Using Paraphrases of Deep Semantic Represention to Support Regression Testing in Spoken Dialogue Systems

Beth Ann Hockey
UC Santa Cruz and BAHRC LLC
Mail Stop 19-26, UCSC UARC
NASA Ames Research Center, Moffett Field, CA 94035–1000
bahockey@bahrc.net

Manny Rayner
University of Geneva, TIM/ISSCO
40 bvd du Pont-d'Arve, CH-1211 Geneva 4, Switzerland
Emmanuel.Rayner@unige.ch

Abstract

Rule-based spoken dialogue systems require a good regression testing framework if they are to be maintainable. We argue that there is a tension between two extreme positions when constructing the database of test examples. On the one hand, if the examples consist of input/output tuples representing many levels of internal processing, they are fine-grained enough to catch most processing errors, but unstable under most system modifications. If the examples are pairs of user input and final system output, they are much more stable, but too coarse-grained to catch many errors. In either case, there are fairly severe difficulties in judging examples correctly. We claim that a good compromise can be reached by implementing a paraphrasing mechanism which maps internal semantic representations into surface forms, and carrying out regression testing using paraphrases of semantic forms rather than the semantic forms themselves. We describe an implementation of the idea using the Open Source Regulus toolkit, where paraphrases are produced using Regulus grammars compiled in generation mode. Paraphrases can also be used at runtime to produce confirmations. By compiling the paraphrase grammar a second time, as a recogniser, it is possible in a simple and natural way to guarantee that confirmations are always within system coverage.

1 Introduction

Design features that enable important functionality in medium vocabulary, mixed-initiative spoken dialogue systems also create challenges for the project cycle, and in particular for regression testing. Two issues that make regression testing particularly difficult are the need for context dependent interpretation, and the use of multiple levels of representation. Both of these features are typically necessary for non-trivial dialogue systems of this type. Multiple levels of processing, as usual, provide necessary modularity. Context dependent interpretation enables responses that are tuned to the current circumstances of the interaction or the world, and frequently helps resolve ambiguity.

The implications for regression testing, though, are less happy. The context of each interaction in the test suite needs to be stored as part of the interaction. Multiple levels of representation that are, for example, useful for doing ellipsis resolution or reference resolution, also complicate testing. If regression testing is done on each separate level of processing, or involves internal representations, small changes to a representation at one level can mean having to revise and rejudge the entire test suite to keep it up to date.

This paper discusses the methodology we have developed to address regression testing issues within the Regulus framework. Regulus (Rayner et al., 2006) is an Open Source toolkit for builting medium

Proceedings of the NAACL HLT Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing, pages 14–21,
Boulder, Colorado, June 2009.

vocabulary spoken dialogue and translation applications, and has been used to build a number of non-trivial spoken dialogue systems. Prominent examples include NASA's Clarissa Procedure Navigator (Rayner et al., 2005), Geneva University's multi-modal mobile-platform Calendar application (Tsourakis et al., 2008), SDS, a prototype in-car system developed by UC Santa Cruz in collaboration with Ford Motors Research which was voted first in Ford's 2007 internal technology fair, and Taxi, a speech-enabled game in which the user interacts with a simulated cab driver to navigate around a map of Manhattan. It has also been used to build the MedSLT medical speech translation system (Bouillon et al., 2008).

The Regulus platform includes tools for developing feature grammars, and compiling them in various ways. In particular, it is possible to compile grammars into generators, and use them to support paraphrasing from the internal semantic representations created during dialogue processing. This capability is key to the newest part of our regression testing approach, and is discussed in detail in Section 3. First, though, Section 2 gives an overview of Regulus and the architecture of Regulus-based systems; we discuss features that complicate regression testing, and how to address these problems within this type of architecture. Section 4 discusses how test suites are constructed and what types of items they may contain. In Section 5 we show how paraphrases can also be included in the run-time architecture. The final section concludes.

2 The Regulus platform

Regulus is an Open Source toolkit for building medium vocabulary grammar-based spoken dialogue and translation systems. The central idea is to base run-time processing on efficient, task-specific grammars derived from general, reusable, domain-independent core grammars. Early versions of Regulus used a single core grammar per language; a detailed description of the core grammar for English can be found in (Rayner et al., 2006, Chapter 9). More recently, there have been attempts to go further, and merge together core grammars for closely related languages (Bouillon et al., 2007).

