Natural Language Engineering **1** (1): 000–000. Printed in the United Kingdom

Robust Parsing and Spoken Negotiative Dialogue with Databases

JOHAN BOYE and MATS WIRÉN
TeliaSonera, Sweden

(*Received 12 April 2006*)

Abstract

This paper presents a robust parsing algorithm and semantic formalism for the interpretation of utterances in spoken negotiative dialogue with databases. The algorithm works in two passes: a domain-specific pattern-matching phase and a domain-independent semantic analysis phase. Robustness is achieved by limiting the set of representable utterance types to an empirically motivated subclass which is more expressive than propositional slot–value lists, but much less expressive than first-order logic. Our evaluation shows that in actual practice the vast majority of utterances that occur can be handled, and that the parsing algorithm is highly efficient and accurate.

1 Introduction

The need for spoken dialogue with databases is rapidly increasing as more and more people access information through various voice-activated terminals. A key issue in the design of such dialogue systems is how to achieve the robustness necessary to cope with spoken language input. Typically, existing systems have been built by taking an off-the-shelf speech recognizer and using one of the following two approaches:

1. If the expected variation in spoken input is small, a grammar-based language model for the speech recognizer is used. The vast majority of deployed commercial systems falls into this category. Since the grammar rules can also encode the interpretation of an utterance, the output from the speech recognizer can be a semantic structure of some kind, rather than just a string of words.
2. If the expected variation in spoken input is great (for example, if there are open prompts), it is generally too difficult to hand-code a grammar that covers the input. In those cases, a statistical language model for the speech recognizer should be used instead. However, this requires a subsequent processing step which maps the recognition result to a semantic representation. Early examples of work in this direction are Ward (1989) and Jackson et al. (1991).

Both of these approaches have been highly successful, as judged by the large number of systems deployed. The development time for either grammars or parsers in Approach 1 and 2, respectively, is usually on the order of a few person-weeks, and yet the end result is at least as good as that of large-scale, linguistically-based parsers. What makes this possible is a suitable combination of system-directed dialogue (putting sufficient constraints on what the user can say at a given point) and simplicity of the semantic representation. Generally, the semantic representation is limited to variable-free slot–value structures capturing the propositional contents of words and phrases critical to the domain.

However, these slot–value structures are not well-suited for representing utterances in *negotiative* dialogue, in which several alternative solutions to a problem can be simultaneously discussed and compared (Larsson 2002). Here, in addition to specification utterances such as "I'd like an apartment with a balcony", the system must be able to represent requests for additional information about an already mentioned object, like "Does that apartment have a balcony?", and even references to different objects in the same utterance, like "Is there anything cheaper than that apartment on King's Street?". (See further Section 2 below.) The following question then presents itself: Is it possible to design a robust parser which is applicable to spoken negotiative dialogue, but which retains the robustness and efficiency of the simpler approaches outlined above? We claim that the answer is yes, and will spend the rest of the paper trying to substantiate this claim.

In essence, our solution is as follows: First, the amount of user initiative and variation involved in spoken negotiative dialogue demands that a statistical language model for the speech recognizer be used. Our solution therefore starts off from Approach 2 above, and more specifically has the following characteristics:

1. The semantic representation is more expressive than variable-free slot–value structures, but still much more restricted than first-order logic, and hence also than general-purpose, logic-based formalisms like that of Minimal Recursion Semantics (Copestake et al. 1999), the CLE (Alshawi 1992) or TEAM (Grosz et al. 1985).
2. No particular design of the speech–language interface is presupposed. Our current implementation uses the simplest design possible, namely, picking the top hypothesis from the N-best list.
3. Parsing is deterministic in the sense that only a single analysis is produced for each utterance.
4. Surface-syntactic analysis (the first step of the parsing algorithm) is guided by patterns motivated by the particular domain model, and is hence domain-dependent.
5. Semantic analysis (the second step of the parsing algorithm) is driven by a small set of domain-*independent*, heuristic rules.

The goal of the paper is thus to find an empirically motivated trade-off between robustness and expressiveness in spoken, negotiative database dialogue: By deliberately restricting the expressiveness of the semantic representation formalism — keeping it sufficiently coarse-grained — we make it suitable for robust parsing. At the same time, we keep the representation sufficiently fine-grained to be able to capture the distinctions actually made in spoken, negotiative database dialogue. Because of this trade-off, we are able to retain efficiency both with respect to development effort and execution time.

Our approach has been used successfully in ADAPT (Bell et al. 2001), a multimodal dialogue system in an apartment domain. A forerunner was used in the travel-planning system SmartSpeak (Boye et al. 1999).

The rest of this paper is organized as follows: Section 2 gives an illustration of negotiative dialogue by way of an example. Section 3 describes how information about the domain is specified in our system. Section 4 and 5 constitute the heart of the paper, describing the semantic formalism and robust parsing algorithm, respectively. Section 6 provides the evaluation of the parser. Section 7 discusses the changes that would be needed to encode more complex domains than ADAPT. Finally, Section 8 gives an overview of related work and Section 9 provides the conclusions.

2 Negotiative dialogue

The following piece of dialogue with the ADAPT system gives an illustration of the kind of negotiative dialogue that we are aiming to handle.

1. **User:** Are there any two-room apartments on the South Side that cost less than two million?
2. **System:** Do you have any other requirements?
3. **User:** A balcony would be nice.
4. **System:** There are seven such apartments. They are shown on the map.
5. **User:** I see... this apartment on Horn Street... how much does it cost?
6. **System:** The apartment on Horn Street costs 1,900,000 kronor.
7. **User:** Okay... is there anything cheaper?
8. **System:** The apartment on Blekinge Street costs 1,500,000 kronor.
9. **User:** Eeh... please tell me more about that apartment.

In utterances 1 and 3, the user homes in on the kind of apartment he wants to consider (the *specification* phase). The system briefly presents a number of results (in utterance 4) by indicating their position on the map, after which the user starts asking questions about the individual apartments, comparing their relative merits (utterances 5, 7 and 9). This *negotiation* phase is what distinguishes negotiative dialogue systems from simpler systems.

Thus, as mentioned in Section 1, the system must be able to distinguish specification utterances like "I'd like an apartment with a balcony" from utterances seeking information about a particular apartment, like "Does that apartment have a balcony?". The system must also be able to handle references to different objects in the same utterance, like "Is there anything cheaper than that apartment on King's Street?".

3 Domain models

We assume that from the user's point of view, the purpose of a dialogue as exhibited in Section2 is to retrieve information about a set of interrelated objects, such as apartments, prices and addresses. The set of all such objects in the domain, together with their relations, constitutes the *domain model* of the system. From the system's point of view, the goal is then to translate each user utterance into an expression denoting a subset of the domain model (namely, the subset that the user is asking for), and to respond by either presenting that subset or ask the user to change the constraints in case the subset cannot be readily presented. (Naturally, this is somewhat idealized, as there are meta-utterances, social utterances, etc. that are not translatable to database queries. Still, 96% of the utterances in our ADAPT corpus, briefly described in Section 6.3, correspond to database queries.)

We will assume that each object in the domain model is typed, and to this end we will assume the existence of a set of *type symbols*, e.g. $\mathsf{apartment}$, $\mathsf{integer}$, $\mathsf{street\_name}$, money etc., and a set of *type variables* $\mathsf{t}_1, \mathsf{t}_2, \ldots$ ranging over the set of type symbols. Each type symbol *denotes* a set of objects in an obvious way, e.g. $\mathsf{apartment}$ denotes the set of apartments. Both type symbols and type variables will be written with a $\mathsf{sans\ serif}$ font, to distinguish them from symbols denoting individual objects and variables ranging over individual objects, which will be written using an *italicized* font. The expression b^{t} is taken to mean the assertion "b is of type t".

