Available online at www.sciencedirect.com
SCIENCE DIRECT®
Speech Communication 48 (2006) 335–353
SPEECH COMMUNICATION
www.elsevier.com/locate/specom

Robust spoken language understanding in a computer game

Johan Boye *, Joakim Gustafson, Mats Wirén

TeliaSonera R&D, Rudsjöterrassen 2, 13680 Haninge, Sweden

Received 22 December 2004; received in revised form 23 June 2005; accepted 27 June 2005

Abstract

We present and evaluate a robust method for the interpretation of spoken input to a conversational computer game. The scenario of the game is that of a player interacting with embodied fairy-tale characters in a 3D world via spoken dialogue (supplemented by graphical pointing actions) to solve various problems. The player himself cannot directly perform actions in the world, but interacts with the fairy-tale characters to have them perform various tasks, and to get information about the world and the problems to solve. Hence the role of spoken dialogue as the primary means of control is obvious and natural to the player. Naturally, this means that robust spoken language understanding becomes a critical component. To this end, the paper describes a semantic representation formalism and an accompanying parsing algorithm which works off the output of the speech recogniser's statistical language model. The evaluation shows that the parser is robust in the sense of considerably improving on the noisy output of the speech recogniser.

Keywords: Spoken language understanding; Robust parsing; Robustness; Dialogue systems; Conversational systems; Computer games; Animated characters

1. Introduction

Computer games provide an excellent application area for research in spoken dialogue technology, requiring an advance of the state-of-the-art on several fronts. Speech input is already used in some commercial computer games as a supplement to the mouse and keyboard, but to date very few commercial games are using voice commands as the primary means of control (*Lifeline*, released in 2004, is one example). More advanced spoken dialogue would have the potential to greatly enrichen computer games. For example, it would allow players to refer to past events and to objects currently not visible on the screen, as well as interacting socially and negotiating solutions with the game characters.

A problem which has to be overcome when designing and implementing such a game is to achieve an acceptable level of spoken input understanding, while at the same time giving the player

* Corresponding author. Tel.: +46 70 5866724.
E-mail address: johan.boye@teliasonera.com (J. Boye).

0167-6393/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.specom.2005.06.015

the impression that he can express himself freely. In order to maximise recognition performance, the only viable option is to use a statistical language model, trained on input from as many users as possible. But then it is necessary to have a robust method of extracting the meaning from the word strings delivered by the speech recogniser, to handle disfluent input and recognition errors.

This paper describes the methods for spoken language interpretation used in the NICE fairy-tale game. The game scenario of the fairy-tale game is basically that of a player interacting with embodied fairy-tale characters in a 3D world via spoken dialogue (in Swedish), as well as graphical gestures via a mouse-compatible input device, in order to solve various problems. The fairy-tale characters communicate with the player using spoken dialogue and gestures. The appearances of the characters, their voices, actions and ways of expressing themselves all contribute to giving the player the impression of fairy-tale characters with distinct personalities. The game is intended for young users (9–15-year olds), and the development of the game has been highly iterative. Several versions of the game have been tried on young users, upon which the data has been analysed and used to improve all aspects of the game.

The robust parsing algorithm, which is the main subject of this paper, proceeds in two steps: a domain-dependent pattern-matching phase and a domain-*independent* rewriting phase. The output of the parser is a typed, tree-structured expression representing the utterance. Previous systems based on pattern matching have been restricted to producing relatively simple semantic structures, such as variable-free slot-filler lists. Unfortunately, such structures are not suitable as input to a dialogue manager in our domain, which involves information-seeking utterances, commands and simple negotiation, and where there is also abundant reference to objects in the 3D world as well as in the discourse. Thus, our system produces more complex semantic structures, tailored to capture the kind of information contained in utterances collected in our domain. Still, our semantic structures are much less complex than general-purpose, logic-based approaches, thereby allowing for efficient and robust processing. Our evaluation also shows that the parser is robust in the sense of considerably improving on the noisy output of the speech recogniser (see Section 5).

In sum, the contribution of the paper is a novel combination of pattern-matching and rewriting which allows for a trade-off between the simple semantic structures typically generated by pattern-matching parsers and the complex structures generated by general-purpose, linguistically-based parsers. In particular, this trade-off allows us to retain the advantages of pattern-matching systems in terms of efficiency and robustness, while capturing the contents of the great majority of utterances manifested in our domain.

2. Game scenario

The scenario and characters are loosely inspired by the fairy-tale universe of H.C. Andersen. The game begins in H. C. Andersen's house in Copenhagen in the 19th century. Andersen has just left on a trip, and has asked one of his fairy-tale characters, Cloddy Hans, to guard his fairy-tale laboratory while he is away. The key device in the laboratory is a fairy-tale machine, which nobody except Andersen himself is allowed to touch (Fig. 1). On a set of shelves beside the machine, various objects are located, such as a key, a hammer, a diamond and a magic wand. By removing objects from the shelves, putting them into suitable slots in the machine and pulling a lever, one lets the machine construct a new fairy-tale in which the objects come to life.

Just before the user enters the game, Cloddy Hans has got the idea of surprising H. C. Andersen with a new fairy-tale on his coming back. There is a problem, however: Each slot is labelled with a symbol which tells which type of object is supposed to go there, but since Cloddy Hans is not very bright, he needs help from the user with understanding these. There are four slots, which are labelled with symbols denoting "useful", "magical", "precious" and "dangerous" things, respectively. Which object goes in which slot is sometimes more obvious (provided you understand the symbols), like the diamond belonging in "precious", and sometimes less obvious, like

Fig. 1. The first scene: Cloddy Hans standing beside the shelves with objects, and in front of the fairy-tale machine.

the knife belonging in "useful" rather than "dangerous".

The first scene thus develops into a kind of "put-that-there" game, where it is the task of the user to instruct Cloddy Hans; tell him where to go, which objects to pick up and where to put them down, etc. If the user does not understand what to say, Cloddy Hans will encourage him or her, give suggestions, and eventually take matters into own hands. Because the initial scene is task-oriented in a straightforward way, the system is able to anticipate what the user will have to say to solve it. The real purpose is not to solve the task, but to engage in a collaborative conversation where the player familiarises himself with the possibilities and limitations of the spoken (multimodal) input capabilities.

In the second scene, the player enters the actual fairy-tale world for the first time, together with Cloddy Hans. The fairy-tale world is a large 3D virtual world (parts of it can be seen in Fig. 2). At the beginning of the second scene, Cloddy Hans encourages the player to explore the immediate surroundings on the small island. While wandering about and looking around, the player discovers that the objects that were put in the fairy-tale machine in the preceding scene are now lying scattered in the grass. Although it is not completely clear to the player at this point, these objects will actually constitute valuable assets when solving various tasks in the world.

The player soon encounters the first problem. Together with Cloddy Hans, he is trapped on a small island, from which he can see the marvels of the fairy-tale world—houses, fields, a wind mill, and many more things—but they are all out of reach. A deep gap separates him from these wonders. There is a drawbridge, which can be used for the crossing, but it is open, and the handle that operates it is on the other side. Fortunately, a girl, Karen, is standing on the other side (Fig. 3).

Karen has a different kind of personality compared to Cloddy Hans. Instead of having Cloddy Hans's positive attitude, she is sullen and uncooperative, and refuses to close the drawbridge. The key to solving this deadlock is for the player to find out that Karen will comply if she is paid: she wants to have one of the fairy-tale objects that are lying in the grass on the player's side of the gap (which object she wants will change each time the game is restarted). Thus, it is the task of the player to find the appropriate object, and use this object to bargain with Karen.