The core grammars are automatically specialised,

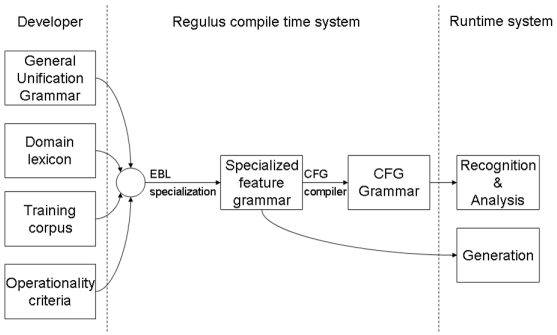

Figure 1: The Regulus compilation path. The general unification grammar is first transformed into a specialised feature grammar. This can then be transformed either into a CFG grammar and Nuance recogniser, or into a generator. and a Nuance recogniser.

using corpus-driven methods based on small corpora, to derive simpler grammars. Specialisation is both with respect to task (recognition, analysis, generation) and to application domain. The specialisation process uses the Explanation Based Learning algorithm (van Harmelen and Bundy, 1988; Rayner, 1988). It starts with a parsed treebank derived from the training corpus, and then divides the parse tree created from each training example into a set of one or more subtrees, following a set of domain- and grammar-specific rules conventionally known in the Machine Learning literature as operationality criteria. The rules in each subtree are then combined, using the unification operation, into a single rule. The set of all such rules constitutes a specialised unification grammar. Each of these specialised unification grammars is then subjected to a second compilation step, which converts it into its executable form. For analysis and generation, this form is a standard parser or generator. For recognition, it is a semantically annotated CFG grammar in the form required by the Nuance engine, which is then subjected to further Nuance-specific compilation steps to derive a speech recognition package. Figure 1 summarises compile-time processing.

The Regulus platform also contains infrastructure to support construction of applications which use the recognisers, parsers and generators as components. In this paper, we will only discuss spoken dialogue system applications. (There is also an elaborate infrastructure to support speech translation systems).

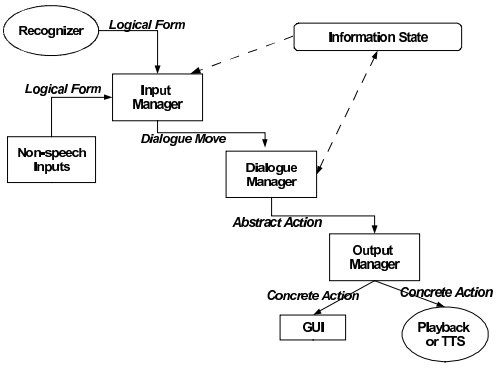

Figure 2: Top-level architecture for Regulus-based spoken dialogue system

At a high level of generality, the architecture is a standard one (Figure 2; cf. for example (Allen et al., 2000)). The central component is the Dialogue Manager (DM), which receives dialogue moves and produces abstract actions. It also manipulates an information state, which maintains context; processing will generally be context-dependent. The DM is bracketed between two other components, the Input Manager (IM) and the Output Manager (OM). The IM receives logical forms, and non-speech inputs if there are any, and turns them into dialogue moves. The OM received abstract actions and turns them into concrete actions. Usually, these actions will be either speaking, though TTS or recorded speech, or manipulation of a GUI's screen area.

In the next section, we examine in more detail how the various components are constructed, and what the implications are for the software development cycle. We will in particular be interested in regression testing.

3 Context, regression testing and paraphrasing

The three main components of the spoken dialogue system — the IM, DM and OM — all transform one or more inputs into one or more outputs. With the current focus on machine learning techniques, a natural thought is to learn the relevant tranformations from examples. Implemented mainly through Partially Observable Markov Decision Processes (POMDPs), this idea is attractive theoretically, but has been challenging to scale up. Systems have been restricted to very simple domains (Roy et al., 2000; Zhang et al., 2001) and only recently have techniques been developed that show promise for use in real-world systems (Williams and Young, 2007; Gasić et al., 2008). The representations required in many systems are more complex than those employed even in the more recent POMDP based work, and there is also the usual problem that it is not easy to obtain training data. In practice, most people are forced to construct the transformation rules by hand; the Regulus framework assumes this will be the case. Hand-coding of dialogue processing components involves the usual software engineering problems that arise when building and maintaining substantial rule-sets. In particular, it is necessary to have a framework that supports efficient regression testing.

