Developing an Evaluation Methodology for Spoken Language Systems

Madeleine Bates, Sean Boisen and John Makhoul

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

Abstract

There has been a long-standing methodology for evaluating work in speech recognition (SR), but until recently no community-wide methodology existed for either natural language (NL) researchers or speech understanding (SU) researchers for evaluating the systems they developed.

Recently considerable progress has been made by a number of groups involved in the DARPA Spoken Language Systems (SLS) program to agree on a methodology for comparative evaluation of SLS systems, and that methodology is being used in practice for the first time.

This paper gives an overview of the process that was followed in creating a meaningful evaluation mechanism, describes the current mechanism, and presents some directions for future development.

1. A Brief History

The work reported in this paper began in 1988, when people working on the DARPA Spoken Language Systems program and others working on other aspects of natural language processing met at a workshop on NL evaluation [1]. At that meeting, considerable time was devoted to clarifying the issues of black-box evaluation and glass-box evaluation.

Since that meeting, the SLS community formed a committee on evaluation, chaired by Dave Pallett of NIST. The charge of this committee was to develop a methodology for data collection, training data dissemination, and testing for SLS systems under development (see [2] and [3]). The emphasis of the committee's work has been on automatic evaluation of queries to an air travel information system.

The first community-wide evaluation using the first version of methodology developed by this committee took place in June, 1990. It is reported in [4].

2. The Issues

Why are systems for NL understanding, or speech understanding , more difficult to evaluate than SR systems? The key difference is that that the output of speech recognition is easy to specify -- it is a character string containing the words that were spoken as input to the system - and it is trivially easy to determine the "right" answer and to compare it to the output of a particular SR system. Each of these steps,

1. specifying the form that output should take,
2. determining the right output for particular input, and
3. comparing the right answer to the output of a particular system,

is very problematic for NL and SU systems.

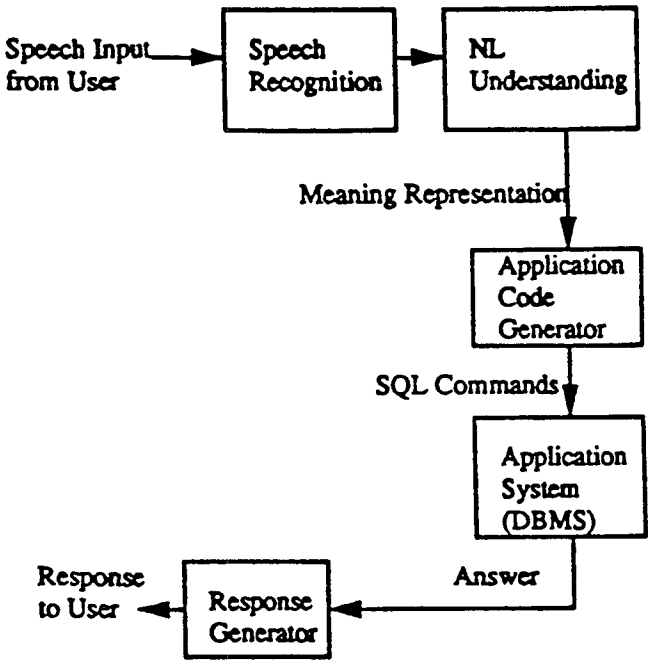

Figure 1: A Typical Speech Understanding System

3. The Goal

The goal of the work was to produce a well-defined, meaningful evaluation methodology (implemented using an automatic evaluation system) which will both permit meaningful comparisons between different systems and also allow us to track the improvement in a single NL system over time. The systems are assumed to be front ends to an interactive application (database inquiry) in a particular domain (ATIS - an air travel information system).

The intent is to evaluate specifically *NL understanding* capabilities, not other aspects of a system, such as the user interface, or the utility (or speed) of performing a particular task with a system that includes a NL component.

4. The Evaluation Framework

The methodology that was developed is very similar in style to that which has been used for speech recognition systems for several years. It is:

1. Collect a set of data as large as feasible, under conditions as realistic as possible.
2. Reserve some of that data as a test set, and distribute the rest as a training set.
3. Develop answers for the items in the test set, and an automatic comparison program to compare those "right" answers with the answers produced by various systems.
4. Send the test set to the sites, where they will be processed unseen and without modifications to the system. The answers are then returned and run through the evaluation procedure, and the results reported.

Figure 2 illustrates the relationship between an SLS system and the evaluation system.

