<sub>t</sub>’s node in T. Append the transformations appearing in S to OPEN.

STEP 8: Reorder OPEN based on some heuristic measure.

STEP 9: Go to STEP 3.

The procedure COMPUTE-PATH ends at either STEP 4 or STEP 5. Exiting from STEP 4 occurs when the search reaches the predetermined limit. Exiting from STEP 5 occurs when a transformation is found that is a direct match to the system's expectation. In either case, the procedure produces a transformation path from the root node r to the last focused expression r<sub>i</sub>. The expression r<sub>i</sub> can be thought of as the system's interpretation of r with respect to E. The path from r to r<sub>i</sub> can be thought of as the sequence of inferencing steps that takes place during the interpretation. In order to more specifically describe how the path discovery relates to determining response-expectation relationships, the notion of conceptual distance is introduced.

Let the path obtained upon exit from the procedure COMPUTE-PATH be the sequence of ordered triples \(\langle (r, o_1, r_1), (r_1, o_2, r_2), \ldots, (r_{n-1}, o_n, r_n), (r_n, o_{n+1}, r_{n+1}) \rangle\), where each triple \((r_{i-1}, o_i, r_i)\), 1 ≤ i ≤ n, indicates that r<sub>i</sub> is obtained by applying the transformation operator o<sub>i</sub> to r<sub>i-1</sub>. Suppose s represents a state evaluator that describes possible relationship existing between r<sub>i</sub> and the system expectation E. Then the sequence of ordered triples, \(\langle (r, o_1, r_1), (r_1, o_2, r_2), \ldots, (r_{n-1}, o_n, r_n), (r_n, s, E) \rangle\) is said to represent the conceptual distance from r to E. For the sake of simplicity, this conceptual distance is expressed in the form of \(\langle (r, o_1, r_1), (r_1, o_2, r_2), \ldots, (r_{n-1}, o_n, r_n), (r_n, s, E) \rangle\).

The above definition states that the conceptual distance from r to E can be described by specifying the two measures, one indicating what the path from r to r<sub>i</sub> looks like and one indi-

\(^3\)Besides the syntactic and semantic analyses, an additional linguistic analysis called pragmatic processing (e.g., speech act analysis [13]) would be required to determine other important aspects of the response such as intention and focus.
SHIN: EXPECTATION-DRIVEN RESPONSE UNDERSTANDING PARADIGM

437

\begin{align*}
C_h(r_k, E) &= \begin{cases} 
Y & \text{if } \ell_e \text{ is identical to } \ell_e \\
E & \text{if } \ell_e \text{ is sub- or super-concept related to } \ell_e \\
N & \text{otherwise}
\end{cases} \\
C_r(r_k, E) &= \begin{cases} 
Y & \text{if the head-concept label of some } \tilde{t}_r^e \text{ is identical to } \ell_e \\
E & \text{if the head-concept label of some } \tilde{t}_r^e \text{ is sub- or super-concept related to } \ell_e \\
N & \text{otherwise}
\end{cases} \\
C_e(r_k, E) &= \begin{cases} 
Y & \text{if the head-concept label of some } \tilde{t}_e^e \text{ is identical to } \ell_e \\
E & \text{if the head-concept label of some } \tilde{t}_e^e \text{ is sub- or super-concept related to } \ell_e \\
N & \text{otherwise}
\end{cases} \\
C_m(r_k, E) &= \begin{cases} 
Y & \text{if for each } c^e_i \text{ there exists } c^e_j \text{ such that } c^e_i = c^e_j; \text{ and for such } i \text{ and } j \text{ the head-concept labels of } \tilde{t}_e^e \text{ and } \tilde{t}_e^e \text{ are identical each other} \\
E & \text{if for each } c^e_i \text{ there exists } c^e_j \text{ such that } c^e_i = c^e_j; \text{ and for such } i \text{ and } j \text{ the head-concept labels of } \tilde{t}_e^e \text{ and } \tilde{t}_e^e \text{ are sub- or super-concept related each other} \\
N & \text{otherwise}
\end{cases} \\
C_p(r_k, E) &= \begin{cases} 
Y & \text{if for some } i, c^e_i = \text{val} \\
N & \text{otherwise}
\end{cases}
\end{align*}

cating how \( r_k \) relates to \( E \). In addition, another measure can be taken into consideration: whether other transformation(s) exists in the OPEN list whose merit value is identical to that of \( r_k \). Overall, depending on how these three measures are combined, the conceptual distance computation can turn into five different cases. Each of these cases is described below.