Objects are either simple, scalar or structured. Objects representable as numbers or strings are simple (such as objects of the type money or $\mathsf{street\_name}$). Scalar objects are sets of objects, for instance $\mathsf{set(apartment)}$. Structured objects have a number of attributes, analogous to C structures or Java reference objects. Typically, structured objects correspond to real-world phenomena on which the user wants information, such as apartments in a real-estate domain, or flights and trains in a travel planning domain. In the domain model used in the ADAPT system, only the $\mathsf{apartment}$ type is structured (but see Section 7 for possible extensions).

We will use the notation $b.a$ to refer to attribute a of a structured object b. For example, an apartment has the attributes *size*, *number_of_rooms*, *price*, *street_name*, *accessories*, etc., with the respective types $\mathsf{square\_meters}$, $\mathsf{integer}$, money, $\mathsf{street\_name}$, $\mathsf{set(accessory)}$, etc. Hence if $b^{\mathsf{apartment}}$ is a true assertion, then so is $(b.size)^{\mathsf{square\_meters}}$.

Thus, a (structured) object o_1 might be related to another (simple, scalar or structured) object o_2 by letting o_2 be the value of an attribute of o_1. For instance, an apartment a is related to "King's Street" by letting $a.street\_name = Kings\_street$. There is a standard transformation from this kind of domain models into relational database schemes (see e.g. Ullman 1988, p. 45), but domain models can also be represented by other types of databases.

For each type, we will assume the existence of a set of variables $x_1, x_2, \ldots$, ranging over the objects of the type. For a variable x, we will take the expression x^{t} to mean "x can assume values of type t".

We will further assume that types are arranged in a subtype hierarchy. The type t_1 is a subtype of t_2 (written as $\mathsf{t}_1 \preceq \mathsf{t}_2$) if x^{t_2} is a true assertion whenever x^{t_1} is a true assertion. For instance, in the ADAPT domain model, $\mathsf{money} \preceq \mathsf{integer}$, since in that domain, x is an integer whenever x is a sum of money.

4 Semantic representation formalism

Utterances may contain explicit or implicit references to other objects than the set of objects sought. For example, when the user says "A balcony would be nice" in utterance 3 of the dialogue fragment of Section 2, the effect is to further restrict the context (the set of apartments) which was obtained after utterance 1. Obviously, an utterance cannot be fully interpreted without taking the context into account. The context-independent interpretation of an utterance is thus a function, mapping a dialogue context (in which the utterance is made) to the final interpretation of the utterance. In our case, a dialogue context is always an object or a set of objects (a subset of the domain model), and the final interpretation denotes the set of objects that are compatible with the constraints imposed by the user. This section describes expressions called "utterance descriptors", which formalize this notion.

4.1 Constraints

Constraints express desired values of variables and attributes, by means of equalities, inequalities and statements expressing membership in (finite) sets. Constraints are built from expressions denoting simple, scalar or complex objects, and variables ranging over such objects. Suppose each of x_1 and x_2 is such an expression or variable, and suppose $x_1^{\mathsf{t}_1}$ and $x_2^{\mathsf{t}_2}$, where either $\mathsf{t}_1 \preceq \mathsf{t}_2$ or $\mathsf{t}_2 \preceq \mathsf{t}_1$. Then the set of constraints are defined as follows:

- $x_1 = x_2$ is a constraint.
- If both t_1 and t_2 are subtypes of $\mathsf{integer}$, then $x_1 < x_2$, $x_1 \leq x_2$, $x_1 \geq x_2$ and $x_1 > x_2$ are constraints.
- If y is of type $set(\mathsf{t}_1)$, then $x_1 \in y$ is a constraint.

The following are all examples of constraints:

- $x.street\_name = King\_street$
- $x.price < 2,000,000$
- $balcony \in x.accessories$
- $x.street\_name \in \{King\_street, Horn\_street\}$

Note that the definition of constraint disallows the relating of values of incompatible types (such as $x.street\_name = 2$).

To define what it means for a variable-free constraint to be *true*, we assume the existence of a function *eval* which evaluates expressions containing dot-notation (e.g. so that $eval(b.price) = 2000000$ if b is an apartment object whose price attribute has the value 2000000). For expressions not containting dots, *eval* is the identity function (e.g. $eval(2000000) = 2000000$).

- $a = b$ is true iff $eval(a)$ and $eval(b)$ are identical.

- $a < b$ is true iff $eval(a)$ is strictly smaller than $eval(b)$. The truth of other kinds of numerical constraints ($a \leq b$, $a > b$, $a \geq b$) is defined analogously.
- $a \in b$ is true iff $eval(a)$ occurs among the members of $eval(b)$.

A conjunction C of constraints is *satisfiable* if there exists a binding σ, mapping the variables in C onto variable-free expressions, such that all constraints in $\sigma(C)$ are true.

4.2 Set descriptors

Set descriptors are expressions denoting subsets of the domain model. They have the form $?x^{\mathsf{t}}\ (P)$, where P is a conjunction of constraints in which the variable x occurs. Such a set descriptor denotes the set of all objects x of type t such that P is a true assertion of x. Thus,

$$?x^{\mathsf{apartment}}\ (x.area = South\_side \wedge\ x.number\_of\_rooms = 2)$$

denotes the set of all apartment objects whose *area* attribute has the value *South_side* and whose *number_of_rooms* attribute has the value 2.

We may also add existentially quantified "place-holder" variables to a set descriptor without changing its semantics. For instance, the set descriptor above is equivalent to:

$$?x^{\mathsf{apartment}}\ \exists y^{\mathsf{integer}}\ (x.area = South\_side\ \wedge\ x.number\_of\_rooms = y\ \wedge\ y = 2)$$

Thus, set descriptors can also have the form $?x^{\mathsf{t}_1}\ \exists y^{\mathsf{t}_2}\ (P)$, where P is a conjunction of constraints in which x and y occur.

4.3 Representing context: Utterance descriptors

As mentioned above, the context-independent interpretation of an utterance is a function, mapping the dialogue context in which the utterance is made to the final interpretation of the utterance. In our case, a dialogue context is always an object or a set of objects (a subset of the domain model), and the final interpretation is a set descriptor, denoting the set of objects that are compatible with the constraints imposed by the user.

Accordingly, the context-independent interpretation of "A balcony would be nice" is taken to be

$$\lambda S^{\mathsf{set(apartment)}}\ \ ?x^{\mathsf{apartment}}\ (balcony \in x.accessories\ \wedge\ x \in S)$$

where S is a parameter that can be bound to a subset of the domain model. Thus the expression above can be paraphrased "I want an apartment from S that has a balcony". The idea is that the ensuing stages of processing within the dialogue interface will infer the set of objects belonging to the context, upon which the functional expression above can be applied to that set, yielding the final answer. In the dialogue example of section 2, S will be bound to the set of apartments obtained after utterance 1.