The following is a manual transcription of an excerpt from a dialogue between a user, Cloddy Hans and Karen. When the excerpt begins, Cloddy Hans and the user are standing at one side of the

Fig. 2. The second scene: a small part of the fairy-tale world. The player and Cloddy Hans start off on the small island on the left-hand side.

Fig. 3. The second scene: Cloddy Hans and Karen at the gap and the open drawbridge.

gap, while the camera is showing Karen standing on the other. The user is trying to convince Karen to lower the bridge, while Cloddy Hans is making some side comments, triggered by Karen's utterances.

Player: Open the bridge ... could you lower the elevator?
Karen: If I am to lower the bridge, I want something in return.
Cloddy: Maybe she will lower the bridge if we give her something nice.
Player: Yes take this take Cloddy's sword.
Karen: If you can't offer something better then forget it ... I saw some shiny stuff fall down from the sky a while ago.
Player: ... yes ...
Cloddy: Girls like shiny things, don't they?
Player: OK Cloddy, fetch the lamp then.
(*The camera swings back to show Cloddy Hans.*)
Cloddy: I can only hold one object at a time.
Player: OK, drop the sword.
(*Cloddy Hans drops the sword on the ground.*)

Cloddy: Do you want me to go to the lamp?
Player: Yes.
Cloddy: OK, I'll do that.
(*Turns around and walks over to the lamp.*)
Player: Pick up the lamp.
Cloddy: OK, I'll do that.
(*Picks up the lamp.*)
Player: Go to ehh what's her name ehh go to that girl ...
Cloddy: Do you want me to go to Karen?
Player: Yes, go to Karen.
Cloddy: I am already on my way.
(*Starts walking. When Cloddy Hans reaches the bridge, the system triggers on this event, and swings the camera over to show Karen.*)
Karen: What did you bring for me?
Player: You can have this lamp that Cloddy has.
Karen: What a piece of junk! Dump it in another fairy-tale.

The fairy-tale character with whom the player is talking is always in camera (i.e. is shown on the screen). The player can control the camera by saying the name of a character. For example, by saying "Cloddy", the camera swings over to show Cloddy Hans. This is also the way for the player to change dialogue partner.

The system can also initiate a camera change and a change of dialogue partner, by triggering on certain events. For instance, whenever Cloddy Hans reaches the gap, the camera automatically swings over to show Karen, and Karen starts talking. There is also a possibility for a character to make side-comments (without being in camera). In the example above, Cloddy Hans triggers on certain utterances by Karen to provide hints to the user ("Maybe she will lower the bridge if we give her something nice", "Girls like shiny things, don't they?").

3. Semantic representation formalism

3.1. Dialogue acts

Utterances are represented by tree-structured expressions, called *dialogue acts*. As an example, the dialogue act representing the user saying to Cloddy Hans; "Pick up the axe":

$$\mathsf{request}(\mathsf{user}, \mathsf{cloddy}, \mathsf{pickUp}(\mathsf{cloddy}, \mathsf{axe}))$$

Here, the topmost symbol (request) indicates the type of dialogue act, the first argument (user) indicates the character issuing the dialogue act, whereas the second argument (cloddy) indicates the intended recipient of the dialogue act. These components are present for all types of dialogue acts. The third component (pickUp(cloddy, axe), in this case) indicates the propositional contents of the dialogue act, in this case the action of picking up the axe. The general form of a request takes the form:

$$\mathsf{request}(\mathsf{x}^{\mathsf{character}}, \mathsf{y}^{\mathsf{character}}, \mathsf{z}^{\mathsf{action}})^{\mathsf{dialogueAct}}$$

where the superscripts indicate type constraints on the subexpressions. The pickUp action can be further decomposed into

$$\mathsf{pickUp}(\mathsf{x}^{\mathsf{character}}, \mathsf{y}^{\mathsf{thing}})^{\mathsf{action}}$$

i.e. the first argument must be a character (who is doing the picking up), and the second argument is a thing (which is picked up).

Anaphoric utterances are represented by means of typed lambda abstractions. For instance, consider the utterance "Pick it up". The meaning of this utterance is obviously depending on the context in which it is said (i.e. what "it" is referring to). Therefore it is reasonable to assert that the meaning of the utterance "Cloddy Hans, put it down on the table" is a function, mapping the relevant part of the dialogue context to an expression of the type dialogueAct. Thus:

$$\lambda \mathsf{y}^{\mathsf{thing}}.\mathsf{request}(\mathsf{user}, \mathsf{cloddy}, \mathsf{pickUp}(\mathsf{cloddy}, \mathsf{y}))$$

(We assume familiarity with the lambda calculus (see Hindley and Seldin, 1986), and its use in natural language semantics (see e.g. Jurafsky and Martin, 2000 chapter 15)). This expression denotes a function taking a thing as argument returning a character as the result (its type is written thing → dialogueAct). Functions of several arguments are represented with nested lambda abstractions, e.g. "Put it down" is

$$\lambda \mathsf{x}^{\mathsf{thing}} \lambda \mathsf{y}^{\mathsf{location}}.\mathsf{request}(\mathsf{user}, \mathsf{cloddy}, \mathsf{putDown}(\mathsf{cloddy}, \mathsf{x}, \mathsf{y}))$$

Domain questions are represented by means of ask expressions, e.g. "What color is the ruby?" is:

$\lambda x^{color}.ask(user, cloddy, x[ruby.color = x])$

Here the expression within square brackets indicates domain constraints imposed on the possible instantiations of x (in this case that x should be the color of the ruby).

Granting of information is represented by tell expressions, e.g. "I'm fourteen years old" is:

$tell(user, cloddy, 14[user.age = 14])$

The offer construction is used for bargaining, e.g. the user saying to Karen "I will give you the ruby" is

$offer(user, karen, ruby)$

Confirmations and disconfirmations are represented by confirm and disconfirm expressions, respectively, e.g. "Yes, do that" is:

$\lambda x^{dialogueAct}.confirm(user, cloddy, x)$

Requests for help and explanations are represented by askForSuggestion and askForExplanation expressions, respectively, e.g. "What should we do now?" is

$\lambda x^{dialogueAct}.askForSuggestion(user, cloddy, x)$

Fig. 4 summarizes the types of dialogue acts to which user input will be mapped in the fairy-tale game, and the types of their arguments. The type niceObject is a superset of all other types in the system.

The possible actions the system can reason about is listed in the table below. The first argument is always the character performing the action; the remainder of the arguments are the other role-players of the action:

Name	Argument structure
goTo	goTo($x^{character}$, y^{place})
pickUp	pickUp($x^{character}$, y^{thing})
putDown	putDown($x^{character}$, y^{thing}, $z^{location}$)
giveTo	giveTo($x^{character}$, y^{thing}, $z^{character}$)
raiseDrawbridge	raiseDrawbridge($x^{character}$)
lowerDrawbridge	lowerDrawbridge($x^{character}$)

Objects of other types (character, place, thing, location, etc.) are represented by argument-free terms (e.g. cloddy, knife, atMachine).

As seen above, the semantic expressions may also include expressions that constrain the set of possible values for a variable or a set of variables, for example:

$x.color = red$

Name	**Argument structure**
request	request($x^{character}$, $y^{character}$, z^{action})
ask	ask($x^{character}$, $y^{character}$, $z^{niceObject}$)
tell	tell($x^{character}$, $y^{character}$, $z^{niceObject}$)
offer	offer($x^{character}$, $y^{character}$, z^{thing})
confirm	confirm($x^{character}$, $y^{character}$, $z^{dialogueAct}$)
disconfirm	disconfirm($x^{character}$, $y^{character}$, $z^{dialogueAct}$)
askForAttention	askForAttention($x^{character}$, $y^{character}$)
askForSuggestion	askForSuggestion($x^{character}$, $y^{character}$, $z^{dialogueAct}$)
askForExplanation	askForExplanation($x^{character}$, $y^{character}$, $z^{dialogueAct}$)

Fig. 4. Types of user dialogue acts.