Case 1: \( s \) indicates a direct match; the length of the path from \( r \) to \( r_k \) is null (i.e., \( r = r_k \)), and no transformation of the same merit as \( r_k \) exists. This case corresponds to the most straightforward expectation-response relationship, Direct Match. When the user-response is in a question form, it can correspond to an Exemplify.

Case 2: \( s \) indicates a direct match; the length of the path from \( r \) to \( r_k \) is not null and no transformation of the same merit as \( r_k \) exists. This case corresponds to an Indirect Match. When the user-response is in a question form, it can also correspond to an Alternative Test.

Case 3: \( s \) indicates a direct match; the length of the path from \( r \) to \( r_k \) is not null and one or more other transformations of the same merit as \( r_k \) exist. This case corresponds to an Indirect Ambiguous Match.

Case 4: \( s \) does not indicate a direct match, but \( r_k \) matches the system's some other prestored expectation \( E' \). This case can correspond to a Mismatch or an Alternative Test if the user-response is in a question form.

Case 5: \( s \) does not indicate a direct match, and it does not match the system's some other prestored expectation \( E' \). This case corresponds to a No Match.

In conclusion, the descriptions listed above indicate that computing the conceptual distance between \( r \) and \( E \) amounts to identifying a specific expectation-response relation between \( r \) and \( E \).

B. Search Heuristics

The evaluation function that is used in the procedure COMPUTE-PATH is described below. This evaluation function, denoted by \( h(\cdot) \), assigns a numeric merit value to each \( c \)-term generated by the procedure. Search reduction is attempted by sorting all the to-be-expanded \( c \)-terms by merit value and choosing the \( c \)-term with the highest merit (the lowest numeric value here) as the next one to be expanded. To keep the cost of computing the evaluation function \( h(\cdot) \) manageable, two techniques are employed: One limits the ways in which each \( c \)-term (obtained from transformation) can be computed, say, \( r_k \), are expressed by the following forms:

\[ E = l_e[c^e_1 : t^e_1, \ldots, c^e_m : t^e_m, \text{val} : \text{val}] \]
\[ r_k = l_e[c^e_1 : t^e_1, \ldots, c^e_p : t^e_p] \]

Suppose \( I \) is an index set, \( I = \{ h, r, e, m, p \} \). Five different meaningful ways of comparing the constituents of \( r_k \) and \( E \) are defined at the top of this page.

Now five weight functions \( \omega_i \), where \( i \in I \), are derived, each of which maps the range of each function described above into natural numbers. Here the weights are assigned in a way that the outcome of \( C_h \) is weighed most heavily, then the outcomes of \( C_r \) and \( C_e \), and then the outcome of \( C_m \), and, finally, the outcome of \( C_p \). The outcome of \( C_h \) is weighed most heavily because the relation between two head-concept labels \( l_e \) and \( l_e \) is the most crucial in determining the likelihood of \( r_k \) turning out to be a direct match to \( E \). An example of the five weight functions are given in Fig. 2. Overall, the evaluation
The merit.

Note here that the proposed weight function to its corresponding output of merit scores, each of which resulted from applying how closely two labels are sub- or superconcept-related can be quantified and factored into the evaluation. Also, comparing the constituents of be developed. First, one can design more sophisticated ways for comparison. One attempt to produce such a similarity LM function is expressed by the following equation:

\[ h(r_k) = \sum_{i \in I} W_i(C_i(r_k, E)). \]

Fig. 2 shows possible combinations of outcomes for \( C_i(r_k, E) \), where \( i \in I \). The figure includes a sequence of merit scores, each of which resulted from applying the proposed weight function to its corresponding output combination. Note here that "-" indicates that the outcome of the corresponding comparison is irrelevant to computing the merit.