An utterance may contain more than one implicit reference to the context. For example, "Is there a cheaper apartment?" (utterance 7 of the dialogue fragment of section 2) contains one implicit reference to a set of apartments from which the selection is to be made, and another implicit reference to an apartment with which the comparison is made (i.e. "I want an apartment from S which is cheaper than the apartment y"). Hence the representation is:

$$\lambda y^{\mathsf{apartment}}\ \lambda S^{\mathsf{set(apartment)}}\ ?x^{\mathsf{apartment}}\ (x.\mathit{price} < y.\mathit{price}\ \wedge\ x \in S)$$

The contextual reasoning carried out by the ADAPT system then amounts to applying this expression to suitable arguments. The system employs a straightforward recency principle when searching for individual objects to be used as arguments. In our example, the most recently mentioned apartment is the apartment mentioned in the preceding utterance (no. 6). As for determining the relevant subset of the domain model (the parameter S), the system assumes that the user is referring to the set of apartments currently indicated on the map. In the example above, this is the set of apartments introduced by utterance 4. If no apartments are currently indicated on the map (as at the beginning of the dialogue), the system assumes that the user is referring to the set of all apartments.

The type discipline employed in the ADAPT system effectively rules out contextual interpretations that make no sense in the domain. For instance, consider the following dialogue fragment:

1. **User:** Does the apartment on Horn Street have a balcony?
2. **System:** Yes.
3. **User:** How much does it cost?

In this example, the system interprets the last question as an inquiry of the value of a "price" attribute. In the ADAPT type discipline, objects of type apartment have a "price" attribute, whereas balconies do not. Thus the last mentioned compatible object (having a "price" attribute) would not be the balcony but rather the apartment on Horn Street.

We thus define an *utterance descriptor* as an expression of the form $\lambda X_1 \cdots \lambda X_n\ U$, where X_i is either a set variable or a typed variable x^{t}, and where U is a set descriptor in which the variables of $X_1, \ldots, X_n$ occur free. Thus, an utterance descriptor is a function taking n arguments (representing the context), returning as result a subset of the domain model.

Yet an example is given by the utterance "How much does the apartment on King's Street cost?", which is represented by

$$\lambda y^{\mathsf{apartment}}\ ?x^{\mathsf{money}}\ (y.\mathit{price} = x\ \wedge\ y.\mathit{street\_name} = \mathit{King\_street})$$

To resolve the reference, the ADAPT system would search for the most recently mentioned apartment y which is compatible with the constraint $y.\mathit{street\_name} = \mathit{King\_street}$.

Utterance descriptors can also contain type variables when sufficient type information is lacking. For instance, "What does it cost?" would be represented by

$$\lambda y^{\mathsf{t}}\ ?x^{\mathsf{money}}\ (y.price = x)$$

Here, the target for reference resolution would be the most recently mentioned object y of any type that has a price attribute (otherwise the constraint $y.price = x$ would be non-sensical).

4.4 Minimization and maximization

In many situations one is interested in the set of objects that are minimal or maximal in some regard, for example, the “biggest apartment” or the “cheapest ticket” (this is usually a singleton set). To cater for this, we will further extend the notion of utterance descriptor, and introduce a limited form of universal quantification. For instance, “Which is the cheapest apartment?” would be represented as:

$$\lambda S^{\mathsf{set(apartment)}}\ ?x^{\mathsf{apartment}}\ \mu price(x \in S)$$

which is used as shorthand for

$$\lambda S^{\mathsf{set(apartment)}}\ ?x^{\mathsf{apartment}}\ (x \in S\ \wedge\ \forall y^{\mathsf{apartment}} \in S : x.price \leq y.price)$$

When applied to a context set S, the function above returns an expression denoting the set of apartments in S whose price attribute has the minimal value.

The general form of this construction is $?x^{\mathsf{t}}\ \mu attribute(P)$, where *attribute* is an attribute of x, and P is a conjunction of constraints. For instance, by replacing $x \in S$ by $x \in S\ \wedge\ x.street\_name = King$ in the first expression above, we get an expression denoting the (singleton set of the) least expensive apartment on King’s Street.

The maximization operator is defined analogously to the minimization operator (“$\leq$” is replaced by “$\geq$”; otherwise the definition is the same).

4.5 Expressive power

Many current commercial spoken-dialogue interfaces rely on a system-driven “slot-filling” dialogue strategy (e.g. “System: *Where do you want to go?*, User: *To Stockholm*, System: *What date?*, User: *May seventh*, etc.). In such cases, the user’s utterances can be represented by lists of slot–filler pairs (e.g. $[destination = Stockholm]$ or $[departure\_date = 7May]$).

The expressiveness of the formalism outlined in the preceding sections goes beyond that of variable-free slot–filler lists, motivated by the phenomena seen in Section 2. For instance, “I’d like an apartment with a balcony” is represented by

$$?x^{\mathsf{apartment}}\ (balcony \in x.accessories)$$

whereas “Does that apartment have a balcony?” is represented by

$$\lambda x^{\mathsf{apartment}}\ ?balcony^{\mathsf{accessory}}\ (balcony \in x.accessories)$$

The slot–filler list $[balcony\ \in\ x.accessories]$ fails to make this distinction. Also utterances involving (implicit) references to several objects, like “Is there a cheaper apartment?” are unrepresentable using variable-free slot–filler lists.

On the other hand, our formalism can only represent a subset of possible database search (SQL) commands. For instance, the only universal quantification allowed is the one provided by the maximization and minimization operators. A limited kind of disjunction is provided by the membership relation, otherwise disjunction is disallowed (the utterance “I want an apartment on Horn Street or on King’s Street” can be represented using the constraint $x.street\_name \in \{King\_street, Horn\_street\}$, whereas the utterance “I want an apartment that costs less than two million or has a balcony” cannot be represented). In this sense, the formalism is less expressive than quantifier-free predicate logic.

As previously mentioned, our choice of formalism is motivated by a trade-off between robustness and expressiveness. Section 6.3 discusses to what extent we have found a suitable trade-off.

5 Robust parsing

This section describes a parsing algorithm that maps speech recogniser output to the utterance descriptors just described. The algorithm consists of two phases, pattern matching (Section 5.2) and rewriting (Section 5.3). In the latter phase, heuristic rewrite rules are applied to the result of the first phase. When porting the parser to a new domain, one has to rewrite the pattern matcher, whereas the rewriter can remain unaltered.

5.1 Meta-constraints

The pattern matching rules in the pattern matcher associate a sequence of constraints and meta-constraints to each pattern. The constraints will eventually end up in the body of the final utterance descriptor, while the sole purpose of the meta-constraints is to guide the rewriting phase. The most commonly used meta-constraint has the form $obj(x^{\mathsf{t}})$ stipulating the existence of an object x of type t which the user has referred to, either explicitly or implicitly. For instance, in the ADAPT parser, the pattern ”apartment” would yield

$$obj(x_1^{\mathsf{apartment}})$$

whereas the pattern ”King’s Street” would yield

$$obj(x_2^{\mathsf{apartment}}),\ x_2.street = Kings\_street$$

where x_1 and x_2 are variables. The pattern-matching rule stipulates the existence of the object $x_2^{\mathsf{apartment}}$, since in the ADAPT domain model, streets can only occur in the context of the *street* attribute of the $\mathsf{apartment}$ type. If the domain model would include also another type ($\mathsf{restaurant}$, say) that also has an attribute *street*, the pattern-matching rule could instead be defined to yield:

$$obj(x_2^{\mathsf{t}}),\ x_2.street = Kings\_street$$

$obj(x^{\mathsf{t}})$	An object x of type t has been explicitly or implicitly referred to
$head\_obj(x^{\mathsf{t}})$	Same as above; additionally, the object denoted by x is the object sought
$x \neq y$	x and y denote different objects
$ambiguous(x, \{a_1, a_2, \ldots, a_n\}, default(a_i))$	x is one of the objects $a_1, a_2, \ldots, a_n$. Unless evidence to is the contrary, x should be equal to a_i

Fig. 1. Different kinds of meta-constraints

where t is a type variable.

