Developing an enriched natural language grammar for prosodically-improved concept-to-speech synthesis

by

Laurette Marais

submitted in accordance with the requirements for the degree of

MASTER OF SCIENCE

in the subject of

COMPUTING

at the

University of South Africa

Supervisor: Aarne Ranta

Co-supervisor: Tertia Hörne

April 10, 2014

Declaration

I declare that *Developing an enriched natural language grammar for prosodically-improved concept-to-speech synthesis* is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

Signature
(Mrs)

Date

Acknowledgements

I would like to thank my husband, Willem, for his unceasing love and support, and my parents, Fransjohan and Laurette Pretorius, for their love and energy in encouraging me to pursue a career in research.

Many thanks are due to my supervisor, Aarne Ranta, for his valuable input and guidance, and for building the wonderful community around GF that challenges and inspires.

I also want to thank my co-supervisor, Tertia Hörne, for her meticulous help in ensuring that this thesis is presented in its best form.

Thanks also to Krasimir Angelov, who patiently answered many questions about various facets of GF.

Finally, I want to thank all my colleagues at the HLT group of the Meraka Institute, CSIR, for providing a supportive and creative work environment. Special thanks to Karen Calteaux, for the opportunity to pursue this topic and travel to both Europe and the USA during the course of my studies, and Daniel van Niekerk and Georg Schlunz for being willing to exchange ideas and share their insights.

Abstract

The need for interacting with machines using spoken natural language is growing, along with the expectation that synthetic speech in this context sound natural. Such interaction includes answering questions, where prosody plays an important role in producing natural English synthetic speech by communicating the information structure of utterances.

CCG is a theoretical framework that exploits the notion that, in English, information structure, prosodic structure and syntactic structure are isomorphic. This provides a way to convert a semantic representation of an utterance into a prosodically natural spoken utterance. GF is a framework for writing grammars, where abstract tree structures capture the semantic structure and concrete grammars render these structures in linearised strings. This research combines these frameworks to develop a system that converts semantic representations of utterances into linearised strings of natural language that are marked up to inform the prosody-generating component of a speech synthesis system.

Key terms

GF; CCG; prosody; intonation; speech synthesis; concept-to-speech; information structure; syntax; question-answering; spoken natural language;

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Talking machines have always been the staple of science fiction literature, but judging by the success of spoken conversational agents like Apple's Siri and Samsung's S-Voice for Android phones, the need for interacting with machines using spoken natural language is growing. Users can issue commands to their phones to perform actions like sending text messages, setting reminders, searching the web etc., but they can also ask questions and have the agent supply an answer. In order to do this, conversational agents like Siri rely on technologies for natural language question answering and, specifically, spoken question answering. This typically includes aspects of speech recognition, language understanding, information retrieval and reasoning, language generation and speech synthesis.

In spoken question answering, the addition of synthetic speech to human-computer interaction provides a richer medium for conveying meaning to the user. Many spoken languages, including English, rely on prosody (which refers to features like stress, intonation, phrasing and accent) to convey meaning in ways that text cannot. In order for synthetic voices to sound more natural, the prosodic properties of human speech must be included in the synthesis process.

Spoken dialogue and question-answering (QA) systems can generally be seen as consisting of five main components: Automatic Speech Recognition, Natural Language Understanding, Task Management (which handles reasoning and information retrieval), Natural Language Generation and Speech Synthesis. Figure 1.1 shows a spoken question-answer system adapted from Jurafsky and Martin (2009).

Traditionally, the speech synthesis problem (or text-to-speech problem) is conceived as having arbitrary natural language text as input, with prosody generated based on the semantics derived from the text. Deriving semantics from arbitrary text is difficult, since natural language text can be underspecified in terms of semantics (Taylor, 2009; Blackburn and Bos, 1999). However, not all text-to-speech (TTS) applications depend on the ability to synthesise arbitrary text. Particularly, in question-answering systems, the reasoning component pro-

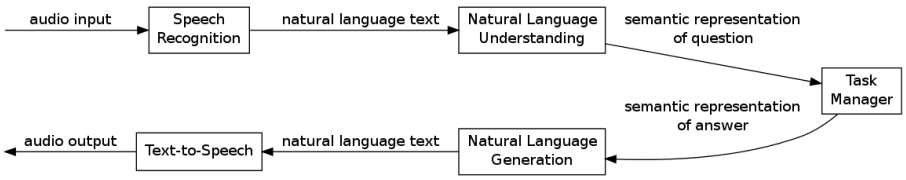

Figure 1.1: A spoken question-answer system

duces a semantic representation of the answer, from which both natural language and synthetic speech is produced. A system using the semantic representation as a starting point for both the natural language generation and speech synthesis components is known as a concept-to-speech (CTS) system. Since the semantics of the utterance serves as the starting point for synthesis, the problem of deriving semantics from the text in an attempt to model prosody disappears.

One aspect of the semantics of an utterance that is particularly relevant to prosody is its information structure. According to Prevost (1995b) "information structure refers to the organization of information within an utterance. In particular, information structure defines how the information conveyed by a sentence is related to the knowledge of the interlocutors and the structure of their discourse". Prevost adopts a two-tiered representation of information structure: on the first tier, at the phrase level, an utterance is divided into its theme and rheme; on the second tier, at the word level, focus is assigned to certain words in both the theme and rheme phrases. He takes theme to refer to "the part of the utterance that links it to previous utterances" and rheme to refer to "the part of the utterance that forms the core of the speaker's contribution" (Prevost, 1995b).

Steedman (2001) argues that it is possible to describe utterances syntactically in such a way that the phrases corresponding to theme and rheme appear as syntactic constituents. That is, when specified correctly, the syntax of an utterance provides knowledge of its information structure. Steedman (1996; 2001) also shows that, in English, the theme and rheme of an utterance are associated with distinct intonational tunes. This shows that there is a structural correlation between information structure, syntax and prosody in natural language.

The purpose of this study is to exploit this structural correlation in order to contribute toward prosodically-improved speech synthesis.

1.1 Problem Statement

CTS systems generally take as input semantic representations "such as database entities, templates or logical forms [...] and transforms them first into grammatical sentences and subsequently, into natural and coherent spoken utterances" (Pan et al., 2002). A spoken question-answering system that uses CTS is shown

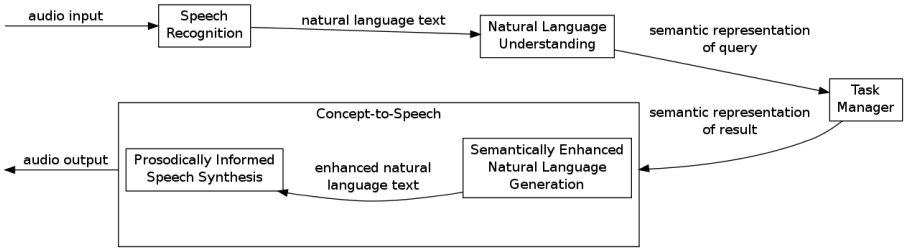

Figure 1.2: A spoken question-answering system that uses CTS synthesis

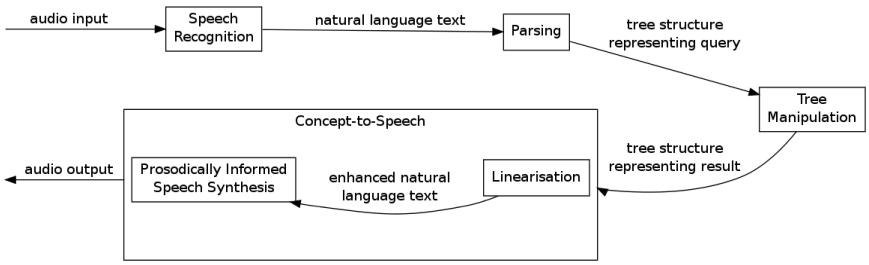

Figure 1.3: A spoken question-answering system that uses GF for CTS synthesis

in figure 1.2.

Grammatical Framework (GF) (Ranta, 2011) is a functional programming language for writing grammars. It supports mapping between semantic constituents and syntactic constituents through its distinction between semantic grammars and syntactic grammars. A semantic grammar provides functions for building tree structures that represent the semantics of utterances in the abstract, while syntactic grammars provide rules about how these tree structures are represented as linear strings in various languages. This means that a semantic grammar essentially plays the role of an interlingua between its corresponding syntactic grammars. The mapping between the semantic grammar and the syntactic grammars is bidirectional, which means that concrete, linear strings can be parsed into tree structures, and tree structures can be linearised into strings. The separation between semantic and syntactic structure in GF allows us to re-envision the CTS structure for a question-answering system as in figure 1.3.

It is clear that using GF simplifies especially the answer generating aspects of question-answering: the creation of an abstract representation of the answer is reduced to manipulating the tree structure representing the question, replacing and adding the relevant components, while the generation of enriched text that serves as input to the synthesiser is reduced to linearising the tree structure into a string that retains the necessary information about the tree structure itself.

GF also provides resources that simplify the linearisation step. A Resource Grammar Library (RGL) for over twenty languages handles grammatical com-

plexity, which means that grammar engineers need only to define how semantic trees are rendered in terms of syntactic structures, leaving linguistic details such as case, number, gender and word order to the RGL.

Therefore, our approach is to extend the existing GF RGL for English to include functions for exploiting the structural correlation between syntax and information structure mentioned in the previous section. The goal is to generate natural language text that is suitably marked up for use in prosodically-improved speech synthesis.

1.1.1 Modelling intonational units

To improve the naturalness of a synthetic voice, appropriate prosodic information, especially concerning intonation and phrasing, must be available to inform the prosodic characteristics of the synthesised speech (Taylor, 2009, p.112). Usually, speech synthesis uses natural language text as its starting point. In order to produce suitable prosody, the text is analysed using several natural language processing techniques, including traditional syntactic analysis (Taylor, 2009, p. 117). This analysis forms the basis for generating suitable prosodic information for each of the syntactic constituents in the utterance. However, traditional syntactic units do not always correspond to intonational units (Prevost and Steedman, 1994a). Instead, Steedman (1996) argues that certain intonational tunes and spoken language phrases are rather associated with the theme (topic) and rheme (comment) of an utterance, respectively. In the sentence "John loves MARY", where *John loves* is the theme and *Mary* the rheme, the intonation, transcribed using ToBI (Tones and Break Indices) (Beckman and Hirschberg, 1994), is represented in the following way:

John	loves	Mary
LH*	LH%	H*LL%

Here, *John loves* and *Mary* are the intonational units, whereas traditionally, *John* (subject) and *loves Mary* (predicate) would be the syntactic units. In order to model these units in a grammar, a more generalised notion of syntax, which would consider *John loves* and *Mary* as syntactic constituents, is needed.

1.1.2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) generalises the notion of surface constituency, "allowing multiple derivations and constituent structures for sentences, including ones in which the subject and verb of a transitive sentence can exist as a constituent, complete with an interpretation" (Prevost and Steedman, 1994a). Such a grammar could treat the phrases *John loves* and *Mary* as constituents of the language that could in turn be assigned certain intonational information. Crucially, the grammar would also provide a semantic interpretation for such constituents (Prevost and Steedman, 1994a). In other words,

this would allow the writing of a grammar that is flexible in terms of matching surface constituents with the theme and rheme of an utterance.

CCG captures all the required syntactic and categorial information in its lexicon. The verb *love*, for instance, might have the type (S\NP)/NP. This means that it combines with an NP to the right and then an NP to the left in order to create an S. For example:

John	loves	Mary
NP	(S\NP)/NP	NP
	S\NP	
S		

This corresponds to the semantic representation (John (loves Mary)).

The verb *love* could, however, be associated with a second type, namely (S/NP)\NP, combining first with an NP to the left and then to the right, yielding the following syntactic interpretation:

John	loves	Mary
NP	(S/NP)\NP	NP
S/NP		
	S	

This, in turn, corresponds to the semantic representation ((John loves) Mary). Hence, CCG provides a suitable theoretical basis for a generalised notion of syntax, which provides the additional property that intonational structure, syntactic structure and information structure are isomorphic (Prevost, 1995b). A flexible way of implementing such a grammar is needed to generate prosodically-marked-up natural language.

1.1.3 Grammatical Framework

A GF grammar consists mainly of two parts: the abstract grammar and a set of concrete grammars. The abstract grammar specifies semantic level relations between constituents, and the concrete grammar allows for these constituents and relations to be linearised in a specific language. More than one concrete grammar could be developed for a given abstract grammar, allowing semantically robust translation between the concrete grammars (Ranta, 2011).

GF also provides a Resource Grammar Library (Ranta, 2009), the purpose of which is "to provide the main grammar rules of different languages, in a form readily usable by application programmers. The library defines the low-level details of morphology and syntax ..." (Ranta, 2011) That is, a single abstract grammar provides the general grammatical structures available in most languages, and a concrete grammar for each language linearises these structures correctly according to its unique morphology and syntax. It also works as a parser of sentences allowed by the grammar. Currently, GF supports over 20

```
UseCl                       -- produces S
    (PredVP                 -- produces Cl
      (UsePN john_PN)
      (ComplSlash           -- produces VP
        (SlashV2a love_V2)
        (UsePN mary_PN)
      )
    )
```

Figure 1.4: GF: John (loves Mary)

```
UseSSlash                    -- produces S
   (UseSlash                 -- produces ClSlash
      (SlashVP               -- produces VPSlash
         (mkNP john_PN)
         (SlashV2a love_V2)
      )
   )
   (mkNP mary_PN)
```

Figure 1.5: GF: (John loves) Mary

languages, including English (Ranta, 2013).

In many cases, the RGL provides more than one parse for a sentence, all of which are grammatically correct (Ranta, 2011). The two GF trees (slightly simplified for clarity) in figures 1.4 and 1.5 correspond to the two sentences linearised as "John loves Mary" in the previous section. Figures 1.6 and 1.7 give graphical representations of these trees.

This example shows that, besides providing traditional categories like VP and NP, the RGL also provides structures for some non-traditional categories such as ClSlash (a clause missing an NP, e.g. "she looks at") and VPSlash (a verb phrase missing an NP, e.g. "looks at"). These categories correspond to S/NP and (S/NP)\NP, respectively, in CCG.

1.1.4 Developing a grammar for prosodically-improved spoken English

As seen in the previous section, the RGL provides structures allowing for multiple parses of the same sentence. In section 2.2 it was shown that a generalised approach to syntax allows different surface structures for sentences to provide

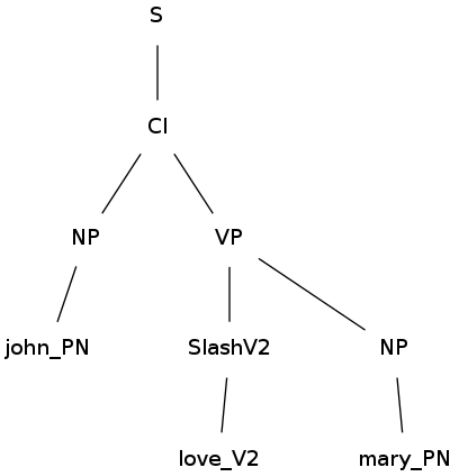

Figure 1.6: John (loves Mary)

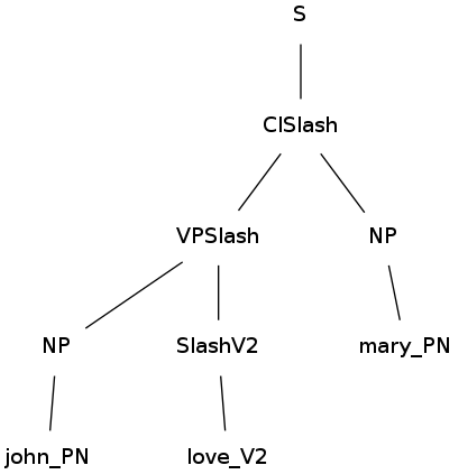

Figure 1.7: (John loves) Mary

a way of modelling intonational units more closely. This research presents an extension to the RGL with both functions that allow the building of sentences in multiple ways and with functions that allow enriched text linearisation that marks up otherwise identical textual representations as having different underlying structures. This extension is used in a spoken question-answering system to produce prosodically-improved speech synthesis.

1.2 Research Question

How can the novel combination of GF, insights from CCG and current speech synthesis techniques be used to develop a grammar that assists in building a prosodically-improved CTS question-answering system in English?

The following sub-questions will be addressed:

1. How should the GF English RGL be extended to provide functions for building trees whose structures encode information structure?
2. How should the information structure encoded in the tree structures be rendered when the tree is linearised?
3. How should the extended GF English RGL be used to build an application grammar for a question-answering system, where question-answer pairs are structurally related in terms of information structure?
4. How can a question-answering system use such an application grammar to compute semantically correct answers to questions?
5. How can the output of the question-answering system be used to improve the prosody in a synthetic voice?

1.3 Methodology

In order to answer the research question, a process of computational modelling, prototyping, validation and testing was followed.

1. The grammatical scope of the research was established as limited to wh-question-answer pairs consisting of main clauses.
2. A set of test sentences was developed to reflect the grammatical scope. The chosen domain of the sentences was results of football games and signings.
3. The sentences were analysed in terms of possible information structural interpretations and then parsed by the GF English RGL to determine the RGL's coverage of information structure.

4. Based on this analysis, the RGL was adapted in the following two steps:
   (a) It was extended to complete its coverage of different information structural interpretations of the sentences. This involved adding functions to the abstract Extra module of the English RGL, along with the corresponding concrete syntax linearisation functions.
   (b) The RGL was also extended to allow for enhanced linearisations of the parse trees to indicate the information structure of each tree.
5. A GF application grammar was developed to showcase the use of the extended RGL. It provides functions for building the question-answer pairs in the test set.
6. A question-answering system was developed in Python to show how questions parsed by the grammar could be interpreted in order to calculate an answer, and manipulated in order to return the corresponding sentence, enhanced with information structural markup to facilitate synthesising speech with improved prosody.
7. The output of the question-answering system was converted in a straight forward way, replacing the markup for information structure with Speech Synthesis Markup language (SSML) markup. This was fed to an SSML-enabled synthesis system to produce audio output.
8. Finally, a perceptual experiment was conducted on the audio output, to confirm that the information structural markup provided by the RGL extension via the application grammar, would indeed result in prosodically-improved synthetic speech.

1.4 Assumptions and limitations

This research focuses on the problem of using CTS in spoken question-answering systems, as opposed to the general TTS problem, and as such does not aim to synthesise arbitrary natural language text. Such unconstrained input may include grammatical errors that cannot be parsed by the RGL. As such, the syntactic scope is at least limited to the coverage of the English RGL. Furthermore, the linguistic scope of the research is limited to wh-questions consisting of only main clauses, as discussed in section 4.2, although possible future work could include widening the linguistic scope.

The MARY synthesis engine will be used to produce speech synthesis. This limits the ways in which the synthesis may be modified to the mechanisms available for MARY. We discuss these in more detail in section 5.5.

1.5 Proposed Contribution

This research proposes to contribute the novel combination of GF with the theory of information structure, syntax and prosody employed in CCG, as well current speech synthesis techniques in order to produce prosodically-improved speech synthesis. This combination of ideas proposes to contribute in two particular ways.

Firstly, GF as an implementation framework for grammars provides parsing and linearising capabilities, a runtime environment that allows directly manipulating syntax trees, as well as a grammar architecture that separates abstract syntax from concrete syntax, which ensures efficient and extensible grammar development in a multilingual setting. Implementing functions in GF based on the work done in CCG with regards to information structure, syntax and prosody, will extend this work with all the advantages offered by the GF environment. In other words, the addition of information structurally motivated functions to the GF English RGL provides the option to exploit the isomorphic relation between information structure, syntax and prosody to develop grammars for prosodically-improved speech synthesis. A spoken question-answering system based on a GF application grammar is developed to illustrate this. Additionally, since GF is particularly suited to multilingual solutions, this research could provide the basis from which similar systems could be bootstrapped for other languages.

Secondly, employing information structure as a means to improve prosody is tested in the context of a state-of-the-art speech synthesis engine. Steady improvements in the naturalness of speech synthesis have been made in the last decade, and it remains to be seen whether the structural correlation exploited in the work on CCG will still result in prosodically-improved speech synthesis using current synthesis techniques.

A note on tree diagrams

In the many GF tree diagrams presented in this document, the tree represents the construction of an utterance from its various category components. Each node is marked up with the function and the resulting type. For example, the root node in some cases is labelled `DeclUtt : Utt`, which means that the function `DeclUtt` is used to produce the type `Utt`. Its children represent the arguments the function takes.