C. Alternative Heuristics

Heuristic measures other than the one proposed above can be developed. First, one can design more sophisticated ways of comparing the constituents of \( r_k \) and \( E \). For example, how closely two labels are sub- or superconcept-related can be quantified and factored into the evaluation. Also, comparisons of the constituents of \( r_k \) and \( E \) may continue on to the nested level of the expressions (i.e., within \( t'_k \) and \( t'_k \)), resulting in more comparison functions and additional weight functions. Second, instead of using comparison functions and their corresponding weight functions, all of the possible ways to describe the similarity between \( r_k \) and \( E \) can be enumerated, and the resulting descriptions can be numerically ordered for comparison. One attempt to produce such a similarity description set and ordering resulted in a total order of 40 meaningful description primitives [2]. Third, in addition to the above methods of estimating the similarity between \( r_k \) and \( E \), the actual shortest transformation cost needed to produce \( r_k \) from \( E \) can be included in the overall heuristic evaluation of \( r_k \). The result is an \( A^* \) search.

Designing an \( A^* \) search is not as straightforward as one might hope. It is difficult to quantify the cost of each transformation step while each operation performs different kinds of transformations. Even if such cost is measurable, it is not clear how that cost can be numerically and meaningfully added to the estimated cost representing the similarity between \( r_k \) and \( E \). One approach that has been attempted in [3] is to divide the transformation operations into three groups: \( G_1 = \{O6, O8\} \), \( G_2 = \{O1, O2, O5\} \), and \( G_3 = \{O3, O4, O7\} \). Different weights are assigned to each group to reflect the different strength of the operations. For example, if \( W_k^E \) is the weight for the operations of \( G_1 \), then weights satisfying the condition \( W_k^E > W_k^G_2 > W_k^G_3 \) are designed. Because the operations of \( G_1 \) do not involve any reasoning, they should be the least costly; the operations of \( G_2 \) involve formal deduction and should be more costly; and the operations of \( G_3 \) involve abduction and should be the most costly. Different choices of \( W_k^E \) can be attempted to see whether its addition to the similarity estimation between \( r_k \) and \( E \) produces any meaningful result. Experimental results of this approach are discussed in Section V.C.

V. EXPERIMENTATION

A. Prototype Architecture

The key idea behind the proposed model is that determining response-expectation relationships amounts to the system recognizing ad hoc user-responses to its own question. To demonstrate that this tenet may indeed work, a prototype system, labeled MIDMAN, has been constructed. The overview of the prototype’s components is given in Fig. 3. Note that the current architecture omits any mechanism for handling plans or task-oriented schemas. The system’s expectation about the user’s potential answers is prestored into the system, because the prototype was designed exclusively for the purpose of determining response-expectation relationships. Given below is a brief description of each function of the processing components presented in Fig. 3.

Transformation Executive: This component is responsible for controlling the overall flow of the response-expectation relationship decision process. At the core of this control program is a search procedure that takes the response as the root node and expands it into a search tree by repeatedly applying the available transformation operators. Applying transformation operators requires access to the prestored knowledge base as well as the supporting database. The search process halts either when a transformation is found that directly matches the expectation, or when the search tree grows beyond a certain boundary.

Transformation Operators: These are eight individual lisp functions that implement, respectively, the eight transformation operations described in Section III-C. Each of these functions takes an input expressed in c-term and transforms it into other forms of c-term. These functions share the prestored knowledge base and the content of the knowledge base (KB) cache.

Control Heuristics: This component is responsible for producing a merit value for each transformation expression that is generated. The merit value is computed by comparing the current transformation with the system’s expectation. The
transformation executive uses these numeric values in order to choose the most meritorious transformation to be expanded next.

**Prestored Knowledge Base:** This component contains part of the necessary domain knowledge, which includes all of the classification assertions and some small portion of the predication assertions that are unfit for storage in the supporting database. Each entry in the knowledge base is a c-term and therefore is ready for use by the transformation operators.

**Relational DBMS:** The relational DBMS stores a large portion of the predication assertions where these are well structured and more accurately termed the “data” rather than the “knowledge.” These assertions are entered and modified via the DML component of the underlying DBMS, and they become available for the transformation process upon request.

**DB Query Generator:** Whenever a particular set of data stored in the database needs to be looked up for some transformation, this component is responsible for generating the corresponding QUEL query expression. For the query generation, this component uses the KB/DB mapping information.

**KB/DB Mapping Information:** This component describes the mapping relationship between the database constituents and the knowledge base constituents. Specifically, it maintains the organizational links that describe how an assertion schema of the prestored knowledge base, or its constituents, are realized in terms of database domains, attributes or relations.

