The CommandTalk Spoken Dialogue System*

Amanda Stent, John Dowding
Jean Mark Gawron, Elizabeth Owen Bratt, and Robert Moore
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
{stent,dowding,gawron,owen,bmoore}@ai.sri.com

1 Introduction

CommandTalk (Moore et al., 1997) is a spoken-language interface to the ModSAF battlefield simulator that allows simulation operators to generate and execute military exercises by creating forces and control measures, assigning missions to forces, and controlling the display (Ceranowicz, 1994). CommandTalk consists of independent, cooperating *agents* interacting through SRI's Open Agent Architecture (OAA) (Martin et al., 1998). This architecture allows components to be developed independently, and then flexibly and dynamically combined to support distributed computation. Most of the agents that compose CommandTalk have been described elsewhere (for more detail, see (Moore et al., 1997)). This paper describes extensions to CommandTalk to support spoken dialogue. While we make no theoretical claims about the nature and structure of dialogue, we are influenced by the theoretical work of (Grosz and Sidner, 1986) and will use terminology from that tradition when appropriate. We also follow (Chu-Carroll and Brown, 1997) in distinguishing *task initiative* and *dialogue initiative.*

Section 2 demonstrates the dialogue capabilities of CommandTalk by way of an extended example. Section 3 describes how language in CommandTalk is modeled for understanding and generation. Section 4 describes the architecture of the dialogue manager in detail. Section 5 compares CommandTalk with other spoken dialogue systems.

* This research was supported by the Defense Advanced Research Projects Agency under Contract N66001-94-C-6046 with the Space and Naval Warfare Systems Center. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency of the U.S. Government.

2 Example Dialogues

The following examples constitute a single extended dialogue illustrating the capabilities of the dialogue manager with regard to structured dialogue, clarification and correction, changes in initiative, integration of speech and gesture, and sensitivity to events occurring in the underlying simulated world. [1]

Ex. 1: Confirmation

U	1	Create a point named Checkpoint 1 at 64 53
S	2	⊙
U	3	Create a CEV at Checkpoint 1
S	4	⊙
U	5	Create a CEV here <*click*>
S	6	⊙ I will create CEV at FQ 643 576

Utterances 1 and 3 illustrate typical successful interactions between an operator and the system. When no exceptional event occurs, CommandTalk does not respond verbally. However, it does provide an audible tone to indicate that it has completed processing. For a successful command, it produces a rising tone, illustrated by the ⊙ symbol in utterances 2 and 4. For an unsuccessful command it produces a falling tone, illustrated by the ⊗ symbol in utterances 12 and 14. [2]

[1] **U** indicates a user utterance as recognized by the system, and **S** indicates a system utterance. Bracketed items in italics are gestures.

[2] The rising and falling tones come from the prediaLogue version of the system. They were added because we observed that users did not always notice when the system had not understood them correctly, and a textual error message alone did not always get the user's attention. These tones also perform basic grounding behavior.

Utterance 6 demonstrates a case where, although the system successfully completed the command, it chose to provide an explicit confirmation. Explicit confirmations can be given at any time. In this case, the system chose to give the confirmation because it performed a nontrivial reference, resolving "here" to the map coordinates given by the gesture, FQ 643 576. Similar situations in which the system gives an explicit confirmation are the resolution of pronouns and elided, definite or plural noun phrases.

Ex. 2: Correction

U	7	Put Objective Golf here <*click*>
S	8	⊙ I will locate Objective Golf at FQ 658 583
U	9	I meant Objective Alpha
S	10	⊙ I will locate Objective Alpha at FQ 658 583

Example 2 contains a correction sub-dialogue. Utterance 9 is a correction of utterance 7, and is interpreted as though the operator had said "Put Objective Alpha here". This illustrates two points. First, since utterance 7 was successful, the system undoes its effects (that is, deletes Objective Golf) before creating Objective Alpha. Second, although the edited utterance contains the word "here", the gesture that was used to resolve that is no longer available. The system keeps track of gestural information along with linguistic information in its representation of context in order to interpret corrections.