As everyone who has tried will know, the thing that makes regression testing difficult for this kind of application is context-dependency. In the worst case, the context is the whole world, or at least the part of it that the system is interacting with, and regression testing is impossible. In more typical cases, however, good architectural choices can make the problem reasonably tractable. In particular, things become enormously simpler if it is possible to encapsulate all the context information in a datastructure that can be passed around. In the dialogue management architecture realised in Regulus (Rayner et al., 2006, Chapter 5), the assumption is that this is the case; it is then possible to use a version of "update semantics" (Larsson and Traum, 2000). The central concepts are those of *dialogue move*, *information state* and *dialogue action*. At the beginning of each turn, the dialogue manager is in an information state. Inputs to the dialogue manager are by definition dialogue moves, and outputs are dialogue actions. The behaviour of the dialogue manager over a turn is completely specified by an *update function* f of the form

$$f : State \times Move \rightarrow State \times Actions$$

Thus if a dialogue move is applied in a given information state, the result is a new information state and a set of zero or more dialogue actions.

3.1 Regression testing

Using the side-effect free framework is certainly a large step in the right direction; it is in principle pos-

sible to construct a regression test suite consisting of 4-tuples of the form

$$\langle InState, Move, OutState, Actions \rangle$$

There are however several problems. First, processing consists of much more than just the update function. It is optimistic to assume that the speech recogniser will be able to produce dialogue moves directly. In simple cases, this may be possible. In more complex cases, extra levels of processing become necessary; in other words, the IM component will generally have a substantial amount of structure.

There are several reasons for this. The representation delivered by the grammar-based speech recogniser is syntax-oriented; it needs to be translated into a semantic form. Again, because of context-dependency, this translation often needs to be carried out in more than one step. For example, in the Calendar application, a question like "When is the next meeting in Switzerland?" might be followed by the elliptical utterance "In England?". Some kind of mechanism is needed in order to resolve this to a representation meaning "When is the next meeting in England?" A separate mechanism is used to perform reference resolution. For instance, the default database for the Calendar application contains one person called "Nikos" and two called "Marianne". The question "Is Nikos attending the meeting?" needs to be converted into a database query that looks up the appropriate record; however, the structurally similar query "Is Marianne attending the meeting?" should produce a disambiguation query. Examples like these motivate the introduction of yet another processing step, which carries out reference resolution.

Of course, different systems will address these issues in different ways; but, whatever the solution, the general point remains that there will usually be many layers of representation. From a system development point of view, the problem is how to structure the regression testing needed in order to maintain the stability of each processing step. The most cautious and direct way to do this is to have a corpus of input/output tuples representing each individual step, but experience shows that this type of solution places an enormous burden on the annotators who are required to judge the correctness or otherwise of the tuples. First of all, under this approach the annotators must be experts capable of reading and understanding internal representations. Second, even very small changes in the system often require complete reannotation of the test corpus; for example, some data structure may have been changed so as to include an extra field. If constant rejudging is required to keep the test suite coherent with the current version of the system, either the testing is abandoned as overly difficult and time consuming, or it is done in a less careful way in order to speed up the process. Neither outcome is satisfactory.

If annotation uses input/output tuples referring to internal representations, the problems we have just named appear inescapable. At the opposite end of the spectrum, a common approach is not to look at internal representations at all, but only at input/output pairs consisting of top-level inputs and outputs. For example, we can pair "When is the next meeting in Geneva?" with "March 31 2009", and "Is Marianne attending the meeting?" with "Which Marianne do you mean?" This is generally, in practice, easier than doing regression testing on internal representations; the key advantages are that, since we are only dealing with pre-theoretical notions, annotation can be performed by non-experts, and annotations remain stable across most changes to internal representations.

Unfortunately, however, new problems arise. First, determining the correct output response for a given input is often tedious and slow. For example, in the Calendar application, this generally involves carrying out a database search. Suitable annotation tools can alleviate the pain here, but then a worse problem arises; it is often possible to produce a correct system response, even if processing is incorrect. For instance, even if the system correctly answers "No" to a yes/no question, this proves very little; the question could have been interpreted in a multitude of ways, and still produced a negative answer. Knowing that a WH-question provides a correct answer says more, but can still often be misleading. Suppose, for example, that we know that the Calendar system correctly answers "None" to the question "What meetings are there during the next week?" and there are no meetings for the next 15 days. We will be unable to tell whether the question has been interpreted as "What meetings are there during the