**KB Cache:** When the supporting database produces the result of processing a query, the result is converted into a c-term form and is held temporarily and locally by the KB cache. The content of the KB cache becomes available for the transformation operators as if it is a logical extension of the prestored knowledge base.

MIDMAN has been implemented on the Zetalisp environment of Symbolics 3670. MIDMAN’s supporting database component is implemented with INGRES (Version 5.0) running on Sun 3/280. MIDMAN interfaces with INGRES via C language function calls over the network file server.

**B. Sample Run**

A sample run from MIDMAN is shown in Fig. 4. The sample run illustrates the sequence of processes that takes place when the user-response is an indirect match to the expectation. The transcript of the run has been edited from an actual one in order to make it brief and readable. For referencing purposes, numbers are attached to some steps of the run.

The current MIDMAN uses a semantic parser to convert the user-response to a c-term. The step (1) shows the outcome of the parsing. The step (2) demonstrates that the system anticipates a university name as the potential answer. If the expression of (1) were a direct match to the expression of (2), then the process would halt at this point. Because this is not the case, the search process begins by expanding the user-response of (1). In (3), the initial expansion results in two transformations. Here each transformation is shown as an ordered triple, say, \((t, o, v)\), where \(t\) is the obtained c-term...
MEDMAN: I ask you a question, and you answer me.
MEDMAN: Where did you do your thesis?
USER: My advisor was John Doe.

Logical expression of the user-response is:
ADVISOR[val:JOHN-DOE]

MEDMAN’s expectation is:
UNIVERSITY[val:TARGET*]

Next transformation begins with:
ADVISOR[val:JOHN-DOE]

Transformations obtained are:
PROFESSOR[val:JOHN-DOE] 01 90
ADVISOR[val:PERSON[val:JOHN-DOE]] 04 90

Next transformation begins with:
ADVISOR[val:PERSON[val:JOHN-DOE]]

Transformations obtained are:
PROFESSOR[val:PERSON[val:JOHN-DOE]] 01 90

Next transformation begins with:
TEACHER[val:JOHN-DOE]

Transformations obtained are:
EMPLOYEE[org:SCHOOL,val:JOHN-DOE] 01 90
Teacher[org:SCHOOL,val:JOHN-DOE] 05 80
Teacher[org:PERSON[val:JOHN-DOE]] 08 90

Next transformation begins with:
EMPLOYEE[org:SCHOOL,val:JOHN-DOE]

(transformation continues) ...

Next transformation begins with:
TEACHER[org:SCHOOL,val:JOHN-DOE]

Search for DB schema pertaining to:
TEACHER SCHOOL

No pertinent DB schema exists.

Transformations obtained are:
EMPLOYEE[org:SCHOOL,val:JOHN-DOE] 01 90
Teacher[org:SCHOOL,val:JOHN-DOE] 05 80
Teacher[org:PERSON[val:JOHN-DOE]] 08 90

Next transformation begins with:
TEACHER[org:SCHOOL,val:PERSON[val:JOHN-DOE]]

(transformation continues) ...

Next transformation begins with:
TEACHER[org:UNIVERSITY,val:JOHN-DOE]

(transformation continues) ...

Next transformation begins with:
EMPLOYEE[org:UNIVERSITY,val:JOHN-DOE]

Search for DB schema pertaining to:
EMPLOYEE UNIVERSITY

Pertinent DB relation has been found: UNIV-EMP

SQL query formation:
SELECT UNIVERSITY,EMPLOYEE FROM UNIV-EMP WHERE EMPLOYEE = 'JOHN-DOE'

Tuples returned from the DB are:

Tuples translated and stored into KB cache are:

EMPLOYEE[org:UNIVERSITY[val:UCONN],val:JOHN-DOE]

Transformations obtained are:
EMPLOYEE[org:UNIVERSITY[val:UCONN],val:JOHN-DOE] 01 80
EMPLOYEE[org:SCHOOL,val:JOHN-DOE] 01 80

Next transformation begins with:
EMPLOYEE[org:UNIVERSITY[val:UCONN],val:JOHN-DOE]

Transformations obtained are:
EMPLOYEE[org:SCHOOL[val:UCONN],val:PERSON[val:JOHN-DOE]] 06 70

Next transformation begins with:
EMPLOYEE[org:SCHOOL,val:PERSON[val:JOHN-DOE]]

Direct match found with:
UNIVERSITY[org:UCONN,val:EMPLOYEE:PERSON[val:JOHN-DOE]] 01 80

No ambiguity found.