In general, if a is an expression of type t, and objects of type t have an attribute att of type s, and b is an expression of type s, then

$$a.att = b$$

is a well-formed constraint.

3.2. Contextual interpretation

As shown above, underspecified utterances are represented by means of lambda abstractions, where the lambda-bound variables act as placeholders for the missing information. The functional lambda expression representing the utterance "Put it down";

$$\lambda x^{thing} \lambda y^{location}.request(user, cloddy, putDown(cloddy, x, y))$$

has two missing pieces of information; the thing x to be put down, and the place y at which to put it down. The dialogue management component of the system is often able to retrieve such information from the preceding dialogue. Consider the dialogue excerpt:

1. *User*: "Cloddy Hans, please pick up the axe and go to the shelf."
2. *Cloddy Hans*: "OK, I'll do that" (*Picks up the axe and walks over to the shelf.)*
3. *User*: "Now put it down."

Here utterance 3 is represented by the lambda expression above. Utterance 1 is represented by a sequence of two expressions:

request(user, cloddy, pickUp(cloddy, axe))
request(user, cloddy, goTo(cloddy, shelf))

The missing information in utterance 3 can now be retrieved by searching the expressions representing utterance 1 for sub-expressions of the appropriate types. To obtain the final interpretation, the lambda expression of utterance 3 is then applied first to axe, and then to shelf, as follows:

((λxλy.putdown(cloddy, x, y) axe) shelf)→
(λy.putdown(cloddy, axe, y) shelf)→
putdown(cloddy, axe, shelf)

Ellipses are represented by means of higher-order functions. Consider the example:

1. *User*: "Cloddy Hans, please pick up the axe."
2. *Cloddy Hans*: "OK" (*picks up the axe)*
3. *User*: "Now the hammer".

In utterance 1, the user wants Cloddy Hans to do something with the hammer, but it is not possible to infer what dialogue act the user is performing without taking the dialogue context into account. Thus a context-independent representation of this utterance must represent the dialogue act by a function, as follows:

$$\lambda f^{thing \rightarrow dialogue\_act}.(f\ hammer)$$

The parameter f is to be bound to a function that takes as argument the information present in the utterance (hammer), and returns the appropriate dialogue act. Constructing this function is the task of the dialogue management component of the system. To this end, it uses a technique reminiscent of Dalrymple et al. (1991). In this example, the representations of preceding utterances are searched in reverse chronological order, to find an expression of type dialogue_act with a subexpression of type thing. In this case, the representation of utterance 1 is such an expression:

$$request(user, cloddy, pickUp(cloddy, axe))$$

Then functional abstraction (reverse functional application) yields an expression of the appropriate type thing → dialogue_act:

$$\lambda y^{thing}.request(user, cloddy, pickUp(cloddy, y))$$

This is actually the function we are looking for, since

(λf thing → dialogue_act.(f hammer) λy thing.request
(user, cloddy, pickUp(cloddy, y)))→
(λy thing.request(user, cloddy, pickUp(cloddy, y))
hammer) →
request(user, cloddy, pickUp(cloddy, hammer))

i.e. "Pick up the hammer".

4. Robust parsing

The robust parsing algorithm consists of two phases, a pattern matching phase and a rewriting phase. In the first phase, a string of words is scanned left-to-right, and a sequence of semantic constraints, triggered by syntactic patterns, are accumulated. The input to this phase is the 1-best hypothesis from the speech recognizer (for a discussion related to this, see Section 5.4). 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.

4.1. Semantic constraints

The most common kind of semantic constraint simply stipulates that the existence of certain objects of certain types can be inferred from the user's utterance. Such constraints are written on the form

$$\mathsf{object}^{\mathsf{type}}$$

For instance, the word "hammer" would trigger the constraint

$$\mathsf{hammer}^{\mathsf{thing}}$$

whereas the phrase "pick up" would trigger the following conjunctive constraints:

$$\mathsf{pickUp}(\mathsf{x},\mathsf{y})^{\mathsf{action}}, \mathsf{x}^{\mathsf{character}}, \mathsf{y}^{\mathsf{thing}}$$

Disequalities are used to express that two objects (of the same type) are necessarily different. For instance, the initial phrase "What is ..." indicates that the user is asking a question. Thus it results in the following list of constraints:

$$\mathsf{ask}(\mathsf{user},\mathsf{x},\mathsf{y})^{\mathsf{dialogue\_act}}, \mathsf{user}^{\mathsf{character}}, \mathsf{x}^{\mathsf{character}}, \quad \mathsf{x} \neq \mathsf{user}, \mathsf{y}^{\mathsf{t}}$$

Obviously, the user is asking someone else than himself; hence the disequality $\mathsf{x} \neq \mathsf{user}$. As "What is ..." does not give any clue to what the user is asking about, the type of the third argument is a variable t.

Equality constraints are used to relate objects with attributes of other objects. For example, the initial phrase "Where is ..." indicates that the user is enquiring about the position of some object. The list of constraints triggered by the syntactic pattern "Where is ..." is:

$$\mathsf{ask}(\mathsf{user},\mathsf{x},\mathsf{y})^{\mathsf{dialogue\_act}}, \mathsf{user}^{\mathsf{character}}, \mathsf{x}^{\mathsf{character}}, \quad \mathsf{x} \neq \mathsf{user}, \mathsf{y}^{\mathsf{location}}, \mathsf{y} = \mathsf{z}.\mathsf{position}, \mathsf{z}^{\mathsf{t}}$$

Here it is possible to infer that the object asked about is a location; hence the type of y is location rather than a variable t. Furthermore, it is assumed that this location is the position of some object z, whose type we do not know (and therefore its type is a variable t). However, z must be an object that has a position attribute.

4.2. Pattern-matching phase

The purpose of the pattern-matching phase is to generate a list of semantic constraints on the basis of the syntactic patterns that appear in the input. Such rules are coded by means of a definite clause grammar (see e.g. Sterling and Shapiro, 1994, Chap. 19), as illustrated by the following example[1]:

$$\mathsf{pickUp\_hints}([\mathsf{pickUp}(\mathsf{X},\mathsf{Y})^{\mathsf{action}}, \mathsf{X}^{\mathsf{character}}, \mathsf{Y}^{\mathsf{character}}|\mathsf{MoreHints}], \mathsf{Tail}) \rightarrow [\mathsf{take}, \mathsf{the}], \mathsf{thing\_hints}([\mathsf{Y}^{\mathsf{character}}|\mathsf{MoreHints}], \mathsf{Tail}).$$

$$\mathsf{pickUp\_hints}([\mathsf{pickUp}(\mathsf{X},\mathsf{Y})^{\mathsf{action}}, \mathsf{X}^{\mathsf{character}}, \mathsf{Y}^{\mathsf{character}}|\mathsf{Tail}], \mathsf{Tail}) \rightarrow [\mathsf{take}].$$

$$\mathsf{thing\_hints}([\mathsf{hammer}^{\mathsf{thing}}|\mathsf{Tail}], \mathsf{Tail}) \rightarrow [\mathsf{hammer}].$$

$$\mathsf{thing\_hints}([\mathsf{sword}^{\mathsf{thing}}|\mathsf{Tail}], \mathsf{Tail}) \rightarrow [\mathsf{sword}].$$

Basically, the algorithm consists in trying to match an initial segment of the input with the right-hand side of such a rule. The rules are tried in the order they are written. If a match is possible, the semantic constraints on the left-hand side are appended to the result list, the matched input segment is

[1] For these rules, we adopt the standard logic programming convention that expressions with an initial capital letter are variables.

discarded, and the process is repeated with the remaining input. If a match is not possible, the first word of the input is discarded, and the process is repeated with the remaining input.