World Context	time=2008-10-14 14:34, speaker=mike
Last Para	(none)
Input	when is the next meeting with mark
Paraphrase	when is [the next meeting attended by mark green]
World Context	time=2008-10-16 09:47, speaker=mike
Last Para	(none)
Input	when is my next meeting with mark
Paraphrase	when is [the next meeting attended by mark green and mike jones]
World Context	time=2007-07-08 15:03, speaker=susan
Last Para	(none)
Input	is there a meeting next week
Paraphrase	are there meetings between Mon Jul 9 2007 and Sun Jul 15 2007
World Context	time=2008-11-17 18:20, speaker=mike
Last Para	(none)
Input	do i have a meeting on friday morning this week
Paraphrase	are there meetings between 06:00 and 12:00 on Fri Nov 21 2008 attended by mike jones
World Context	time=2008-11-12 10:19, speaker=mike
Last Para	when is [the next meeting attended by mike jones]
Input	will alex participate
Paraphrase	will that meeting be attended by alex miller
World Context	time=2007-07-08 15:56), speaker=susan
Last Para	are there meetings on Mon Jul 9 2007
Input	how about on tuesday
Paraphrase	are there meetings on Tue Jul 10 2007

Table 1: Examples of regression testing tuples in the English Calendar system. Each tuple shows the current world context (timestamp and speaker), the preceding paraphrase, the input, and the paraphrase produced from it.

next 7 days?", as "What meetings are there during the 7 day period starting this Sunday?" or as "What meetings are there during the 7 day period starting this Monday?" Examples like these mean that regression testing often fails to catch bugs introduced by system changes.

3.2 Paraphrasing dialogue moves

To summarise: when carrying out regression testing, we have two competing requirements. First, we need to be able to access internal representations, since they are so informative. At the same time, we prefer to work with human-readable, pretheorically meaningful objects, which will be stable at least under most small changes in underlying representations. There is, in fact, a good compromise between these goals: we define a transformation which realises the dialogue move as a human-readable string, which we call a *dialogue move paraphrase*. So, for example, consider the possible interpretations when, on March 6 2009, a user asks "What meetings are there during the next week?". If "What meetings are there during the next week?" is interpreted as "What meetings are there during the next 7 days?", then the paraphrase might be "What meetings are there between Fri Mar 6 and Thu Mar 12 2009?"; if the interpretation is "What meetings are there during the 7 day period starting this Monday?", then the corresponding paraphrase would be "What meetings are there between Mon Mar 9 and Sun Mar 15 2009?" Regression testing can be carried out using paraphrases of dialogue moves, rather than the dialogue moves themselves.

The paraphrase mechanism is implemented as a Regulus grammar, compiled in generation mode, which directly relates a dialogue move and its surface form. We have found that it is not hard to design "paraphrase grammars" which produce outputs

fulfilling the main design requirements. Regression testing is carried out on tuples consisting of the preceding paraphrase, the world context (if any), the input, and the resulting paraphrase. Examples of such tuples for the English Calendar grammar are shown in Table 1; in Calendar, the world context consists of the utterance time-stamp and the speaker.

A tuple combines the results of IM and DM (but not OM) processing for a given example, and presents them in a pre-theoretically meaningful way. Although they are not as fine-grained as tuples for individual processing steps, they are stable over most system changes. In the opposite direction, they are far more fine-grained than straight system input/system output tuples. They are much easier to judge than both of the other types of tuple. The bottom line, at least as far as we are concerned, is that a regression testing database of paraphrase-based tuples can actually be maintained without inordinate effort, implying corresponding gains for system stability. Previously, this was impossible.

The idea of creating paraphrases from dialogue moves is of course not new; in previous work, however, they have generally been used at runtime to provide feedback to the user as to how their input has been interpreted by the system. Although in the current discussion we have been more concerned with their use in regression testing, we have in fact also employed them for the more traditional purpose.

We return to this theme in Section 5. First, we describe in more detail where our test suites come from.

4 Collecting test suites

The tradition in the speech engineering community is that a test suite consists of a list of recorded wavfiles, together with accompanying transcriptions. The Nuance platform contains a fair amount of infrastructure, in particular the `batchrec` utility, for processing lists of wavfiles. These tools are very useful for computing measures like WER, and there is a strong temptation to try to build on top of them. After a while, however, we discovered that they meshed poorly with the the basic goals of regression testing in a spoken dialogue system, which revolve around speech *understanding* rather than speech *recognition*. There are two central problems. One of them is context-dependence, which we have already discussed at length. The other is the fact that many applications require that the IM process both speech and non-speech actions, with the sequence and even the timing of actions being important.