Evaluation: Indirect Match

Conceptual distance computed:
ADVISOR[val:JOHN-DOE]
PROFESSOR[val:JOHN-DOE] 01
TEACHER[val:JOHN-DOE] 01
Teacher[org:SCHOOL,val:JOHN-DOE] 03 80
Teacher[org:PERSON[val:JOHN-DOE]] 08 90

Teacher[org:SCHOOL,val:PERSON[val:JOHN-DOE]] 08 90

Teacher[org:UNIVERSITY,val:JOHN-DOE]

(transformation continues) ...

Teacher[org:UNIVERSITY,val:PERSON[val:JOHN-DOE]]

(transformation continues) ...

Conclusion: *TARGET* = UCONN

Fig. 4. An example run with MIDMAN.

(e.g., PROFESSOR[val:JOHN-DOE]), o is the transformation operator identifier that has been used to produce the c-term t (e.g., 01) and the merit value assigned to the c-term (e.g., 90), based on the heuristics presented in Section IV-B.

Because both transformations of (3) have the identical merit, the next step (4) shows that the second transformation in (3) is chosen arbitrarily for the next expansion. In (5), the expansion produces two transformations whose merits exceed its predecessor's. Among these two transformations, the next step shows that the first one is chosen arbitrarily for the next expansion. The search continues as long as no transformation emerges that is a direct match to the expectation.

The step (6) shows a transformation that has emerged in (5) and is currently chosen for the next expansion. A transformation of the type (6) triggers a need to check the database schemas to see whether the supporting database contains any particular information such as which school employs the teacher John Doe. The step (7) shows how the database schema search begins using the terms TEACHER and SCHOOL. Unfortunately, it is concluded that no such information is available under those terms in the database. Therefore, the expansion process continues without the benefit of using the database, and this process results in six transformations as given in (8). The step (9) shows the next expansion choice of the most meritorious transformation. The search process continues.

Step (10) shows a transformation that has emerged at some previous step and has been chosen for the next expansion. Step (11) demonstrates the system’s attempt to check whether the supporting database contains any employment relationship.
such as which university employs John Doe. Note that in step (11), the database schema search begins with the terms EMPLOYEE and UNIVERSITY. Step (12) shows that the needed information is available in the relation, namely, UNIV-EMP. The system uses the KB/DB mapping information for the discovery.

Before going to the next step, an important observation needs to be made regarding the two database schema searches attempted in (7) and (11). Both searches attempt to find whether the information about John Doe's employment is available from the supporting database. The actual run transcript demonstrates that the search of (7) initially uses the wrong search terms, TEACHER and SCHOOL, and fails, but the search of (11) chooses the right search terms, EMPLOYEE and UNIVERSITY, and succeeds. Had the search of (11) also failed as in (7), the attempt to find John Doe's employment information would continue by using other related terms that would be subsequently provided by the next transformation processes. This trial-and-error method for finding the right search terms can be seen as a way of modeling one's attempt to perform a database search without remembering the right relation or attribute names to use.

Step (13) shows that an SQL query expression is generated in order to request the name of the university that employs John Doe. In (14), a tuple is returned as the answer to the query from the supporting database. In (15), the meaning described by the tuple, say, "John Doe is an employee of the University of Connecticut," is encoded into a c-term and stored into the KB cache. In (16), the first expression illustrates how the information stored in the KB cache is used to transform the expression in (10) into a more specific one. Last, step (17) shows the expression that is chosen for the next expansion. The result of that expansion is shown in (18).

The step (19) shows that an expression among the three transformations of (18) is a direct match to the expectation. The search process would halt here. However, an additional check tests whether any other similar direct match expressions exist. If there were, the search would conclude with an indirect ambiguous match. The step (20) confirms that no other direct match expression exists. In (21), the search concludes that the original response is an indirect match, because it was not a direct match initially; but a direct match expression is derived eventually. The step (22) summarizes the entire search path from the user-response (1) to the derived direct match expression of (19). Finally, in (23), the process is completed by concluding the name of the university that the system believes the user-response could have meant.