The different kinds of meta-constraints are summarized in Figure 1. We will illustrate their use in the next section.

5.2 Pattern matching phase

The purpose of the pattern matching phase is to generate a sequence of constraints and meta-constraints based on the syntactic patterns that appear in the input. Pattern matching rules are encoded using a Definite Clause Grammar (see e.g. Sterling and Shapiro 1994, chapter 19). An example showing such rules is given below (in which we adopt the standard logic programming convention that expressions with an initial capital letter are variables).

$\mathsf{apartment\_hints([obj(X^{apartment}), obj(Y^{street\_name}), X.street\_name = Y \mid Tail], Tail)} \rightarrow$
$\quad\mathsf{apartment\_phrase(X),}$
$\quad\mathsf{[on],}$
$\quad\mathsf{street(Y).}$

$\mathsf{apartment\_hints([obj(X^{apartment}) \mid Tail], Tail)} \rightarrow$
$\quad\mathsf{apartment\_phrase(X).}$

$\mathsf{street\_hints([obj(Y^{street\_name}), obj(Z^{apartment}), Z.street\_name = Y \mid Tail], Tail)} \rightarrow$
$\quad\mathsf{street(Y).}$

$\mathsf{street(king\_street)} \rightarrow$
$\quad\mathsf{[kings, street].}$

The algorithm first tries to match an initial segment of the input with the right hand side of such a rule. If a match is possible, the semantic constraints on the left hand side are appended to the result list, the matched input segment is discarded, and the process is repeated with the remaining input. If no match is possible, the first word of the input is discarded, and the process is repeated with the remaining input.

As an example, reconsider the utterance "I'm looking for an apartment on King's Street". If the utterance has been correctly recognized, the first pattern would be triggered, resulting in the constraints:

$$obj(x^{\mathsf{apartment}}),\ obj(kings\_street^{\mathsf{street\_name}}),\ x.street\_name = kings\_street$$

However, the utterance might have been misrecognized as "I'm looking for an apartment of King's Street", or the user might have hesitated ("I'm looking for an apartment on ehh King's Street"). In neither case the first rule would trigger; hence, the pattern matching phase would fall back to the two non-contiguous fragments "apartment" and "King's Street", yielding the constraints:

$$\begin{aligned}&obj(x^{\mathsf{apartment}}),\ obj(kings\_street^{\mathsf{street\_name}}),\ obj(z^{\mathsf{apartment}}),\\&\qquad z.street\_name = kings\_street\end{aligned}$$

That is, the link between the apartment and "King's Street" is missed (but will be recovered in the second phase of the parsing algorithm, presented in section 5.3).

As can be seen from the example, longer syntactic patterns are likely to convey more precise information, but on the other hand they are more brittle. Therefore longer patterns are applied before shorter patterns, so the parser can use structure whenever present in the input, and degrade gracefully on noisy input.

5.3 Rewriting phase

In the rewriting phase, a number of heuristic rewrite rules are applied (in a fixed order) to the sequence of constraints and meta-constraints, resulting in an utterance descriptor (after removing all meta-constraints). The most important rules are:

- Unify as many objects as possible.
- Resolve semantic ambiguities.
- Identify the object sought.
- Identify contextual references.

5.3.1 Object unification

The first rewriting rule to be applied is the object unification rule. Suppose pattern matching has resulted in:

$$obj(x_1^{\mathsf{apartment}}),\ obj(x_2^{\mathsf{t}}),\ x_2.street = Kings\_street$$

Then checking whether the two objects x_1 and x_2 are unifiable amounts to checking whether their types are compatible (which they are, as t is a type variable), and checking whether an $\mathsf{apartment}$ has an attribute *street* (which is true). Therefore the result after applying the rule is

$$obj(x_1^{\mathsf{apartment}}),\ x_1.street = Kings\_street$$

In the list

$$obj(y^{\mathsf{money}}),\ obj(x^{\mathsf{apartment}}),\ x.price = y,\ obj(2000000^{\mathsf{integer}})$$

the expressions y and 2000000 are unifiable, since y is a variable of type money, and 2000000 is of type integer, a supertype of money. The resulting expression is 2000000^{money}. Application of the rewrite rule thus yields:

$$obj(2000000^{\mathsf{money}}),\ obj(x^{\mathsf{apartment}}),\ x.price = 2000000$$

Unification of two objects can be prevented by an explicit disequality constraint. So, the following list would be unmodified by the the object unification rewrite rule:

$$obj(x^{\mathsf{apartment}}),\ obj(y^{\mathsf{apartment}}),\ x \neq y$$

An example where such a disequality constraint is useful can be found in section 5.4 below.

5.3.2 Resolution of semantic ambiguities

Some pattern matching rules give rise to semantic ambiguities. For instance, in the ADAPT parser, the pattern "costs" yields the following constraints and meta-constraints:

$$\begin{aligned} &obj(x^{\mathsf{apartment}}),\ obj(y^{\mathsf{money}}),\ x.z = y, \\ &\quad ambiguous(z, \{price, monthly\_fee\}, default(price)) \end{aligned}$$

The variable z represents the ambiguity in this case; z can either be the "price" attribute or the "monthly fee" attribute of the apartment x. The ambiguity resolution rewrite rule tries to resolve this ambiguity by unifying the equation $x.z = y$ with another equation in the list (if possible). Otherwise, the default value for z ("price") in this case is chosen. See Section 5.4 for an example of the application of this rule.

5.3.3 Identification of the object sought

This rule identifies the object the user wants information about, using the following heuristics.

1. Firstly, the object sought is assumed to be the leftmost *head_obj* in the sequence of constraints and meta-constraints, if such an expression appears at all in the sequence. For instance, in the ADAPT parser, the pattern "How much" yields the following list of constraints:

$$head\_obj(y^{\mathsf{money}}),\ obj(x^{\mathsf{apartment}}),\ x.price = y$$

   which would be rewritten into

$$?y^{\mathsf{money}}\ (obj(x^{\mathsf{apartment}}),\ x.price = y)$$

2. If no *head_obj* appears in the sequence, the next step is to look for a variable which is *not* of a structured type (see Section 3 for the meaning of "structured" objects), and which appears only once in the constraints. For instance, in the sequence

   $$obj(x^{\mathsf{apartment}}),\ obj(y^{\mathsf{money}}),\ x.price = y$$

   the object sought would be assumed to be y, as y appears only once among the constraints, in the equation $x.price = y$. (Although y also occurs in a meta-constraint $obj(y^{\mathsf{money}})$, that occurrence does not count for the purpose of this rewriting rule.) This case actually adds to the robustness of the parsing algorithm (see the example in Section 5.4).
3. Finally, if none of the two cases above apply, the object sought is assumed to be the leftmost *obj* in the sequence. So, if the sequence is

   $$obj(x^{\mathsf{apartment}}),\ obj(2000000^{\mathsf{money}}),\ x.price = y,\ y < 2000000$$

   the object sought would be assumed to be x, and the list would be rewritten into

   $$?x^{\mathsf{apartment}}\ (obj(2000000^{\mathsf{money}}),\ x.price = y,\ y < 2000000)$$