Ex. 3: Structured Sub-dialogue

U	11	CEV advance to Objective Golf
S	12	⊗ There are 2 CEVs. Do you mean 100A11 or 100A12?
U	13	100A11
S	14	⊗ There is no point named Objective Golf. Which point should 100A11 proceed to?
U	15	I said Objective Alpha
S	16	⊙ 100A11 will proceed to Objective Alpha

Example 3 illustrates a structured discourse segment containing two subsegments. Utterance 11 is uninterpretable for two reasons: the reference to "CEV" is ambiguous, and Objective Golf does not exist. The first difficulty is resolved in discourse segment 12-13, and the second in discourse segment 14-16. Notice that the operator is not required to answer the question posed by the system in utterance 14, but is free to correct the system's misunderstanding of utterance 11 even though it is not the immediately prior utterance. This is true because utterance 13 (the most recent utterance) is interpreted as if the operator had said "100A11 advance to Objective Golf".

Ex. 4: Required Argument

U	17	A12 conduct crater breach facing north
S	18	⊗ Which point should 100A12 conduct crater breach oriented north at?
U	19	<*click*>
S	20	⊙ 100A12 will conduct crater breach oriented north at FQ 542 595

Example 4 demonstrates a case where, although there are no errors in the operator's utterance, the system requires additional information before it can execute the command. Also note that the question asked by the system in utterance 18 is answered with an isolated gesture.

Ex. 5: Delayed Response

U	21	A13 continue to Checkpoint 1 in a column formation.
S	22	⊗ There is no A13. Which unit should proceed in a column formation to Checkpoint 1?
U	23	What units are there?
S	24	⊙ There are 2 units: 100A11 and 100A12.
U	25	A11 proceed to Checkpoint 1.
S	26	⊙ 100A11 will proceed in a column formation to Checkpoint 1.

In example 5, the system asks a question but the operator needs to perform some other activity before answering it. The question asked by the system in utterance 22 is answered by the operator in utterance 25. Due to the intervening material, the most natural way to answer the question posed in utterance 22 is with a

complete sentence, rather than an isolated noun phrase.

Ex. 6: Changes in the Simulated World

U	27	A12 advance to Objective Bravo.
S	28	⊗ There is no Objective Bravo. What point should 100A12 advance to?
U	29	Create it at 635 545.
S	30	⊙
S	31	Should 100A12 proceed to Objective Bravo?
U	32	Yes.
S	33	⊙ 100A12 will proceed to Objective Bravo.

Example 6 demonstrates the use of a *guard*, or test to see if a situation holds. In utterance 27, a presupposition failure occurs, leading to the open proposition expressed in utterance 28. A guard, associated with the open proposition, tests to see if the system can successfully resolve "Objective Bravo". Rather than answering the question in utterance 28, the operator chooses to create Objective Bravo. The system then tests the guard, which succeeds because Objective Bravo now exists. The system therefore takes dialogue initiative by asking the operator in utterance 31 if that operator would like to carry out the original command. Although, in this case, the simulated world changed in direct response to a linguistic act, in general the world can change for a variety of reasons, including the operator's activities on the GUI or the activities of other operators.

3 Language Interpretation and Generation

The language used in CommandTalk is derived from a single grammar using Gemini (Dowding et al., 1993), a unification-based grammar formalism. This grammar is used to provide all the language modeling capabilities of the system, including the language model used in the speech recognizer, the syntactic and semantic interpretation of user utterances (Dowding et al., 1994), and the generation of system responses (Shieber et al., 1990).

For speech recognition, Gemini uses the Nuance speech recognizer. Nuance accepts language models written in a Grammar Specification Language (GSL) format that allows context-free, as well as the more commonly used finite-state, models.[3] Using a technique described in (Moore, 1999), we compile a context-free covering grammar into GSL format from the main Gemini grammar.

This approach of using a single grammar source for both sides of the dialogue has several advantages. First, although there are differences between the language used by the system and that used by the speaker, there is a large degree of overlap, and encoding the grammar once is efficient. Second, anecdotal evidence suggests that the language used by the system influences the kind of language that speakers use in response. This gives rise to a consistency problem if the language models used for interpretation and generation are developed independently.