Chapter 2

Literature Review

Generating natural language for use in prosodically-informed speech synthesis systems for question-answering encompasses several topics, including computational linguistics, natural language generation, the relationship between meaning, prosody and syntactic structure, discourse structure, and dialogue management. As pointed out by Prevost (1995b), topics such as these have been approached from fields as diverse as syntax, semantics, phonology, theoretical computer science and artificial intelligence, each having different research goals and methodologies.

In this chapter, we review various attempts to address the problem of prosodically-natural spoken language generation by surveying the research into concept-to-speech systems, with particular focus on research into the relationship between prosody, information structure and syntax.

2.1 Prosody in speech synthesis

A growing appreciation for the role prosody plays in communication has led to interest in modelling prosody in order to produce more "natural sounding" synthetic speech. In particular, prosody can often be appealed to in disambiguating structural or scope ambiguities (Hirschberg, 2002). According to Steedman (2012) "in spoken English, information-structural distinctions are to an unusual degree conveyed by intonational prosody".

When attempting to model prosody for synthetic speech, "one must first determine the prosodic inventory of the language to be modelled and which aspects of that inventory can be varied by speakers to convey differences in meaning" (Hirschberg, 2006a). In many languages, this results in a focus on prominence (accent) and phrasing. In English, for example, speakers tend to make content words prominent, especially when the concepts introduced by them are new to

the discourse. Prosodic phrasing is a result of humans "chunking" their utterances into meaningful units of information. Several systems have been developed to describe prominence and phrasing features, of which the most popular are the Edinburgh Festival Tilt system, the ToBI system, developed for different varieties of English prosody, the IPO contour stylisation techniques developed for Dutch, and the Fujisaki model developed for Japanese (Hirschberg, 2006a).

However, prosody research in itself has faced methodological challenges, which according to Xu (2012) is mainly due to the so-called lack of reference problem.

> By reference I mean a pivot that serves as both a starting point of inquest and a point that one can comfortably fall back on. In segmental research, for example, word identity serves as such a reference, because it is consciously accessible whether the language under study has a writing system or even whether the human informant is literate. Thus we can confidently investigate the phonetic properties that distinguish one word from another. But words also give us a false sense of certainty, because their ease of access may lead to the assumption that what underlies the lexical contrast, namely, the phonemes, are also easily accessible to individual speakers (Xu, 2012).

Easy access to phonemes, Xu maintains, is not the case, referring to work by Mattingly (1972) and Liberman et al. (1989). Also, in the case of prosody, as opposed to words, "very little of its functionality is orthographically represented, except for the punctuations whose meanings are at best ambiguous" (Xu, 2012).

Xu divides the research on modelling prosody into three approaches.

1. In the case of analysis by transcription, which includes the examples mentioned above, the goal is not a predictive system, but rather a descriptive one, making it difficult to apply such models in producing natural prosody in synthetic speech.
2. In the case of analysis by introspection, the obvious danger is the subjective nature of the research, resulting in it remaining merely hypothetical. While analysis by hypothesis testing provides a more rigorous framework for research by allowing falsifiability, Xu lists several challenges faced in this approach. These include the division between prosodic function and the encoding thereof in speech, as well as ecological validity: "how much of what is observed in a controlled experiment is applicable to everyday speech". This aspect is especially of importance when considering prosody production in synthetic speech.
3. Finally, challenges facing the approach of analysis by computational modelling include decisions concerning whether the computational model should be prescriptive or predictive, how such a model could be evaluated and the correct degree of freedom, that is, "the number of free parameters needed to completely specify a model".

While the current work does not concern prosody *per se*, but rather effecting a change in the prosodic realisation of a synthetic voice based on the information structure of an utterance, these concerns are of particular importance when conducting perceptual experiments with prosodically-altered speech.

Originally, attempts to model prosody in synthetic speech were approached from a text-to-speech perspective, a paradigm in which systems are able to read aloud unrestricted text. Essentially, text-to-speech (TTS) comprises two components, namely text analysis and speech generation (Taylor, 2000). However, since the prosodic features of an utterance "are highly dependent on the informational structure, on the linguistic structure, and on the situational context of the utterance" (Alter et al., 1997), this approached has not been entirely successful in producing appropriate prosody in synthetic speech. For example, traditionally, the task of varying contour in TTS systems was confined to assigning appropriate contour to declaratives and questions. Increasing knowledge of intonational meaning has been slow to find its way into speech synthesis systems (Hirschberg, 2002). According to Feng et al. (2012), recent research in pitch accent and prosody prediction for TTS has had only limited success. The problem of determining the boundaries of prosodic phrases in TTS is especially difficult when the text itself is ambiguous. Hirschberg (2002) reports that the given/new status of prosodic phrases "is modelled in text-to-speech systems at best by collecting stems of previously uttered items in a fixed window, or a paragraph or other orthographic unit".

Determining phrase boundaries usually relies on the text analysis performed in the first phase of text-to-speech synthesis. The components that produce prosodic assignment are either rule-based, in which case the system relies especially on syntactic information, or corpus-trained, where features such as part-of-speech, sentence length and type and punctuation are considered. However, prosodic phrase boundaries do not necessarily occur at the boundaries of syntactic units (Prevost and Steedman, 1994a), and so some success has been achieved by incorporating more sophisticated constituent information. However, systems seem to benefit most from allowing users to explicitly control the prosody of the utterance via (usually system specific) markup (Hirschberg, 2006a).

The limitations of text-based approaches can partly be ascribed to the fact that the written form of an utterance is a poor knowledge source for its prosodic realisation (Alter et al., 1997; Blackburn and Bos, 1999; Taylor, 2009), since prosodic structure contributes to communicating the intented meaning of the utterance (Veilleux, 1997). In reaction to this, there has been an evident shift of focus in the computational linguistics community towards so-called concept-to-speech (CTS) systems (Haji-Abdolhosseini, 2003).

2.2 Concept-to-speech synthesis

Walker and Rambow (2002) separate the task of language generation into deciding, firstly, what to say, and secondly, how to say it.

Spoken language systems used to identify these two decisions with separate modules for natural language generation and speech generation, with prosodic assignment seen as the responsibility of the latter process. CTS systems, on the other hand, make use of tight interaction between these processes. Specifically, "a [natural language] generation system may utilize options of speech synthesis during its decisions of tactical or strategic generation, e.g., reflect the information structure either by intonational cues or via morpho-syntactic variations" (Alter et al., 1997). This means that the speech generation module is passed more than just the text to be synthesised: some form of markup is typically used to inform the prosody-generating component (Hirschberg, 2006a). Essentially, CTS systems can be described as taking as input semantic representations, such as "database entities, templates or logical forms" and, after transforming them into grammatical sentences, producing as final output "natural and coherent spoken utterances" (Pan et al., 2002). The rationale is that the naturalness is improved by avoiding the loss of information structural knowledge that occurs when a semantic representation is rendered in ambiguous or underspecified text and then converted to synthetic speech.

In the generation of prosodically-natural speech synthesis via enriched text, as in CTS systems, there is an interplay between information structure as present in the semantic representation, syntactic structure as present in the textual representation, and prosodic structure as present in the final synthetic speech. It will be useful to look at how the relationships between these aspects have been studied so far. Since information structure is the starting point in CTS systems, we first consider attempts to understand or model the relationship between information structure and prosody, and then consider attempts to understand or model the relationship between information structure and syntax. Finally, particular focus is given to attempts to unify all three aspects in one theory.

2.2.1 Information structure, prosody and syntax

Steedman (2012) echoes Xu's (2012) point that, across languages, the various markers of information structure are not well-understood, and he ascribes this to our limited ability to detect and classify them in spoken language.

> Semantically, almost all of their effects to which we have conscious access appear to be secondary implicatures arising from more primitive meaning elements relating to interpersonal propositional attitude, whose nature can only be inferred indirectly (Steedman, 2012).

Nonetheless, because of the role of prosody in communicating information structure, especially in English and other Germanic languages (Theune, 2002), numerous studies have made use of information structure as a basis for more accurately modelling prosody in CTS systems. Some studies attempt to understand the relationship between information structure and prosody, while others use this knowledge to compute more natural sounding prosody in synthetic speech.

In their study of the features that are most informative for modelling prosody, Pan et al. (2002) use SURGE as the natural language generating component, selecting from the features available in the system to learn prosodic associations. Their results lead them to conclude, among other things, that part-of-speech information and syntactic function are most effective for learning pitch accent associations, and that the boundaries and length of semantic/syntactic constituents are additionally effective for learning break indices.

Following a different approach, (Prevost, 1995b) identifies two tiers in information structure that serve to inform prosody: the given/new distinction informs intonational tunes and breaks at the phrase level, while focus informs pitch accents at the word level. At the phrase level, Prevost refers to the given part of a sentence as the *theme* and the new or interesting part of the sentence as the *rheme*. At the word level, he considers in detail contrastive focus, where contrast is derived from a word's alternative set, restricting the sets of alternatives to "those entities and propositions explicitly mentioned in the discourse".

While terminology varies widely in the literature, a similar distinction is maintained by others also. For example, Gundel and Fretheim (2006) classify givenness-newness as being either relational or referential. Relation givenness-newness refers to the division of the sentence into a new and a given component, roughly corresponding to Prevost's theme and rheme, while referential givenness-newness "involves a relation between a linguistic expression and a corresponding non-linguistic entity in the speaker/hearers mind, the discourse (model), or some real or possible world", corresponding to Prevost's interpretation of focus. Steedman (2000) also points out that several authors refer to two kinds of "focus", "using the term to cover both comment/rheme and phonological focus". Büring (2007) again refers to two ways in which prosody reflects information contained in an utterance, namely Narrow Syntactic Mapping, where prosody reflects sentence-internal structure such as syntax, and Extraneous Feature Mapping, where prosody reflects discourse-related information, which he sees as unrelated to syntax. However, he has in mind traditional notions of syntax. Steedman (2001), on the other hand, proposes a view of syntax that he argues is in fact isomorphic to both information structure and prosodic structure. Steedman (2012) also draws together several conceptualisations of information structure as essentially reflecting Prevost's two-tiered model of information structure, commenting that the prevailing consensus in the literature may have been obscured "by the numerous superficially differing nomenclatures that have been applied".

The last 20 years have produced several studies on including information structure in order to produce better prosody in synthetic speech. Some have specifically factored in the influence of syntax on prosody (Prevost and Steedman, 1993, 1994a,b; Hiyakumoto et al., 1997; Theune, 2002; Li et al., 2012), while others have either limited their research to single syntactic templates (Kügler et al., 2012) or focused on word-level accents (Spyns et al., 1997; Hitzeman et al., 1998; Yagi et al., 2005).

Specifically, much of the work by Prevost and Steedman (1993) and others use Combinatory Categorial Grammar (CCG) as a unifying framework for information structure, syntax and prosody. What is unique to this approach is the use of

non-traditional syntactic constituents for modelling intonational units. The argument is made for an isomorphism between information structure, syntax and prosody, and in fact, that "the Surface Syntax of natural language acts as a completely transparent interface between the spoken form of the language, including prosodic structure and intonational phrasing, and a compositional semantics" (Steedman, 2001). In terms of relating utterances to the discourse, that is, assigning accents on word-level to contrasting constituents, Hiyakumoto et al. (1997) used WordNet synsets to determine alternative sets for words, assigning focus based on the presence of synonyms and antonyms earlier in the discourse. They note that determining focus is an easier problem than identifying theme and rheme phrases in arbitrary text.

Theune (1998), on the other hand, deals with the issue of contrastive accent by comparing the data structures that give rise to utterances. If the data type of a new utterance is the same as that of a recent utterance, their fields are compared, and wherever the new utterance has different values, the corresponding surface forms are assigned contrastive accents. This work is followed by research on generating contrastive accent in the CTS system D2S (Theune, 2002), where the effects of syntax are taken into account as well. Utterances are produced from syntactic templates, which carry syntactic information used to compute prosody.

More recent work by Kügler et al. (2012) also follows the route of syntactic templates, but they use only a single template in their experiment. While their system is work in progress, the 2012 publication is intended to motivate the use of IS-enriched speech synthesis. The experiment they describe confirms that in a complex discourse situation, IS-enriched synthetic speech is preferable to a system's default output. They used the MARY TTS system to produce the default output, and manually manipulated the output to produce IS-enriched synthetic speech (Kügler et al., 2012). Performing experiments with manually manipulated audio has the advantage of presenting to users synthetic speech that has been finely tuned to reflect the appropriate information structure (IS). Their research does not yet, however, address the issue of manipulating the default output of the MARY TTS system automatically based on information structure. It also does not address the role of syntactic structure in computing prosody.

Other work on improving prosody based on information structure includes that of Hitzeman et al. (1998), which focuses on accent placement on noun phrases, followed by Hitzeman et al. (1999) with an annotation scheme for information structure. They argue for markup of linguistic information, which they define as including rhetorical relations that function on the level of utterances (such as concession, amplification, contrast, etc.), and relations that function on the level of noun phrases. These relations include reference type, such as first mention and anaphor, syntactic type, where a distinction is made between different kinds of noun phrases and other structures, and semantic type, where there is a distinction between proper nouns and other nouns. Opting for linguistic (or in some cases information structural) markup supports a modular approach, since it allows for simplified use with different synthesisers. They contrast this choice with that of Hiyakumoto et al. (1997) and Bierner (1998), who mark up the

generated text with explicit prosodic information.

Finally, recent work by Li et al. (2012) uses the information structure annotation scheme proposed by Calhoun et al. (2005) to mark contrastive word pairs. They use an HMM-based synthesiser, trained on two sets of recorded data: the first set is recorded with natural prosody, while the second is recorded with exaggerated emphasis on contrastive word pairs within the sentence. Detection of contrastive word pairs is done with support vector machines, while the emphasis is realised via the HMM-based synthesiser. Their experiments show that participants were able to successfully identify contrastive words on the basis of the prosody assigned to them. However, they conclude that over-emphasis decreases the perceived naturalness of synthetic speech.

2.2.2 Modelling information structure and syntax computationally

As mentioned in the previous section, CCG has been the basis of some computational research into improvements to prosody in CTS in which syntax plays an integral role. In fact, the argument is for a compositional semantics that is reflected in both the prosodic structure as well as the syntactic structure. According to Steedman (2012) "The semantics is surface-compositional (Hausser, 1984), in the sense that logical forms can be derived directly via surface-syntactic derivation, and constitute the only level of representation in the grammar". The relevant semantics in the case of question-answering systems is the information structure of utterances.

A computational grammar framework that also follows a compositional approach to semantics and syntax is Grammatical Framework (GF). One of the main features of a GF grammar is that it requires an abstract syntax, which is a typed grammar for specifying semantics compositionally, and one or more concrete syntaxes, which are concrete grammars for linearising the semantic components as strings in different languages. It borrows this notion of separation between abstract syntax and concrete syntax from programming language design, but applies it to natural language (Ranta, 2003).

In terms of representing semantic information and converting it to natural language, GF has been used in several ways, including in multimodal grammar applications (Bringert et al., 2005), dialogue systems (Ranta and Cooper, 2004; Larsson and Ljunglof, 2008), generating GF grammars from OWL ontologies (Dannélls et al., 2012) and also verbalising ontologies using GF and the Lexicon Model for Ontologies (*lemon*) (Davis et al., 2012).

Research on multimodal grammars is based on the flexibility of the architecture of GF, which allows several concrete grammars for a single abstract grammar. According to Davis et al. (2012), such concrete grammars may take the form of "a table output as a LATEX file or in Excel format" or "a format suitable for speech synthesis", or even, as in their example application, a "graphical presentation of a route".

GF has been used to provide natural language descriptions of semantic concepts for ontologies written in OWL, as well as ontologies represented in *lemon* (Pérez et al., 2006; Dannélls et al., 2012; Angelov and Enache, 2012). Large parts of SUMO, an open-source ontology, have been translated to GF (Enache, 2010).

GF has also been used directly with the dialogue system GoDiS, providing a full specification of the dialogue system as a GF grammar. GoDiS is an Information State Update (ISU-based) dialogue system that provides "general and fairly sophisticated accounts of several common dialogue phenomena such as interactive grounding, accommodation, multiple conversational threads, and mixed initiative" (Larsson and Ljunglof, 2008). The semantic representations used by GoDiS are called dialogue moves, which could be translated by a suitable grammar into natural language utterances such as declarative sentences, questions and imperatives.

GF has also been used in speech-related research, though the focus has been on recognition and speech translation instead of synthesis (Bringert, 2007). A GF speech recognition compiler was developed, targeting formats such as Nuance GSL, SRGS, JSGF and HTK SLF (Bringert, 2008).

No work on GF, however, has been done in exploiting the abstract-concrete syntax division to explicitly carry information structure in the way suggested by work in CCG, that is, to attempt to exploit the isomorphism between information structure, syntax and prosody as argued for by Steedman (2000) and others. The purpose of this research, therefore, is to show how GF can be used as a framework for question-answering in which the information structure is captured in the semantic composition of an utterance, where the concrete syntactic structure reflects this, and where this leads to a linearisation that is marked up with information for speech synthesis with improved prosody.

Chapter 3

GF: A syntax-semantics interface

Grammatical Framework (GF) can be described as a categorial grammar formalism, a special-purpose, functional language for grammars, and also a natural language processing (NLP) framework (Ranta, 2012). These three descriptions capture different aspects of GF: firstly, as a grammar formalism based on type theory; secondly, as a programming language for writing grammars for parsing and linearising natural language; and thirdly as a framework equipped with resources and tools that assist in the development of grammar-based applications. We look at each of these three aspects in order to introduce the various features of GF that are relevant to this research.

3.1 GF as a grammar formalism

According to Ranta (2003),

> The development of GF started as a notation for *type-theoretical grammars*, which use Martin-Löf's type theory to express the semantics of natural language. The first implementation was released in 1998 at Xerox Research Centre Europe in Grenoble, with focus on multilingual authoring via a type-theoretical pivot language.

Since then, GF has developed into a functional programming language and the focus has shifted towards applications, with GF used as a "high-level and reliable grammar formalism" for linguists, as well as an "elegant and efficient tool for building natural language applications" for programmers (Ranta, 2003).

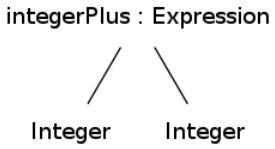

Figure 3.1: Compositional functions create tree structures

A key feature of GF is that it distinguishes between abstract and concrete syntax (Dannélls et al., 2012). The abstract syntax describes the hierarchical, semantic structure of a language, whereas the concrete syntax describes how this is realised when the language is written down. More specifically, the abstract syntax defines categories and typed rules for obtaining categories compositionally, while the concrete syntax defines how each category is to be linearised and how the linearisations of categories are obtained compositionally via the rules. In effect, GF extends a logical framework (which is a version of constructive type theory) with a notation for concrete syntax (Ranta, 2003).

For example, suppose we wanted to express the concept 'integer-plus' in an infix language. We could define two categories in the abstract syntax, `Expression` and `Integer`, and the following rule for building an `Expression` from `Integer`s

```
integerPlus : Integer -> Integer -> Expression
```

This function composes the `Expression` category from two instances of the `Integer` category, and can be represented as a tree structure, as in figure 3.1, which reflects the semantics of the function.

The concrete syntax would then contain the linearisation types for each category, and define how these are achieved by applying the appropriate linearisation rule. The following shows how this is done. The keyword `++` denotes token concatenation.

```
Expression = {s : Str}
Integer = {s : Str}

integerPlus x y = {s = x.s ++ "+" ++ y.s }
```

If we now wanted to express the same concept in a prefix language, we could use the same abstract syntax (i.e. the same semantic tree structures), but define another concrete syntax in the following way:

```
Expression = {s : Str}
Integer = {s : Str}
```

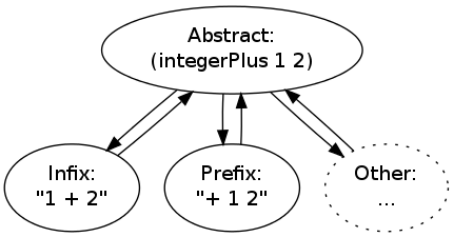

Figure 3.2: GF abstract and concrete syntaxes

```
integerPlus x y = {s = "+" ++ x.s ++ y.s }
```

In fact, since the two languages share an abstract syntax, we could easily generate the same statement in both languages. GF goes even further, however, and provides the ability to parse strings recognised by a concrete grammar, which means that we have a reliable two-way translator between our infix and prefix languages: parsing with one concrete syntax and linearising with the other (Bringert, 2008). Additionally, adding translation to and from more languages simply requires writing a concrete syntax for each new language. (See figure 3.2)

This distinction between abstract and concrete syntax is quite clear when designing programming languages, as the concrete syntax is purposefully designed with certain principles in mind, such as simplicity and readability etc., to reflect the semantics of the abstract syntax. This is not the case for natural languages, since they arise naturally and there is no question there of design. However, GF applies the idea of distinguishing between abstract and concrete syntax to natural language: the concrete syntax is "designed" in such a way as to ensure that it looks exactly like the natural language (Ranta, 2003). That is, we could add a natural language to the translation system described above by adding a concrete syntax for English that looks like this

```
Expression = {s : Str}
Integer = {s : Str}

integerPlus x y = {s = "the sum of" ++ x.s ++ "and" ++ y.s }
```

Once natural language enters the picture, however, linguistic features such as number, gender, case, tense etc., have to be dealt with as well. GF as a functional language offers ways to encode such linguistic features, and GF as an NLP framework offers libraries in which this has been implemented for over twenty languages. We now look at the language features that are specifically relevant to this research.