5.3.4 Identification of contextual references

As explained in Section 4, contextual references are represented by means of lambda-bound variables. This rewrite rule searches the list of constraints and meta-constraints for expressions of the form $obj(x^{\mathsf{t}})$ where t is a structured type, in which case x is made into a lambda-bound variable. For example, the expression

$$?y^{\mathsf{money}}\ (obj(x^{\mathsf{apartment}}),\ x.price = y)$$

is transformed into

$$\lambda x^{\mathsf{apartment}}\ ?y^{\mathsf{money}}\ (x.price = y)$$

5.4 Example

We will now use the utterance "I'd like an apartment on Horn Street that is cheaper than the apartment on King's Street" to illustrate both pattern-matching and several rewrite rules in the ADAPT system. First of all, "apartment on Horn Street" yields

$$obj(x_1^{\mathsf{apartment}}), x_1.street = Horn\_street$$

Furthermore, the word "cheaper" yields the sequence

$$\begin{aligned}&head\_obj(x_2^{\mathsf{apartment}}), obj(x_3^{\mathsf{apartment}}), x_2 \neq x_3,\\ &\quad obj(y_2^{\mathsf{money}}), x_2.z = y_2,\\ &\quad obj(y_3^{\mathsf{money}}), x_3.z = y_3, y_2 < y_3,\end{aligned}$$

$$\mathit{ambiguous}(z, \{\mathit{price}, \mathit{monthly\_fee}\}, \mathit{default}(\mathit{price}))$$

which is appended to the first sequence. Finally, the phrase "apartment on King's Street" causes the sequence

$$\mathit{obj}(x_4^{\textsf{apartment}}), x_4.\mathit{street} = \mathit{Kings\_street}$$

to be appended.

In the rewriting phase, objects are first unified in a left-to-right order. Thus x_1 and x_2 are unified, but the meta-constraint $x_2 \neq x_3$ prevents unification of x_2 and x_3. Instead, x_3 and x_4 are unified. The ambiguity of z is then resolved using the default (binding z to *price*). Next, the variable x_2 is identified as the main object, and the implicit contextual reference argument S is added:

$$\begin{aligned}&\lambda S^{\textsf{set(apartment)}}\ ?x_2{}^{\textsf{apartment}}\ (x_2.\mathit{street} = \mathit{Horn\_street},\\ &\quad x_2 \in S, \mathit{obj}(y_2^{\textsf{money}}), x_2.\mathit{price} = y_2,\\ &\quad x_2 \neq x_3, x_3.\mathit{street} = \mathit{Kings\_street},\\ &\quad \mathit{obj}(y_3^{\textsf{money}}), x_3.\mathit{price} = y_3, y_2 < y_3)\end{aligned}$$

Finally, the variable x_3 is identified as a contextual reference. After removing meta-constraints, this results in:

$$\begin{aligned}&\lambda x_3{}^{\textsf{apartment}}\ \lambda S^{\textsf{set(apartment)}}\ ?x_2{}^{\textsf{apartment}}\\ &\quad (x_2.\mathit{street} = \mathit{Horn\_street}\ \wedge\ x_2 \in S\\ &\quad \wedge\ x_3.\mathit{street} = \mathit{Kings\_street}\ \wedge\ x_2.\mathit{price} < x_3.\mathit{price})\end{aligned}$$

Because of the algorithm's ability to infer relationships between pieces of information gathered from different, possibly isolated, patterns, it is very robust in the presence of speech recognition errors. In the example above, the algorithm really only relies on the fragments "Horn Street", "cheaper" and "King's Street".

6 Evaluation

To evaluate the parser, we used a corpus of utterances independently collected by Edlund and Nordstrand (2002) with the ADAPT system. The corpus was collected using 24 subjects who were given the task of finding an apartment in central Stockholm that they were potentially interested in acquiring. A majority of the subjects said that they had used standard apartment search tools on the Web, but none of them had used or seen ADAPT before. Also, none of them had professional knowledge of speech technology.

6.1 Basic results

The evaluation is based on running the parser on a set of 300 randomly selected, unseen utterances from the Edlund–Nordstrand corpus. To obtain a reference with

Table 1. *Parsing and recognition accuracy*

	Spoken input	Recognized input	Transcribed input
Speech recognizer			
Sentence accuracy	39.7%		
Word accuracy	65.6%		
Parser			
Semantic accuracy		66.6%	93.0%
Concept accuracy		83.7%	96.7%

which to assess the output of the parser, we manually constructed the correct analysis for each of the 300 utterances. The results of the evaluation are shown in Table 1.

The top half of the table shows the accuracy of the speech recognizer. Thus, 39.7% of the utterances were recognized perfectly, and the overall word accuracy was 65.6% (that is, the word error rate was 34.4%). The bottom half of the table shows the accuracy of the parser. By “semantic accuracy” we mean the proportion of utterances for which the output of the parser completely matches the correct analysis. (It is thus the semantic analogue of sentence accuracy.) “Concept accuracy” is defined as

$$CA = 100\ (1 - \frac{u_S + u_I + u_D}{u})\ \%$$

where u_S, u_I and u_D is the number of semantic units that are substituted, inserted and deleted, respectively, and where u is the total number of semantic units in the reference. A semantic unit is here defined as a constraint in the body of an utterance descriptor, a λ-bound variable or a ?-bound variable in the head of an utterance descriptor. (Concept accuracy is thus the semantic analogue of word accuracy; compare Boros et al. 1996.)

The middle column of the table shows how the parser performs on actual, recognized input. As can be seen, the parser produces completely correct results for 66.6% of the input sentences, in spite of the fact that only 39.7% of them were perfectly recognized. Similarly, the concept accuracy of the output is 83.7%, in spite of the word accuracy being only 65.6%. Finally, the rightmost column shows how the parser performs on transcribed input (perfect speech recognition).

A similar evaluation of a previous version of the system was reported in Boye and Wirén (2003), based on another (distinct) set of 300 utterances from the Edlund–Nordstrand corpus. The result of this evaluation was used for guiding several improvements to the pattern-matching rules as well as the addition of one heuristic rewrite rule (namely, the one described in Section 5.3.3). However, no changes to the algorithm as such or to any other parts of the system were needed.

6.2 Robustness of the parser

The figures in Table 1 show that the parser is robust in the sense that it outputs utterance descriptors with a significantly higher degree of accuracy than the strings output by the speech recognizer, in effect reconstructing meaning from noisy input. To arrive at a more in-depth analysis of these overall figures, we can compare the sentence accuracy of the speech recognizer with the semantic accuracy of the parser. By looking at the analysis of each recognized utterance, we obtain the following four cases:

1. *Wrong recognition, wrong analysis*: This corresponds to lack of robustness in the sense that the parser failed to (completely) reconstruct the correct meaning from the incorrect recognition.
2. *Wrong recognition, correct analysis*: This shows instances of successful robust analysis.
3. *Correct recognition, wrong analysis*: This corresponds to lack of coverage of the parser or lack of expressiveness of the semantic representation.
4. *Correct recognition, correct analysis*: This shows basic coverage.

The distribution of the 300 parsed utterances and accompanying analyses with respect to the four cases is as follows:

1. *Wrong recognition, wrong analysis*: 33.1%.
2. *Wrong recognition, correct analysis*: 27.1%.
3. *Correct recognition, wrong analysis*: 0.3%.
4. *Correct recognition, correct analysis*: 39.4%.