The grammar used in CommandTalk contains features that allow it to be partitioned into a set of independent top-level grammars. For instance, CommandTalk contains related, but distinct, grammars for each of the four armed services (Army, Navy, Air Force, and Marine Corps). The top-level grammar currently in use by the speech recognizer can be changed dynamically. This feature is used in the dialogue manager to change the top-level grammar, depending on the state of the dialogue. Currently in CommandTalk, for each service there are two main grammars, one in which the user is free to give any top-level command, and another that contains everything in the first grammar, plus isolated noun phrases of the semantic types that can be used as answers to wh-questions, as well as answers to yes/no questions.

3.1 Prosody

A separate Prosody agent annotates the system's utterances to provide cues to the speech synthesizer about how they should be produced. It takes as input an utterance to be spoken, along with its parse tree and logical form. The output is an expression in the Spoken Text Markup Language[4] (STML) that annotates the locations and lengths of pauses and the locations of pitch changes.

[3] GSL grammars that are context-free cannot contain indirect left-recursion.

[4] See `http://www.cstr.ed.ac.uk/projects/ssml.html` for details.

3.2 Speech Synthesis

Speech synthesis is performed by another agent that encapsulates the Festival speech synthesizer. Festival[5] was developed by the Centre for Speech Technology Research (CSTR) at the University of Edinburgh. Festival was selected because it accepts STML commands, is available for research, educational, and individual use without charge, and is open-source.

4 Dialogue Manager

The role of the dialogue manager in CommandTalk is to manage the representation of linguistic context, interpret user utterances within that context, plan system responses, and set the speech recognition system's language model. The system supports natural, structured mixed-initiative dialogue and multimodal interactions.

When interpreting a new utterance from the user, the dialogue manager considers these possibilities in order:

1. Corrections: The utterance is a correction of a prior utterance.
2. Transitions/Responses: The utterance is a continuation of the current discourse segment.
3. New Commands/Questions: The utterance is initiating a new discourse segment.

The following sections will describe the data structures maintained by the dialogue manager, and show how they are affected as the dialogue manager processes each of these three types of user utterances.

4.1 Dialogue Stack

CommandTalk uses a dialogue stack to keep track of the current discourse context. The dialogue stack attempts to keep track of the open discourse segments at each point in the dialogue. Each stack frame corresponds to one user-system discourse pair, and contains at least the following elements:

- an atomic *dialogue state* identifier (see Section 4.2)
- a semantic representation of the user's utterance(s)
- a semantic representation of the system's response, if any
- a representation of the background (i.e., open proposition) for the anticipated user response.
- focus spaces containing semantic representations of the items referred to in each system and user utterance
- a gesture space containing the gestures used in the interpretation of each user utterance
- an optional guard

The semantic representation of the system response is related to the background, but there are cases where the background may contain more information than the response. For example, in utterance 28 the system could have simply said "There is no Objective Bravo", and omitted the explicit follow-up question. In this case, the background may still contain the open proposition.

Unlike in dialogue analyses carried out on completed dialogues (Grosz and Sidner, 1986), the dialogue manager needs to maintain a stack of all open discourse segments at each point in an *on-going* dialogue. When a system allows corrections, it can be difficult to determine when a user has completed a discourse segment.

Ex. 7: Consecutive Corrections

U	34	Center on Objective Charlie
S	35	⊗ There is no point named Objective Charlie. What point should I center on?
U	36	95 65
S	37	⊙ I will center on FQ 950 650
U	38	I said 55 65
S	39	⊙ I will center on FQ 550 650

In example 7, for instance, when the user answers the question in utterance 36, the system will pop the frame corresponding to utterances 34-35 off the stack. However, the information in that frame is necessary to properly interpret the correction in utterance 38. Without some other mechanism it would be unsafe to ever pop a

[5] See `http://www.cstr.ed.ac.uk/projects/festival.html` for full information on Festival.

frame from the stack, and the stack would grow indefinitely. Since the dialogue stack represents our best guess as to the set of currently open discourse segments, we want to allow the system to pop frames from the stack when it believes discourse segments have been closed. We make use of another representation, the *dialogue trail*, to let us to recover from these moves if they prove to be incorrect.