3.2 GF as a functional language

GF is a functional language designed specifically for writing grammars. It borrows constructs from Haskell and ML, such as algebraic datatypes, higher-order functions and pattern matching, while the "type theory used in the abstract syntax part of GF is inherited from logical frameworks, in particular ALF" (Ranta, 2012).

In the discussion in section 3.1, the idea of defining categories and rules was introduced, and it was shown how this structure is reflected in both the abstract and concrete syntaxes. In GF, the keyword `cat` is used to introduce all the categories in the abstract syntax, and `lincat` is used in the concrete syntax to define each the linearisation category of each category in the abstract syntax. Linearisation categories define the object that is produced when linearising a category, and typically take the form of records (Ranta, 2011). In the abstract syntax, the keyword `fun` is used to declare functions for building trees, and in the concrete syntax, the keyword `lin` is used to define the linearisation rules for building linearisation objects from their arguments (Ranta, 2011).

Records are a way of capturing various linguistic aspects of language categories. For example, in the English Resource Library (Ranta, 2012), the category for noun phrases, `NP`, has the following `lincat` definition:

```
lincat NP = {s : NPCase => Str ; a : Agr} ;
```

This means that the linearisation category of `NP` is a record with two fields: `s`, of which the type is a table from `NPCase` to `Str`, and `a`, which has type `Agr`. `NPCase` and `Agr` are so-called parameter types, while `Str` is the native GF type for strings.

The `a` field holds the agreement information necessary to ensure that the noun phrase agrees with the verb phrase. The agreement information, which is inherent to the noun phrase, uses the parameter type `Agr`. This is similar to Haskell and ML's algebraic data types (Ranta, 2012) in that it defines the possible values that an instance of type `Agr` can take as follows:

```
param Agr = AgP1 Number | AgP2 Number | AgP3Sg Gender | AgP3Pl ;
```

This captures the notion that there is first and second person agreement, which both take number as a parameter, while third person singular agreement takes gender as a parameter. Third person plural agreement takes no parameters. This information is necessary for linearising the verb phrase correctly.

On the other hand, the `s` field encapsulates the notion that the surface form, or linearised form, of the noun phrase is variable, and that it is generated based on the relevant case. It does this using a table, which maps values of type `NPCase`

(also a parameter type, like `Agr`) to the correct surface string. The operator `!` is used for choosing the correct string from the table. For instance, if we wanted to linearise an object noun phrase, we would select the correct string by using `np.s!NPAcc`. `NPAcc` is one of the legal values of the parameter type `NPCase` and represents the accusative case.

Another feature of GF that ought to be mentioned is its operations, introduced with the keyword `oper`. While `fun` functions in GF are used to construct trees, and in terms of functional programming they behave as constructors, operations correspond to functions in any typed functional programming language. They are able to take arguments and return results, can be overloaded and are defined using lambda abstraction. Operations have many uses in GF, especially for defining *smart paradigms*, where pattern matching on strings is used to determine the correct inflexion patterns for words in the grammar (Ranta, 2011).

The preceding discussion shows that GF uses records, tables and parameters to model the various inherent and variable features of lexical and phrasal categories, as well as operations to abstract away certain functionality from the tree-constructing `fun` functions.

3.3 GF as an NLP framework

3.3.1 Resource Grammar Library

An important resource available to GF users is the Resource Grammar Library (RGL) and API. From a software perspective, the RGL functions as a software library and in fact as the standard library of GF, while from a linguistic perspective it is essentially a set of parallel grammars (Ranta, 2009). It consists of an abstract syntax and concrete syntaxes for more than twenty languages. In contrast to the application grammars discussed so far, where the abstract syntax focuses on capturing semantic relations between categories, the focus of the abstract grammar is syntactic, that is, on capturing syntactic relations between syntactic categories. By syntactic categories we mean both lexical categories, such as *noun* or *determiner*, and phrasal categories, such as *noun phrase* or *prepositional phrase*. The RGL's abstract syntax defines the common phrasal structures between languages, and leaves language-specific lexical structures and their morphology to the concrete syntax of each language. According to Ranta, the RGL can be thought of as a "linguistic ontology" (Ranta, 2011):

> It is a domain of linguistic objects, organized in linguistic categories and following strict rules of combination. Two millennia of research and education in the Western grammatical tradition have set up a rather standard set of concepts, which in the 20th century has also been formalized as precise mathematical models, and in later decades, as computer implementations.

Ranta admits that the classical grammar concepts do not "fit" *all* languages, but argues that "having a common conceptual framework has both practical and theoretical advantages", including re-use of previous work, especially in developing multi-lingual applications, as well as the availability of precise concepts to compare languages (Ranta, 2011).

The library has two layers, namely the core grammar and the user API, and uses a module structure to group relevant functions together. While the common syntactic structures are accessed via the `Syntax`L modules of each language's concrete syntax and the language specific morphology via the `Paradigms`L modules (where L represents the language code), some languages have syntactic structures that are not shared by other languages. In these cases, a module named `Extra`L is included, which extends the grammar with the unique structures of the language. The `Syntax`L module is built from the core grammar using grammar composition, with the main parts of the core grammar being `Grammar`L (for all syntactic combination rules), `Lexicon`L (containing a test lexicon), `Lang`L, which combines the latter two modules, and `All`L, which combines `Lang`L with `Extra`L (Ranta, 2009).

The categories covered by the resource grammar start at the suprasentential level with `Text` (representing lists of phrases separated by punctuation), `Phr` (phrases) and `Utt` (utterances, which can be whole sentences or subsentential phrases). `Utt`s are constructed from sentence types such as `S` (regular sentences) and `QS` (question sentences). Sentence types are built from clause types (e.g. `Cl` and `QCl`), by fixing the polarity and tense. It follows that clause types have variable polarity and tense, and as such correspond to the notion of the sentence radical. Clause types themselves are built by the processes of predication (the combination of a subject and predicate) and complementation (where the main verb is *be*). Subjects of sentences have type `NP` (noun phrase), while predicates have type `VP` (verb phrase). A `VP` has a verb (e.g. `V`, `V2`, etc.) as its head and can be modified by sentence adverbs (`AdV`), such as *always*, and ordinary adverbs (`Adv`), such as *here*. `NP`s are formed mainly from common nouns (`CN`), determiners (`Det`) and pronouns (`Pron`). `CN`s can be modified by adjectival phrases (`AP`), and relative clauses (`RCl`) (Ranta, 2009). This overview is not intended to be a full description of the categories in the Resource Grammar library, but rather to give an idea of its hierarchical structure for building syntax trees.

The Resource Grammar API consists of a set of overloaded functions of the form `mk`C for constructing phrasal categories C, *word_*C for constructing lexical categories C, and *descr*C for any other function constructing C (Ranta, 2009). Phrasal (or syntactic) functions take arguments, e.g. `mkS : NP -> VP -> S`, while lexical functions take no arguments and produce the members of the closed word classes, e.g. `i_Pron` for the pronoun "I" and `with_Prep` for the proposition "with".

These overloaded functions allow the linguistic details to be hidden from the user of the library, who only needs to know how the domain concepts in the abstract syntax are expressed in natural language (Ranta, 2011). For instance, constructing the question "How does Jack play?", requires knowing that the

```
fun
  mkUtt : QS -> Utt ;
  mkQS : QCl -> QS ;
  mkQCl : IAdv -> Cl -> QCl ;
  mkCl : NP -> V -> Cl ;
  mkNP : PN -> NP ;
  mkPN : Str -> PN ;
  play_V : V ;
  how_IAdv : IAdv ;
```

Figure 3.3: Typical API functions

```
lin mkUtt (mkQS (mkQCl how_IAdv
                   (mkCl (mkNP (mkPN "Jack")) play_V)));
```

Figure 3.4: A tree built from API functions

categories needed are an interrogative adverb (how) and a clause formed from a noun phrase (Jack) and a verb (play). It also requires knowledge of the API functions in figure 3.3, which can be found in the online reference (Ranta, 2012).

From this knowledge, the tree in figure 3.4 can be constructed. As it happens, this structure also yields the question in a semantically correct way in many other languages. This means that for sufficiently similar languages, code may be re-used in application grammars using a so-called functor, which further reduces the burden upon the application programmer (Ranta, 2011).

Slash Categories

A group of categories particularly relevant to this research is the so-called slash categories. A slash category CSlash represents a category C that is "missing" its noun phrase. The notion of slash categories is inspired by GPSG (Ranta, 2009). The categories available in the RGL are `SSlash`, `ClSlash` and `VPSlash`. Slash verb phrases are formed by providing a verb with all its complements except one of the noun phrases. In the case of ditransitive verbs (such as pay, in "I pay you twenty euros"), a function is provided for constructing the slash verb phrase where the second noun phrase ("twenty euros") is not provided. A `VPSlash` is then used to form a `ClSlash`, which in turn is used to form an `SSlash`. Ranta explains the reason for defining only a few rules, instead of a general slash category constructor, as follows (Ranta, 2009):

```
lin mkQS (mkQCl (mkIP which_IQuant man_N)
            (mkClSlash
                (mkNP (mkPN "John"))
                (mkVPSlash see_V2))
```

Figure 3.5: Question constructed from a ClSlash

> Semantically, extraction (slash formation) can be interpreted as function abstraction. Thus CSlash is as semantic type equal to `NP -> C`. Slash propagation is function composition. Both these interpretations could be expressed using higher-order abstract syntax (functions that take functions as arguments; see Ranta 2004). But the results would be overgenerating, for instance, because of island constraints. Thus we have chosen to approach full coverage from below, by using weak rules to cover only the cases that are certain. The approach is inspired by Combinatory Categorial Grammar (CCG) where a set of combinators, such as function composition, are used instead of unrestricted lambda binding.

These categories have so far been used to form certain kinds of questions. For example, the question "Which man does John see?" can be constructed from the GF Resource Grammar API as in figure 3.5.

The slash clause (`ClSlash`) is formed from the subject noun phrase "John" and the slash verb phrase (`VPSlash`) "see", with "see" being a transitive verb missing its object noun phrase. The novel use of slash categories for modelling information structural components such as theme and rheme is discussed in section 5.2.2.

3.3.2 Previous work with GF

GF can be compiled into a runtime format, called Portable Grammar Format (PGF) (Angelov et al., 2009), and can be used by applications written in Haskell, Java, Javascript, C and Python (Angelov and Enache, 2012). It is also supported by runtime systems in Haskell (Angelov, 2009) and C (Ranta, 2012). Most recently, GF has been used in the online multilingual translation project MOLTO for applications that include adding multilinguality to semantic wiki pages (Fuchs et al., 2013), providing knowledge management software for museums via ontology verbalisation (Ranta et al., 2012), high quality translation of patents (Enache et al., 2012) and the ability to query knowledge bases using natural language questions (Mitankin et al., 2010).

Previous projects using GF include the educational project WebALT (Caprotti, 2006), the verification tool KeY (Ahrendt et al., 2005) and the dialogue sys-

tem research project TALK (Ljunglöf et al., 2006). In TALK, GF was used for building extensible, modularised and even multimodal dialogue systems. The proof-of-concept system was able to understand sentences about public transport using both speech recognition and mouse clicks (Bringert et al., 2005). In the domain of speech, GF has also been used to implement speech translation. A grammar compiler was built to support the generation of grammars in various formats, including Nuance GSL, SRGS, JSGF and HTK SLF (Bringert, 2008).

Chapter 4

Modelling information structure, syntax and prosody

The purpose of this chapter is to lay the foundation for the discussion of the implementation of an enriched natural language grammar for prosodically-improved speech synthesis. This is done by looking at how information structure, syntax and prosody are understood for the purposes of this research.

4.1 Information structure

The view of information structure followed here is essentially a slightly simplified version of Prevost (1995a) and others (Steedman, 1991, 2000, 2001; Calhoun et al., 2005), i.e. a two-tiered system: one tier is concerned with the given/new distinction at the phrase level, distinguishing between theme (given) and rheme (new); the second tier is concerned with contrastive focus at the word level. Kruijff-Korbayová and Steedman (2003) identify some distinguishing features of theories of information structure, which serve as a useful grid along which to clarify the view of information structure that is taken here.

One such feature is the recursivity of information structure partitioning. Theories differ over the level at which information structure partitions an utterance, whether at the sentence level, clause level of perhaps lower levels of the syntax hierarchy. Within these structures, some theories allow "mild" recursivity, while others allow either unlimited recursivity or none at all. Following Steedman (2001), we partition at the sentence level and allow for no recursivity.

Each sentence is seen as consisting of a single theme and rheme. The subtrees

```
answer :: GUtt -> GUtt
answer p = case p of
  GQUtt (GWhoTeamQ (GVPTheme action team))
   ->  GDeclUtt (GWhoTeamA (get_rheme action team)
                           (GVPTheme action team))
```

Figure 4.1: Haskell code for pattern matching on a tree

representing theme and rheme attach directly to the root node of the utterance, and if they appear contiguously in the linearisation of the utterance, they are enclosed with markup indicating which phrase represents the "rheme" and which the "theme". However, in some cases the rheme appears mid-sentence (see example (1)), so to clarify the markup, a third label, namely "tail", is taken from Vallduvi (1993) to mark the parts of the theme that appear after the rheme in a sentence. As to the level of orthogonality in this information structure scheme, contrastive focus is seen as a completely orthogonal feature that occurs in both the theme and rheme.

(1) a. Which game did Barcelona win yesterday?
    b. (Barcelona won)$_{theme}$ (the game against Ajax)$_{rheme}$ (yesterday)$_{tail}$.

The advantage of using this view together with GF to model the information structure of wh-question-answer pairs is that the process of constructing answers from their preceding wh-questions is fairly simple. To illustrate this, we use the functional language Haskell to show how pattern matching can achieve this construction. In the example in figure 4.1, all types starting with `G` are Haskell data types that represent the functions in the GF application grammar. The program takes GF trees as input, and these are converted by the GF Haskell API into native Haskell trees. The function `answer` takes such a Haskell tree representing a question and returns the tree that represents its answer. The first branch of the `case` statement is shown below. Figure 4.2 shows an example of a GF tree that would match the branch. Figure 4.3 shows and example of a resulting GF tree.

The code in figure 4.1 shows that the syntactic unit representing the theme in the answer is reused directly from the question. In this particular case, the syntactic unit is a traditional verb phrase. However, using the conceptualisation of syntax that underlies Combinatory Categorial Grammar (Steedman, 2001), similar transformations become possible in which whatever information structural unit representing the theme is reused from the question is also a well-defined syntactic unit. Additionally, it corresponds directly to an intonational unit that can be marked up in the linearisation of the tree in order to produce enhanced natural language, which in turn may inform the prosody generation of a speech synthesis module.

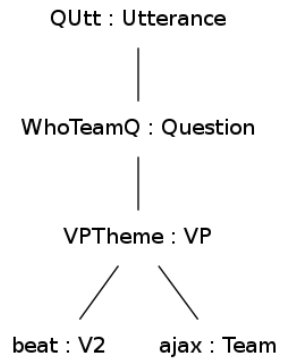

Figure 4.2: Who beat Ajax?

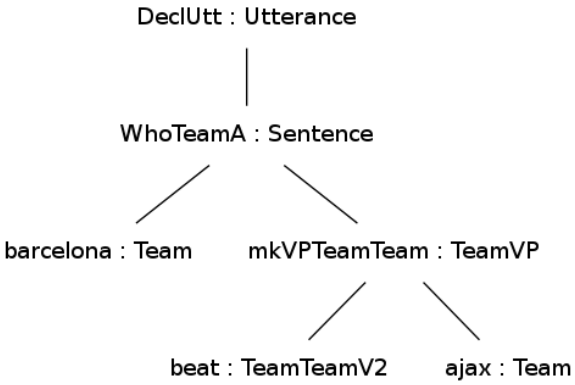

Figure 4.3: (Barcelona)$_{rheme}$ (beat Ajax)$_{theme}$.

Another distinguishing feature discussed by Kruijff-Korbayová and Steedman (2003) is that of the level at which the analysis of information structure occurs. Some theories analyse the surface syntactic constituents, while others perform analysis at the level of semantic representation. We follow Steedman (2001) in considering semantic representation and syntactic structure to be isomorphic, and thus the level of analysis that is done on the semantic representation is also directly applicable at the syntactic level. This relates well to the approach taken by GF, where semantic categories in the abstract syntax are associated with syntactic categories in the concrete syntax.

Finally, Kruijff-Korbayová and Steedman (2003) refer to the issue of the underlying discourse semantics assumed by various theories of information structure. Certain theories leave the question of semantics at the intuitive level, while others take a specific formal approach to it. These can be divided into those using a form of "update" semantics and those using Alternative Semantics. Once again, following Steedman (2001), we take the latter approach, set out by Rooth (1992). The two alternative sets used in this scheme is that of the theme alternative set and the rheme alternative set. The two alternative sets present in each utterance govern the contrastive focus assigned to words in the theme and rheme, respectively. For example, given the question-answer pair in example (2), the theme of the answer is "Barcelona won" and the rheme is "yesterday's game". The rheme alternative set, then, are all those games that Barcelona could have won. The word "yesterday" distinguishes the particular game in question, and therefore receives contrastive focus (Steedman, 2012).

(2) a. Which game did Barcelona win?
    b. Barcelona won yesterday's game.

Another consideration regarding discourse semantics is the notion of recency when identifying elements new to the discourse. Yule (1981) and Krifka (2008) both refer to the phenomenon of the transitory status of newness. Yule (1981) distinguishes between the states "new", "current" and "displaced", arguing that as the event of some element being mentioned recedes in time, the status of the element changes. Consequently, the current element is defined as the most recently established element in the discourse, while all elements mentioned before it are said to be displaced. Applying this to the notion of contrastive focus in this research means that newly mentioned elements can only be said to be contrastive if they contrast current elements, not displaced ones. Therefore, this notion of the decay of an element's newness must be factored in when determining which elements in an utterance ought to be marked as focused.

The view taken of information structure can be summarised as follows.

1. Each utterance consists, non-recursively, of two parts, a theme and a rheme. This is the first tier of an utterance's information structure.
2. The second tier identifies the elements that receive contrastive focus, and, this tier being orthogonal to the first, such elements can appear in both the theme and the rheme of the utterance.

3. The underlying discourse semantics is based on Alternative Semantics, and also employs the notion of recency to determine whether an element is contrastive to a previously mentioned element in the discourse.

The abstract syntax of GF, which defines the legal tree structures in a grammar, is the vehicle for encoding the two tiers of information structure. GF's linearisation functionality is employed to produce strings of enriched natural language that are marked up to show the underlying information structure contributed by both tiers. To show how this is achieved, we must first look more closely at the two tiers of information structure.

The first tier, where the theme/rheme distinction is made, manifests on the phrase level and communicates the structure of the sentence as consisting of two information structural parts that are related to the discourse context with regards to their givenness/newness. The second tier manifests on the word level and communicates the contrastive relationship of specific elements of the utterance to previously mentioned elements in the discourse. This research treats the two tiers as completely orthogonal, and therefore the mechanisms for determining their contribution to the tree structure of the utterance are required to be independent. Consequently, in the utterance produced by the system, the theme/rheme structure of the tree is determined by analysing the question that is to be answered, while the focused status of elements in the tree are determined by analysing recently established discourse elements, distinguishing between current and displaced non-new elements. How this is reflected in the abstract syntax tree is illustrated in the following example.

Suppose the question "Which game did Chelsea win?" has the answer "Chelsea won the game against Ajax". With regards to the first tier of information structure, the theme of the answer is "Chelsea won", while the rheme of the answer is "the game against Ajax". Suppose also that no element in the theme is contrastive to some element in the discourse, and therefore, no element in the theme is focused. In the rheme, however, "Ajax" is focused, because it is the word that distinguishes this rheme from others that are possibilities in the discourse context (Steedman, 2012). Figure 4.4 shows how the two-way division of the sentence into `ContestTheme` and `ContestRheme` represents the first tier of information structure, while the function `ajax`, whose return type is `Team`, is wrapped by the function `focusTeam`, also of type `Team`, to represent the second tier of information structure. In section 4.1 we discuss the linearisation of this structure into enriched natural language.

