Robust Parsing of Utterances in Negotiative Dialogue

Johan Boye and Mats Wirén

Telia Research[1]
johan.boye@teliasonera.com, mats.wiren@teliasonera.com

Abstract

This paper presents an algorithm for domain-dependent parsing of utterances in negotiative dialogue. To represent such utterances, the algorithm outputs semantic expressions that are more expressive than propositional slot-filler structures. It is very fast and robust, yet precise and capable of correctly combining information from different utterance fragments.

1. Introduction

The rapidly increasing number of spoken-dialogue systems have led to numerous robust parsers for limited domains having been constructed during the last decade. By "parsing" we here mean a mapping from the input utterance to a context-independent semantic representation. By "robust" we mean that the parser will give a reasonable result even on very noisy input.

In simple spoken-dialogue applications, parsing can be interleaved with speech recognition (if the language model is grammar-based). The vast majority of launched commercial systems fall into this category. However, spoken-dialogue systems aiming at less system control and more user initiative usually have a statistical *n*-gram recogniser, due to the difficulties of constructing a grammar with sufficient coverage. In this case (which is what we consider in this article), a separate parser is needed to transform the output from the recogniser into a semantic representation.

Generally speaking, these robust parsers have been highly successful. Distinguishing features are: very fast execution, short development times (down to a few person weeks), and performance as good as or better than state-of-the-art parsers based on large-coverage grammars. Such robust parsers are usually based on phrase-spotting, and output slot–filler structures representing the propositional contents of the input utterance. Thus, precision is traded for robustness, execution speed and ease of development.

However, propositional slot–filler structures are not sufficient in cases where more user initiative is involved, such as negotiative dialogue (for example, travel planning, appointment scheduling, or apartment browsing). In this paper, starting from a domain model in the form of a relational database, we derive a richer semantic representation formalism suitable for negotiative dialogue. Furthermore, we describe and evaluate a parsing algorithm for mapping speech-recognition output to this representation. The parsing algorithm is very fast, yet precise and capable of correctly combining information from different utterance fragments. It has been used successfully in Adapt (Bell et al. [1]), a multimodal dialogue system in an apartment domain. A forerunner was used in the travel-planning system SmartSpeak (Boye et al. [2]).

2. Negotiative spoken dialogue systems

The following piece of dialogue with Adapt [1] provides an illustration of negotiative dialogue:

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:** Ok... is there anything cheaper?

A distinguishing factor, compared to most system-driven dialogue systems, is that the dialogue may concern several objects simultaneously, allowing the user to compare their relative merits, as in utterance 5 and 7 above. This means, to begin with, that the system must be able to represent objects (that can be referred to) and not just properties of objects, as shown in utterance 5. Furthermore, the system must be able to distinguish what is asked for (the cost) from what is given (the apartment on Horn street). In the rest of the paper, we explore the implications of these additional requirements for both parsing and semantic representation.

3. Back-end: Relational database

We assume that the domain information is represented (or at least representable) as a relational database. Figure 1 shows a relation "apartment" (which is a simplified version of the domain model of Adapt [1]).

We assume that each attribute is typed, so that e.g. the attribute *price* has the type *money, size* has the type

[1] Address: Vitsandgatan 9B, 12386 Farsta, Sweden

ID	Street name	No	Size	No of rooms	Price	Monthly fee	Accessories	
31213	Horn street	134	76	2	1,900,000	2,830	balcony, tiled stove, ...	...
61623	King's street	73	105	4	3,500,000	4,113	parquet floor, ...	...
...	...	...	...	...	...	...	....	...

Figure 1: A domain model represented by a relational database

square meters, etc. Types are arranged in a hierarchy, so that e.g. both *money* and *square meters* are subtypes of *integer* (which entails that values in both the *price* column and the *size* column may be compared numerically). Some attributes have types which consist of an enumeration of all possible values, e.g. the type of *street name* is an enumeration of all the street names in Stockholm.

This kind of type information is essential for the semantic representation of utterances, and indeed for the parsing algorithm (see Section 5).

4. Semantic representation formalism

The semantic output from the parser consists of "utterance descriptors", which are expressions that have a natural mapping to a sequence of database operations. The by far most applicable kind of utterance descriptor is of the form *?x.P* (paraphased as "Find *x* such that *P*"), where P (the *body* of the expression) is a list of constraints, and where x (the *head* of the expression) denotes the sought object. We may write *t:x* instead of *x* to indicate that *x* should be of type *t*. For instance, the utterance "I'm looking for an apartment on King's street" would be represented as

?apartment:x.(x.street = Kings_street)

Here the type *apartment* refers to members of the database relation with the same name (i.e. rows in the table in Figure 1).

The constraints in the body *P* specify the desired values of database attributes, as well as relations between database values and other values. The following are all examples of constraints (assuming *x* denotes an apartment):

- *x.street = Kings_street*
- *x.price < 2,000,000*
- *balcony ∈ x.accessories*

The set of well-formed constraints are completely determined by the type information of the back-end relational database. For instance, the second constraint is well-formed since the *price* attribute is numerical. The third constraint is well-formed since the *accessories* attribute has a list type.