The dialogue trail acts as a history of all dialogue stack operations performed. Using the trail, we record enough information to be able to restore the dialogue stack to any previous configuration (each trail entry records one operation taken, the top of the dialog stack before the operation, and the top of the dialog stack after). Unlike the stack, the dialogue trail represents the entire history of the dialogue, not just the set of currently open propositions. The fact that the dialogue trail can grow arbitrarily long has not proven to be a problem in practice since the system typically does not look past the top item in the trail.

4.2 Finite State Machines

Each stack frame in the dialogue manager contains a unique *dialogue state* identifier. These states form a collection of finite-state machines (FSMs), where each FSM describes the turns comprising a particular discourse segment. The dialogue stack is reminiscent of a recursive transition network, in that the stack records the system's progress through a series of FSMs in parallel. However, in this case, the stack operations are not dictated explicitly by the labels on the FSMs, but stack push operations correspond to the onset of a discourse segment, and stack pop operations correspond to the conclusion of a discourse segment.

Most of the FSMs currently used in CommandTalk coordinate dialogue initiative. These FSMs have a very simple structure of at most two states. For instance, there are FSMs representing discourse segments for clarification questions (utterances 23-24), reference failures (utterances 27-28), corrections (utterances 9-10), and guards becoming true (utterances 31-33). CommandTalk currently uses 22 such small FSMs. Although they each have a very simple structure, they compose naturally to support more complex dialogues. In these sub-dialogues the user retains the task initiative, but the system may temporarily take the dialogue initiative. This set of FSMs comprises the core dialogue competence of the system.

In a similar way, more complex FSMs can be designed to support more structured dialogues, in which the system may take more of the task initiative. The additional structure imposed varies from short 2-3 turn interactions to longer "form-filling" dialogues. We currently have three such FSMs in CommandTalk:

- The Embark/Debark command has four required parameters; a user may have difficulty expressing them all in a single utterance. CommandTalk will query the user for missing parameters to fill in the structure of the command.
- The Infantry Attack command has a number of required parameters, a potentially unbounded number of optional parameters, and some constraints between optional arguments (e.g., two parameters are each optional, but if one is specified then the other must be also).
- The Nine Line Brief is a straight-forward form-filling command with nine parameters that should be provided in a specified order.

When the system interprets a new user utterance that is not a correction, the next alternative is that it is a continuation of the current discourse segment. Simple examples of this kind of transition occur when the user is answering a question posed by the system, or when the user has provided the next entry in a form-filling dialogue. Once the transition is recognized, the current frame on top of the stack is popped. If the next state is not a final state, then a new frame is pushed corresponding to the next state. If it is a final state, then a new frame is not created, indicating the end of the discourse segment.

The last alternative for a new user utterance is that it is the onset of a new discourse segment. During the course of interpretation of the utterance, the conditions for entering one or more new FSMs may be satisfied by the utterance. These conditions may be linguistic, such as presupposition failures, or can arise from events that occur in the simulation, as when a guard

is tested in example 6. Each potential FSM has a corresponding priority (error, warning, or good). An FSM of the highest priority will be chosen to dictate the system's response.

One last decision that must be made is whether the new discourse segment is a subsegment of the current segment, or if it should be a sibling of that segment. The heuristic that we use is to consider the new segment a subsegment if the discourse frame on top of the stack contains an open proposition (as in utterance 23). In this case, we push the new frame on the stack. Otherwise, we consider the previous segment to now be closed (as in utterance 3), and we pop the frame corresponding to it prior to pushing on the new frame.

4.3 Mechanisms for Reference

CommandTalk employs two mechanisms for maintaining local context and performing reference: a list of salient objects in the simulation, and focus spaces of linguistic items used in the dialogue.

Since CommandTalk is controlling a distributed simulation, events can occur asynchronously with the operator's linguistic acts, and objects may become available for reference independently of the on-going dialogue. For instance, if an enemy unit suddenly appears on the operator's display, that unit is available for immediate reference, even if no prior linguistic reference to it has been made. The ModSAF agent notifies the dialogue manager whenever an object is created, modified, or destroyed, and these objects are stored in a salience list in order of recency. The salience list can also be updated when simulation objects are referred to using language.