Considering the view taken from Combinatory Categorial Grammar that the relationship between between semantic, syntactic and intonational structure is isomorphic (Steedman, 2001), in GF terms, it can be summarised by saying that the information structure is encoded in the abstract syntax, while the syntax and prosody are encoded in the concrete syntax. Consequently, the grammar discussed in section 5.3, defining utterances in the domain of football results, at once encodes the information structure, the syntactic structure as well as the prosodic structure. By employing GF's linearisation functionality, the grammar is used to produce enriched natural language utterances for use in

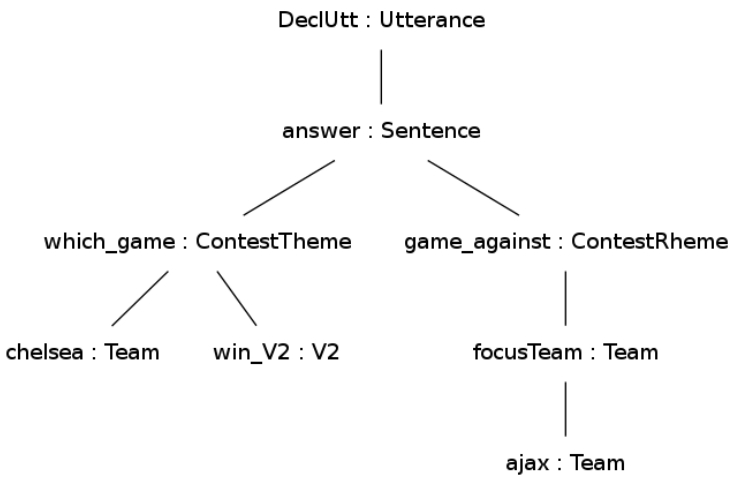

Figure 4.4: (Chelsea won)$_{theme}$ (the game against AJAX)$_{rheme}$

speech synthesis.

4.2 Syntax

Having determined which issues impact the abstract syntax, it is necessary to address some issues relating to syntax that manifest at the level of the concrete syntax. In other words, given that semantic - or more specifically in this case, information structural - categories in the abstract syntax are identified with syntactic categories in the concrete syntax, what is left is the issue of linearising the syntactic categories using GF.

Syntax itself may be used to convey information structure. An example of this is clefting, where the syntactic constituents are ordered so that new information appears at the end of the sentence (Ward and Birner, 2006). For example, the question "Which team beat Chelsea last season?" could have, among others, the following two answers:

(3) a. Barcelona beat Chelsea last season.
b. The team that beat Chelsea last season is Barcelona.
c. Barcelona.

Since the focus of this research is to improve the prosody of concept-to-speech systems, clefting and other syntactic means to communicate information structure are not considered. Also, the minimal answer in sentence (c) is also no considered. While in certain cases the rheme itself is a possible answer, at least

three arguments can be made in favour of repeating the theme of the answer utterance in a spoken question-answering system.

The first and most practical argument is that repeating the theme implicitly acknowledges the content of the question. Since automatic speech recognition can fail even when the system is sufficiently confident of its recognition, repeating the theme in the answer ensures that the user knows that the right question is being answered.

Secondly, in cases where the theme contains a focused element, repeating the phrase provides useful discourse continuity. A theme-focused element contrasts a recently mentioned element in the discourse, and acknowledging it as such by repeating the theme with the appropriate element focused, communicates the full information structure of the answer.

Thirdly, although conjunctions of questions are out of the current scope, answering such questions by giving only the rhemes as answers is unnatural and could even be confusing. Consider the following question and some possible realisations of the same answer.

(4) a. Who won yesterday's game and who beat Arsenal?
b. Wigan and Chelsea.
c. Wigan won yesterday's game and Chelsea beat Arsenal.
d. Wigan and Chelsea beat Arsenal.
e. Wigan won yesterday's game and Chelsea.

What is particularly important to note is that when answering conjoined questions, both themes should be repeated or left out. Sentence (d) and (e) show clearly the ambiguities and unnatural utterances that might arise from being inconsistent in this respect. However, it should also be clear that sentence (c) is a more understandable answer to the question than sentence (b). In fact, (b) can also be an answer to only "Who beat Arsenal?", and it might not be clear to the user whether both questions have been answered by the system.

Therefore, in all cases, sentences similar to example (3a) are used, and it is left to the prosody generation component of the synthesis system to communicate the correct information structure.

The syntactic scope of this research must also be delineated. Since the correlation between the intonational tunes (i.e. prosody) and information structure of wh-questions is relatively well-understood (Hirschberg, 2006b; Prevost, 1995a; Steedman, 2001), it provides a suitable first delineation of the linguistic scope of the research. A second, practical delineation is the decision to focus on utterances containing verbs that take either one, two or no objects; that is, transitive, ditransitive or intransitive verbs. Verbs that take sentence-like arguments, such as "say" in "She says that John saw London", or "ask", which takes a direct object and a question as arguments as in "He asked me if John saw London", are not covered. Other verbs not covered include those that take infinitive verbs

(want to read) and adjectival phrases (became rich). This results in the broad syntactic scope being limited to wh-question pairs containing only main clauses.

Some limitations on the kind of syntactic constituents that occur lower down in the hierarchy, such as noun phrases, are also imposed. These decisions were informed by the chosen domain, as well as by the practical consideration of including the necessary syntactic diversity to demonstrate the effect of the two-tiered view taken of information structure. Therefore, the majority of noun phrases consist of atomic proper names relating to football teams and players. However, the ability to modify common nouns that appear in noun phrases (such as "game" and "goal") with genitive constructions (such as "yesterday's game"), prepositional phrases ("the game against Ajax") and adjectival phrases ("the second goal") is supplied. It is also possible to modify verb phrases and slash verb phrases with adverbs indicating time ("beat Chelsea yesterday").

4.3 Prosody

In this section, we discuss the proposed markup scheme for producing enriched natural language. This scheme should provide a synthesis system with the information required to produce prosodically-improved speech synthesis. The purpose of the markup scheme is to make explicit the information structure of the utterance, so it should be general enough to be usable with different synthesis systems, while being detailed enough to provide all the information required to produce the appropriate prosody.

Synthesis engines may choose to follow one of two approaches when using enriched natural language input. On the one hand, the markup that forms part of the enriched natural language may be used in the training of a prosodically-informed synthetic voice. The information encoded in the markup would, along with the analysis of the natural language component, be incorporated into the features of the data on which the prosodic behaviour of the voice is trained. On the other hand, it may also be used directly to explicitly manipulate an existing voice. In this case, the markup would form the basis on which specific commands are issued to the system to alter aspects of the synthesised speech, whether it be pitch, volume, duration etc. These considerations also point to the need for the markup scheme to be sufficiently general so that it may be useful in both approaches.

One possible system for communicating the desired prosody resulting from the underlying information structure is autosegmental-metric (AM) theory, "which describes contour solely in terms of a small number of compound tones defined in terms of as few as two abstract pitch-levels, high (H) and low (L), from which actual contours can be derived algorithmically" (Steedman, 2012). Steedman and others (Steedman, 1991; Prevost and Steedman, 1993; Steedman, 2012), have argued extensively that certain tunes described using the AM system are typically associated with theme and rheme. Specifically, theme is associated with the tune L+H*LH%, which means that it consists of a L+H* pitch accent,

an L phrase accent and an H% boundary tone. Sentence initial rheme is associated with H*L (a H* pitch accent and L phrase accent) and sentence final rheme with H*LL% (a H* pitch accent, L phrase accent and L% boundary tone) (Steedman, 1991; Hirschberg and Pierrehumbert, 1986).

As is evident from analysing the meaning of the AM descriptions, the tunes described above result from an interaction between the two tiers of information structure. The different pitch accents indicate on which words the focus lies, while the tunes as a whole communicate whether the phrase is a theme or a rheme. However, if the mechanism for determining focus does not require knowledge of whether the element to be focused belongs to a theme or a rheme, thereby confirming the orthogonal nature of the two tiers of information structure, then for this kind of markup to reflect the correct pitch accent requires that the linearisation process knows the location of the focused item in the tree. However, GF's linearisation process is compositional, which means that information can only be passed up the tree, not down. One solution is to produce markup which, instead of communicating directly the desired prosody, rather indicates the information structure of the utterance. That is, focused items are simply marked as focused, and not as theme-focused or rheme-focused, and phrases are marked as theme or rheme.

The alternative is to relax the requirement of orthogonality of the two tiers somewhat, and to mark elements specifically as theme-focused or rheme-focused, via the AM pitch accents. However, this would lead us, firstly, to sacrifice simplicity during the process of constructing the abstract syntax tree. Secondly, it seems prudent to assume as little as possible before the linearisation phase with regards to prosodic realisation. For example, in some cases the rheme appears mid-sentence, with the theme part being split into a theme and tail. In such cases it might not be clear how the prosody of the tail should be realised in a universal way.

Finally, no information is lost by simply marking elements as focused, since their presence within the phrase marked as theme or rheme will make it clear what kind of pitch accent they should receive. Retaining the information structural designations within the markup scheme also allows for the implementation of other theories regarding prosodic realisation of the two tiers of information structure.

Figure 4.5 shows how the information structure of the rheme of the example in section 4.1 relates to the result of the linearisation by GF, using a purely information structurally informed markup scheme. The scheme uses XML-style tags, with the names "theme" and "rheme" (and "tail", if applicable) for the first tier, and "foc" for the second tier. The way in which this particular scheme is implemented is described in the next chapter.

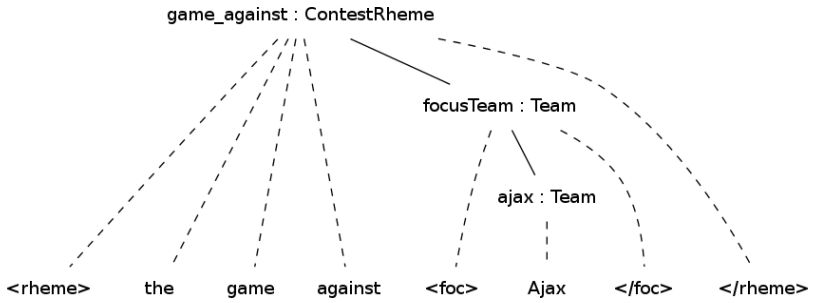

Figure 4.5: the game against AJAX

Chapter 5

Implementation

5.1 Introduction

This chapter is structured around the research questions put forward on page 19. Section 5.2 covers the extension of the GF English Resource Grammar Library (RGL) in order to answer the first two subquestions:

1. How should the GF English RGL be extended to provide functions for building trees whose structures encode information structure?
2. How should the information structure encoded in the tree structures be rendered when the tree is linearised?

In section 5.3 we discuss the development of an application grammar in order to answer the third subquestion:

3. How should the extended GF English RGL be used to build an application grammar for a question-answering system, where question-answer pairs are structurally related in terms of information structure?

Section 5.4 shows the implementation of a question-answering system in order to answer the fourth subquestion:

4. How can a question-answering system use such an application grammar to compute semantically correct answers to questions?

Finally, in section 5.5 we discuss the approach taken to produce prosodically–improved synthetic speech from the output of the question-answering system in order to answer the final subquestion:

5. How can the output of the question-answering system be used to improve the prosody in a synthetic voice?

This chapter presents the work done in order to answer the research questions. In chapter 6, this work done is evaluated with regards to the research questions.

5.2 Extending the GF English Resource library

In order to handle the two tiers of information structure in GF, two goals must be achieved. Firstly, in terms of marking phrase-level theme and rheme, the GF English RGL must be extended with the required functions for constructing sentences from various syntactic categories that correspond to themes and rhemes. Secondly, a way of marking syntactic categories as focused must be supplied.

Regarding the phrase level extension, the functions available in the RGL must be useful for the construction of both wh-questions and their corresponding answers. The wh-questions in view introduce information that appears in the answer as the theme, while the rheme of the answer contains the information that was sought for. The questions considered here have simple main clauses as answers where the rheme is either the subject, the object or some adverbial phrase. It follows that the remaining syntactic categories that constitute the answer sentence, that is, the theme, are introduced in the question. Regarding the marking of focus, it should be possible to mark syntactic categories lower down in the syntax hierarchy as focused during the linearisation process.

In GF terms, the requirement, therefore, is that it should be possible to build an answer tree from two syntactic categories, corresponding to the information structural categories of theme and rheme, and a question tree from only the syntactic category corresponding to the theme. This offers the ability to compute answer trees from question trees in a straightforward way. This syntactic division must be exploited in the linearisation process by marking the two linearisation categories (`lincat`s) as either theme or rheme. For marking focus, a similar marking capability should exist for categories lower down in the syntax hierarchy. Both the theme/rheme marking and the focus marking should be useful for the production of prosodically–natural synthetic speech. This requires a flexible approach, in which the markup should be user defined to some degree.

This chapter deals with the marking of `lincat`s first, and then discusses the extension of the RGL with the categories and functions necessary to model information structure as syntactic structure.

```
lincat ClSlash = {
  s : ResEng.Tense => Anteriority => CPolarity => Order => Str;
  c2 : Str ;
  a : Agr ;
  gapInMiddle : Bool } ;
```

Figure 5.1: Linearisation category for `ClSlash`

```
oper mark : Str -> Str -> ClSlash -> ClSlash =
   \m,n,cls -> lin ClSlash
      { s = \\t,ant,c,o
            => "<"+m+">" ++ cls.s!t!ant!c!o ++ "</"+m+">" ;
        c2 = "<"+n+">" ++ cls.c2  ++ "</"+n+">" ;
        a = cls.a ;
        gapInMiddle = cls.gapInMiddle } ;
```

Figure 5.2: `mark` overloaded for `ClSlash` and two strings

5.2.1 Operation for marking categories

In GF, while `fun` functions are used to construct trees, operations (or `oper`s) use lambda abstraction to abstract away similar behaviour in `fun` functions. These operations can also be overloaded for different categories (Ranta, 2011). An `oper` named `mark` was defined in a new resource module called MarkupEng.gf (see appendix C, subsection C.1.1). Its function is to take one or two strings and a syntactic category, and add XML-style tags to the string components of the language category. For example, for the categories with type {`s : Str`} (e.g. `Prep`), the operation with one string is defined as

```
oper mark : Str -> {s : Str} -> {s : Str} =
       \m,x -> { s = "<"+m+">" ++ x.s ++ "</"+m+">" } ;
```

The linearisation category for `ClSlash` is shown in figure 5.1. Figure 5.2 shows how `mark` is overloaded for this category.

Note that if the `c2` field is empty, the tags around it still appear. The strings passed as arguments are used as tag names, and could typically have values such as 'theme', 'rheme', 'emph' or 'focus'. So for instance, we might expect to mark a `ClSlash` with 'theme', since themes are identified at the phrase level, and a `Prep` with 'focus', since focus is identified at the word level. In section 5.3 we show how the operation was used to mark both tiers of information structure in an application grammar.

Linearisation type	Category
{s : NPCase => Str ; a : Agr}	NP
{s : AForm => Str}	A
{s : Case => Str ; n : Number ; hasCard : Bool}	Num
{s : Agr => Str ; isPre : Bool}	AP
{s : Str}	Prep, Subj
{s : Str}	AdV, Predet
{s1,s2 : Str ; n : Number}	Conj
...	Det
{s : Agr => Str}	Comp
...	ClSlash
...	Cl
...	VPSlash
...	VP

Table 5.1: List of categories for which `mark` is overloaded

Table 5.1 shows a list of the categories for which `mark` is overloaded. (The linearisation types of the more complex categories are not shown.)

5.2.2 New functions

In order to mark the information structure of utterances, the English RGL was extended with new functions in the ExtraEng.gf module (see appendix C, subsections C.1.2 and C.1.3). The scope of the current work is limited to simple main clauses of declarative sentences, since these are typically the kinds of utterances that a spoken question-answer system would be expected to produce as answers to wh-questions. The extension of the RGL was guided by a test set of 40 wh-question-answer pairs (see appendix B, section B.1), covering among them essentially three different ways of constructing answers to wh-questions from theme and rheme phrases.

As mentioned in chapter 4, main clauses are seen as consisting of two information structural parts: theme ("what the participants have agreed to talk about") and rheme ("what the speaker has to say about the theme") (Prevost and Steedman, 1993; Prevost, 1995a; Steedman, 2001; Prevost and Steedman, 1993). In some cases, this division coincides with a two-way division that can be expressed in a syntactically traditional way, e.g. `NP -> VP -> Cl`. In this example, the rheme of the utterance is the subject noun phrase, and the theme is the verb phrase. Note that this runs somewhat contrary to the notion in traditional semantics that the subject of an utterance is the topic (or theme) about which the verb phrase makes a comment. Here, the theme (or topic) consists of both the verb and any applicable objects in the sentence, and the subject of the utterance is the rheme (or comment).

Objects and adjuncts could also be rhemes, however, and so in the case of objects, the theme would constitute the subject and the object, and in the case

of adjuncts, the theme would constitute the original clause, and the rheme would be an adverbial phrase. Constructing clauses in these two ways aren't typical in traditional syntax, but prove to be useful, since they provide a way to describe syntax as isomorphic to information structure.

The idea in GF is then to be able to provide at least three different ways for clauses to be constructed from two arguments. The first, from an NP and a VP, is already available. The categories for the second, that is, from a ClSlash and an NP, are also available, as are the categories for the third, but they have not been used in this way before. In the following sections, we firstly consider the case of predication, where a clause is formed from a `NP` and a `VP`, secondly the case where a clause is formed by adding an adjunct (or adverbial phrase) to a complete clause, and finally the case where slash categories are used, that is, where a clause is formed from a `ClSlash` and an `NP`. The discussion focuses on the phrase level marking of theme and rheme. Marking focus is presented in more detail in the discussion of the application grammar in section 5.3.

Clauses via predication

This is the simplest case, since the functionality for building clauses in this way already exists in the RGL. All that remained to be done was to ensure that the marking operations for `NP`s and `VP`s behaved correctly. While the marking of the noun phrase is straight forward, since it always appears contiguously, marking the verb phrase required a different approach. This is because the order in which the various elements of the verb phrase appear in the final linearisation depends on parameters that are fixed higher up in the syntax tree, such as mood and polarity. Therefore, besides the `mark` operation being overloaded to handle the "default" case (a declarative sentence with positive polarity and indicative mood), other operations were defined for handling the imperative and infinitive moods, and for the case where the verb phrase appears in a question. These functions mark the verb phrase at the phrase level, but operations for marking individual elements of the verb phrase (such as the auxiliary verb, the main form of the verb, the infinitive form of the verb etc.) were also added to deal with situations in which individual elements are required to be marked as focused at the word level.

Example (5) shows the linearisation of an utterance where the information structure corresponds to the syntactic structure achieved through predication. The tree in figure 5.3 illustrates this syntactic structure. Note that the root node has two children, and these children represent the theme and rheme of the utterance. The GF code for achieving the correct linearisation is as follows.

```
PredVP (mark "rheme" (UsePN chelsea_PN))
       (mark "theme" (ComplSlash (SlashV2a beat_V2)
                                 (UsePN arsenal_PN)))
```

(5) <rheme> Chelsea </rheme> <theme> beat Arsenal </theme>

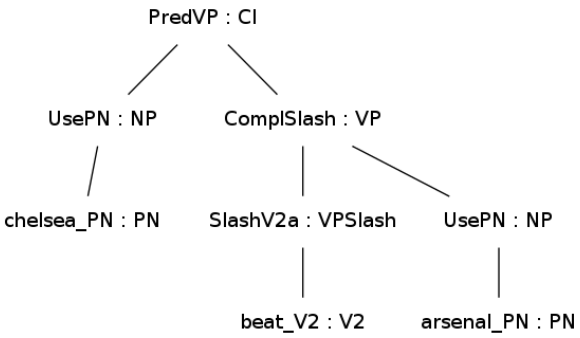

Figure 5.3: (Chelsea)$_{rheme}$ (beat Arsenal)$_{theme}$

Adding adverbial phrases (adjuncts)

In GF, as is typically the case, adverbs take verb phrases or sentences as arguments and produce modified verb phrases and sentences, respectively. In the case of verb phrases there is a distinction between adverbs that appear before and after the verb, as in "always sleep" and "sleep here" (Ranta, 2013). The distinction is made at the lexical level, with adverbs like "always" having type `AdV` and adverbs like "here" having type `Adv`. The `VP`'s `lincat` provides a field of type `Str` for the `AdV`, and a single complement field of type `Str` for the objects of the verb and the `Adv`s. The RGL also provides operations for inserting string elements before or after the complement. In most cases, this works well for English, since the object, if present, is always predictably the first element of the complement, and adjuncts can be added at the end of the complement without compromising grammaticality. However, this approach does not always allow modification of the verb phrase to result in the most natural rendering. Consider the following examples:

(6) a. How did Chelsea beat Arsenal yesterday?
b. Chelsea beat Arsenal yesterday well.
c. Chelsea beat Arsenal well yesterday.