For example, as we have already seen, time is a central concept in the Calendar system. If the user says “Are there any meetings this afternoon?” the system interprets her as meaning “Are there any meetings from now until the end of the afternoon?” This means that the exact clock time for each utterance is important. In the Taxi application, the taxi is continually in motion, even when the user is not talking. The simulator sends the IM an non-speech message several times a second, giving the taxi’s new position and heading. This information is passed to the DM, updating its state, and is essential for correct interpretation of commands like “Turn right at the next corner”.

Considerations like these finally convinced us to move to a different strategy, in which offline regression testing more closely models the runtime behaviour of the application. At runtime, the system produces a time-stamped log of all input passed to the IM, including both speech and non-speech messages, in the sequence in which they were received. Each speech message is paired with a pointer to the recorded wavfile which produced it. Sets of such logs make up the test suite. Offline testing essentially re-runs the sequence of time-stamped records. Wavfiles are passed to a recognition server, which returns recognition results; time-stamps are used to set a notional internal clock, which replaces the real one for the purposes of performing temporal calculations. The test harness was quite easy to implement, and solves all the problems that arose from close adherence to a more speech recognition oriented test framework.

5 Using paraphrases at run-time

As mentioned in Section 3.2, paraphrase grammars can also be used at runtime, in order to provide a direct confirmation to the user showing how the system has interpreted what they have said. This is not a compelling design for every system; in a speech-only system, constant direct confirmation using paraphrases is in most cases unnatural and te-

dious. It is, however, a potentially valid strategy in a multi-modal system where it is possible to present a visual display of the paraphrase. In such a system, if paraphrases are regularly displayed to a user, there is, however, a good possibility of lexical and/or syntactic entrainment. Entrainment increases the likelihood that the user will produce the paraphrase language, which means that it would be valuable to be able to process that language through the system.

In the Regulus framework, this problem can be very straightforwardly addressed. Since the paraphrase grammar is a valid Regulus grammar, it can be compiled into a Nuance grammar, and hence into a second recognition package. At runtime, this package can be used in parallel with the main system recogniser. Because the paraphrase grammar is designed to directly relate surface language to dialogue moves, dialogue moves are generated directly, skipping the Input Manager processing. In particular, since the original point of the paraphrase grammar is to restate the user's content in a way that resolves and disambiguates underspecified material, there is no need for resolution processing. Figure 3 shows the dialogue system architecture with the additional paraphrase processing path.

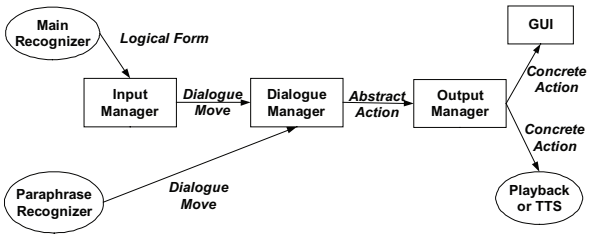

Figure 3: Regulus dialogue architecture with a processing path for paraphrases added. The paraphrase recognizer sends a dialogue move directly to the Dialogue Manager.

Although it may seem preferable to include the paraphrase coverage in the main recogniser coverage, we have found, somewhat to our surprise, that this is not nearly as straightforward as it first appears. The problem is that the two grammars are designed for very different tasks; the recognition grammar is intended to capture natural user language, while the paraphrase grammar's job is to produce unambiguous surface renderings of resolved semantic representations. Although we have endeavoured to make the paraphrase language as natural as possible, it is hard to avoid at least a few marginal constructions, which do not fit well into the structure of the normal recognition grammar; even if we did try to include them, the burden of keeping the two different grammars in synch would be considerable. From a software engineering point of view, it is far simpler just to maintain the two grammars separately, with each of them generating its own version of the recogniser.

We tested the paraphrase grammar recognizer for the Calendar application using paraphrases taken from a previous run log and recorded by the two authors. There were 249 recorded paraphrases total used. Because the Calendar paraphrase grammar had originally been designed with only visual display in mind, some augmentation of the paraphrase grammar was needed to cover the spoken versions of the paraphrases. There is often more than one possible spoken version corresponding to a written representation as was the case for this data. For example with a paraphrase such as "when are meetings on Sat Jan 3 2009", "Sat" could be pronounced "sat" or "Saturday", "3" could be "third" or "three", "Jan" could be produced as either "jan" or "January" and "2009" could be "two thousand nine" or "two thousand and nine". With the paraphrase component structured as a standard Regulus grammar, all that was needed was to add lexical items to cover the spoken variants. These additions were restricted to the recognition use of the paraphrase grammar and not used for generation. Word Error (WER) was 4.43% for the paraphrase grammar recognizer, Sentence Error (SER)was 34.53% and Semantic Error (SemER)was 17.9%. This SemER was calculated on untuned n-best. Clearly it is not possible to compare with the main recognizer on the same data, but for a rough comparison, we can look at numbers reported for the Calendar application in (Georgescul et al., 2008). That paper reports WER of 11.17% and SemER of 18.85% for the 1-best baseline. The SemER on the paraphrase grammar is 21.5% for 1-best. The paraphrase grammar recognizer has much better WER because it is so much more restricted than the main recognizer. However, the sentences covered by the paraphrase grammar are much longer than those covered in the main grammar, and this difference is reflected in the poorer performance by paraphrase grammar when measured in terms of Se-