Clearly, Case 1 is most interesting if we want to understand why the parser sometimes fails, and how robustness can be improved. A further analysis of the recognized utterances in this case reveals that in the vast majority of instances, the parser had no possibility of producing the right analysis due to some semantically important word having been misrecognized. Altogether, there are only eight instances of Case 1 (corresponding to 8% of Case 1) in which all the necessary words were present in the input. In all of these instances, the reason for the incorrect analysis was a coverage leak (and not a limitation of the formalism). Similarly, a coverage leak is the reason for the single faulty analysis that constitutes Case 3.

It should be pointed out, however, that in most of the intances in which a semantically important word had been misrecognized, the parser still produced a partially correct result. A typical example is the following:

Spoken input:	"Hornsgatan tjugonio" ("Twenty-nine Horn Street")
Recognized input:	"Hornsgatan tjugo nu ja" ("Twenty now yes Horn Street")

Here, the parser produced an analysis where the street number is incorrect, but the street name is correct.

Case 2 is most interesting if we want to understand how the parser actually manages to achieve robustness. Most of the instances of Case 2 are utterances where the recognizer had misrecognized or erroneously inserted words that do not belong to any semantically relevant pattern. A typical example is the following:

Spoken input:	"Finns det någon lägenhet i Gamla stan?" ("Is there any apartment in the Old Town?")
Recognized input:	"Finns det det var lägenhet den Gamla stan mmm mmm?" ("Is there there was apartment it Old Town mmm mmm?")

There are some exceptions, however, like:

Spoken input:	"Finns det en tvåa på Kungsholmen?" ("Is there a two-room apartment on Kungsholmen?")
Recognized input:	"Finns det en tvåa på Kungsholmen eller fem?" ("Is there a two-room apartment on Kungsholmen or five?")

Here the fragment "or five" yields a semantic constraint which cannot be combined with the other constraints resulting from the rest of the utterance. Consequently it is discarded in the rewriting phase of the parsing algorithm, and hence does not damage the analysis.

6.3 Expressiveness of the semantic representation formalism

The figures above indicate that the parser meets the basic goal of robustness in the sense that it outputs utterance descriptors with a significantly higher degree of accuracy than the strings output by the speech recognizer. The remaining question is then whether the formalism is sufficiently expressive. Although the analysis in Section 6.2 is a strong indicator that this is the case with the present data set, it is interesting to consider utterances from other corpora, specifically the kinds of utterances that *cannot* be represented by the formalism.

To get a handle on this, we have looked at two other database-oriented corpora, one from the same domain and one from a different domain. The former is the ADAPT corpus, comprising 1 858 user utterances (Bell et al. 2000), and the other one is the SmartSpeak travel-planning corpus (Boye et al. 1999), comprising 3 600 user utterances. Both corpora are the results of Wizard-of-Oz data collections. In both cases the wizards tried to promote user initiative as well as to simulate near-perfect speech understanding.

Below is a list of sentence types that are not representable as utterance descriptors, but instances of which are found in at least one of the corpora. The list was obtained by manually checking several hundred utterances from each of the two corpora and, in addition, searching the entire corpora for a variety of constructions judged to be critical.

1. Constructions involving a function of more than one structured object: "How many two- or three-room apartments are there around here?"
2. Complex and–or nesting: "A large one-room apartment or a small two-room apartment."
3. Selection of elements from a complementary set: "Are there any other apart-

ments around Medborgarplatsen that are about 50 square meters big and that are not situated at the ground floor?"

4. Comparatives involving implicit references where the comparison is made with a *set* of objects rather than with a single object. To illustrate, assume that several flight alternatives and their departure times have been up for discussion previously in the dialogue. The user then asks: "Is there a later flight?", requesting a flight which is later than all the previously mentioned ones. (To determine whether such a comparison is made with a set of objects or with a single object, it is in general not sufficient to look only at the last utterance. Thus, to handle this, the context-independent representation of the utterance must cater for both possibilities, thereby allowing contextual analysis to make the final verdict.)

Altogether, the most common of these types is (1), which accounts for 0.4 % in the apartment corpus but does not show up at all in the travel corpus. In none of the other cases do the number of occurrences exceed 0.05 % of a single corpus. We take this as a further indication that the expressiveness trade-off is a reasonable one for the kind of task and domains that we have been considering.

As a comparison, TEAM (Grosz et al. 1985) handles many constructions that our parser cannot handle, such as "Is the smallest country the least populous" (comparison involving two superlatives) and "For the countries in North America, what are their capitals?" ("each" quantification). However, constructions like these do not show up in our corpora of spoken negotiative dialogue.

We set out by claiming that negotiative dialogue demands that we go beyond variable-free slot–filler structures. Indeed, even a quick look at the corpora reveals that a substantial part of the utterances *do* require the added expressiveness. Thus, in addition to trivial specification utterances such as "I'd like a two-room apartment on the South Side", one encounters in our corpora numerous instances like the following that can be represented by our formalism, but not in general by variable-free slot–filler structures:

1. Seeking information about a particular aspect of an apartment, like "Does that apartment have a balcony?", as opposed to specifications such as "I'd like an apartment with a balcony".
2. Comparative constructions involving explicit references to different objects in the same utterance, such as "I'd like an apartment on Horn Street which is cheaper than the apartment on King's Street".
3. Comparatives involving implicit references, such as "Is there anything cheaper?".
4. Superlatives: "The cheapest apartment near Karlaplan."
5. Combinations of a comparative and selection of a minimal element: "When is the next flight?", which can be paraphrased as "Give me the earliest flight that departs after the flight that you just mentioned."

6.4 Discussion

Based on the arguments of the previous section, we conclude that the expressiveness of the semantic representation formalism can be narrowed down relative to first-order logic without harming its practical usefulness with respect to the kind of application dealt with here (spoken negotiative dialogue with databases). Because of the restrictions of expressiveness (in other words, the deliberate coarse-grainedness of the formalism), we can use a form of robust parsing where we neither have to analyse every word in the utterance, nor analyse all relations between the words in the utterance.

As an illustration of this point, the system is unable to represent (and understand) the utterance "Do all apartments that cost more than 2 million have a balcony?". The system would understand the utterance as a request for an apartment that costs more than 2 million and has a balcony, by triggering on "apartments", "more than 2 million" and "balcony", allowing recognition errors on all remaining words. Thus robustness is increased.

As yet another illustration, a common source of error in spoken-language understanding is the recognition of function words, due to their typically being short and non-prominent. Consider, for example, the utterance "How much does it cost?", which is frequently used in the ADAPT domain. Here, our parser achieves increased robustness by not necessarily depending on correct recognition of the pronoun "it" to infer that the utterance involves an anaphoric relation. Rather, the two crucial patterns are "How much" and "cost", which together allow the parser to draw the required conclusion (for lack of more information).

7 Possible extensions

In the ADAPT domain model, the apartment type is the only structured type. However, in many interesting domains there would be a need for several different structured types. Consider for instance an extension of the apartment domain encoding also the properties of the neighbourhood of the apartment (e.g. locations and other properties of schools, metro stations, restaurants, etc.), or an extension of the domain encoding the properties of individual rooms of each apartment. Each of these cases introduces new considerations. In the former case, utterances like "What street is it?" would be ambiguous, since there would now be several kinds of objects that have a street name attribute. In the latter case, a system should be able to correctly interpret and respond to queries like "Is there an apartment with a living-room at least 30 square meters big?".