For instance, suppose the input is "take the ehh hammer". The first rule is not applicable in this case because of the inserted "ehh", but the second rule is applicable, since the input begins with "take". The following two words ("the" and "ehh") are discarded as they do not match any rule. Finally, the last word "hammer" matches the third rule. The accumulated semantic constraints are:

$$\mathsf{pickUp}(x, y), x^{\mathsf{character}}, y^{\mathsf{thing}}, \mathsf{hammer}^{\mathsf{thing}}$$

In case the input is "take the hammer", without the inserted hesitation "ehh", the first rule matches the whole input string. In this case, the variable Y is set to hammer, and the output is:

$$\mathsf{pickUp}(x, \mathsf{hammer}), x^{\mathsf{character}}, \mathsf{hammer}^{\mathsf{thing}}$$

As can be seen from these examples, longer syntactic patterns are likely to convey more precise semantic information, but on the other hand they are more brittle, as the probability increases that recognition errors and disfluencies like "ehh" prevent matching. Moreover, longer patterns are less likely to occur in the input anyway. Therefore rules should be ordered as in the example, with longer patterns appearing before shorter patterns, so that the parser can capitalize on structure whenever present in the input, and degrade gracefully on noisy input.

Graphical pointing gestures also generate semantic constraints. If the user clicks on the hammer, the system's gesture recognizer contributes with the semantic constraint $\mathsf{hammer}^{\mathsf{thing}}$. Thus, an utterance "pick this up" accompanied by a click on the hammer results in the same list of constraints as above. The only limitation is that the click must not occur after the user has finished speaking (in which case the graphical input will be grouped with the next utterance instead).

In the example above, the presence of the filler word "ehh" made the parser miss the link between the hammer and the second argument of pickUp. However, this link will be recovered in the second phase of the parsing algorithm, presented next.

4.3. Rewriting phase

In the rewriting phase, the list of constraints aggregated in the pattern-matching phase is rewritten using four rewrite rules: *object merging, constraint inference, filtering* and *abstraction.*

4.3.1. Object merging

The first rewriting step, object merging, amounts to unifying objects of the same type. The rewriting rule can be formulated generally as follows:

> *Starting from the left, terms are unified with their nearest unifiable neighbour to the right.*

Here "unifiable" means that the ensuing list of semantic constraints (after unification) must be consistent. For instance, in a list containing the three constraints

$$x^{\mathsf{character}}, y^{\mathsf{t}}, y.\mathsf{nextTo} = z$$

x and y are *not* unifiable, even though the type of y is a variable, since a character does not have a nextTo attribute. However, in the example of the previous section:

$$\mathsf{pickUp}(x, y), x^{\mathsf{character}}, y^{\mathsf{thing}}, \mathsf{hammer}^{\mathsf{thing}}$$

y and hammer can be unified, resulting in

$$\mathsf{pickUp}(x, \mathsf{hammer}), x^{\mathsf{character}}, \mathsf{hammer}^{\mathsf{thing}}$$

The object merging process can be controlled by properly ordering the constraints in pattern matching rules, and by the use of disequality ($\neq$) constraints. This was demonstrated previously in the example:

$$\mathsf{ask}(\mathsf{user}, x, y)^{\mathsf{dialogue\_act}}, \mathsf{user}^{\mathsf{character}}, x^{\mathsf{character}}, x \neq \mathsf{user}, y^{\mathsf{t}}$$

where the disequality constraint $x \neq \mathsf{user}$ prevents unification of x and user.

4.3.2. Constraint inference

Consider the utterance "Go to the hammer", giving the following list of constraints:

$$\mathsf{goTo}(x, y)^{\mathsf{action}}, x^{\mathsf{character}}, y^{\mathsf{place}}, \mathsf{hammer}^{\mathsf{thing}}$$

At first, it seems as uncomplicated a sentence as "Take the hammer", discussed previously. But "Go to the hammer" actually poses bigger natural language understanding problems, because the domain encoding is strictly typed so that characters cannot go to things, only to places. Essentially the system must reason as follows:

The user wants me to go to some place x.
The hammer is at location y.
So x should be the place which is next to y.

This kind of reasoning is embodied in the following graph algorithm. First create a list of sets where every expression is put in a set of its own:

$$\{\mathsf{goTo}(x,y)^{\mathsf{action}}\},\{x^{\mathsf{character}}\},\{y^{\mathsf{place}}\},\{\mathsf{hammer}^{\mathsf{thing}}\}$$

Then sets are merged according to the following rule.

4.3.2.1. Set merging rule. Two sets X and Y should be merged if X contains an expression x which is a subexpression of some expression $y \in Y$.

This leaves the following list of sets:

$$\{\mathsf{goTo}(x,y)^{\mathsf{action}},x^{\mathsf{character}},y^{\mathsf{place}}\},\{\mathsf{hammer}^{\mathsf{thing}}\}$$

If there is only one remaining set at this stage, the algorithm halts. If there is more than one set, we choose the smallest set and apply the following rule:

4.3.2.2. Constraint adding rule. Given a set X, choose an object x and one of its attributes att, and add to X the expressions $x.\mathsf{att} = y$ and y^{t} (where att's values are of type t).

If the object denoted by this expression has an attribute att, we introduce the value of att as a new expression. In the example, objects of class thing have an attribute position, whose value is of type location. This gives us:

$$\{\mathsf{hammer}^{\mathsf{thing}},\mathsf{hammer.position} = l, l^{\mathsf{location}}\}$$

This set can still not be merged with the other set in the list, so we choose the same set again and re-apply the constraint adding rule. Objects of type location have an attribute nextTo whose value is of type place. Adding this link gives us:

$$\{\mathsf{hammer}^{\mathsf{thing}},\mathsf{hammer.position} = l, l^{\mathsf{location}},\ l.\mathsf{nextTo} = p, p^{\mathsf{place}}\}$$

Now the full list of constraints, after applying object merging (Section 4.3.1), is:

$$\mathsf{goTo}(x,y)^{\mathsf{action}},x^{\mathsf{character}},y^{\mathsf{place}},\mathsf{hammer}^{\mathsf{thing}},\ \mathsf{hammer.position} = l, l^{\mathsf{location}}, l.\mathsf{nextTo} = y$$

The set merging rule would place all these expressions in the same set, and therefore the algorithm terminates, returning the list of constraints above as the result. There is now a link from the second argument of goTo to the hammer; "the place y which is next to the location l where the hammer is".

A depth-first version of the algorithm can be concisely formulated as follows:

```
Given a list L of constraints:
while (true) {
    perform object merging (section 5.3.1);
    put each constraint in L in a set of its own,
      producing a list L′ of sets;
    apply the set merging rule to L′,
      producing L″;
    if L″ contains a single set,
        return this set as the result;
    else {
        choose a set in L″, and an
          expression in this set, and apply
          the object adding rule;
        Let L be the list of all the
          constraints
          in all the sets in L″;
    }
}
```

The actual implementation is breadth-first rather than depth-first, in order to find the shortest path connecting all constraints. Moreover, the algorithm only proceeds to a certain depth, to prevent looping.