The salience list is not part of the dialogue stack. It does not reflect attentional state; rather, it captures recency and "known" information.

While the salience list contains only entities that directly correspond to objects in the simulation, focus spaces contain representations of entities realized in linguistic acts, including objects not directly represented in the simulation. This includes objects that do not exist (yet), as in "Objective Bravo" in utterance 28, which is referred to with a pronoun in utterance 29, and sets of objects introduced by plural noun phrases. All items referred to in an utterance are stored in a focus space associated with that utterance in the stack frame. There is one focus space per utterance.

Focus spaces can be used during the generation of pronouns and definite noun phrases. Although at present CommandTalk does not generate pronouns (we choose to err on the side of verbosity, to avoid potential confusion due to misrecognitions), focus spaces could be used to make intelligent decisions about when to use a pronoun or a definite reference. In particular, while it might be dangerous to generate a pronoun referring to a noun phrase that the user has used, it would be appropriate to use a pronoun to refer to a noun phrase that the system has used.

Focus spaces are also used during the interpretation of responses and corrections. In these cases the salience list reflects what is known now, not what was known at the time the utterance being corrected or clarified was made. The focus spaces reflect what was known and in focus at that earlier time; they track attentional state. For instance, imagine example 6 had instead been:

Ex. 6b: Focusing

U	40	A14 advance there.
S	41	⊗ There is no A14. Which unit should advance to Checkpoint 1?
U	42	Create CEV at 635 545 and name it A14.

At the end of utterance 42 the system will reinterpret utterance 40, but the most recent location in the salience list is FQ 635 545 rather than Checkpoint 1. The system uses the focus space to determine the referent for "there" at the time utterance 40 was originally made.

In conclusion, CommandTalk's dialogue manager uses a dialogue stack and trail, reference mechanisms, and finite state machines to handle a wide range of different kinds of dialogue, including form-filling dialogues, free-flowing mixed-initiative dialogues, and dialogues involving multi-modality.

5 Related Work

CommandTalk differs from other recent spoken language systems in that it is a command and control application. It provides a particularly

interesting environment in which to design spoken dialogue systems in that it supports distributed stochastic simulations, in which one operator controls a certain collection of forces while other operators simultaneously control other allied and/or opposing forces, and unexpected events can occur that require responses in real time. Other applications (Litman et al., 1998; Walker et al., 1998) have been in domains that were sufficiently limited (e.g., queries about train schedules, or reading email) that the system could presume much about the user's goals, and make significant contributions to task initiative. However, the high number of possible commands available in CommandTalk, and the more abstract nature of the user's high-level goals (to carry out a simulation of a complex military engagement) preclude the system from taking significant task initiative in most cases.

The system most closely related to CommandTalk in terms of dialogue use is TRIPS (Ferguson and Allen, 1998), although there are several important differences. In contrast to TRIPS, in CommandTalk gestures are fully incorporated into the dialogue state. Also, CommandTalk provides the same language capabilities for user and system utterances.

Unlike other simulation systems, such as QuickSet (Cohen et al., 1997), CommandTalk has extensive dialogue capabilities. In QuickSet, the user is required to confirm each spoken utterance before it is processed by the system (McGee et al., 1998).

Our earlier work on spoken dialogue in the air travel planning domain (Bratt et al., 1995) (and related systems) interpreted speaker utterances in context, but did not support structured dialogues. The technique of using dialogue context to control the speech recognition state is similar to one used in (Andry, 1992).

6 Future Work

We have discussed some aspects of CommandTalk that make it especially suited to handle different kinds of interactions. We have looked at the use of a dialogue stack, salience information, and focus spaces to assist interpretation and generation. We have seen that structured dialogues can be represented by composing finite-state models. We have briefly discussed the advantages of using the same grammar for all linguistic aspects of the system. It is our belief that most of the items discussed could easily be transferred to a different domain.