In the domain model used by the ADAPT system, all objects except apartments are *second-class* in the sense that they can only appear in the context of an apartment. If the user mentions a balcony, an adress or a price, the system always infers that the balcony, the address or the price is an attribute of an apartment, even though no apartment has been explicitly mentioned. Only apartments are *first-class* objects in the sense that no such inference is called for. In the ADAPT domain model, apartments (the first-class objects) are represented by means of structured objects, and second-class objects are represented as simple objects or sets of simple objects (for instance, a price is represented as an integer, and a kitchen is represented as a set of atomic terms representing the equipment in the kitchen; stove, dishwasher, microwave oven, and so on).

The two extensions described above represent two different kinds of departures from the ADAPT domain model. In the first case, there is more than one type of first-class object. In the second case, also second-class objects may be structured (i.e. second-class objects are no longer equated with simple objects or sets of simple objects).

Using the notions of first-class and second-class objects, the two extensions would be quite straightforward. Firstly, as briefly mentioned earlier, ambiguous utterances like "What street is it?" are represented using type variables:

$$\lambda y^{\mathsf{t}}\ ?x^{\mathsf{street\_name}}\ (y.street\_name = x)$$

Since there are several types of first-class objects that have a street name attribute, the type of the inferred first-class object is ambiguous and therefore represented by a variable. Here the first argument could be resolved against any object that has a street name attribute.

In the case where also individual rooms are modelled as structured objects, "Is there an apartment with a living-room at least 30 m^2 big?" would be represented using an existentially quantified variable y, as explained in section 4.2:

$$\lambda S^{\mathsf{set(apartment)}}\ ?x^{\mathsf{apartment}}\ \exists y^{\mathsf{room}}\ (y \in x.rooms\ \wedge\ y.type = living\_room \\ \wedge\ y.space > 30\ \wedge\ x \in S)$$

Furthermore, we assume that only first-class objects (rather than only structured objects) can be retrieved from the dialogue context, so lambda variables are always of a first-class type. This is not a restriction, since second-class objects are always associated with a first-class object. For instance, "I would like a bigger living-room" would be seen as a constraint on the desired apartment (i.e. "I would like an apartment that has a bigger living-room"). Such an utterance would thus be represented as:

$$\lambda S^{\mathsf{set(apartment)}}\ \lambda z^{\mathsf{apartment}}\ ?x^{\mathsf{apartment}}\ \exists y^{\mathsf{room}}\ \exists w^{\mathsf{room}} \\ (y \in x.rooms\ \wedge\ y.type = living\_room\ \wedge\ w \in z.rooms \\ \wedge\ w.type = living\_room\ \wedge\ y.space > w.space\ \wedge\ x \in S)$$

To make the parser correctly handle the extensions, two minor modifications of the rewriting phase are required: In step 2 of Section 5.3.3, the algorithm should look for variables of second-class types rather than objects of non-structured types. In Section 5.3.4, free variables of first-class types should be made lambda-bound, whereas free variables of second-class types should be made existentially quantified.

8 Related work

The study of text-based natural-language interfaces to databases has a long tradition, going back at least to around 1970, and with a culmination already in the 1980s (for an excellent overview, see Androutsopoulos et al. 1995). The high-end systems of this time, for example, TEAM (Grosz et al. 1985), LOQUI (Binot et al. 1991) and CLE/CLARE (Alshawi 1992, Alshawi et al. 1992) used deep syntactic analysis and powerful logical languages for semantic representation, and were able to engage in dialogue involving complex phenomena such as quantification, anaphora and ellipsis. Naturally, the task was facilitated by the fact that no noise from speech recognition was introduced.

When moving from written to spoken input, it is necessary to address the robustness problems suffered by the kind of systems mentioned above. One approach, pioneered by Ward (1989), is to rely on shallow pattern matching, and to resort to a relatively coarse-grained semantic formalism (essentially variable-free slot–filler lists). Other examples in the same vein are Jackson et al. (1991) and Aust et al. (1995). These approaches work well for system-directed dialogue in small domains, but the semantic representation is too weak to serve as a vehicle for more advanced forms of interaction, such as negotiative dialogue.

Several attempts at synthesizing or combining deep and shallow approaches have been made. One possibility is to "robustify" a deep-processing method, either by picking the largest grammatical fragment (Boye et al. 1999), or by trying to connect the smallest set of grammatical fragments that span the entire utterance (for example, van Noord et al. 1999 and Kasper et al. 1999).

Another possibility is to extend a shallow pattern-matching approach with the capability of handling general linguistic rules. For example, the parser of Milward and Knight (2001) makes use of linguistically motivated rules, representing the analysis as a chart structure. Semantic interpretation is carried out by mapping rules that operate directly on the chart. These rules incorporate task-specific as well as structural (linguistic) and contextual information, and are hence domain-dependent. By giving preference to mapping rules that are more specific (in the sense of satisfying more constraints), grammatical information can be used whenever available. As a comparison, our approach shares the trait of being able to combine constraints from many fragments of the utterance. However, this is achieved by a general procedure (unifying objects whose type information is compatible) rather than by using lots of special-purpose mapping rules. Also, the semantic representation produced by Milward and Knight's parser is still limited to that of variable-free slot–filler lists.

A third possibility is to refrain from committing to a choice by letting shallow and deep processing work in parallel. In translation, this has been achieved by having different processors add partial translation results to a single chart structure (Frederking and Nirenburg 1994, Rayner et al. 2000). Likewise, in dialogue systems it has been achieved by letting one deep and one shallow processor output separate semantic analyses obeying the same format, and to have a preference mechanism that picks the best of these according to some metric (Boye et al. 1999).

A general approach to combining shallow and deep processing is provided by Mimimal Recursion Semantics (MRS; Copestake et al. 1999, Copestake 2003) and similar formalisms such as Hole Semantics (Bos 1995, Blackburn and Bos 2003). Here, the idea is to use a flat, highly factorized representation of first-order logic with generalized quantifiers as a means for representing analyses from both shallow and deep processing. This opens up the possibility of combining results from deep and shallow processing, for example, invoking deep processing whenever shallow processing returns an underspecified semantic representation. A related notion is used by Alexandersson et al. (2000).

Similar to our utterance descriptors, MRS was developed in the context of a dialogue system (namely, Verbmobil). There is thus a large corpus that contains MRS descriptions for utterances from face-to-face dialogues in an appointment-scheduling and travel-arrangement domain (Oepen et al. 2002). Another similarity is that both MRS and utterance descriptors constitute syntactically flat representations (in the latter case, this is because constraints are never embedded within one another). The crucial difference, however, is the expressiveness: MRS is as expressive as first-order logic, and is explicitly designed to handle semantic composition in general grammatical frameworks such as HPSG. In contrast, our utterance descriptors are much less expressive than first-order logic, motivated by what we have found to characterize spoken negotiative dialogue with databases (see Section 4.5).

As for contextual reasoning, Section 4.3 gives a detailed description of how this is carried out in the ADAPT system. General semantic frameworks such as Discourse Representation Theory (DRT; Kamp and Reyle 1993) and Centering Theory (Grosz et al. 1995) use elaborate mechanisms in the form of accessibility relations and ranking of forward-looking centers for establishing anaphoric relations. This is further exploited in applications such as DIPPER (Bos et al. 2003). Again, however, our approach is more restricted, and replaces these elaborate mechanisms with a few very simple principles based on the *typing* of objects in the domain.