(7) a. How did Jane play the Bach in the concert last week?
b. Jane played the Bach in the concert last week beautifully.
c. Jane played the Bach beautifully in the concert last week.
d. Jane played the Bach beautifully.

Currently, if we wanted to reuse the verb phrase of the question in the answer, the `lincat` of the verb phrase allows rendering of only (6b) and (7b), although they are not quite as felicitous as their c-counterparts. The reason that only the b-sentences can be rendered is because the new adverb can only be inserted

before the complement ("well Arsenal yesterday"), which would be ungrammatical, or after the complement ("Arsenal yesterday well"). This is consistent with the general trend in English to present new information (the rheme) towards the end of the sentence (Ward and Birner, 2006; Hajičová et al., 1995). However, it is clear that in some contexts it would be more natural to use (6c) and (7c), and leave it to the intonation contour of the utterance to make it clear what the new information is. In any case, since the c-sentences are clearly grammatical, it would be useful to be able to construct them by reusing the verb phrases in the questions. A simple solution would be to have the `lincat` of the verb phrase provide a field for the object and a separate field for the adjuncts. This would make it possible to provide an operation for adding new adjuncts between the object and the adjuncts introduced in the question.

Another argument for changing the RGL's `VP lincat` by splitting the complement field in two could be made by referring to the utterance in (7d). Although ellipsis does not form part of the current work, having two fields for the complement instead of one would enable one to provide an operation on the verb phrase introduced in the question that would linearise only the object and the new adjunct, while leaving out the "old" or "given" adjuncts.

Given the way the current `lincat` of the verb phrase is defined in GF, only the b-sentences can be rendered. This limitation does allow for a slight simplification to be made when answers that have adjuncts as their rhemes are constructed from their wh-questions. Since the RGL only allows the creation of `Cl`s where the verb phrase is linearised as the rightmost element, and since `Cl`s are always linearised with their complement element in the rightmost position, simply adding adjuncts to the right of `Cl`s results in the same string as adding adjuncts to the right of `VP`s. Therefore, the function `AddAdjunct : Cl -> Adv -> Cl`, which simply adds the linearised adjunct to the end of the token list, was added to the RGL. It provides a straight-forward way of constructing the answer from its theme and rheme: the `Cl` constituent represents the theme and the `Adv` constituent represents the rheme. Figure 5.4 shows the use of the function `AddAdjunct` to construct a tree in this way. The resulting linearisation is shown in example (8).

(8) <theme> Chelsea beat Arsenal </theme> <rheme> today </rheme>

ClSlash: a clause missing an object NP

The existing `ClSlash` category is used in GF for questions such as "Which goal did Torres score?", where the NP "Torres" and verb "score" make up a `ClSlash` that is missing its object NP. Now, a function was added that uses this same category to produce answers such as '(Torres scored)$_{theme}$ (the second goal)$_{rheme}$', where the theme is represented by the `ClSlash` and the rheme by the object NP. This structure can then be exploited to be marked for information structure. For example, the tree in figure 5.5 produces the following linearisation, when marked appropriately:

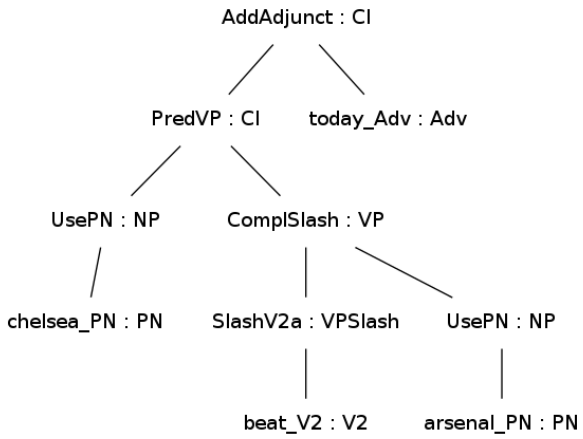

Figure 5.4: (Chelsea beat Arsenal)$_{theme}$ (today)$_{rheme}$

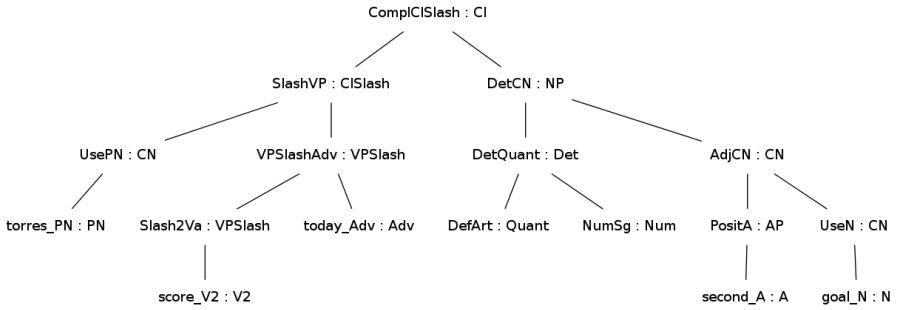

Figure 5.5: (Torres scored)$_{theme}$ (the second goal)$_{rheme}$ (today)$_{tail}$

(9) <theme> Torres scored </theme> <rheme> the second goal </rheme> <tail> today </tail>

That is, when ComplSlash is marked as `ComplClSlash (mark "theme" "tail" clslash) (mark "rheme" object)`, where `clslash` and `objectnp` have types `ClSlash` and `NP`, respectively, the string in example (9) is produced. Note that the rheme consists of the entire phrase "the second goal". A typical question resulting in this answer is "Which goal did Torres score today?". Strictly speaking, the only entirely new part of the answer would be the word "second". However, the entire phrase "the second goal" constitutes a prosodic phrase, and so is identified as the rheme (see example 51 on p.115 of Steedman (2001)). It is left to the focus-marking tier of information structure to determine that the pitch accent inside the rheme must be placed on the word "second", since it is the element that distinguishes this rheme from other possible rhemes.

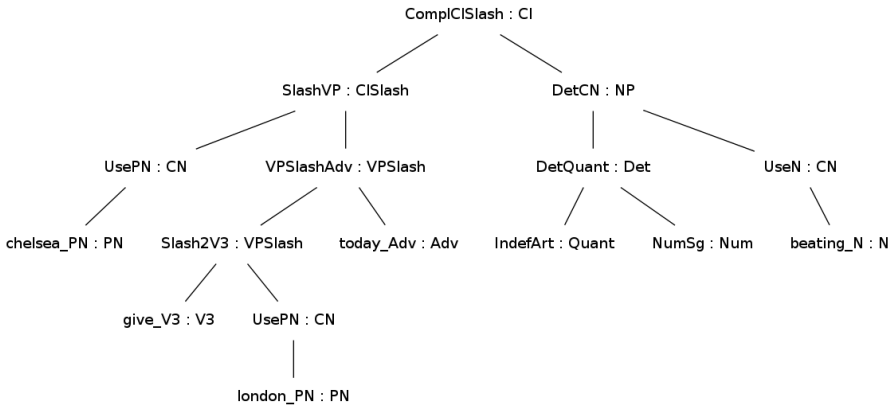

Figure 5.6: (Chelsea gave Arsenal)$_{theme}$ (a beating)$_{rheme}$ (today)$_{tail}$

The `ClSlash` category is constructed from an `NP` and a `VPSlash`, which is a verb phrase missing an object. Additional functions were added to provide more ways of constructing `VPSlash`es from transitive as well as ditransitive (or three-place) verbs. In the latter case, either of the objects may be missing. To illustrate the case where the direct object ("a beating") is missing, the tree in figure 5.6 produces

(10) <theme> Chelsea gave Arsenal </theme> <rheme> a beating </rheme> <tail> today </tail>

The tree in figure 5.7 shows the case where the indirect object ("Terry") is missing, and produces

(11) <theme> Chelsea paid </theme> <rheme> Terry </rheme> <tail> twenty thousand pounds today </tail>

This case, where the indirect object is the missing `NP`, required the addition of the function `Slash3V3a`. While the function `Slash3V3` allows construction of a `VPSlash` from `V3` and a single `NP`, the `NP` is assumed to represent the indirect object. The function `Slash3V3a` was added, which similarly allows the construction of a `VPSlash` from a `V3` and a single `NP`, but in this case the `NP` is assumed to be the direct object.

As with regular `VP`s, the presence of adjuncts in sentences built from `VPSlash`es had to be addressed. The function for completing `ClSlash`es with their missing `NP`s must place the `NP` after the verb, but before the adjuncts. The `ClSlash` `lincat` provides two fields: the first, `s`, representing the clause as a table that results in a string when tense, anteriority, polarity and order are fixed; the second, `c2`, a field of type string. The only way to correctly insert the noun phrase after the verb but before the adjuncts would be to ensure that the adjuncts

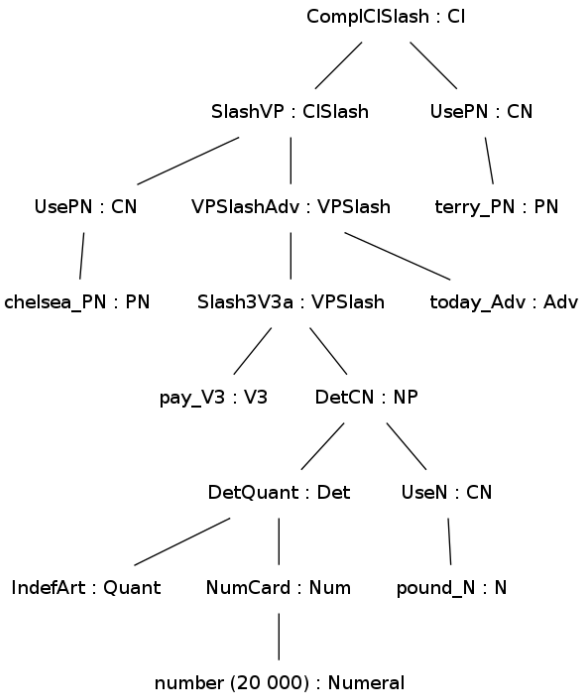

Figure 5.7: (Chelsea paid)$_{theme}$ (Terry)$_{rheme}$ (twenty thousand pounds today)$_{tail}$

of the clause are in the `c2` field. When the `ClSlash` is constructed, it receives the contents of its `c2` field from the `c2` field of the `VPSlash` from which it is constructed. Therefore, the adjuncts of the `VPSlash` should be in its `c2` field. However, the RGL only provides a function that adds adverbs directly to the complement field of the `VPSlash`. The result is that reusing the `VPSlash` when constructing an answer from a question produces sentences with ungrammatical word order.

(12) <theme> Ajax beat yesterday </theme> <rheme> Chelsea </rheme> <tail> </tail>

In the example given above, the adjunct "yesterday" appears before the object, instead of after it in the section marked by the `tail` tag. To remedy this, a function `VPSlashAdv` was created that takes a `VPSlash` and an `Adv` and produces a `VPSlash` in which the `Adv`'s string is placed in the `c2` field. Building the `VPSlash` in this way and then using the `ComplClSlash` function to supply the missing noun phrases correctly yields

(13) <theme> Ajax beat </theme> <rheme> Chelsea </rheme> <tail> yesterday </tail>

5.2.3 Conclusion

By adding the functions and operations discussed above, it was possible to use the RGL to build an application grammar that shows how three different kinds of wh-questions and their corresponding answers could be made available to a question-answering system. The next section discusses the application grammar, which handles wh-questions about football results.

5.3 Building an application grammar

5.3.1 Theme and rheme

The goals of the application grammar developed to model wh-question-answer pairs about football were, firstly, to show how information structure categories could be used to model the semantics of wh-questions and their answers, secondly, how reflecting this in the concrete syntax is simplified through the extension of the English RGL, and thirdly, to provide a basis for a proof-of-concept question-answering system whose output would ultimately result in prosodically–improved speech synthesis.

In chapter 3, an overview of the architecture of a GF grammar was given, emphasising the division between abstract syntax and concrete syntax. This

```
-- example: Who beat Ajax?
AgentQuestion : AgentTheme -> Question ;
AgentAnswer : AgentTheme -> AgentRheme -> Answer ;

-- example: Whom did Barcelona beat?
PatientQuestion : PatientTheme -> Question ;
PatientAnswer : PatientTheme -> PatientRheme -> Answer ;

-- example: When did Barcelona beat Chelsea?
ModQuestion : ModTheme -> Question ;
ModAnswer : ModTheme -> ModRheme -> Answer ;
```

Figure 5.8: Application grammar abstract syntax functions based on information structure

division gives rise to the interlingua structure of a GF grammar, where the central module is the semantic abstract syntax, while the terminal modules are the language-specific concrete syntaxes. The idea is for the semantics of the grammar's domain to be specified compositionally in the abstract syntax, leaving it to the concrete syntaxes to determine how the abstract categories relate to natural language categories in the various languages. In many cases, there is a clear mapping between the parts that make up an abstract utterance and the parts that make up a natural language utterance. For example, abstractly, describing some thing as having a certain property requires a function over two elements: a thing and a property. Concretely, in English, these two elements correspond to the subject noun phrase and an adjectival phrase, respectively.

In the case of wh-questions, it would therefore be ideal if the information structural components were the building blocks of utterances, with the concrete syntax mirroring this structure. Consider the GF functions in figure 5.8, which are based loosely on *thematic relation*, where `Mod` is used as an umbrella term for relations such as location, time, manner etc.

These functions define different ways of building questions and answers using information structural categories. It is then up to the concrete syntax to determine how the categories (`AgentTheme`, `ModRheme` etc.) should be represented and linearised. Where abstract categories correspond to syntactic categories in the RGL, these can be used directly. In section 5.2 we showed which categories in the RGL correspond to the different kinds of themes and rhemes used in figure 5.8. For example, `AgentTheme` is a `VP`, `PatientTheme` is a `ClSlash` and `ModRheme` is an `Adv`. The functions added to the RGL, as discussed in section 5.2, simplifies the process of mapping the functions over abstract categories to functions over linearisation categories. The API, which serves all languages in the RGL, is defined for functions the languages have in common, but not for functions that are particular to a specific language. Figure 5.9 shows the linearisation functions for the example grammar. Note that API functions are used

```
lincat
Question = QS ; Answer = S ;
PatientTheme = ClSlash ;
PatientRheme = NP ;

lin
PatientQuestion patientTheme =
    mkQS pastTense (mkQCl whoSg_IP patientTheme) ;

PatientAnswer patientTheme patientRheme =
    mkS pastTense (ComplClSlash patientTheme patientRheme) ;
```

Figure 5.9: Application grammar concrete syntax functions based on information structure

```
PatientAnswer patientTheme patientRheme =
    mkS pastTense (ComplClSlash (mark "theme" patientTheme
                                (mark "rheme" patientRheme) ;
```

Figure 5.10: A concrete syntax function using the `mark` operation

whenever they are available. Otherwise, the functions defined in the extended RGL are used directly.

Using the `mark` functionality discussed in section 5.2, the final line in figure 5.9 could be changed as indicated in figure 5.10.

This strategy shows how the first two goals of the application grammar can be achieved in general. The specific football application grammar that was developed uses this approach, and covers questions starting with "who", corresponding to the `Agent` question-answer pair shown above, questions starting with "whom", "which", "how much" and "how many", corresponding to the `Patient` question-answer pair, and "when", corresponding to the `Mod` question-answer pair. In the case of the `Patient` pairs where the verb in question was ditransitive, for "whom" questions the answer constituted the indirect object ("Whom did Ajax offer twenty thousand euros"), and for "how much" and "how many" questions the answer constituted the direct object ("How much did Ajax offer Daniels").

The general strategy described above, however, is not specific enough to enforce domain semantics. For example, suppose the verbs `sign_V2` and `score_V2` was available in the grammar, as well as a team, `Barcelona`, a player, `Messi`, and an amount of goals, for example, `num_goals`. We would expect sentences like "Barcelona signed Messi" and "Messi scored three goals". That is, `Barcelona`

and Messi can both act as AgentRhemes, while Messi and num_goals can both act as elements of AgentThemes. However, there is no mechanism for disallowing the sentences "Messi signed three goals" or "Barcelona scored Messi".

The first step is to distinguish between types of "things" in the domain, such as agents and patients, or anything that would typically be expressed as a noun phrase in natural language. In the football grammar, the following can be defined: Player, Team, Amount, Point and Contest. The second step is to identify the action verbs as having certain types, beyond merely the syntactic requirement of identifying them as intransitive, transitive or ditransitive. Specifically, each action type should be defined in terms of which agent type and patient type(s) are applicable to it.

In GF, this would most elegantly be handled by so-called *dependent types*, which offer a way to make types dependent on parameters. Instead of using the categories in the previous paragraph directly as types for the "things" in the grammar, we use them for defining the kinds of semantic relations needed in the grammar. Then, we make our "things" dependent on having a certain "kinds". For example, we could have the following functions for Things.

```
ajax    : Thing Team ;
chelsea : Thing Team ;
messi   : Thing Player ;
```

Next, we make the actions dependent on the kinds of their agents and patients in the following way.

```
beat  : Action Team Team ;
sign  : Action Team Player ;
```

This is propagated up the tree. A verb phrase from a transitive verb combines an Action with a Thing, which represents the patient. We construct a verb phrase for a transitive verb in the following way.

```
mkVP2 : (agent : Kind) -> (patient : Kind) ->
        Action agent patient -> Thing patient ->
        VP agent ;
```

The Action defines the Kinds of its agent and patient, and the function ensures that the second Kind on which the Action is dependent corresponds to the Kind of the Thing passed as the patient. This constructs a VP of a certain Kind, which represents the Kind of Thing it expects as an Agent. The following function for constructing the complete clause, shows similarly how a semantically correct clause is constructed by ensuring that the Kind of the Thing and the VP correspond.

```
mkClause : (agent : Kind) -> Thing agent -> VP agent -> Clause ;
```

Unfortunately, since there is currently no support in the GF C-runtime for dependent types, these kinds of semantic dependencies must be modelled in a different way in the grammar. That is, the types in the grammar must explicitly indicate which actions are applicable to which agents and patients. Below are some examples of the agents, patients and actions in the grammar with their types.

```
ajax    : Team ;
chelsea : Team ;
messi   : Player ;

beat    :   TeamTeamV2 ;
sign    :   TeamPlayerV2 ;
score   :   PlayerPointV2 ;
offer   :   TeamPlayerAmountV3 ;
```

The convention followed was that the agent is named first, then the patients, if applicable, and then the syntactic type of the verb itself. For example, `sign` would be an action a `Team` does to a `Player`. These actions could now be used to construct the information structural category of theme.

In the case of agent-questions, verbs are supplied with patients, if applicable, to construct the theme, as shown in figure 5.11. Similarly, in patient-questions, verbs are supplied with an agent and all required patients but one. Figure 5.12 shows the direct object being supplied, while figure 5.13 shows the indirect object being supplied. In mod-questions, verbs are supplied with both the agent and all patients to construct the theme, like in figure 5.14. Note that in all cases the `Question` node has only one child.

Each question function has a corresponding answer function, listed in table 5.2. The function types listed in the last column show the different kinds of themes and rhemes. All italicised types represent rhemes, while all the other types, except `Sentence`, represent themes. In fact, the question function types can be inferred from the answer function types by removing the rheme type in each case.

Figures 5.15, 5.16, 5.17, 5.18 show the answers to the wh-questions discussed on page 61. The captions show the text that is output by the grammar. The tags `rheme`, `theme` and `tail` were chosen to make it clear which syntactic constituents represent the information structural constituents. A simple pass over the output of the grammar replaces these tags with Speech Synthesis Markup Language (SSML) tags. This is discussed in more detail in section 5.5.