The most significant difficulty with this work is that it has been impossible to perform a formal evaluation of the system. This is due to the difficulty of collecting data in this domain, which requires speakers who are both knowledgeable about the domain and familiar with ModSAF. CommandTalk has been used in simulations of real military exercises, but those exercises have always taken place in classified environments where data collection is not permitted.

To facilitate such an evaluation, we are currently porting the CommandTalk dialogue manager to the domain of air travel planning. There is a large body of existing data in that domain (MADCOW, 1992), and speakers familiar with the domain are easily available.

The internal representation of actions in CommandTalk is derived from ModSAF. We would like to port that to a domain-independent representation such as frames or explicit representations of plans.

Finally, there are interesting options regarding the finite state model. We are investigating other representations for the semantic contents of a discourse segment, such as frames or active templates.

7 Acknowledgments

We would like to thank Andrew Kehler, David Israel, Jerry Hobbs, and Sharon Goldwater for comments on an earlier version of this paper, and we have benefited from the very helpful comments from several anonymous reviewers.

References

F. Andry. 1992. Static and Dynamic Predictions: A Method to Improve Speech Understanding in Cooperative Dialogues. In *Proceedings of the International Conference on Spoken Language Processing*, Banff, Canada.

H. Bratt, J.Dowding, and K. Hunicke-Smith. 1995. The SRI Telephone ATIS System. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 218–220, Austin, Texas.

A. Ceranowicz. 1994. Modular Semi-Automated Forces. In J.D. Tew et al.,

editor, *Proceedings of the Winter Simulation Conference*, pages 755–761.

J. Chu-Carroll and M. Brown. 1997. Tracking Initiative in Collaborative Dialogue Interactions. In *Proceedings of the Thirty-Fifth Annual Meeting of the ACL and 8th Conference of the European Chapter of the ACL*, Madrid, Spain.

P. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. 1997. QuickSet: Multimodal Interaction for Distributed Applications. In *Proceedings of the Fifth Annual International Multimodal Conference*, Seattle, WA.

J. Dowding, J. Gawron, D. Appelt, L. Cherny, R. Moore, and D. Moran. 1993. Gemini: A Natural Language System for Spoken Language Understanding. In *Proceedings of the Thirty-First Annual Meeting of the ACL*, Columbus, OH. Association for Computational Linguistics.

J. Dowding, R. Moore, F. Andry, and D. Moran. 1994. Interleaving Syntax and Semantics in an Efficient Bottom-Up Parser. In *Proceedings of the Thirty-Second Annual Meeting of the ACL*, Las Cruces, New Mexico. Association for Computational Linguistics.

G. Ferguson and J. Allen. 1998. TRIPS: An Intelligent Integrated Problem-Solving Assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI.

B. Grosz and C. Sidner. 1986. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, 12(3):175–204.

D. Litman, S. Pan, and M. Walker. 1998. Evaluating Response Strategies in a Web-Based Spoken Dialogue Agent. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 780–786, Montreal, Canada.

MADCOW. 1992. Multi-Site Data Collection for a Spoken Language Corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 200–203, Harriman, New York.

D. Martin, A. Cheyer, and D. Moran. 1998. Building Distributed Software Systems with the Open Agent Architecture. In *Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, Blackpool, Lancashire, UK. The Practical Application Company Ltd.

D. McGee, P. Cohen, and S. Oviatt. 1998. Confirmation in Multimodal Systems. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 823–829, Montreal, Canada.

R. Moore, J. Dowding, H. Bratt, J. Gawron, Y. Gorfu, and A. Cheyer. 1997. CommandTalk: A Spoken-Language Interface for Battlefield Simulations. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 1–7, Washington, DC. Association for Computational Linguistics.

R. Moore. 1999. Using Natural Language Knowledge Sources in Speech Recognition. In Keith Ponting, editor, *Speech Pattern Processing*. Springer-Verlag.

S. M. Shieber, G. van Noord, R. Moore, and F. Pereira. 1990. A Semantic Head-Driven Generation Algorithm for Unification-Based Formalisms. *Computational Linguistics*, 16(1), March.

M. Walker, J. Fromer, and S. Narayanan. 1998. Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 1345–1351, Montreal, Canada.