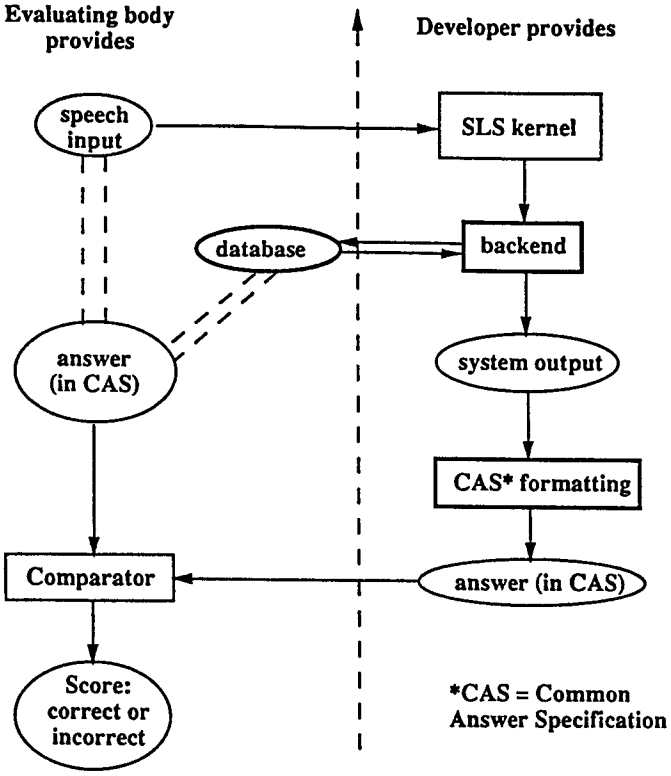

Figure 2: The Evaluation Framework

4.1 Collecting Data

A method of data collection called "Wizard scenarios" was used to collect raw data (speech and transcribed text). This system is described in [5]. It resulted in the collection of a number of human-machine dialogues. This data exhibits some interesting characteristics.

Because much of the data in the database is represented in encoded form (e.g., "L" for "Lunch", and "F" for "First Class"), and because the names of the database fields (which show up as column headers in the answers shown to the users) are often abbreviated or cryptic, many of the users' questions centered on finding out the meaning of some code which was printed as part of the answer to a previous question.[1] This is a characteristic of this particular database, and is not necessarily shared by other databases or domains.

The amount of data shown to the user influences the range of subsequent queries. For example, when the user asks for a list of flights and the answer includes not just the flight numbers but also the departarture and arrival times and locations, the meal service, and so on, there is never any need for a follow-up question like "When does flight AA123 get in?" or "Is lunch served on that flight?".

The language obtained in Wizard scenarios is very strongly influenced by the particular task, the domain and database being used, and the amount and form of data returned to the user. Therefore, restricting the training and test sets to data collected in Wizard scenarios in the ATIS domain means that the language does not exhibit very many examples of some phenomena (such as quantification, negation, and complex conjunction) which are known to appear frequently in database interfaces to other domains.

4.2 Classifying Data

One of the first things to become clear was that not all of the collected data was suitable as test data, and thus it was desirable that the training data be marked to indicate which queries one might reasonably expect to find in the test set.

The notion emerged of having a number of classes of data, each more inclusive than the last, so that we could begin with a core (Class A) which was clearly definable and possible to evaluate automatically, and, as we came to understand the evaluation process better, which could be extended to other types of queries (Classes B, C, etc.).

Several possible classification systems were presented and discussed in great detail. The one which has been agreed on at this time, Class A, is defined in Appendix A; Classes B, C, etc. have not yet been defined.

4.3 Agreeing on Meaning

How do you know whether a system has given the right answer to a question like "List the mid-day flights from Boston to Dallas", "Which non-stop flights from Boston to Dallas serve meals?", or "What's the fare on AA 167?"

It is necessary, for cross-site comparisons, to agree on the meaning of terms such as "mid-day", "meals" (e.g., does that

1For example, 23 of the 90 queries in the June 1990 test set were requests for the meaning of an abbreviation or code.

mean any food at all, or does it include full meals but exclude snacks), and "the fare of a flight" (because in this database, flights don't have simple fares, but a flight together with a class determine the fare).

The process of defining precisely what is meant by many words and phrases is still not complete, but Appendix B lists the current points of agreement. Without this agreement, many systems would produce very different answers for the same questions, all of them equally right according to the systems' own definitions of the terms, but not amenable to automatic evaluation.