4.3.3. Filtering

The next step is to filter the list of semantic constraints by removing all *implied* constraints. A constraint c in the list L is implied if

- c is a variable-free expression of the form $a = b$ or $a \neq b$ or,
- c is an non-variable expression of the form a^t, appearing as a subexpression of some other constraint b^s in L.

In the first case, trivially true facts like $axe = axe$ or $axe \neq hammer$ are removed. In the second case, the existence of the object a^t is implied by the existence of the object b^s. So for instance, in the list

$$request(user, cloddy, goTo(cloddy, y))^{dialogueAct}, goTo(cloddy, y)^{action}, cloddy^{character}, user^{character}, y^{place}$$

the three constraints $cloddy^{character}$, $user^{character}$ and $goTo(cloddy, y)^{action}$ are implied by the constraint $request(user, cloddy, goTo(cloddy, y))^{dialogueAct}$, and are therefore removed. However, the expression y^{place}, being a variable, is kept. This results in:

$$request(user, cloddy, goTo(cloddy, y))^{dialogueAct}, y^{place}$$

4.3.4. Abstraction

The point of the abstraction step is to transform the list of semantic constraints into a combinator by binding all free variables. When the dialogue act is known, this is straightforward. So, for instance, the list of constraints above is transformed to the following combinator by abstraction on y:

$$\lambda y^{thing}.request(user, cloddy, goTo(cloddy, y))$$

This expression, of type $thing \rightarrow dialogueAct$, is returned as the final answer of the parsing process.

A slightly more complex situation arises if the dialogue act is not known (i.e. there is no constraint of type dialogueAct in the list of constraints). Consider, for instance, the elliptical utterance "the hammer", leading to the singleton list

$$hammer^{thing}$$

Here, a new function symbol $f^{thing} \rightarrow dialogue\_act$ has to be introduced, as explained in Section 4.2. The final result is:

$$\lambda f^{thing \rightarrow dialogueAct}.(f\ hammer)$$

If the list of semantic constraints contains several expressions, the same process is repeated. So, for instance, the list

$$hammer^{thing}, axe^{thing}$$

is represented as

$$\lambda f^{thing \rightarrow (thing \rightarrow dialogueAct)}.((f\ hammer)axe)$$

That is, f should be bound to a function which is applied to hammer, returning a function which is applied to axe, returning a dialogue act.

4.4. Domain-dependent rewriting phase

We started Section 5 by claiming that the rewriting phase is domain independent, and thus does not need modification when moving to a new domain. Nevertheless, it can be very useful also to be able to define domain-dependent rewriting rules for resolving those types of underspecifications that are always resolved in the same way in the domain.

Such heuristic rewrite rules are expressed as combinators a^s. If the resulting expression b from the previous rewriting process is of type $s \rightarrow t$, then b will be applied to a. As an example, consider the utterance:

Ehh ... put down ehh ... let's see the pencil

The parsing algorithm just presented yields the following result:

$$\lambda f^{action \rightarrow dialogue\_act} \lambda x^{character} \lambda z^{location}.\ (f\ putDown(x, pencil, z))$$

This expression adequately represents all underspecifications in the utterance: Someone should put down a pencil somewhere, and the user is saying something about it. However, there are several reasonable assumptions we can make in order to simplify this expression, namely:

1. The user is making a request to Cloddy Hans.
2. Cloddy Hans is the one who should put down the pencil.

The point here is that these assumptions are made without considering the dialogue context. This can be done, since at least in the first scene of the game (see Section 2), Cloddy Hans is the only character present, and the scenario is all about the user instructing him where to put various things. So the two heuristics (1) and (2)

above are domain-specific rather than dialogue-context-specific.

The first heuristic, that an utterance about an action is a request to perform that action, can itself be expressed by a combinator:

$$\lambda x^{\text{action}}.\text{request}(\text{user}, \text{cloddy}, x)^{\text{dialogue\_act}}$$

Applying our expression to this heuristic combinator yields:

$$(\lambda f^{\text{action} \rightarrow \text{dialogue\_act}}\ \lambda x^{\text{character}}\ \lambda z^{\text{location}}.(f\ \text{putDown}(x, \text{pencil}, z))\ \lambda x^{\text{action}}.\text{request}(\text{user}, \text{cloddy}, x)) \rightarrow$$

$$\lambda x^{\text{character}}\ \lambda z^{\text{location}}.\ (\lambda x^{\text{action}}.\text{request}(\text{user}, \text{cloddy}, x))\ \text{putDown}(x, \text{pencil}, z)) \rightarrow$$

$$\lambda x^{\text{character}}\ \lambda z^{\text{location}}.\text{request}(\text{user}, \text{cloddy}, \text{putDown}(x, \text{pencil}, z))$$

The second heuristic, that the user is talking to Cloddy Hans, can be expressed simply as the following combinator:

$$\text{cloddy}^{\text{character}}$$

Applying our expression to the cloddy combinator yields:

$$(\lambda x^{\text{character}}\ \lambda z^{\text{location}}.\text{request}(\text{user}, \text{cloddy}, \text{putDown}(x, \text{pencil}, z))\ \text{cloddy}) \rightarrow$$

$$\lambda\ z^{\text{location}}.\text{request}(\text{user}, \text{cloddy}, \text{putDown}(\text{cloddy}, \text{pencil}, z))$$

The final expression is taken to be the (context-independent) interpretation of the user's utterance. The last parameter z might be bound as a result of context-dependent processing (see Section 3.2).

5. Evaluation

5.1. Corpora and data-collection methodology

To evaluate the parser, we used 3400 utterances from our corpora collected at four different occasions over a 5-month period (Bell et al., 2005). The subjects were children, aged 9–15. At the first data collection occasion, the subjects played the first scene only. At the second occasion, the subjects played the first scene, and then were allowed to explore the fairy-tale world together with Cloddy Hans. At the two last occasions, the subjects played two entire scenes, including the negotiation with Karen in order to cross the bridge. The 3400 utterances contain 810 unique words and 11,925 tokens, of which 1715 tokens are outside the system's present vocabulary of 525 words (i.e. the out-of-vocabulary rate is 14.4%).

To allow for extended user sessions where the player was able to explore the scenarios without being hindered by occasional errors due to imperfect speech recognition or understanding, the system was run in *supervised* mode. This meant that a human operator was supervising the interaction from behind the scene, and had the opportunity to interfere and correct the speech recognition result whenever he judged that the original result would seriously disturb the progression of the dialogue. He was also allowed to edit the system's response back to the user before this was output in cases where it would likewise have disturbed the progression of the dialogue.

It should be emphasized that the purpose of using supervised mode in the data collection was purely to ensure that the game (and hence the *dialogue*) was moving forward in those cases where there was otherwise a risk that it would be stalled or that repetitious errors would occur. Most importantly, all performance figures presented here are based on the recognition results obtained *before* any editing by the human operator. Hence, there is no "contamination" of the figures from the point of view of measuring the quality of parsing as such (since the domain of parsing is limited to single user turns). Actually, we believe that if supervised mode has any effect on the difficulty of the parsing task, it is rather to make it *harder*, since what supervised mode does is to occasionally "help" a fairytale character to address the player in a more coherent and intelligent fashion than would otherwise have been possible.