Three things should be noted: firstly, that each `Answer` has two children, corresponding to the theme and rheme of the utterance; secondly, that the theme-child subtree corresponds exactly to the single child subtree of the `Question`s

Question function	Answer function	Answer function type
`WhoTeamQ`	`WhoTeamA`	*Team* `-> TeamVP -> Sentence`
`WhoPlayerQ`	`WhoPlayerA`	*Player*`-> PlayerVP -> Sentence`
`WhenQ`	`WhenQ`	`FClause ->` *TimeA* `-> Sentence`
`WhichPointQ`	`WhichPointA`	`PointClS->` *Point* `-> Sentence`
`WhichContestQ`	`WhichContestA`	`ContestClS ->` *Contest* `-> Sentence`
`HowManyQ`	`WhichPointA`	`PointClS ->` *Point* `-> Sentence`
`WhomPlayerQ`	`WhomPlayerA`	`PlayerClS ->` *Player* `-> Sentence`
`WhomTeamQ`	`WhomTeamA`	`TeamClS ->` *Team* `-> Sentence`
`WhomPlayerQ2`	`WhomPlayerA2`	`PlayerClS2 ->` *Player* `-> Sentence`
`HowMuchQ`	`HowMuchA`	`AmountClS2 ->` *Amount* `-> Sentence`

Table 5.2: Question and answer functions

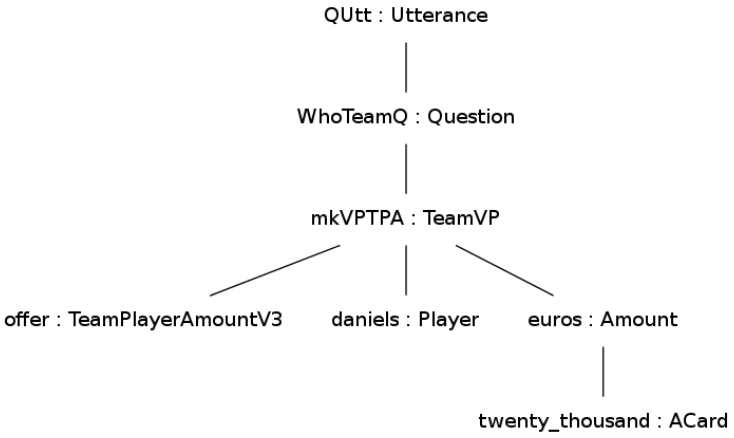

Figure 5.11: Who offered Daniels twenty thousand euros?

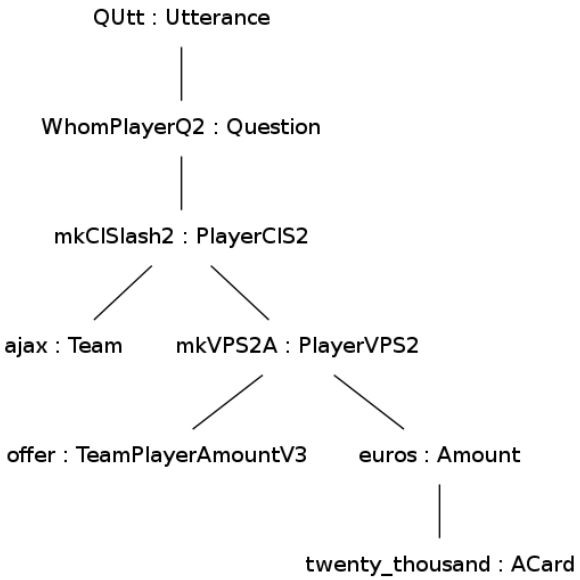

Figure 5.12: Whom did Ajax offer twenty thousand euros?

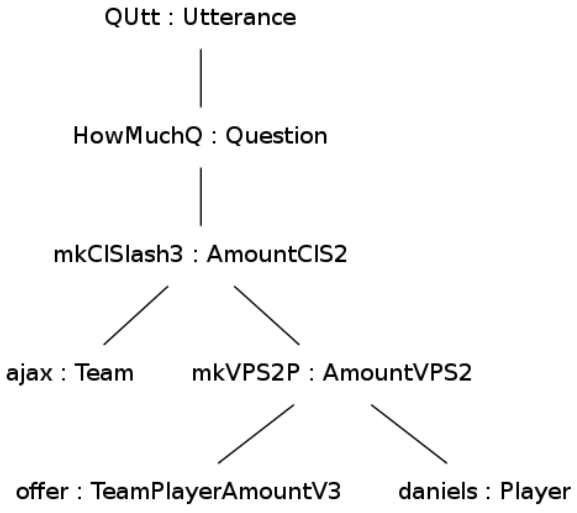

Figure 5.13: How much did Ajax offer Daniels?

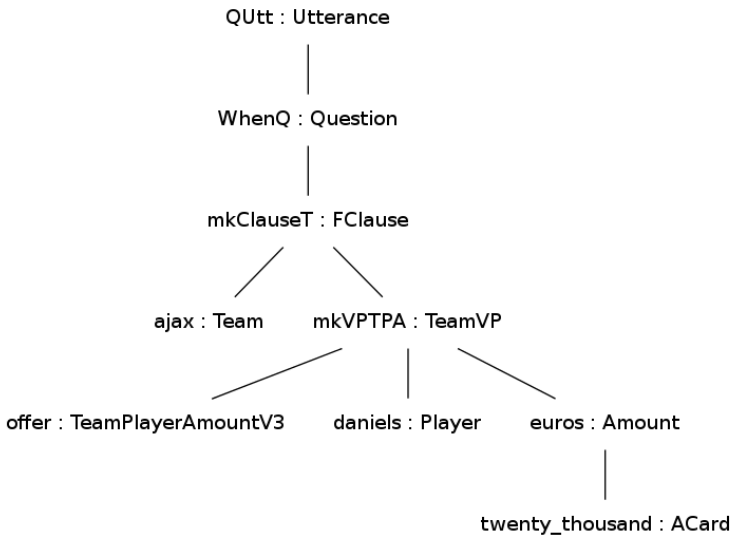

Figure 5.14: When did Ajax offer Daniels twenty thousand euros?

in figures 5.11 to 5.14; and thirdly, that the function names for constructing `Question` types can be converted to their unique corresponding `Answer` function names by replacing the final `Q` with an `A`. It is therefore clear that the computation of the answer tree from the question tree consists simply of combining the theme-child with the rheme-child under the corresponding answer type.

5.3.2 Focus

Orthogonal to the notion of theme and rheme as information structural categories is the notion of focus. Both the rheme and theme may contain focused elements, as discussed in chapter 4. The first step in marking focus is to identify the types in the abstract syntax capable of carrying focus. Usually, this should correspond to categories low down in the syntax hierarchy. In the football grammar, the following types were identified as the types that would carry focus: `Player`, `Team`, `Amount`, `TimeA` (time as an adverbial phrase), `TimeN` (time as a certain point in time, i.e. a noun) and the number types `ACard` (for amounts of money), `FCard` (for numbers of goals) and `FOrd` (ordinal numbers for talking about specific goals).

A wrapper function that returns the same type was created for each type. In the concrete syntax, this function invokes the RGL's `mark` operation with the tag `<foc>`. As discussed in chaper 4, focus is a function of the salience or contrastiveness of an element, and this is dependent on the discourse context of the utterance. Therefore, the focus in an answer cannot be computed by inspecting the abstract syntax tree of the question. It must be handled by

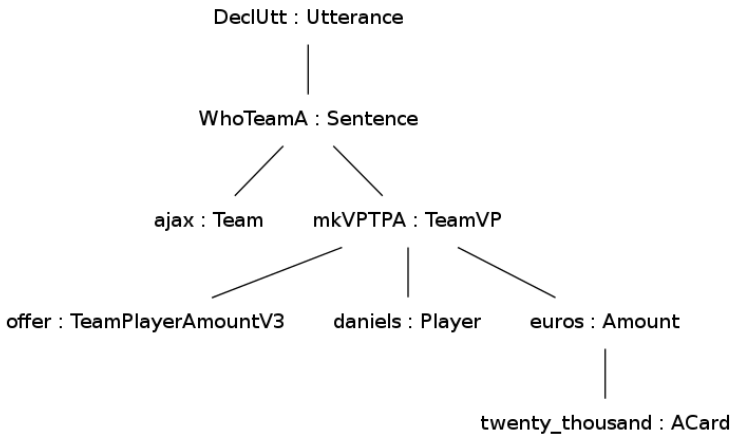

Figure 5.15: <rheme> Ajax </rheme> <theme> offered Daniels twenty thousand euros </theme>

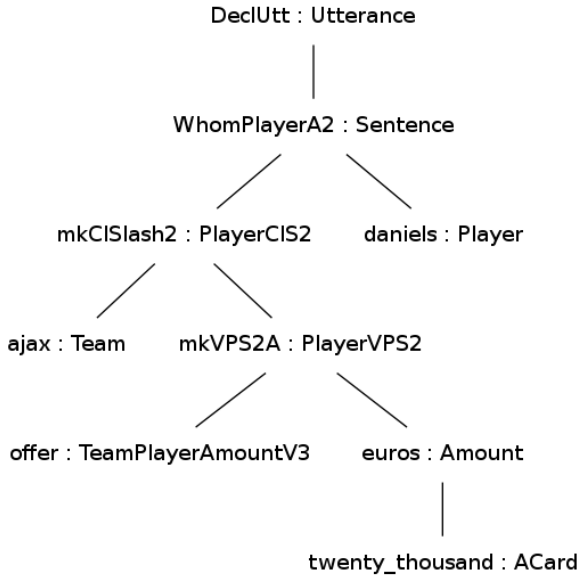

Figure 5.16: <theme> Ajax offered </theme> <rheme> Daniels </rheme> <tail> twenty thousand euros </tail>

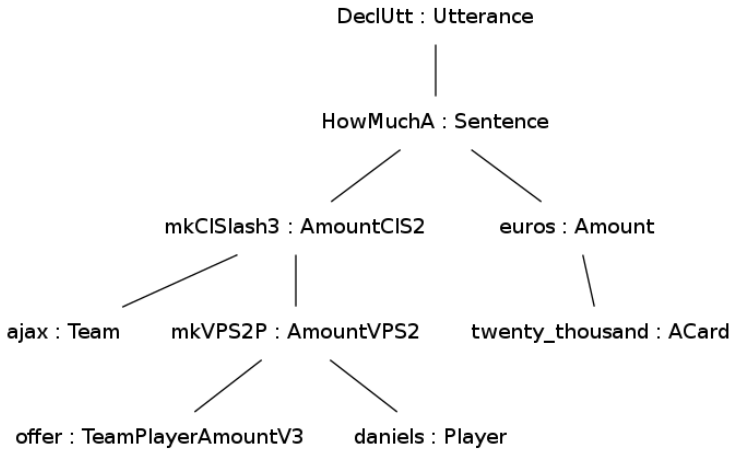

Figure 5.17: <theme> Ajax offered Daniels </theme> <rheme> twenty thousand euros </rheme>

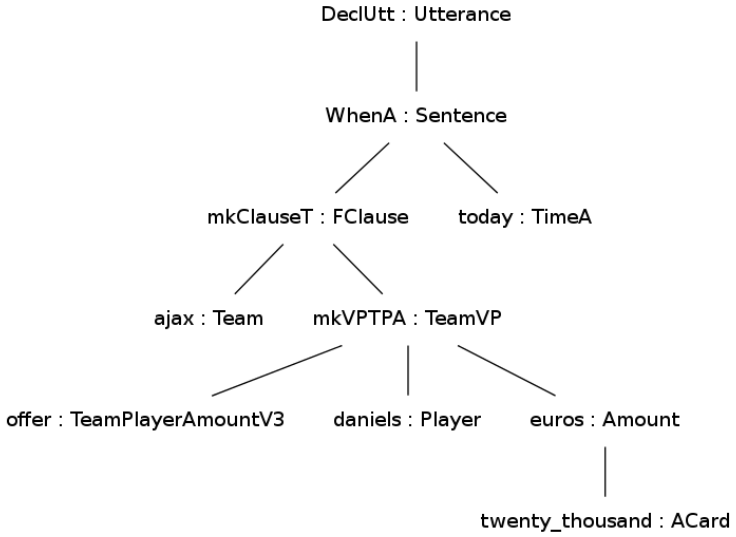

Figure 5.18: <theme> Ajax offered Daniels twenty thousand euros </theme> <rheme> today </rheme>

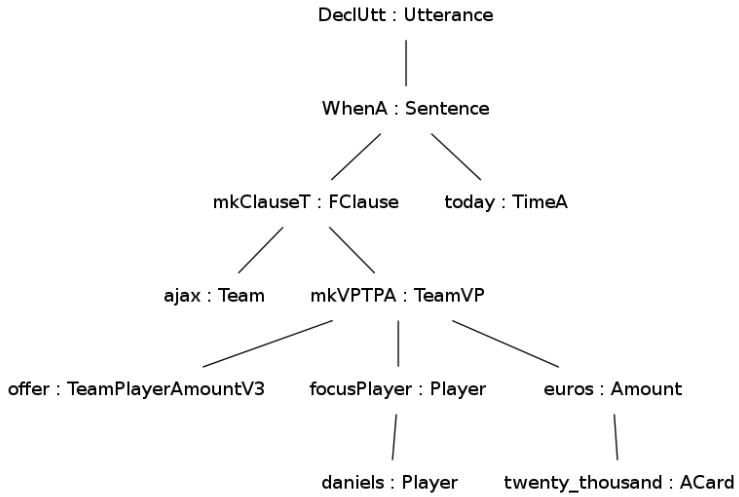

Figure 5.19: <theme> Ajax offered <foc> Daniels </foc> twenty thousand euros </theme> <rheme> today </rheme>

keeping track of the discourse state. This is discussed further in section 5.4.

This section concludes with an example of a focus function that is applied to an element in the theme of an answer. Suppose that the mention of the player "Daniels" is contrastive in the discourse. The answer tree of figure 5.18 is modified to the tree in figure 5.19, with the caption showing the linearisation.

5.4 A Question-Answering system

This section describes a question-answering system, `football_qa.py`, developed in Python and using the Python bindings to the GF C-runtime. In order to use the application grammar described in section 5.3, the grammar is converted to a PGF (portable grammar format) file. The Python bindings to the GF C-runtime allow general functionality available for PGF, such as parsing and linearising of utterances defined in the grammar, as well as ways of inspecting and constructing trees. The approach is not type-safe, however, so care must be taken to construct trees that are defined in the grammar to avoid runtime errors. The Python module has one access method, `reply`, which takes a question string and returns an answer string. Internal methods are defined for traversing the tree and analysing its nodes. Some methods illustrating how the system works are given in appendix C in subsection C.1.4.

The construction of the answer tree is a two-step process, and involves determining the structure of the answer tree, and analysing the question in order to compute the content of the rheme of the answer. Random answers may be

Prolog query	Question
sign(X,daniels,today)	Who signed Daniels today?
sign(ajax,X,yesterday)	Whom did Ajax sign yesterday?
sign(ajax,daniels,X)	When did Ajax sign Daniels?
sign(barcelona,X,_)	Whom did Barcelona sign?
offer(X,fischer,euros,twenty_thousand,_)	Who offered Fischer twenty thousand euros?

Table 5.3: Prolog queries from questions

produced by analysing the question to determine the type of the rheme, returning random content of the correct type, and constructing the answer tree by copying the theme and supplying the new content as rheme. This indicates how simple the process of constructing the answer is once the content of the rheme has been determined.

Of course, in a real question-answering system, the content of the rheme should depend on the content of the question, i.e. the content of the theme. One could think of the question, or in effect, the theme, as representing a query to be presented to a database in order to retrieve a result. Prolog-style queries are easy to read and are able to express the kinds of queries implicit in the wh-questions considered here. While traversing the theme to analyse its content, we therefore construct a Prolog query of the form $verb(arg1, arg2, ...)$, where the verb of the question becomes the name of the relation, and the subjects, objects or adverbs become its arguments. Once the query is constructed, it may be presented to a Prolog fact base in order to retrieve the content of the rheme of the answer. Table 5.3 shows examples of some queries inferred from questions presented to the question-answering system.

Since the focus of the research is not question-answering *per se*, but rather improving the prosody of spoken question-answering systems, a dummy mechanism was developed for resolving the constructed Prolog queries in the prototype system. However, the construction of the Prolog-style queries proves the essential point: that the meaning of the theme subtree can be determined by traversing and analysing it, and that this can be presented in the form of a query that can be resolved.

The matter of computing the answer tree was shown in section 5.3 to be straight forward. The response type is a declarative utterance, and so the root node is of type `DeclUtt`, as opposed to `QUtt`, which represents a question utterance. The question type is identified, and its corresponding answer type is found. This serves as the child of the root node. The question tree is then traversed in order to determine the query it represents. When this is known, the rheme part of the answer can be computed. The answer type is then supplied with two children: the theme child carried over from the question, and the newly computed rheme child. For figures showing such trees in more detail, see section 5.3.

An element is marked as focused in both the theme and the rheme if it is contrastive. We first discuss how theme-focus is handled in the question-answering

```
> who signed Fischer

<rheme> Barcelona </rheme> <theme> signed Fischer </theme>

> whom did Chelsea sign

<theme> <foc> Chelsea </foc> signed </theme>
<rheme> Daniels </rheme> <tail> </tail>
```

Figure 5.20: Example 1 of interaction with question-answering system

```
> who scored three goals

<rheme> Cole </rheme> <theme> scored three goals </theme>

> who scored two goals

<rheme> Messi </rheme> <theme> scored <foc> two </foc>
goals </theme>
```

Figure 5.21: Example 2 of interaction with question-answering system

system, and then how rheme-focus is handled.

The way in which theme-focus is determined is based on the contrastiveness of the elements solely in relation to elements that have been made salient by previous answers. Consider the short sequence of questions to the system shown in figure 5.20. Because "Barcelona" was mentioned in the first answer, "Chelsea", which is part of the theme, is constrastive to it in the second, and so receives focus. Another example is the sequence in figure 5.21.

The mechanism for keeping track of contrastiveness involves storing salient, or recently-mentioned, elements. A queue-like structure is maintained for all the types in the grammar capable of carrying focus. When an element of a type capable of carrying focus is presented in either the question or in the answer, it is added to the appropriate queue. To model the notion of decay in the discourse state, the queue has a fixed length. With each question-answer cycle, the oldest elements in all the queues are removed. Checking for contrast, therefore, involves checking whether it is non-salient and whether a member of its alternative set has been mentioned recently, which amounts to checking whether the queue of its associated type is non-empty and does not contain the element in question.

Rheme-focus has an extra dimension to it, especially pertaining to questions starting with "which" and "how many". In this grammar, questions starting

```
> which game did Ajax win

<theme> Ajax won </theme> <rheme> <foc> today's </foc>
game </rheme> <tail> </tail>

> which game did Chelsea win

<theme> <foc> Chelsea </foc> won </theme> <rheme> the game
against <foc> Barcelona </foc> </rheme> <tail> </tail>
```

Figure 5.22: Example 3 of interaction with question-answering system

with "who", "whom", "how much" etc., have as their rheme parts mostly atomic concepts in this grammar. In such cases, where all the words in the rheme are completely new (and perhaps contrastive), the focus tag is omitted. However, the "which"-questions and "how many"-questions have a slightly different dynamic. In asking "How many goals did Messi score?", the notion of "goal" is mentioned. This explicitly invokes the alternative set of specific goals, which in this case are cardinal numbers, and it also makes the notion of "goal" salient in the discourse. Even if this were the first question in a sequence, we would still expect the answer to put focused emphasis on the *number* of goals scored: "Messi scored <foc> two </foc> goals". The rheme constitutes the phrase "two goals", but the focus should not fall on the phrase as a whole. With the notion of "goal" being made salient by the question, the element that distinguishes the answer from other possible answers is the number.

Similarly, consider the interaction sequence in figure 5.22. The answer to the first question assigns focus to the noun "today". It is the element of the rheme that distinguishes it from other possible answers, because the notion of "game" has been explicitly introduced. The second answer shows, on the one hand, theme-focus, with "Chelsea" being contrasted with the previously mentioned "Ajax", and on the other hand it shows rheme-focus on "Barcelona", since it is the element that distinguishes it from other possible answers, with "game" already being a salient concept.

With this as output from a question-answering system, we can now turn to the process of using the output to produce prosodically-improved speech synthesis.

5.5 Producing speech with improved prosody

So far in this work, no attempt has been made to make the notion of "prosodically-improved" speech synthesis precise. To do this now, we briefly consider challenges facing the actual realisation of prosody in speech synthesis, and then look at criteria for measuring the success of speech synthesis approaches in produc-

ing improved prosody. The rest of the section describes the process that was followed to produce such prosodically-improved speech synthesis.

5.5.1 Realising prosody in speech synthesis

Two problems face the realisation of improved prosody in speech synthesis: firstly, that of deciding *which* prosodic effects are to be realised for a given utterance, and secondly, of deciding *how* such effects are, in fact, to be realised in the synthetic speech (Taylor, 2009). It should be clear that evaluating the success of solutions to the first problem also depends on the success of solutions to the second problem. That is, limitations on successful realisation of chosen prosodic effects necessarily affects the evaluation of the choice of prosodic effects itself in the resulting synthetic speech.