4.4 Developing Canonical Answers

It is not enough to agree on meaning, it is also necessary to have a common understanding of what is to be produced as the answer, or part of the answer, to a question.

For example, if a user asks "What is the departure time of the earliest flight from San Francisco to Atlanta?", one system might reply with a single time and another might reply with that time plus additional columns containing the carrier and flight number, a third system might also include the arrival time and the origin and destination airports. None of these answers could be said to be wrong, although one might argue about the advantages and disadvantages of terseness and verbosity.

It was agreed that, for the sake of automatic evaluation, a canonical answer (the minimum "right" answer) should be developed for each Class A query in the training set, and that the canonical answer should be that answer retrieved by a canonical SQL expression. That is, the right answer was defined by the expression which produces the answer from the database, as well as the answer retrieved. This ensures A) that it is possible to retrieve the canonical answer via SQL, B) that even if the answer is empty or otherwise limited in content, it is possible for system developers to understand what was expected by looking at the SQL, and C) the canonical answer contains the least amount of information needed to determine that the system produced the right answer.

Thus it was agreed that for identifying a "flight" the unique flight id would be used, not the carrier and flight number (since there may be several different flights called AA 123 connecting different cities with different departure times and other characteristics).

What should be produced for an answer is determined both by domain-independent linguistic principles [2] and domain-specific stipulation (Appendix B). The language used to express the answers is defined in Appendix C.

4.5 Developing Comparators

A final necessary component is, of course, a program to compare the canonical answers to those produced by various systems. Two programs were constructed to do this, one written in Common Lisp by BBN and one written in C by NIST. Their functionality is substantially similar; anyone interested in obtaining the code for these comparators should contact Bill Fisher at NIST or Sean Boisen at BBN.

The task of answer comparison is complicated substantially by the fact that the canonical answer is intentionally minimal, but the answer supplied by a system may contain extra information. Some intelligence is needed to determine when two answer match (i.e. simple identity tests won't work).

4.6 Choosing a Test Set

For the first evaluation, a test set of 90 Class A queries was chosen from dialogues collected by 4 subjects who were not represented in the training set. We believe that makes the test set harder than necessary, since it is clear that there is not enough training data to illustrate many of the linguistic forms used in this domain, and there is also a strong indication that new users tend to stick with a particular way of asking questions.

4.7 Presenting Results

Expressing results can be almost as complicated as obtaining them. Originally it was thought that a simple "X percent correct" measure would be sufficient, however it became clear that not all systems could answer all questions, and that there was a significant difference between giving a wrong answer and giving no answer at all, so the results were presented as: Number right, Number wrong, Number not answered. How harshly systems should be judged for giving wrong answers was not determined.

5. Strengths of this Methodology

It forces advance agreement on the meaning of critical terms and on at least minimal information to be included in the answer.

It is objective, to the extent that a method for selecting testable queries can be defined, and to the extent that the agreements mentioned above can be reached.

It requires less human effort (primarily in the creating of canonical examples and answers) than non-automatic, more subjective evaluation.. It is thus better suited to large test sets.

Flexible comparison means system developers need not develop a completely separate system for evaluation purposes. With only minor formatting changes, substantially the same system can be used for other purposes.

It can be easily extended, as discussed in section 7 below.

6. Weaknesses of this Methodology

It does not distinguish between merely acceptable answers and very good answers (although the comparators could be made to take this into account if multiple canonical answers with associated acceptability levels could be provided).

It does not distinguish between some cases, and may thus give undue credit to a system that "over answers". For example, if the system prints out the carrier, flight number, arrival and departure times and locations, and meal service every time it is asked about a flight, then the answers to "What is the arrival time of AA 123", "What is the destination of AA 123", "What meal is served on AA 123" and "List flight AA 123" could all produce exactly the same answer and be scored correct on all of them, since the canonical answers would be a subset of the information printed (note that it still must correctly distinguish flight AA 123 from other flights).

It cannot tell if a system gets the right answer for the wrong reason. This is an inevitable problem with "black-box" evaluation, but it can be mitigated by use of larger test sets.

It does not adequately measure the handling of some phenomena, such as extended dialogues.

7. Suggestions for The Future

Our experience thus far has shown that the methodology of developing well-defined test set criteria, combined with automatic evaluation of canonical answers, is a useful one.

7.1 Challenge Training Set

One of the strongest needs of the SLS community at this time is more training data. Instead of a few hundred training queries, several thousand are needed. One way of obtaining more data is simply to continue to run Wizard scenarios to collect them, but this process is rather slow, and tends to yield numerous examples of very similar queries.