Utterances may contain explicit or implicit references to other objects than the object sought. For instance, "Is there a cheaper apartment?" contains an implicit reference ("Is there a cheaper apartment than the one you just mentioned?"). Such references are represented by lambda-bound variables, as in the example:

λ apartment:y ?apartment:x.(x.price<y.price)

(For an explanation of how to interpret the *λ*-notation, see e.g. [5], chapter 15). The idea is that the ensuing processing stages of the dialogue system will infer what object the user is referring to, and the functional semantic expression above can then be applied to that object. As an example, consider the dialogue excerpt in Section 2, where the expression above is the representation of utterance no 7. Applying that expression to the object representing the apartment mentioned in utterance 6, would result in

?apartment:x.(x.price < 1,900,000)

Another example of the use of lambda-bound variables is the utterance "How much does it cost", which is represented as:

λapartment:y ?money:x.(y.price = x)

Another form of utterance descriptor is used to represent superlatives such as "Where is the biggest apartment?" or "Which is the cheapest?", but because of the restricted space we will not discuss these kinds of expressions.

5. Robust parsing algorithm

The robust parsing algorithm consists of two phases, a pattern matching phase and a rewriting phase. In the latter, 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. Pattern matching phase

In the first phase, a string of words[1] is scanned left-to-right, and a sequence of constraints and *meta-constraints*, triggered by syntactic patterns, are collected. The constraints will eventually end up in the body of the final utterance descriptor, while the purpose of the meta-constraints is to guide the rewriting phase.

The syntactic patterns can be arbitrarily long, but of course the longer the pattern, the less frequently it will appear in the input (and the more sensitive it will be to

[1] Currently, 1-best output from the speech recogniser is used.

recognition errors, disfluencies etc.). On the other hand, longer syntactic patterns are likely to convey more precise information.

The solution is to try to apply longer patterns before shorter patterns. As an example, reconsider the utterance "I'm looking for an apartment on King's street, and suppose that "apartment on S" (where S is a street), "apartment" and "King's street" are all patterns used in the first phase. If the utterance has been correctly recognised, the first pattern would be triggered. However, the utterance might have been misrecognised 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 both cases the pattern "apartment on S" would fail, so the pattern matching phase would have to fall back on the two separated patterns "apartment" and "King's street," and let the rewriting phase infer the relationship between them.

5.2. Meta-constraints

The following list enumerates some examples of kinds of meta-constraints (*Var* represents a variable):

obj(Type:Var)	An object of type Type has been mentioned.
head_obj(Type:Var)	Same as above + the object denoted by Var is likely to be the object sought.
$Var_1 \neq Var_2$	Var_1 and Var_2 denote different objects.
ambiguous(Var, {a_1, a_2, ... a_n}, default(a_i))	*Var* is one of the objects a_1, a_2, ... a_n. If no evidence is against, *Var* should be equal to a_i.

The pattern matching rules in the pattern matcher associate a sequence of constraints and meta-constraints to each pattern. For instance, in the Adapt parser[1], the pattern "apartment" would yield

$$obj(apartment:x_1)$$

whereas the pattern "King's street" would yield

$$obj(apartment:x_2),\ x_2.street = Kings\_street$$

where x_1 and x_2 are variables. The existence of the object $apartment:x_2$ is inferred, since in the database domain model, streets can only occur in the context of the *street* attribute of the *apartment* relation. If the domain model would include also another relation (*restaurant*, say) that also has an attribute *street*, the pattern could instead yield:

$$obj(t:x_2),\ x_2.street = Kings\_street$$

where *t* is a variable.

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 a utterance descriptor (after removing all meta-constraints). The most important rules are:

1. Unify as many objects as possible.
2. Identify the object sought
3. Identify lambda-bound variables
4. Identify minimisation or maximisation attributes (if any)
5. Determine scope of negations (if any)
6. Resolve ambiguities

The first rule works as follows: Suppose pattern matching has resulted in:

$$obj(apartment:x_1),\ obj(t:x_2),\ x_2.street = Kings\_street$$

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

$$obj(apartment:x_1),\ x_1.street = Kings\_street$$

As for the second rule, the object sought is assumed to be the leftmost *head_obj* in the sequence. Failing that, it is assumed to be the leftmost *obj*. In the sequence above, that means $apartment:x_1$, which results in:

$$?apartment:x_1.\ (obj(apartment:x_1),\ x_1.street = Kings\_street)$$

Since no other rewrite rule is applicable, the final result is obtained by removing all meta-constraints:

$$?apartment:x_1.\ (x_1.street = Kings\_street)$$

The restricted space prevents a detailed discussion of all the rewrite rules. Some more illustration of their application is provided in the next section.