This research primarily concerns the computation of plausible prosodic effects for utterances based on their information structure. However, attempting to show how this leads to prosodically-improved speech synthesis depends on producing speech synthesis output that shows some prosodic improvement, which is in turn dependent on current techniques for realising prosodic effects. Recognising the limits of such techniques is helpful in deciding how best to measure the success of this work.

There are, broadly speaking, two ways in which a synthetic voice with certain prosodic behaviour can be developed from marked-up input. The first is to train a voice on data that has been marked with the necessary information, so that the resulting prosody is implicitly modelled when the voice is built. The second is to use an existing voice and explicitly manipulate its signal during the synthesis process, based on prosodic information in the input. The first approach would require specially prepared data, while the second would require a signal processing module capable of effecting the necessary changes. In either case, a module that supports the processing of the markup is needed.

Speech Synthesis Markup Language (SSML) is a W3C standard for speech synthesis, and its essential role is "to provide authors of synthesisable content a standard way to control aspects of speech such as pronunciation, volume, pitch, rate, etc. across different synthesis-capable platforms" (Burnett et al., 2004). Its tags and attributes for prosody and style provide what might be called interpreted and explicit control. An example of an interpreted tag would be "emphasis", which simply instructs the synthesiser to emphasise the content enclosed in the tag. It leaves the decision on how to achieve emphasis to the synthesiser, which might choose to modify the volume, pitch, rate, etc. in a way that fits its own requirements. An example of a tag with an explicit attribute is "prosody", whose "duration" attribute takes values in seconds or milliseconds, and directly instructs the synthesiser to give the enclosed content the exact duration specified (Burnett et al., 2010).

MARY (Schröder and Trouvain, 2003) is an open-source text-to-speech (TTS) engine that supports SSML input. It is distributed with several voices that can

be manipulated using SSML. This makes it a suitable fit for this research.

5.5.2 Criteria for assessing prosody in speech synthesis

Taylor notes that the two main goals of speech synthesis is, firstly, to communicate the desired message, and secondly, to have the voice sound human-like. In the literature, these two goals are referred to as intelligibility and naturalness (Taylor, 2009).

Let us consider naturalness first. In order to make a synthetic voice sound more human-like, the first course of action would be to improve the sound quality of the voice. This would include, for example, reducing buzzing, scratches and other mechanical noise. The next step would be to attempt to mimic other characteristics of human voices, particularly the prosodic aspects of human speech: appropriate intonation contours, phrase breaks, boundary tones and pitch accents. There have been various approaches to measuring the naturalness of synthetic speech, specifically with regards to prosody, including informal inspection of the signals produced, formal objective evaluations and formal subjective evaluations. Objective evaluations are usually concerned with goodness of fit between a synthetic voice and the original prosody, while subjective evaluations make use of perceptual tests. Such perceptual tests have themselves taken many different forms, including directly judging whether the synthetic speech is natural, whether the prosody itself is natural, and comparing synthetic speech with natural speech (Xu, 2012).

Intelligibility is for the most part considered to be solved, if by it one means that the general meaning, or propositional content, of the utterance can be understood by the user. Taylor refers to this as "decoding the message", and in this respect, one might even argue that this problem has been solved for several decades (Taylor, 2009). However, if it is the case that information structure is encoded via prosody in a language such as English, it follows that improving the prosody of a system should also improve its intelligibility. Here, however, the notion of intelligibility must be broadened to include communicating not only the propositional content of utterances, but also information structural content.

Consequently, for a system to claim that it produces "prosodically-improved" speech synthesis, it should produce a measured improvement of at least one, and ideally both, intelligibility and naturalness. Given the scope of this research, the primary criterium for evaluating the claim of prosodically-improved speech synthesis is improved intelligibility of information structural content. That is, the goal is to test whether there is a difference between the intelligibility of state of the art synthesis and modified synthesis based on the output of the kind of markup generated from a system as described in section 5.4.

In this section we discuss the attempt to produce prosodically-improved synthetic speech, specifically with regards to intelligibility of information structural content. The method, using SSML, is discussed, and the approaches taken to convert the output of the question-answering system to SSML markup are

shown. The limitations of the approach taken are discussed in relation to the two problems mentioned above, i.e. firstly, of deciding which prosodic effects to realise, and secondly, of realising these effects. Also, the decisions made with regards to which aspects of improvement were aimed for, are reviewed.

The following chapter evaluates the four stages of the pipeline; from extending the RGL, to producing an application grammar exploiting it, to the development of a system that produces answers to questions with information structural markup, and finally to producing prosodically-modified speech synthesis. The evaluation of the last stage includes a discussion of the results of a perceptual experiment conducted in order to compare the intelligibility of modified and unmodified speech synthesis, to determine whether "prosodically-improved" speech synthesis was achieved.

5.5.3 Speech synthesis with SSML

SSML elements and attributes are divided into three categories: document structure, text processing and pronunciation; prosody and style; and other elements. The second group, prosody and style, are primarily relevant to this research. These elements include the "voice" element, for choosing either specific voices, or voices that meet certain requirements, such as gender and age. The "emphasis" element has been mentioned before, the "break" element provides ways to insert silent breaks in the synthesis, while the "prosody" element provides control over the attributes "pitch", "contour", "range", "rate", "duration" and "volume". Most of these attributes can take so-called interpreted or explicit values. For example, the "pitch" attribute may take values in Hz to indicate the exact pitch with which the content should be synthesised, or it may take relative changes to the baseline pitch value in either Hz or percentages to indicate with which pitch the content should be synthesised relative to the baseline pitch value. These are clearly explicit commands to the synthesiser. However, it may also take the values "x-low", "low", "medium", "high", "x-high" or "default" (Burnett et al., 2010). The decision of how to realise these values are left to the synthesiser, and can therefore be called interpreted.

The advantage of the explicit values is clearly the precise control over the attributes it provides. However, it also requires precise use. That is, it is not very intelligent and its successful use is subject both to precise knowledge about the content to be synthesised, such as the expected duration and which syllables are to be emphasised, as well as knowledge of the output produced by the synthesiser, such as the pitch contour it produces for the input.

For example, suppose the word "Chelsea" is to receive a pitch accent. One might use the "prosody" tag with its "contour" attribute in the following way:

```
<prosody contour="(5%,+80%) (50%,0%)"> Chelsea </prosody>
```

The above markup indicates that at the 5% duration mark, the pitch level

should be raised to 80% of the baseline value, and at 50% duration it should return to the baseline value. This would, in general, produce a reasonable pitch accent on the word "Chelsea", and perhaps many other two-syllable words whose emphasis is on the first syllable. However, suppose the word to receive pitch accent is "United". Here, the natural emphasis of the word falls on the second syllable, and this syllable should therefore carry the pitch accent. Now, the relative duration of the first syllable is needed in order to know at which point the pitch level should be raised. It might be something like the following:

```
<prosody contour="(20%,+80%) (60%,0%)"> United </prosody>
```

However, producing this markup requires that decisions be made up front that are more effectively dealt with by the synthesiser itself. In fact, unless the exact behaviour of the synthesiser on all possible input is known beforehand, there is no reliable way of predicting precisely enough the way in which pitch accents should be realised. To complicate matters further, so far we have assumed that the synthesiser produces a sufficiently neutral prosody that can be manipulated safely by these explicit commands. However, it may be that certain words produced by the synthesiser already bear a pitch contour that resembles a pitch accent. Explicit manipulation of this might result in very unnatural output. These are the kinds of limitations on the explicit controls of SSML that must be born in mind when using it to produce prosodically-modified speech synthesis.

The interpreted values might simplify the problem, since they leave it to the synthesiser to make the decisions discussed above, and since it is already relied upon to make intelligent choices regarding the issues of syllable emphasis, markup commands could be interpreted intelligently. However, this binds the user of the system to the choices made by the synthesiser, which might not fit in well with the intended use. In the case of MARY and this work, this was unfortunately the case. This meant that a controlled approach had to be followed: both the vocabulary of the input, as well as the permissible structures, were controlled to produce reasonable results via explicit control commands. Fortunately, the syntactic scope of the grammar was not influenced, and all three kinds of wh-question-pairs could be presented to the system, and their prosody modified to an acceptable degree.

To illustrate the sufficiency of the information structural markup involving the tags "theme", "rheme", "tail" and "focus", a script was used that replaced the information structural tags with SSML markup, based only on the tags themselves. No other analysis of the output of the question-answering system was done. Due to the complicated nature of explicit control via SSML as discussed above, a simplified scheme for prosody was followed. Specifically, due to the unpredictability of the pitch contour of the synthesis output, the phrase level information structure was only indicated by the "break" element. A break was introduced between the theme and rheme phrases of the utterance, although not between the rheme and tail parts, if they were present. Focus was achieved by simulating pitch accents on the content using the "prosody" tag's "contour" attribute along with its "rate" attribute. The following is an example of an

utterance that has been marked up for both theme and rheme, as well as focus. The attributes and values of the "speak" element are omitted.

```
<speak ... >
 Barcelona won <break time="3ms" />
 <prosody rate="0.9" contour="(5%,+60%) (10%,+60%) (50%,0%)">
 yesterday's </prosody> game
</speak>
```

The values of the attributes were arrived at informally and were influenced by the general behaviour of the voice that was chosen. The male UK English voice, distinguished as "dfki-spike-hsmm", was chosen due to its relatively neutral and monotonous prosody. This ensured more predictability, especially in terms of pitch contour. Incidentally, this shows that when following the explicit control route described here, prosodically-rich data could present more of a risk to successful synthesis than data that exhibits less prosodic variation.

The grammar used in the question-answering stage to produce marked-up output was limited to include only pitch-bearing words where the emphasis falls on the first syllable. In terms of teams, examples include "Chelsea" and "Barcelona", players include "Fischer" and "Messi", while the adverb "yesterday" was allowed and "today" was not. The result was an improvement in the prosody of the speech synthesis, as is argued in the next chapter.

Chapter 6

Evaluation

In evaluating this research, we have to determine whether the implemented system puts us in a position to answer the research questions. The system consists of four components, each intended to answer certain subquestions, and these serve as the first guidelines for identifying the specific goals of each component in the system. Since the system is concatenation of dependent processes, the success of each component can be gauged by viewing it from the point of view of the component that directly depends on it. For the purposes of this discussion, we refer to this directly dependent component as the successor. Besides the relevant subquestions, an important question to be asked of each component is then: How does the component enable the successful implementation of its successor?

Therefore, with the success of each component being partly measured by its successor's use of it, we have to evaluate the system in reverse, so to speak. We start by considering the question-answering (QA) system, which produces the markup required for appropriately modified speech synthesis. Then we consider the application grammar that forms the basis of the question-answering system. We continue by considering the extension to the English Resource Grammar Library (RGL), which provides the application grammar with the required functions and operations. Finally, having thus evaluated the process of producing modified speech synthesis based on information structure markup, we report the results of a perceptual experiment to evaluate whether the modified speech synthesis is in fact an improvement in terms of its prosodic realisation.

6.1 Question-answering system

The purpose of the question-answering component is to produce answers, in the form of text strings, that are marked in such a way that it can be used to produce prosodically-improved speech synthesis. We elected for the question-

answering system to use markup that communicates the information structure of the utterance. As explained in section 5.5, the markup is abstract and could be used either in the training of a new synthetic voice or in the control of an existing synthetic voice via synthesis engines with support for SSML markup. We have elected the second approach for this research. The question-answering system may therefore be regarded as successful if the output can readily be converted into the desired SSML.

The first aspect to evaluate is the ability of the question-answering system to produce correct answer strings when provided with questions. In order to test this, 100 random questions were generated from the grammar and fed to the question-answering system. The output was checked by hand to confirm that the information structure marked in the answers were correct.

The next step is to evaluate the usefulness of the question-answering system's output. SSML, being a means of communicating the realisation of prosodic features, is used essentially to encode theory of prosody. To test the usefulness of our markup, we have to be able to produce SSML that encodes an established theory, and the tunes for theme and rheme described by Steedman (1991), following Hirschberg and Pierrehumbert (1986) and others, provide just this.

Suppose we have the following sentence, with its autosegmental-metric (AM) description.

(CHELSEA	won)	(the game against	ARSENAL).
L+H*	LH%		H*LL%

Here, the theme contains a focused element, "Chelsea", as well as other elements, while the rheme consists of unfocused elements followed by one focused element. In this case we might expect to replace the AM description with SSML as follows:

AM	SSML
L+H*	<prosody contour="(5%,-50%) (50%,+50%)>"
LH%	<prosody contour="(5%,-20%) (50%,+20%)> "
H*LL%	<prosody rate="0.9" contour="(5%,+70%) (50%,0%)">

The desired SSML markup for the entire sentence would be as follows:

```
<speak ... >
<prosody contour="(5\%,-50\%) (50\%,+50\%)> Chelsea </prosody>
<prosody contour="(5\%,-20\%) (50\%,+20\%)> won </prosody>
the game against
<prosody rate="0.9"
         contour="(5\%,+70\%) (50\%,0\%)> Arsenal </prosody>
<\speak>
```

Does the output of the question-answering system allow us to produce this SSML? For this sentence, the question-answering system would produce the following.

```
<theme> <foc> Chelsea </foc> won </theme>
<rheme> the game against <foc> Arsenal </foc> </rheme>
```

It is clear that we can infer from this output that the "Chelsea" element is theme-focused, that "won" makes up the rest of the theme, that "the game against" is part of the rheme, and that the rheme-focused element is "Arsenal". Having identified this structure, these elements can simply be wrapped in the SSML tags that corresponds to their information structure status, producing the desired SSML shown above. Whether such SSML markup produces prosodically-improved speech synthesis is addressed in section 6.4. At the moment, however, our concern is whether the question-answering system allows us to produce the desired SSML.

The example shows that the required mechanism involves, firstly, identifying the focused and unfocused sequences of words that make up the theme and rheme of the utterance, and secondly, assigning to each sequence an enclosing `prosody` tag with the appropriate attribute values, based on the information structural status of the sequence. In the example, a sequence identified as theme-focused is enclosed with `<prosody contour="(5%,-50%) (50%,+50%)>` `</prosody>`, while a sequence identified as rheme-unfocused does not receive any modification via the `prosody` tag.

The markup supplied by the grammar completely specifies the sequences of words that have the same information structural status in the two-tiered scheme, and this is sufficient for assigning the desired SSML markup. We may therefore conclude that the question-answering system produces useful output for modifying speech synthesis according to the theory of prosody of Steedman (2012) and others. This is a partial answer to research subquestion 5, relating to the ability of the question-answering system's output to lead to prosodically-improved speech synthesis.

6.2 Application grammar

The purpose of the application grammar is to provide the functions necessary for the question-answering system to produce answers from wh-questions. The grammar may be evaluated in terms of its coverage, as well as its contribution to the success of the question-answering system, whose input and output structures it defines. We have already established the successful functioning of the question-answering system. What remains is to understand how the application grammar enables the question-answering system to analyse its input and produce the correct output.

A test set of 40 question sentences was developed, which was intended to cover the required scope of the grammar (see appendix B, subsection B.1.1). In fact, the set of 40 sentences functions as a specification for the coverage of the grammar. If the grammar successfully parses each sentence, and if the test set can

be shown to cover the required linguistic scope of the research, the coverage of the application grammar may be deemed sufficient.

Let us consider the linguistic coverage of the test set first. The sentences cover the three distinct ways of constructing an answer from a theme and rheme, as discussed in 5.2, as well as instances of theme-focus and rheme-focus (see chapter 4). This means that they cover the cases in which the subject, object or some adjunct of a sentence functions as its rheme, while the other parts of the sentence constitute the theme. Therefore, the sentences provide coverage of the general syntactic structures required for the purposes of this research.

Secondly, the grammar was developed to parse all 40 sentences used as a specification. The results appear in appendix B in subsection B.1.2. The final coverage size of the grammar, when allowing at most one focused element in the theme and in the rheme, and with the lexicon reduced to one function per type, is 252 answer sentences. If focus were disregarded, the grammar would cover 45 answer sentence types, along with the 45 corresponding question sentence types. These 45 sentence types increase to almost 6000 when a function for each category in the lexicon is added to the grammar.

In addition, the effectiveness of the grammar is measured by its ease of use in the question-answering component, i.e., how well does the grammar structures facilitate the analysis of the incoming questions, and how are these structures reusable in the production of the answers returned?

The grammar defines 10 question functions, which cover the three kinds of theme-rheme distinctions discussed in section 5.2.2. They are listed in table 5.2 with the corresponding answer functions and their functions types. In section 5.3 we showed the structural correspondence between the questions in the grammar and the answers.

Analysing the question requires an understanding of the elements present in the theme in order to construct a query to determine the content of the rheme, while producing an answer requires using the question's corresponding answer function to produce the correct abstract syntax tree by supplying the answer with its theme-child and the newly determined rheme-child. The latter step is quite simple: it consists of looking up the question function's corresponding answer function, and presenting as its arguments the theme, copied from the question, and the rheme, determined from a query. It is the former step, that of analysing the question in order to compute the content of the rheme, that presents more of a challenge.

From the construction of Prolog-style queries, as discussed in section 5.4 and shown in table 5.3, it is clear that the traversal of the theme-structures is successful. Each Prolog query presents the verb of the question as the name of its predicate, while the various subjects, objects and adverbs are presented as its arguments. This reflects on the application grammar's design, and its incorporation of all the necessary elements of the question into its theme structure.

Indicating focus in the answer structure is similarly successful. The application

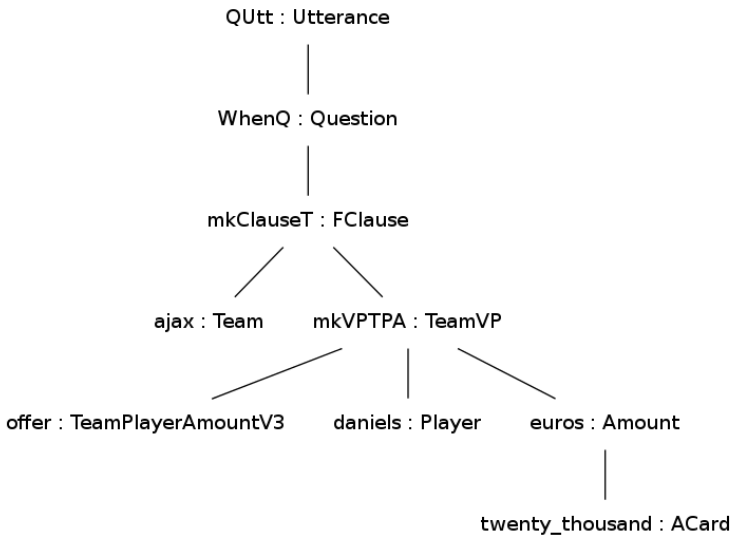

Figure 6.1: (Ajax) (offered Daniels twenty thousand euros)

grammar provides specific categories capable of carrying focus, as well as a function for each that ensures that such elements are marked up correctly. Once an element is determined as carrying focus, updating the answer tree structure simply requires inserting such focus functions above the appropriate elements. Figure 6.1 shows an answer tree where no elements are focused, while figure 6.2 shows the effect of applying focus to the element "Daniels" in the theme.

It is clear that the application grammar covers the required scope, and admits successful interaction with it for analysing questions and producing answers.

6.3 RGL extension

The purposes of extending the English RGL are, firstly, the addition of functions for building tree structures that encode information structure, and secondly, the addition of operations for linearising tree structures so that the underlying information structure is communicated in the resulting string.

In section 6.2 we showed that the application grammar met the scope requirements of the test set of sentences. This necessarily means that the RGL must also meet the scope requirements of the test set. In fact, coverage of the test sentence cases is the first aspect on which the RGL extension is to be evaluated. As discussed in section 5.2, the extension allows for three different ways in which to construct an answer to a wh-question from a theme and a rheme.

Without the extension, the RGL is able only to parse the (unmarked) answers in

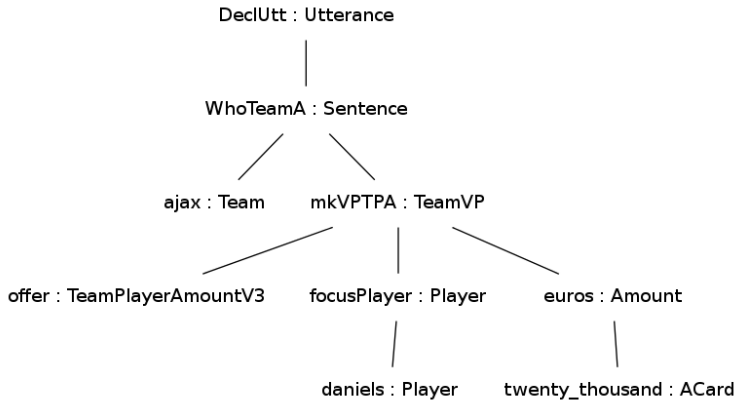

Figure 6.2: (Ajax) (offered DANIELS twenty thousand euros)

the test set as being instances of predication. That is, in each case, the two-way divide of the answer coincides with the information structure associated with who-answers, or answers to questions typically starting with "who". However, with the extension, the unmarked answers of the test set are parsed in multiple ways, via the functions added to the RGL, as discussed in section 5.2.2. The different parse trees represent different information structures, and so we may conclude that the RGL extension achieves its first goal of encoding information structure directly in tree structures.