We suggest that a separate training set, called the Challenge Set be created. To form this set, each of the five system-developing sites would create, using any means they wish, 500 Class A queries, together with canonical SQL and answers for them. Each site would be encouraged to include queries that show the scope of their system and that create a challenge for other sites to match. Every site would then be required to report on the results of running their system on the challenge set. This use of "crucial examples" is similar to more traditional linguistic methods.

7.2 Beyond Class A

7.2.1 Context

The existing Class A definition excludes all sentences whose interpretation requires context outside the sentence itself, i.e. "Which of those flights are non-stop?". The only obstacle to including such sentences is agreement on what discourse phenomenon to allow, what the meanings in context should be, and when the context should be reset (because sentences have been excluded for other reasons). The existing evaluation framework naturally extends to such cases, assuming the system can produce answers without user interaction.

A proposal has been made [6] to standardize output displays in an attempt to reset context for the evaluation of discourse. This would make it possible to evaluate queries containing references that are display-specific, but not many queries in the training data are of this type, and we believe that there are simpler ways of evaluating common discourse phenomena.

7.2.2 Ambiguity

A simple extension to the language for expressing answers would allow more than one answer to be returned for a query. At a minimum, this could be used to give several alternatives: an answer matching any alternative would then be scored as correct. For example, the answer to the query "What is the distance from the San Francisco airport to downtown" could be either the distance to San Francisco or the distances to San Francisco and Oakland (since both of those cities are served by the San Francisco airport). A more sophisticated approach would be to assign different weights to these alternatives, so systems obtaining the preferred reading would score the highest.

References

1. Palmer, M., T. Finin and S. Walter, *Report on the Workshop on the Evaluation of Natural Language Processing Systems*, unpublished report of a RADC sponsored workshop, 1988.

2. Boisen, Sean , Lance A. Ramshaw, Damaris Ayuso, and Madeleine Bates, *A Proposal for SLS Evaluation*, in Proceedings of the DARPA Speech and Natural Language Workshop, October 1989.

3. Ramshaw, Lance A and Sean Boisen, *An SLS Answer Comparator*. SLS Note No. 7, BBN Systems and Technologies Corporation, Cambridge, MA, May 25, 1990.

4. Pallett, David S., et al, *DARPA ATIS Test Results June 1990*, this volume

5. Hemphill, Charles, *TI Implementation of Corpus Collection*, this volume.

6. Hirschman, Lynette, et al, *Beyond Class A: A Proposal for Automatic Evaluation of Discourse*, this volume.

7. Pallet, David S., William M. Fisher, Jonathan G. Fiscus, *Tools for the Analysis of Benchmark Speech Recognition Tests*, Proceedings of ICASSP 1990, p. 97.

Appendix A: The Current[2] Definition of "Class A"

Class A queries will be identified by exception. Class A queries will be those that are none of the following:

1. context dependent
2. vague, ambiguous, disambiguated only by context, or otherwise failing to yield a single canonical db answer
3. grossly ill-formed
4. other unanswerable queries
5. queries from a noncooperative subject.

These exclusionary categories are described below.

A.1. Context dependent

There seem to be two broad subcategories here:

a. queries containing explicit reference to a preceding answer or question, such as "What classes of service are available on those flights?"

b. queries whose scope is implicitly assumed to be limited by a preceding answer or question, such as "Which flights go to Dallas?" in a context that limits attention to some particular set of flights to Washington DC.

It is noted that some queries in the second subcategory could, is isolation, also receive a reasonable context-independent interpretation. For example, in context, "Please list an interpretation of the classes," is likely to mean the classes displayed in the preceding answer, and thus is context dependent. It also could have a reasonable use referring to all classes. Such queries will be specially marked in the process of selecting class A queries.

A.2. Vague, ambiguous, disambiguated only by context, or otherwise failing to yield a single canonical database answer.

Some of the particular cases noted so far include:

a. attachment ambiguities. These will be excluded ONLY if it is not possible for an ordinary person to pick the preferred reading (without resort to context).

b. "What does X mean?" These are out, unless X is an abbreviation code that has a table that expands the code into a descriptive word, phrase, or set of attributes. The query is not acceptable, however, if X is a code that has more than one possible meaning according to what field it appears in, unless there is disambiguating context WITHIN the query, such as "What does fare code X mean?"

c. "Give me information about X." These queries could be allowed, if someone will produce a table of allowable instances of X together with what information should be provided. Pending that happening, these queries are out.