5.2. Units of measurement

Naturally, the quality of the results delivered by the parser, and ultimately the degree of

understanding of an utterance, is contingent on the quality of the input delivered by the speech recognizer. The quality of this input is estimated by the standard measures of sentence accuracy and word accuracy, whereas the quality of the final results are measured in terms of *semantic accuracy* and *concept accuracy*. By semantic accuracy we mean the proportion of utterances where the output of the parser *exactly* matches the correct analysis. Semantic accuracy can thus be seen as the semantic analogue of sentence accuracy. In contrast, concept accuracy is based on the number of semantic units that are substituted, inserted and deleted, and can thus be seen as the semantic analogue of word accuracy (Boros et al., 1996).

In order to calculate concept accuracy, we need a rigorous definition of a "concept". For all semantic expressions (except lambda abstractions), we will consider a "concept" to be a node in the tree making up the semantic expression. For instance, the expression

$$\mathsf{ask\_for\_attention}(\mathsf{user}, \mathsf{cloddy})$$

can be seen as a tree with the root node labeled ask_for_attention, and two leaf nodes labeled user and cloddy, respectively. So this expression has three concepts, but for the purpose of calculating concept accuracy, we will not count user (the first argument of a dialogue act), since it is always assumed that the dialogue act originated from the user.[2] Hence for expressions that are not lambda abstractions, the number of concepts equals the number of nodes in the tree making up the expression, minus one.

For lambda expressions, we simply do the same calculation for the body of the expression. For instance, the expression

$$\lambda x^{\mathsf{thing}}.\mathsf{request}(\mathsf{user}, \mathsf{cloddy}, \mathsf{pickUp}(\mathsf{cloddy}, x))$$

is considered to have the concepts present in the body of the lambda expression, namely request, user, cloddy, pickUp, cloddy, x^{thing}. Out of these, we include all concepts except user for the purpose of calculating concept accuracy.

An error occurs when a concept c appears in the semantic analysis of the input, but the corresponding place in the correct semantic analysis is occupied by a different concept d. If neither c or d are variables, the error is a substitution; if c is a variable but not d, the error is a deletion; if d is a variable but not c, the error is an addition.

5.3. Basic results

When constructing the set of 3400 correct analyses, altogether 509 utterances (15%) were judged not to be representable within the semantic formalism. These unrepresentable utterances ranged from fragments that could mean just about anything (e.g. "*Was it*"), through unanticipated requests (e.g. "*Kill the girl*") and musings ("*I thought as much*"), to complicated counterfactual statements ("*If you had taken the sword earlier you would have been able to cut the cloth to pieces now*"). Note that some of these unrepresentable utterances are not only problematic for the parser, but also pragmatically very difficult, which means that it is not always possible for the system to produce a coherent response.

In the tables below, we report sentence accuracy both with respect to the complete set of 3400 utterances and with the set of 2891 utterances that actually had a complete semantic representation. For the set of 3400 utterances, we judged an analysis to be correct or incorrect as follows: If the parser *failed* to produce an analysis for an unrepresentable utterance (giving as output "failed_act"), we took that output as being correct on the grounds that signalling that no analysis can be produced is the most that we could reasonably expect the parser to do in that case. (Following such an output from the parser, the dialogue manager would then try to repair the dialogue.) On the other hand, if the parser *did* produce an analysis for an unrepresentable utterance, we made the pessimistic assumption that that output was completely erroneous.

[2] This is not true for nested dialogue acts, however, as in one example from our corpus; "Tell Karen to lower the bridge", represented as:

$$\mathsf{request}(\mathsf{user}, \mathsf{cloddy}, \mathsf{request}(\mathsf{cloddy}, \mathsf{karen}, \mathsf{windDown}(\mathsf{karen}))).$$

Here, the user is requesting that Cloddy Hans make a request, so the first argument of the second request is cloddy, not user.

An analogous method was used to determine concept accuracy. Failure of the parser to produce an analysis for an unrepresentable utterance is counted as one instance of correct (the presence of ''failed_act''), whereas the analysis of an unrepresentable utterance will be counted as one deletion (missing ''failed_act'') plus one insertion for each additional semantic unit.

The results are shown in Table 1 below. The top of the table shows the accuracy of the speech recognizer. 30.6% of the recognized utterances were perfectly recognized, and the word accuracy was 38.6% (that is, the word error rate was 61.4%). These very poor figures are largely due to the fact that the subjects were children, and that speech recognition in particular is much less reliable for children than for adults. Furthermore, in our data the recognition results varied a lot between speakers. For some children, recognition was consistently dismal, whereas for others recognition worked quite well. That is, there was a kind of ''recognize-everything-or-recognize-nothing'' tendency, which explains the fact that the difference between sentence accuracy and word accuracy is small. This tendency was further amplified by the fact that the dialogues were long (the mean length of the dialogues was on the order of 90 turns). This allowed the children for which recognition worked well to gradually learn how to express themselves within the coverage of the system's understanding capabilities, making recognition work even better for them.

Table 1
Spoken language understanding results

	Speech input (%)	Recognized input (%)	Transcribed input (%)
Speech recognizer			
Sentence accuracy	30.6		
Word accuracy	38.6		
Parser			
Semantic accuracy (all)		48.6	84.8
Semantic accuracy (representable)		49.1	90.2
Concept accuracy (all)		53.2	86.4
Concept accuracy (representable)		50.5	92.6

The bottom part of the table shows the accuracy of the parser. The robustness of the parsing algorithm can be seen by comparing the first and second columns. The parser managed to recover the correct analysis for 48.6% of the utterances, in spite of the fact that only 30.6% were perfectly recognized. Similarly, the concept accuracy of the parser output is 53.2%, although the word accuracy is only 38.6%. These figures are further commented in Section 5.5.1.

The third column shows how the parser performs on transcribed (perfectly recognized) input. Here the semantic accuracy is 90.2% for the utterances that could be represented; that is, the parser fails to produce the correct analysis for only 9.8% of the utterances. Basically, the latter figure shows the coverage leaks, whereas the difference between 90.2% and 84.8% (that is, 5.4%) shows the extent to which the parser produces unwarranted analyses beyond the scope of the semantic formalism.

5.4. Further experiments

The parser's performance on transcribed input can be seen as a ''roof'' which will never be attained because of the inevitable distorsion of the input caused by the speech recognizer. A more realistic ''roof'' for the parser can be obtained by looking at *N*-best output from the speech recognizer, and more specifically the extent to which a (more) correct hypothesis is present there, as compared to it being the top hypothesis (1-best). To determine the effects of using *N*-best output, three experiments were run. First, sentence and word accuracy were computed using 10-best output from the speech recognizer for the set of 3400 utterances. Thus, for word accuracy, the best hypothesis compared to the transcribed utterance in terms of the number of substitutions, insertions and deletions at the word level was picked out from the 10-best list. The resulting sentence accuracy and word accuracy are shown in Table 2.

As could be expected, this ''oracle algorithm'' (always picking the best hypothesis) gave a significant improvement of both sentence and word accuracy (38% and 42% relative, respectively). Although the result does not alter the fundamental picture of the speech recognizer as constituting the

Table 2
Speech recognition results using 1-best and 10-best hypotheses

Speech recognizer	1-best (%)	10-best (%)
Sentence accuracy	30.6	42.1
Word accuracy	38.6	55.0

Table 3
Spoken language understanding results for 1-best and 10-best recognition hypotheses

Parser	1-best (%)	10-best (%)
Semantic accuracy (all)	48.6	65.4
Semantic accuracy (representable)	49.1	66.3
Concept accuracy (all)	53.2	70.4
Concept accuracy (representable)	50.5	72.3

main bottleneck for robust understanding, it still shows that something may be gained by looking at *N*-best rather than 1-best.