Finally, an interesting approach which is also based on trying to find an empirically useful subclass of database questions is provided by the PRECISE natural-language database interface (Popescu et al. 2003). Here, a class of "semantically tractable questions" is defined, each of which yields a provably correct answer. Questions outside of this class, on the other hand, are answered with an error message indicating that the system does not understand the input question. Hence, PRECISE is "robust" in the sense that no incorrect answer will ever be produced; on the other hand, it lacks graceful degradation, which is a property typically associated with robustness. Also, since PRECISE is text-based and has no dialogue capabilities, it is not directly comparable to our approach.

9 Conclusion

As stated in Section 1, the goal of the paper has been to find an empirically motivated trade-off between robustness and expressiveness in spoken, negotiative dialogue with databases. To this end, we have designed a semantic formalism which strikes a compromise between variable-free slot–filler lists and more powerful representations, such as first-order logic. Our parsing algorithm is able to combine information from many fragments of an utterance by a general heuristic procedure, essentially consisting in unification of objects whose type information is compatible. The evaluation has shown that this method is highly robust; it yields correct results for a considerable amount of incorrectly recognized utterances, and it is only rarely led astray. Since our algorithm does not engage in any complex grammatical analysis, it is also very fast. We have thus shown that it is possible to go beyond variable-free slot–filler structures without having to revert to deep linguistic analysis. In other words, we have demonstrated that the applicability of shallow, pattern-matching parsers is wider than one might have thought.

10 Acknowledgements

The authors would like to thank Joakim Gustafson for discussions on the topics of this paper, Jens Edlund and Magnus Nordstrand for letting us use their corpus for our evaluation, and three anonymous reviewers for thorough and illuminating comments on draft versions. This work was supported by the EU/HLT-funded project NICE (IST-2001-35293).

References

Alexandersson, J., Engel, R., Kipp, M., Koch, S., Küssner, U., Reithinger, N. and Stede, M. (2000). Modeling Negotiation Dialogs. In: W. Wahlster (ed.) *Verbmobil: Foundations of Speech-to-Speech Translation*, Springer, pp. 441–451.

Alshawi, H. (1992). *The Core Language Engine.* The MIT Press.

Alshawi, H., Carter, D., Crouch, R., Pulman, S., Rayner, M. and Smith, A. (1992). CLARE — A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine. Final report, SRI International.

Androutsopoulos, I., Ritchie, G. and Thanish, P. (1995). Natural Language Interfaces to Databases — An Introduction. *Journal of Natural Language Engineering* 1(1), pp. 29–85.

Aust, H., Oerder, M., Seide, F. and Steinbiss, V. (1995). The Philips automatic train timetable information system. *Speech Communication* 17, pp. 249–262.

Bell, L., Boye, J., Gustafson J. and Wirén, M. (2000). Modality Convergence in a Multimodal Dialogue System. *Proc. Götalog*, pp. 29–34.

Bell, L., Boye, J. and Gustafson J. (2001). Real-time Handling of Fragmented Utterances. *Proc. NAACL Workshop on Adaptation in Dialogue Systems.*

Binot, J-L., Debille, L., Sedlock, D. and Vandecapelle D. (1991). Natural Language Interfaces: A New Philosophy. *SunExpert Magazine*, pp. 67–73, January 1991.

Blackburn, P. and Bos, J. (2005). *Representation and Inference for Natural Language.* CSLI Publications.

Boros, M., Eckert, W., Gallwitz, F., Görz, G., Hanrieder, G. and Niemann, H. (1996). Towards Understanding Spontaneous Speech: Word Accuracy vs Concept Accuracy. *Proc. Fourth International Conference on Spoken Language Processing* (*ICSLP 96*), pp. 1009–1012.

Bos, J. (1995). Predicate Logic Unplugged. *Proc. Tenth Amsterdam Colloquium*, ILLC/Department of Philosophy, University of Amsterdam, Amsterdam, Holland.

Bos, J., Klein, E., Lemon, O. and Oka, T. (2003). DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. *Proc. 4th SIGdial Workshop on Discourse and Dialogue*, Sapporo, Japan.

Boye, J., Wirén, M., Rayner, M., Lewin, I., Carter, D. and Becket, R. (1999). Language Processing Strategies and Mixed-Initiative Dialogues. *Proc. IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems.*

Boye, J. and Wirén, M. (2003). Robust Parsing of Utterances in Negotiative Dialogue. *Proc. 8th European Conference on Speech Communication and Technology* (*Eurospeech 2003*).

Copestake, A., Flickinger, D., Sag, I. and Pollard, C. (1999). Minimal Recursion Semantics: An introduction. `http://lingo.stanford.edu/sag/papers/copestake.pdf`. To appear in *Research on Language and Computation.*

Copestake, A. (2003). Report on the Design of RMRS. Deliverable D1.1a, available at `http://www.eurice.de/deepthought/`.

Edlund, J. and Nordstrand, M. (2002). Turn-taking Gestures and Hour-Glasses in a Multimodal Dialogue System. *Proc. ISCA workshop on Multimodal Dialogue in Mobile Environments*, Kloster Irsee.

Frederking, R. and Nirenburg, S. (1994). Three Heads are Better than One. *Proc. Fourth Conference on Applied Natural Language Processing* (*ANLP 1994*), Stuttgart, Germany.

Grosz, B., Appelt, D., Martin, P. and Pereira, F. (1985). TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. Technical note 356, SRI International.

Grosz, B., Joshi, A., and Weinstein, S. (1995). Centering: A Framework for Modeling the Local Coherence of Discourse. *Computational Linguistics*, 21(2), pp. 203–225.

Jackson, E., Appelt, D., Bear, J., Moore, R. Podlozny, A. (1991). A Template Matcher for Robust NL Interpretation. *Proc. DARPA Speech and Natural Language Workshop*, Morgan Kaufmann.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic. Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Kluwer, Dordrecht.

Kasper, W., Kiefer, B., Krieger, H., Rupp C., Worm, K. (1999). Charting the depth of robust speech processing. *Proc. 37th Annual Meeting of the ACL, College Park, Maryland.*

Larsson, S. (2002). Issue-based Dialogue Management. Ph.D. Thesis, Göteborg University, ISBN 91-628-5301-5.

Milward, D. and Knight, S. (2001). Improving on Phrase Spotting for Spoken Dialogue Processing. *Proc. Workshop on Innovation in Speech Processing* (*WISP 2001*), Stratford-upon-Avon, UK.

van Noord, G., Bouma, G., Koeling, R. and Nederhof, M-J. (1999). Robust Grammatical Analysis for Spoken Dialogue Systems. *Journal of Natural Language Engineering*, 5(1), pp. 45–93.

Oepen, S., Flickinger, D., Toutanova, K. and Manning, C. D. (2002). LinGO Redwoods: A rich and dynamic treebank for HPSG. *Proc. First Workshop on Treebanks and Linguistic Theories* (*TLT2000*), Sozopol, Bulgaria.

Popescu, A., Etzioni, O. and Kautz, H. (2003). Towards a Theory of Natural Language Interfaces to Databases. In *Proc. International Conference on Intelligent User Interfaces* (*IUI 2003*), Miami, Florida.

Rayner, M., Carter, D., Bouillon, P., Digalakis, V. and Wirén, M., editors (2000). *The Spoken Language Translator.* Cambridge University Press.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. Second edition. The MIT Press.

Ullman, J. (1988). *Database and Knowledge-base Systems*, Volume I. Computer Science Press.

Ward, W. (1989). Understanding Spontaneous Speech. *Proc. DARPA Speech and Natural Language Workshop*, pp. 137–141, Philadelphia, Pennsylvania.