mER. The paraphrase language is long, very unnatural, yet we are able to produce a level of recognition performance that is quite usable.

Given the ability to recognize with the paraphrase grammar, a question which we hope to be able to investigate empirically is the effect that entrainment from exposure to the longer and less natural paraphrases actually has on user language, which initially tends to be biased towards short, natural-sounding utterances, with frequent use of ellipsis. This is a interesting topic for future research.

6 Conclusions

The dialogue move paraphrase mechanism provides a useful approach to streamlining regression testing without abandoning necessary detail. In non-trivial spoken dialogue systems, it is generally necessary to have a number of levels of representation. Our approach provides a middle ground between tracking each of these levels in the test suites, creating a excessive maintenance burden, and keeping only top level inputs and outputs, which is too coarse-grained to catch many errors. The Regulus framework provides the opportunity to implement this mechanism as a Regulus grammar, which makes the compilation into recognisers, parsers and generators available. While generation with the paraphrase grammar supports the described improvement in regression testing methodology, compiling the paraphrase grammar into a recogniser allows us to ensure that paraphrases used as confirmations can also be processed if directed at the dialogue system. The framework has been used with several fairly different kinds of applications, and appears to have a major impact on the overhead associated with maintenance of a useful regression testing regime.

References

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. 2000. An architecture for a generic dialogue shell. *Natural Language Engineering, Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 1–16.

P. Bouillon, M. Rayner, B. Novellas, M. Starlander, M. Santaholma, Y. Nakao, and N. Chatzichrisafis. 2007. Une grammaire partagée multi-tâche pour le traitement de la parole: application aux langues romanes. *TAL*.

P. Bouillon, G. Flores, M. Georgescul, S. Halimi, B.A. Hockey, H. Isahara, K. Kanzaki, Y. Nakao, M. Rayner, M. Santaholma, M. Starlander, and N. Tsourakis. 2008. Many-to-many multilingual medical speech translation on a PDA. In *Proceedings of The Eighth Conference of the Association for Machine Translation in the Americas*, Waikiki, Hawaii.

Milica Gasić, Simon Keizer, Francois Mairesse, Jost Schatzmann, Blaise Thomson, Kai Yu, and Steve Young. 2008. Training and evaluation of the his pomdp dialogue system in noise. In *Proceedings of the 9th SIGDIAL Workshop on Discourse and Dialogue*.

Maria Georgescul, Manny Rayner, Pierrette Bouillon, and Nikos Tsourakis. 2008. Discriminative learning using linguistic features to rescore n-best speech hypotheses. In *The IEEE Workshop on Spoken Language Technology*, Goa, India.

S. Larsson and D. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340.

M. Rayner, B.A. Hockey, J.M. Renders, N. Chatzichrisafis, and K. Farrell. 2005. A voice enabled procedure browser for the international space station. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (interactive poster and demo track)*, Ann Arbor, MI.

M. Rayner, B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.

M. Rayner. 1988. Applying explanation-based generalization to natural-language processing. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 1267–1274, Tokyo, Japan.

N. Roy, J Pineau, and S. Thrun. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of ACL*, Hong Kong.

N. Tsourakis, M. Georghescul, P. Bouillon, and M. Rayner. 2008. Building mobile spoken dialogue applications using regulus. In *Proceedings of LREC 2008*, Marrakesh, Morocco.

T. van Harmelen and A. Bundy. 1988. Explanation-based generalization = partial evaluation (research note). *Artificial Intelligence*, 36:401–412.

JD Williams and SJ Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*.

B. Zhang, Q Cai, J. Mao, E. Chang, and B Guo. 2001. Spoken dialogue management as planning and acting under uncertainty. In *Proceedings of Eurospeech*, Aalborg, Denmark.