A.3. Grossly ill-formed queries

As long as the query is interpretable, only utterances that appear not to be attempts to speak normal conversational English will be excluded. For example, we should exclude attempts to speak some imagined form of "computerese" rather than normal English: "Origin Dallas, destination Boston, list flights."

A.4. Other unanswerable queries

a. queries not given a database answer by the wizard. This may include some queries that pass all our tests, but if the wizard did not generate a DB query, then we don't have anything to evaluate on.

b. utterances that cannot be interpreted as queries, or that are incoherent.

c. queries that request information not in the database.

d. queries that refer to the way that information is presented.

e. "meta queries" about system capabilities or structure or limits of the database.

A.5. Queries from a noncooperative subject

Utterances that are clearly designed to try to break the system should be excluded: "Given that city A is Oakland and city B is Fort Worth show me all flights from A to B."

A.6. Additional Comments

Minor syntactic or semantic ill-formedness -- if the query is interpretable, it will be accepted, unless it is so ill-formed that it is clear that it is not intended to be normal conversational English.

Presupposition failures -- all presuppositions about the number of answers (either existence or uniqueness) will be ignored. These are the only types of presupposition failures noted to date. Any other types of presupposition failure that make the query truly unanswerable will presumably result in the wizard being unable to generate a database query, and will be ruled out on those grounds.

Multi-sentence utterances -- These will not automatically be ruled out. The examples cited so far are clearly interpretable as expressing multiple constraints that can be combined into a single query.

---

[2] As distributed by Robert Moore of SRI on May 10, 1990.

Appendix B: Current Definitions of Key Concepts in the ATIS Domain

B.1 Basics

A large class of tables in the database have entries that can be taken as defining things that can be asked for in a query. In the answer, each of these things will be identified by giving a value of the primary key of its table. These tables are:

Table Name	English Term(s)	Primary Key
aircraft	aircraft, equipment	aircraft_code
airline	airline	airline_code
airport	airport	airport_code
city	city	city_code
compound_class	service classes	fare_class
day	names of the days	day_code
fare	fare	fare_code
flight	flight	flight_code
food_service	meals	meal_code
ground_service	ground transportation	city_code, airport_code, transport_code
month	months	month_number
restriction	restrictions	restrict_code
state	names of states	state_code
time_zone	time zones	time_zone_code
transport	transport code	transport_code

B.2 Special meanings

In this arena, certain English expressions have special meanings, particularly in terms of the database distributed by TI in the spring of 1990. Here are the ones we have agreed on: (In the following, "A.B" refers to field B of table A.)

B.2.1. Flights.

A flight "between X and Y" means "from X to Y".

In an expression of the form "flight number N", where N is a number, N will always be interpreted as referring to the flight number (flight.flight_number). "Flight code N" will unambiguously refer to flight.flight_code. "Flight N" will refer to flight.flight_number if N is in the range 0 <= N <= 9999 but to flight.flight_code if N >= 100000.

A "one-way" flight is a flight with a fare whose one-way cost is non-empty.

Principle: if an attribute "X" of a fare, such as "one-way" or "coach", is used as a modifier of a flight, it will be interpreted as "a flight with an X fare".

B.2.2. Fare (classes).

A "one-way" fare is one with a non-empty one-way cost.

In determining what is the "cheapest fare", one-way fares will be included.

A "coach" fare is one whose compound_class.class_type = "COACH". Similarly, the fare modifiers "first class", "business class", and "thrift class" refer to values of the compound_class.class_type field.

A reference to ranking of fares, e.g. "fares that are Y class or better", will be interpreted as a reference to the rank of the associated base fare (class_of_service.rank).

A "discounted fare" is one whose compound_class.discounted = "YES".

An "excursion fare" is one with a restriction code (fare.restrict_code) that contains the string "AP", "EX", or "VU".

A "family fare" is the same thing as an "excursion fare".

A "special fare" is one with a non-null restriction code (fare.restrict_code).

B.2.3. Time.

The normal answer to otherwise unmodified "when" queries will be a time of day, not a date or a duration.

The answer to queries like "On what days does flight X fly" will be a list of day.day_code fields, not a flight_days string.

B.2.4. Units.

All units will be the same as those implicit in the database (e.g. feet for aircraft.wing_span, but miles for aircraft.range_miles, durations in minutes).

B.2.5. Meals.

For purposes of determining flights "with meals/meal service", snacks will count as a meal.

"List the types of meal" should produce one tuple per meal, not a single meal_code string.