In a second experiment, the corresponding results for the semantic level were computed, shown in Table 3. Here, the second column shows the results for the hypotheses whose analyses from the parser corresponded most closely to the correct analyses in terms of the number of substitutions, deletions and insertions of semantic units.

The results again show a significant improvement (between 32% and 43% relative, respectively), indicating great potential gains by using *N*-best rather than 1-best. However, the problem then is to find a set of effective criteria which can be applied at run-time, and by which the best candidate from the *N*-best list can be found in as many cases as possible.

An obvious solution is to defer the decision of which hypothesis is (semantically) best, by sending analyses of all hypotheses on the *N*-best list to the next processing step in the system, which is the dialogue manager. The dialogue manager would then be able to use contextual expectations to find the best analysis on the list. For instance, if Cloddy Hans had posed a question to the user in the preceding turn, the system can sift through the list of analyses, looking for an expression that seems to represent an answer to the question. However, exactly how the system may use its knowledge about the current context is a topic of further research, and we will evaluate various possibilites in the future.

5.5. Discussion

5.5.1. Robustness

As can be seen in Table 1, the parser is robust in the sense that the semantic accuracy of the produced output exceeds the sentence accuracy of the input, or alternatively, the concept accuracy of the produced output exceeds the word accuracy of the input. A reasonable question at this point is whether this robustness merely is due to the fact that semantically important words happen to be recognized correctly more often than words in general.

To be able to answer this question, we first need a definition of what a "semantically important" word is. A reasonable definition, we think, is that any word that occurs in the pattern of at least one of the parser's pattern matching rules is semantically important (since there is at least one context in which that word contributes to the semantic analysis).

Using this definition of semantic importance, we made the following calculations. Of the 10,206 semantically important tokens that were uttered, 6084 were present in the recognizer's output whereas 4122 were missing. The corresponding figures for all 11,925 uttered tokens are 6777 recognized, 5148 missing. This means that if X is a semantically important token occurring in the corpus, X has a 60% chance of being correctly recognized, whereas if X is any token in the corpus, X has a 57% chance of being recognized. This gives some support to the hypothesis that semantically important words are being recognized correctly more often, although the difference is not sufficiently big to explain the robustness effect altogether.

If our starting point instead is the recognized tokens, we see that of the 11,326 semantically important tokens occurring in an output string from the recognizer, 6160 were actually uttered whereas 5166 were erroneously inserted. The corresponding figures for all 13,034 recognized tokens are 6848 uttered and 6186 erroneously inserted. This means that if X is a semantically important

token occurring in a recognized string, X has a 54% chance of actually having been uttered, whereas if X is any recognized token, X has a 52.5% chance of actually having been uttered. Again, semantically important tokens are erroneously inserted less often than tokens in general, although the difference is very small.

Many utterances that have not been correctly recognized but anyway yield a correct analysis are indeed examples where semantically unimportant words have been omitted, inserted or substituted for other semantically unimportant words. Examples include "*nej jag vill att du ska ta lampan*" ("*no I want you to take the lamp*"), recognized as "*nej jag vill att du ska ta lampan ehh*" ("*no I want you to take the lamp ehh*"), or "*och nu går du fram till tjejen*" ("*and now you go up the girl*"), recognized as "*att det går fram till tjejen*" ("*that it goes to the girl*").

However, there are also some other kinds of examples. Some utterances contain enough redundancy for the parser to able to recreate the correct analysis even when recognition errors occur, e.g. "*ja det vill jag*" ("*yes I want that*"), recognized as "*vad det vill jag*" ("*what I want that*"). Here both "yes" and "I want that" give rise to a confirm dialogue act, so the misrecognition of "yes" does not have a harmful effect. In some utterances, words occurring in a long pattern are misrecognized, but there is a shorter pattern yielding the same constraints that matches instead. An example is "*gå och lägg det i maskinen då*" ("*go and put it in the machine then*"), recognized as "*och lägg det i maskinen då*" ("*and put it in the machine then*"). Here "go and put it" and "put it" are taken to mean the same thing, so the misrecognition of "go" does not matter. In the example "*den går inte att röra*" ("*you can't move it*"), recognized as "*spring hör inte det är det*" ("*run don't hear it is it*") , the Swedish verb "gå" is used in a sense not meaning "go" or "walk". Since "gå" is recognized incorrectly, the parser is not led astray.

Furthermore, the algorithm is robust in the presence of false starts (like "*go go to the machine*") and clarifications within an utterance (like "*go to it to the machine that is*"), and thus it is robust in the presence of misrecognitions leading to such constructions (such misrecognitions are also present in the corpus).

Summing up, the robustness of the parsing algorithm is to some extent due to the fact that words contributing to the parser's semantic output are recognized more reliably than words in general. There are, however, a multitude of other factors which all contribute to the robustness of the algorithm.

5.5.2. Shortcomings

As already mentioned, the most common reason for incorrect analyses being produced by the parser is misrecognition; that essential words are missing in the input or have been erroneously inserted. The remaining problems can be roughly grouped into different categories, having to do with lexical coverage leaks, commonly misrecognized words, lexical ambiguities, complex grammar, pragmatic ambiguities, and semantic and ontological insufficiencies. These categories are not clear-cut; many utterances can be said to belong to two different groups.

One group consists of utterances running into problems caused by semantic and ontological insufficiencies. This group includes many completely reasonable utterances that, at present, cannot be represented within the semantic formalism, e.g. requests for instructions in specific situations ("Am I supposed to, you know, pull things?", "How do you usually do this?"), questions concerning Cloddy Hans's mental state ("Are you having a good time?"), instructions ("Kill her", "Pick some flowers", "Break something"), complex spatial references ("The second last slot", "Go to the left, that is, your left") and various comments ("I just told you", "I don't give a damn", "I was just kidding"). But it also contains completely unexpected input which we will *not* try to incorporate into the system's repertoire. One boy liked to think of the fairy-tale machine as a time-travel machine, and tried to explain the concept to Cloddy Hans ("you can use it to travel into the future and backwards in time", etc.).

Commonly misrecognized words pose problems in those cases where the substitution of one word for another completely alters the meaning of the utterance, e.g. "What is the fairy-tale machine?"

and ''Where is the fairy-tale machine?''. Here the Swedish words for ''what'' (''vad'') and ''where'' (''var'') are very similar-sounding, and thus easily misrecognized.

Lexical ambiguities are rare in this domain, but point to a fundamental problem to the extent that they occur. The parsing algorithm is deterministic and produces one output expression only; hence it sometimes has to make premature decisions that eventually turn out to be wrong. An example is ''Varför går inte det?'' (Why doesn't that work?/ Why is that impossible?). The word ''går'' has two meanings in Swedish; it may also mean ''walk'' or ''go''. Therefore the parser falsely triggers on the two patterns ''varför'' and ''går'', and interprets the utterance as a question about why Cloddy Hans does not go to some (unspecified) place.

There are a few utterances in the corpus that seem to call for a more grammatical parsing method. One such example is ''Are all the gadgets that were lying on the shelf lying on the grass here?'', asked by a subject when he entered the second scene (this utterance is also semantically complex; a yes/no-question concerning a universally quantified implication).

Finally, there are some pragmatic ambiguities, where it is unclear what dialogue act the user is actually making. An example is ''Can you do that?'', where it is not clear whether the user is making a request or whether he is enquiring about Cloddy Hans's capabilities. However, such utterances would cause problems for any spoken language understanding method.