5.4. Example

Consider the utterance "Is there anything cheaper?". In the Adapt parser, the pattern "cheaper" would yield

$$head\_obj(apartment:x_1),\ obj(apartment:x_2),\ x_1 \neq x_2,\ obj(money:y_1),\ x_1.z = y_1,\ obj(money:y_2),\ x_2.z = y_2,\ y_1 < y_2,\ ambiguous(z, \{price, monthly\_fee\}, default(price))$$

The meta constraint $x_1 \neq x_2$ is added to prevent unification of $apartment:x_1$ and $apartment:x_2$, which are referring to two different apartments. The rewriting phase can apply three rules: the ambiguity will be resolved (binding *z* to *price*), the variable x_1 will be chosen as the main object, and the variable x_2 will be lambda-bound. After removing meta-constraints, this results in:

$$\lambda apartment:x_2\ ?apartment:x_1(x_1.price = y_1 \wedge x_2.price = y_2 \wedge y_1 < y_2)$$

which evidently is equivalent to:

$$\lambda apartment:x_2\ ?apartment:x_1(x_1.price < x_2.price)$$

6. Evaluation

To evaluate the parser, we tested it on 300 unseen utterances from the corpus described in Edlund [3]. The algorithm proved to be very fast; the average running time was below 10ms per sentence (on a Pentium 4) .

The accuracy results are summarised in Figure 2.

	Transcriptions	Recognitions
Sentence accuracy	N/A	36.7%
Word accuracy	N/A	62.9%
Match	92.3%	58.7%
Precision	97.0%	81.4%
Recall	95.9%	72.0%
Concept accuracy	95.9%	73.2%

Figure 2: Parser accuracy

The first two rows describe the accuracy of the input strings. About 37% of the utterances were perfectly recognised, and the overall word accuracy was about 63% (i.e. the word error rate was about 37%).

The last four rows describe the semantic accuracy, i.e. to what extent the semantic output from the parser corresponded to the meaning of the input utterance. The basic semantic unit is defined to be a constraint in the body of the utterance descriptor, a λ-bound variable, or a ?-bound variable in the head of the utterance descriptor. The left column indicates how well the parser performs given perfect speech recognition, while the right column shows how well it performs on actual recogniser output.

The "Match" column shows the number of sentences which were completely correctly analysed. "Precision" is the number of produced correct semantic units divided by the number of produced semantic units. "Recall" is the number of produced correct semantic units divided by the number of correct semantic units. "Concept accuracy" is the semantic equivalent of word accuracy (for a definition, see e.g. van Noord et al [8]).

Finally, we would like to comment on the robustness of the approach. The parser manages to deliver perfectly correct results for 59% of the input sentences, although only 37% of them were perfectly recognised. Another indication of the robustness is the fact that the concept accuracy of the output was 73%, although the word accuracy was only 63%.

7. Discussion and related work

Approaches to domain-dependent robust parsing for spoken input can be largely divided into two types. One approach is based on general-purpose linguistic rules, "robustified" by homing in on the largest grammatical fragment [2], or the smallest set of grammatical fragments that span the whole utterance [6, 8]. The other approach is based on pattern matching, extracting information-carrying units of the utterance based on domain-dependent patterns (which may or may not correspond to a complete grammatical phrase) [4]. Also hybrid systems have been built [2].

An interesting attempt at synthesising these approaches is provided by Milward and Knight [7]. Their parser 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. By giving preference to mapping rules that are more specific (in the sense of satisfying more constraints), grammatical information can be used whenever available.

Our approach shares the trait of being able to maximize the combination of 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. Because our algorithm does not engage any complex grammatical analysis, it is also very fast. Furthermore, we have shown that it is possible to go beyond propositional slot-filler structures without having to revert to grammar-based analysis. In other words, we have demonstrated that the applicability of robust, concept-spotting parsers is wider than one might have thought.

Acknowledgements: The authors like to thank the Adapt group at Telia Research and KTH. Special thanks to Jens Edlund et al for letting us use their test corpus. The work described in this paper was supported by the EU/HLT funded project NICE (IST-2001-35293).

8. References

[1] Bell, L., Boye, J. and Gustafson, J. "Real-time Handling of Fragmented Utterances", NAACL Workshop on Adaptation in Dialogue Systems, 2001.

[2] Boye, J., Wirén, M., Rayner, M., Lewin, I, Carter, D. and Becket, R. "Language Processing Strategies and Mixed-Initiative Dialogues", IJCAI Dialogue Workshop, 1999.

[3] Edlund, J. and Nordstrand, M. "Turn-taking Gestures and Hour-Glasses in a Multi-modal Dialogue System". ISCA workshop on Multimodal Dialogue in Mobile Environments, Kloster Irsee, 2002.

[4] Jackson, E., Appelt, D., Bear, J., Moore, R. Podlozny, A. "A Template Matcher for Robust NL Interpretation". DARPA Speech and Natural Language Workshop, Morgan Kaufmann, 1991.

[5] Jurafsky, D. and Martin, J. "Speech and Language Processing"', Prentice-Hall, 2000.

[6] Kasper, W., Kiefer, B., Krieger, H., Rupp C., Worm, K. "Charting the depth of robust speech processing", ACL, 1999.

[7] Milward, D. and Knight, S. "Improving on Phrase Spotting for Spoken Dialogue Processing", WISP, 2001.

[8] van Noord, G., Bouma, G., Koeling, R., Nederhof, M. "Robust Grammatical Analysis for Spoken Dialogue Systems". Journal of NLE, 5(1), 1999, pp. 45—93.