B.2.6. "With"-modification clauses.

"Show me all the flights from X to Y with their fares" will require the identification of both flights and their fares (so if there are 2 flights, each with three fares, the answer will have 6 tuples, each with at least the flight_code and fare_code). In general, queries asking for information from two or more separate tables in the database will require the logical union of fields that would identify each table entry separately.

B.2.7. Itinerary.

The "itinerary" of a flight refers to the set of all non-stop legs of that flight. When an "itinerary" is asked for, each leg of the flight will be identified by the origin and destination cities for that leg, e.g. (("BOS" "ATL") ("ATL" "DFW")).

B.2.8. "What kind of".

"What kind of X is Y" queries, where Y is clearly a kind of X, will be interpreted as equivalent to "what does Y mean?", where Y is a primary key value for the table referred to by X (see 10 below).

B.2.9. Classes of Service

References to classes of service will be taken as referring to the contents of the compound_class table (not the class_of_service table).

Queries about (unmodified) "class X", e.g. "What is class X?", will be interpreted as referring to the set of compound_class.fare_class entries for which "X" is the fare_class, not the base_class, e.g. '(("X"))', not '(("XA")("XB"))'.

B.2.10. Requests for meaning.

Requests for the "meaning" of something will only be interpretable if that thing is a code with a canned definition in the database. Here are the things so defined, with the fields containing their decoding:

Table	Key Field	Decoding Field
aircraft	aircraft_code	aircraft_type
airline	airline_code	airline_name
airport	airport_code	airport_name
city	city_code	city_name
code_description	code	description
column_table	heading	column_description
day	day_code	day_name
food_service	meal_code	meal_description
interval	period	begin_time, end_time
month	month_number	month_name
state	state_code	state_name
time_zone	time_zone_code	time_zone_name
transport	transport_code	transport_description

B.2.11. Stops.

A request for a flight's stops will be interpreted as asking for the final stop in addition to intermediate stops.

B.2.12. Yes/no Questions.

Literal yes-or-no questions, that is, queries that in normal discourse would be best answered with "yes" or "no" if taken literally, may be answered by either a boolean value ("YES/TRUE/NO/FALSE") or a relation, expressed as a table. Any non-null relation will be considered equivalent to "YES" or "TRUE" and the null relation will be considered equivalent to "NO" or "FALSE".

B.2.13. Near.

A city and an airport will be considered "near" (or "nearby") each other iff the city is served by the airport, and two cities will be considered "near" (or "nearby") each other iff there is an airport that serves them both.

B.2.14. American.

When it is clear that an airline is being referred to, the term "American" by itself will be taken as unambiguously referring to American Airlines.

B.2.15. Vague Queries.

Vague queries of the form "Give me information about X" or "Describe X" or "What is X", where X refers to a table, will be interpreted as equivalent to "Show me X". When X is an attribute or attribute value, such queries will be interpreted as "What does X mean?".

Appendix C: Current Definition of CAS Answers

The following BNF describes the syntax of the Common Answer Specification (CAS) for the ATIS domain:

answer --> *scalar-value* | *relation* | **NO_ANSWER**
boolean-value --> **yes** | **true** | **no** | **false**
number-value --> *integer* | *real-number*
relation --> (*tuple**)
scalar-value --> *boolean-value* | *number-value* | *string*
tuple --> (*value*$^+$)
value --> *scalar-value* | **NIL**

We assume as primitives the values *integer*, *real-number*, and *string*. Integers and reals are not distinguished, and only non-exponential real numbers are allowed. Strings must always be enclosed in double quotes (e.e., "DFW"), are case-sensitive, and should be upper-case (since strings in the ATIS database are). The special tokens **yes**, **no**, **true**, **false**, **no_answer**, and **nil** are not case-sensitive.

Answer relations must be derived from the existing relations in the database, either by substituting and combining relations or by operations like averaging, summation, etc. NIL as the representation of missing data is allowed as a special case for any value, so a legal answer indicating the costs of ground transportation in Boston would be:

(("L" 5.00) ("R" nil) ("A" nil) ("R" nil))

Empty tuples are not allowed (but empty relations are). All the tuples in a relation must have the same number of values, those values must be of the same respective types (boolean, string, or number), and the types in the answer must be the same as the types in the database (i.e., database values like "1335" cannot be converted from strings to numbers in answer expressions).

Acknowledgement

This work was supported by the Defense Advanced projects Agency and monitored by the Office of Naval Research under Contract No. N00014-89-C-0008.