6. Related work

Approaches to robust parsing can be divided into data-driven and symbolic methods, the former of which have been the focus of a steadily growing interest during the last decade. One strand of work in this area deals with syntactic parsing in the sense of deriving a constituent structure or a dependency structure (for example, Collins, 1999; Charniak, 2000; Nivre and Scholz, 2004), but without the specific requirement of producing output that serves the needs of a dialogue manager. Another strand of work, namely, ''How may I help you'' type systems, explicitly aims at integrating robust understanding with a dialogue system, but with a semantic representation that is limited to atomic categories. Thus, parsing here corresponds rather to classification of utterances into a small set of categories—for example, 15 in the classic ATT ''How may I help you'' system (Gorin et al., 1997), and generally not more than a few hundred in more recent systems. We are thus not aware of any approaches that make use of automatic, data-driven methods to derive the kind of complex semantic structures that are needed by a dialogue manager in a domain like ours.

Turning to symbolic, rule-based approaches to robust parsing, one option, pioneered by Ward (1989), is to rely on pattern matching and to use a relatively coarse-grained semantic representation, such as a variable-free slot-filler list. Other instances of work in this shallow-parsing direction are Jackson et al. (1991) and Aust et al. (1995).

However, conversational applications such as the one described here tend to require more fine-grained semantic formalisms in order to sufficiently capture the meaning of user utterances. For example, variable-free slot-filler lists are not suitable for negotiative dialogue, in which several alternative solutions are simultaneously discussed and compared (Boye and Wirén, 2003b; Larsson, 2002). On the other hand, the computational price for adopting a general-purpose logic-based formalism and general semantic reasoning is likely to be too high in an application where savvy users will not accept having to wait for the system to come back with an answer.

Several attempts at finding a suitable trade-off by synthesizing the shallow and logic-based approaches have been made. One possibility is to ''robustify'' some general-purpose linguistic method, either by homing in on the largest grammatical fragment (Boye et al., 1999), or on the smallest set of grammatical fragments that span the whole utterance (see, for example van Noord et al., 1999 and Kasper et al., 1999). Another possibility is to extend the 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. By giving preference to mapping rules that are more specific (in the sense of satisfying more constraints), grammatical information can be used whenever available. However, the semantic representations produced are still limited to that of variable-free slot-filler lists. In contrast, Boye and Wirén (2003a,b) put forward a more fine-grained formalism in which a type system is used instead of general semantic reasoning; hence, the system is still much more restricted than general-purpose logic-based formalisms. The parser and semantic formalism presented here constitute a further development and application to new domain of that framework.

7. Conclusions

In this paper, we have attempted to tackle what we believe is a very hard problem, namely, spontaneous spoken dialogue between children and human-like characters in a 3D fairy-tale environment. The particular problem that we have dealt with is robust parsing (that is, context-independent analysis), but we have also shown how contextual interpretation is carried out within our framework.

Not surprisingly, speech recognition is the major bottleneck with a word accuracy at just 39%. Moreover, even if we use an "oracle" to pick the hypothesis from the 10-best list that comes closest to the transcription, the word accuracy is still only 55%. These poor recognition figures are due to the fact that the subjects were children.

So how can we do robust parsing given this bottleneck resulting from speech recognition? The fast answer is that with a concept accuracy at 53%, the parser still manages to reconstruct a great deal of meaning from the very noisy input. Moreover, this figure is obtained just using 1-best hypotheses. By using 10-best output from the speech recognizer, it is possible with the current parser to attain a concept accuracy of 70%. There is thus potentially a lot to be gained by looking at *N*-best rather than 1-best.

To sum up, we have described a framework for robust parsing of spoken utterances which proceeds in two steps: a domain-dependent pattern-matching phase and a domain-*independent* rewriting phase. Previous systems based on pattern matching have been restricted to producing relatively simple semantic structures, such as variable-free slot-filler lists. Unfortunately, such structures are not suitable as input to a dialogue manager in our domain, which involves information-seeking utterances, commands and simple negotiation, and where there is also abundant reference to objects in the 3D world as well as in the discourse. Our system instead produces a semantic representation that constitutes a trade-off between the simple structures typically generated by pattern-matching parsers and the complex structures generated by general-purpose, lingusitically-based parsers. In particular, this trade-off allows us to retain the advantages of pattern-matching systems in terms of efficiency and robustness, while capturing the contents of the great majority of utterances manifested in our domain.

Acknowledgements

This research was carried out within the EU 5th framework project NICE (IST-2001-35293). The NICE homepage can be found at http://www.nice-project.com. The authors would like to thank the other members of the consortium, in particular Liquid Media (http://www.liquid.se) for providing the wonderful 3D virtual world. The authors also gratefully acknowledge the insightful comments made by two anonymous reviewers.

References

Aust, H., Oerder, M., Seide, F., Steinbiss, V., 1995. The Philips automatic train timetable system. Speech Comm. 17, 249–262.

Bell, L., Boye, J., Gustafson, J., Heldner, M., Lindström, L., Wirén, M., 2005. The Swedish NICE corpus—Spoken dialogues between children and embodied characters in a computer game scenario. In: Proceedings of Interspeech'05, Lisbon, Portugal.

Boros, M., Eckert, W., Gallwitz, F., Görz, G., Hanrieder, G., Niemann, H., 1996. Towards understanding spontaneous speech: word accuracy vs concept accuracy. Proc. ISCLP'96, 1009–1012.

Boye, J., Wirén, M., Rayner, M., Lewin, I., Carter, D., Becket, R., 1999. Language processing strategies and mixed-initiative dialogues. In: Proc IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Stockholm, Sweden.

Boye, J., Wirén, M., 2003. Robust parsing of utterances in negotiative dialogue. In: Proc. Eurospeech, Geneva, Switzerland.

Boye, J., Wirén, M., 2003. Negotiative spoken-dialogue interfaces to databases. In: Proc. Diabruck (7th Workshop on the Semantics and Pragmatics of Dialogue), Wallerfangen, Germany.

Charniak, E., 2000. A maximum-entropy-inspired parser. In: Proc. NAACL (North American Chapter of the Association for Computational Linguistics).

Collins, M., 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. Dissertation, University of Pennsylvania.

Dalrymple, M., Shieber, S., Pereira, F., 1991. Ellipsis and higher-order unification. Linguist. Philos. 14 (4), 399–452.

Gorin, A.L., Riccardi, G., Wright, J.H., 1997. How may I help you?. Speech Comm. 23 113–127.

Hindley, R., Seldin, J., 1986. Introduction to combinators and λ-calculus. Cambridge University Press.

Jackson, E., Appelt, D., Bear, J., Moore, R., Podlozny, A., 1991. A template matcher for robust NL interpretation. In: Proc. DARPA Speech and Natural Language Workshop, Morgan Kaufmann.

Jurafsky, D., Martin, J., 2000. Speech and Language Processing. Prentice Hall.

Kasper, W., Kiefer, B., Krieger, H., Rupp, C., Worm, K., 1999. Charting the depth of robust speech processing. In: Proc. ACL.

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

Milward, D., Knight, S., 2001. Improving on phrase spotting for spoken dialogue systems. In: Proc WISP.

Nivre, J., Scholz, M., 2004. Deterministic dependency parsing of English text. In: Proc. COLING 2004, Geneva, Switzerland.

van Noord, G., Bouma, G., Koeling, R., Nederhof, M.-J., 1999. Robust grammatical analysis for spoken dialogue systems. J. Nat. Language Eng. 5 (1), 45–93.

Sterling, L., Shapiro, E., 1994. The Art of Prolog, 2nd ed. The MIT Press, Berlin.

Ward, W., 1989. Understanding spontaneous speech. In: Proc. DARPA Speech and Natural Language Workshop, Philadelphia, USA, pp. 137–141.