C. Implementation Results

Current implementation of MIDMAN has been tested with three system-initiated questions, each from a different domain. Each question was answered with 5-6 responses from different categories: direct, indirect, ambiguous, and nonsequitur. The knowledge base contains c-terms of the order of magnitude $10^2$, and the back-end relational database contains six different sets of relations, where each relation contains tuples of the order of magnitude 10. All together, 17 ad-hoc user responses have been tested. These tests are based on the A* search discussed in Section IV-C. Two techniques were employed to further enhance the efficiency of the search: Any transformation that violates the selectional restrictions (à la Katz and Fodor in [22]) is prohibited, and any c-term that is generated redundantly and that is more costly is abandoned. The test results concur with the human judgment of the response interpretations, although the system-produced paths tend to include too many detailed steps in traversing the concept.
hierarchy. This tendency is illustrated by the path found in Fig. 4.

For empirical analysis, three items have been counted from the system output of each response process: $T$ for the total number of c-term transformations generated during the search, $E$ for the total number of c-terms expanded in the search tree (i.e., $T$ less the closed nodes of the search tree), and $L$ for the length of the interpretation path found. Two measurements were made as suggested in [31]: $P = L/E$ called penetrance to measure how "elongated" or "bushy" the search is, and $B$ called effective branching factor to measure how "focused" the search is toward the goal. In addition, an additional measure $D = L/E$, called directedness, has been computed as an alternative measure similar to $P$.

Fig. 5 summarizes the evaluation of the program’s heuristic power. It includes the statistics from 15 tested examples. $P$ as well as $D$ decreases as $L$ or $T$ increases, which is typical for heuristic search. Because of the nature of the measure, $D$ is coarser than $P$. $B$ stabilizes approximately at 1.3 and indicates that the search is reasonably focused. In this empirical study, $P$ ranges from 0.23 to 0.54 where $L$ ranges from 6 to 9. This result is not inconsistent with the analytical study given in [31, Fig. 2.13], where, for $B = 1.3$, $P$ ranges from 0.21 to 0.37 for the same range of $L$. Special attention is given to the nonsequitur case. It shows that the search is still focused, though the search is given up at the upper bound, which has been set a priori as 15.

VI. CONCLUSION AND FUTURE WORK

Designing a dialogue system to process ad hoc user responses to its own question is a recognizably difficult problem. Most of the existing dialogue systems have relied on menu-driven or command-driven approaches as a quick and easy solution. Consequently, these approaches do not establish a true sense of the two-way communication between man and machine.

In this paper, a model has been proposed that allows the user to respond to system-initiated questions in an ad hoc manner. This model uses the system’s expectation of the potential answers as the focus for interpreting the user-response. The interpretation process uses the domain knowledge and, given the user-response, determines a specific expectation–response relationship type. The interpretation process has been modeled as a search problem, and heuristics have been suggested to reduce the search space.

Determining expectation–response relationships constitutes only the initial step. Further work needs to be done to investigate how expectation–response relationship determination can be tied into the overall dialogue management facilities. For example, it needs to be determined how an ambiguous match would affect the system’s formulation of a disambiguation counterresponse; how an indirect match would dictate the way the system accepts the user response where the system’s conclusion itself could have been based on plausible reasoning; and whether the system should respond cooperatively when mismatch occurs. Overall, how each expectation–response relationship affects the system’s counter response generation merits further studies.

ACKNOWLEDGMENT

The author would like to acknowledge M. Anderson for his implementation of the prototype system. The author appreciates P. Kot for her editorial corrections and the anonymous referees for their helpful comments and suggestions.

REFERENCES

D.-G. Shin (M'92) received the B.S.E. degree in electronics engineering from Han Yang University, Seoul, Republic of Korea, in 1979, and the M.S.E. and Ph.D. degrees in computer science and engineering from the University of Michigan, Ann Arbor, in 1981 and 1985, respectively.

Currently, he is an Associate Professor in the Department of Computer Science and Engineering, University of Connecticut, Storrs. Before joining the University of Connecticut, he taught briefly at Korea Advanced Institute of Science and Technology and Han Yang University. His current research interests include intelligent database systems, semantic data modeling, user interface design, and discourse planning.

Dr. Shin is a member of the Association for Computing Machinery and the IEEE Computer Society.