In addition to these functions for achieving multiple parses of unmarked text, the extended RGL also enables the communication of the information structure in the linearisation of the tree, via operations for linearising syntactic categories in the RGL with user specified tags. We give an overview of how this feature was used to enable the application grammar to produce appropriately marked-up linearisations for its answer functions.

The operations for marking syntactic categories in the RGL were used to communicate both tiers of information structure. The overloaded `mark` operation was used to mark categories that represent themes and rhemes in the application grammar, as well as categories capable of carrying focus. Table 6.1 shows an example of each of the three ways of dividing an answer into theme and rheme with the use of the `mark` operation.

To mark categories capable of carrying focus, functions in the application grammar used the `mark` operation as well. This was done in a similar way as the marking of theme and rheme. A typical example is the function for marking a `Team` element as focused, presented in figure 6.3.

The use of the overloaded `mark` operation of the RGL extension shows how the second goal of the extension is achieved. With both goals achieved, we may conclude that the RGL extension successfully encodes and communicates the information structure required to produce modified synthetic speech. We

```
-- abstract syntax function
focusTeam : Team -> Team ;

-- concrete syntax linearisation function
focusTeam t = mark "foc" t ;

-- typical result
<foc> Barcelona </foc>
```

Figure 6.3: A function for focusing `Team` elements

Theme-rheme division	Example
Predication	`WhoTeamA s vp = mkS pastTense (mkCl (mark "rheme" s) (mark "theme" vp))`
Adverbial phrases	`WhenA cl t = mkS pastTense (AddAdjunct (mark "theme" cl) (mark "rheme" t))`
Slash categories	`WhomTeamA cl answer = mkS pastTense (ComplClSlash (mark "theme" "tail" cl) (mark "rheme" answer))`

Table 6.1: Marking theme and rheme

now turn to the issue of determining whether this modification is, in fact, and improvement.

6.4 Perceptual experiment

6.4.1 Aim and hypothesis

In section 5.5 we provided a definition of "prosodically-improved speech synthesis" as well as a slightly adapted definition of intelligibility that better describes the aim of this research. We also explained that this work focuses on this notion of intelligibility, which extends beyond propositional content to include information structural content. The aim of the perceptual experiment described in this section is therefore to show that the SSML output produced by the system and discussed in section 5.5 results in speech synthesis with improved intelligibility.

We therefore adopt the following hypothesis:

> Some form of markup of the kind discussed so far will result in speech synthesis with improved intelligibility of information structural content.

6.4.2 Experiment setup

For the perceptual experiment, we elected a simplified SSML scheme, in order to avoid the risks described in section 5.5.3. However, the scheme is expressive enough to take into account all the necessary aspects of information structure: theme and rheme boundaries are indicated by phrase breaks of 3 milliseconds, rheme phrases are spoken at a slower rate (90% of the original rate) and pitch accents are produced by changing the f_0 contour to rise with 70% initially and fall back to the original pitch halfway through the word. Although significantly simplified from the tunes described by Steedman (1991), this scheme nevertheless captures the orthogonal nature of the information and its effects on prosody.

Furthermore, we employed the open-source TTS system, MARY, to process and apply the SSML markup to its synthesis process. The essence of the perceptual test was to determine whether participants could better identify correctly the information structural content of speech produced by MARY from SSML input, than they could identify the same in speech produced by MARY with only bare text as input. In order to understand to what degree the various tiers of information structure contribute to the intelligibility of the modified synthetic speech, the markup scheme was applied in three different ways: one set of utterances were produced from input where only phrase breaks were included, another set of utterances were produced from input where only pitch accents were included, and yet another set of utterances were produced where both phrase breaks and pitch accents were included. A set of utterances were also produced where no prosodic modification was attempted at all. The entire set consisted of fourteen utterances produced in the four ways mentioned here, totalling 56 utterances of synthetic speech. Some of the utterances contained the same words, but were intended to have different prosodic realisations.

Three participants for the experiment were chosen from among mother-tongue speakers of South African English with some experience in speech technology. The latter requirement was decided on in order to ensure some familiarity with the inherent limitations of speech synthesis in general. None of the participants, however, are directly involved in speech synthesis research.

The experiment consisted of presenting the participant with the synthetic speech of an answer to a wh-question. The possible wh-questions it could be an answer to were also presented, along with an option labelled "Cannot tell". The participant was asked to choose from among these options, based on the synthetically spoken answer. For example, a speech snippet may consist of the words "Chelsea beat Barcelona yesterday". The possible questions this could be an answer to are:

1. Who beat Barcelona yesterday?
2. Whom did Chelsea beat yesterday?
3. When did Chelsea beat Barcelona?

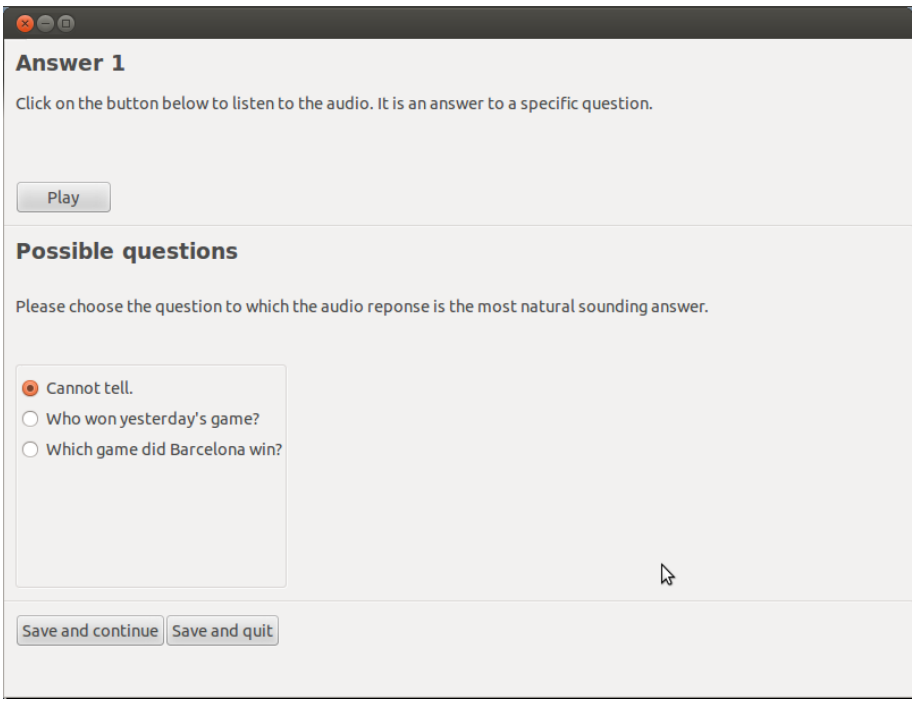

Figure 6.4: Screen shot of perceptual experiment

The 56 utterances were presented to the participants randomly, and the order of the options were also shuffled for each utterance. The participants were allowed to listen to the utterance as many times as they chose. Figure 6.4 shows a screen shot of the experiment.

6.4.3 Results

The results for the options chosen by each participant were categorised as either "correct", "wrong" or "cannot tell". We assume that an utterance's information structure is "intelligible" if participants tend to interpret it "correctly". We discuss the results as summarised in table 6.2 on page 87.

In 5 of the 14 sentences, the markup made only a small improvement to the "correctness" of the options chosen, since participants were "wrong" in most cases, whether the speech was generated with markup or without. These sentences include two "when"-questions, a "which"-question and two "whom"-questions.

Two sentences were almost always "correctly" interpreted by the participants. Perhaps it is significant that these two sentences were both "who"-sentences, which could be an indication that the MARY voice's existing prosodic behaviour favours such an interpretation to some degree. Even here, however, a small improvement was achieved by the marked-up speech.

		Correctly interpreted	
Sentence group	Group size	No markup	With Markup
Mostly wrong	5	7%	18%
Mostly correct	2	83%	94%
Improved intelligibility	6	22%	65%
Reduced intelligibility	1	67%	22%

Table 6.2: Summary of results

Six sentences indicate that some form of markup resulted in significantly more "correct" interpretations, averaging a correct interpretation 65% of the time, while the unmarked speech averaged a correct interpretation only 22% of the time. It should be mentioned that the three different kinds of markup faired equally well in yielding "correct" interpretations. The sentences include examples of all wh-elements, except "when".

Finally, for one sentence the markup speech was entirely misleading, yielding a correct interpretation only 22% of the time, while its corresponding unmarked utterance was correctly interpreted 67% of the time. The question for was "Which game did Barcelona win?", and the answer was "Barcelona won yesterday's game". However, the unmodified speech of the answer already contains a significant pitch accent on the word "game". On top of this, the rheme-focused element, "yesterday", receives a pitch accent in the marked-up speech, which exaggerates the focus on the rheme elements. The same risk was identified by Li et al. (2012).

This shows that, due to a degree of unpredictability, SSML-modified speech synthesis may, in certain cases, lead to synthesis that is so misleadingly unnatural that the default, unmodified pitch contour is preferable. Figure 6.5 shows the unmodified speech and figure 6.6 the speech with both a break and a modified pitch contour. In both cases the vertical dotted red line indicates the start of the rheme's first word, "yesterday". The top bar shows the amplitude of the utterance, the bottom shows its spectrogram, and the blue line in the bottom bar shows the f_0 contour.

6.4.4 Conclusion

The results tend to confirm the hypothesis, since in almost half of the cases, a significant improvement in the intelligibility of the intended information structure was observed, while more than half of the cases indicate good intelligibility. Only one case resulted in reduced intelligibility, but this could be a direct result of the unpredictability of using SSML to modify the prosodic behaviour of a synthetic voice.

Given the simplicity of the markup scheme followed in this research, the results indicate strongly that a finer-grained, more intelligent application of the markup produced by the question-answering system would result in significantly

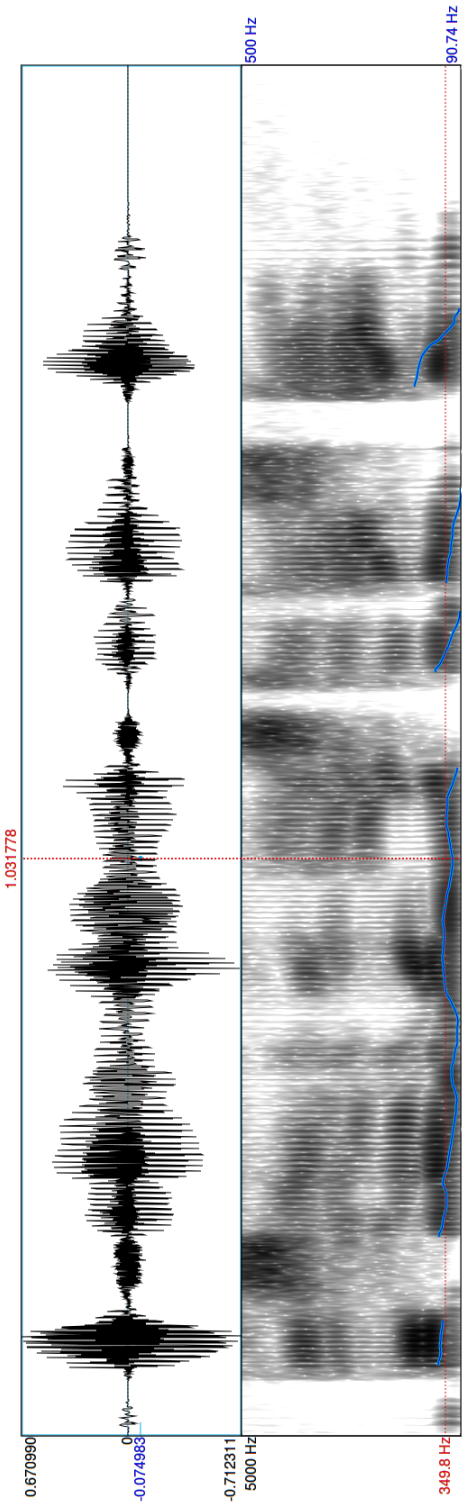

Figure 6.5: Amplitude, spectrogram and f_0 contour for unmodified speech

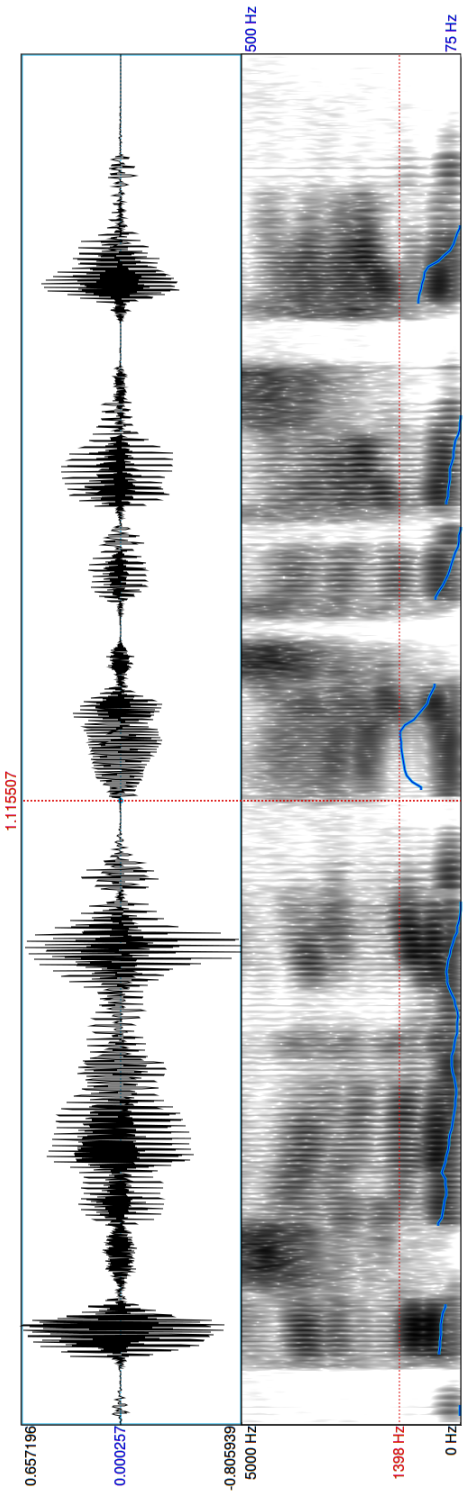

Figure 6.6: Amplitude, spectrogram and f_0 contour for modified speech

improved prosody in speech synthesis. Suggestions for such a system are presented in the next chapter.

6.5 Answering the research questions

We now revisit the research questions stated in chapter 1 in order to determine whether they have been answered.

With regards to subquestions 1 and 2, section 6.3 shows that information structure is successfully encoded in the tree structures built from the extended RGL, and that the resulting linearisations communicate clearly what the underlying information structure of the trees are.

Subquestion 3 relates to the development of an application grammar that provides functions for representing wh-question-answer pairs in a structurally related way. We saw in section 6.2 that such an application grammar was developed and used in a question-answering system.

The question-answering discussed in section 6.1 demonstrates the ability of the question-answering system to receive wh-questions as input and to produce suitably marked answers as output, using the application grammar developed for this purpose, as referred to in subquestion 4.

The final subquestion is addressed in section 6.1, where the output of the question-answering system is evaluated directly, as well as in section 6.4, where the result of processing the output of the question-answering system produces modified speech synthesis. The perceptual experiment discussed in section 6.4 indicates that an improvement in the prosody of speech synthesis was achieved.

In the next chapter we consider the main research question in light of the spoken question-answering system developed and discussed in the preceding chapters.

Chapter 7

Conclusion

We now return to the main research question. The question, "How can the novel combination of GF and insights from CCG be used to develop a grammar that assists in building a prosodically-improved CTS question-answering system in English?" must be addressed with reference to three main aspects:

1. The novel combination of GF with insights from CCG;
2. The development of a grammar;
3. The goal of the research is prosodically-improved concept-to-speech synthesis in an English question-answering system.

GF has been used recently to extend AceWiki (Kaljurand and Kuhn, 2013), which is an online semantic wiki. This wiki allows for adding wh-questions about the content of the wiki that are automatically interpreted and answered. GF was used to extend the wiki, which uses Attempto Controlled English (ACE), to nine other languages. Each statement of fact or question in each of the other languages was translated with GF to ACE, interpreted and answered in ACE, and translated back into the original language.

This research is significantly different from the approach followed by Kaljurand and Kuhn (2013) and Fuchs et al. (2013) in that it does not use ACE, but instead uses GF trees directly to interpret the questions asked. It would be quite possible to extend this work to other languages supported by GF in a way similar to AceWiki. However, this would require the RGLs of the other languages to also support the information structural categories and functions present in the extended English RGL, as discussed in chapter 5. It is this point that illustrates the contribution of this research, in answer to the main research question:

1. Following Steedman (2001), we assumed an isomorphic relation between information structure, syntactic structure and prosodic structure. We

extended the GF English RGL with functions that, in a constrained way, allow for creating similar syntactic structures as would be possible in CCG.

2. We then exploited these structures in the creation of a grammar that was used in a question-answering system for producing enriched natural language.

3. This output was then shown to produce prosodically-improved speech synthesis when transformed into appropriate SSML input to the MARY speech synthesis engine.

There are two clear directions for future work. The first concerns the grammar development aspects of the research. This would involve the further extension of the RGL to cover more verb types, as mentioned in section 4.2, and would also include research into the information structural interpretation of utterances containing such verbs. Furthermore, the scope of the questions and answers themselves may be extended beyond wh-question-answer pairs.

The second direction concerns the generation of prosody in speech synthesis from the output of a question-answering system as described in section 5.4. The approach taken here uses off-the-shelf technology. However, SSML does not contain tags for marking information structure itself. Instead, the information structure markup from the question-answering system is interpreted via a theory of prosody Steedman (1991), and the abstract description of this theory is converted into SSML commands, via a empirical process, to produce reasonable output. However, this leads to the limitations described in section 5.5.3.

One possible approach to take might be the extension of SSML with tags for marking information structure, similar to the "emph" (for emphasis) tag. The "emph" tag allows the synthesis engine to determine the prosodic realisation of the emphasised content in an intelligent way, taking into account the contribution of its natural language processing module. Similarly, information structure tags - perhaps "theme", "rheme" and "focus" - would allow a more intelligent prosodic realisation of the content. It would solve, for instance, the problem of how to correctly place the pitch accent on a word by simply identifying and altering its stressed syllable.

A second possibility is the use of the the markup from the question-answering system while training a voice. The information structure markup would be analysed during the natural language processing step and would form part of the features of the data used for training. The voice's prosodic behaviour associated with the information structure in the sentences would then be modelled implicitly. Since there is no external alteration made on the synthesised speech that is produced from such a system, this approach might result not only in improved intelligibility of the information structural content, but also in an improvement of the naturalness of the utterance.

Finally, an important possible extension of this work must be pointed out. The abstract syntax that uses information structure as its basis for constructing question and answer utterances is inherently language independent. In English,

the information structure is communicated via specific prosodic patterns in spoken language. Other languages, however, may employ other means of realising information structure. The work described here provides a framework within which to explore these strategies for communicating information structure in a multilingual environment. This would enable building multilingual question-answer systems with prosodically-improved speech synthesis.

This work has shown that information structure plays a noticeable role in producing prosodically-improved speech synthesis in the context of current speech synthesis techniques. Furthermore, this research has shown that the notion of information structure as the basis for describing the structure of question-answer pairs in an abstract syntax grammar could be coupled with the parsing, linearising and tree-manipulating functionality provided by GF to improve the prosody of concept-to-speech synthesis for spoken question-answering in English.

It is clear, however, that several questions remain to be answered, in the direction of approaches to generating prosodically-improved speech synthesis, and also of further exploring the relationship between information structure and syntax via monolingual as well as multilingual grammars.

Appendix A

Bibliography

Ahrendt, Wolfgang; Baar, Thomas; Beckert, Bernhard; Bubel, Richard; Giese, Martin; Hähnle, Reiner; Menzel, Wolfram; Mostowski, Wojciech; Roth, Andreas; Schlager, Steffen, and Schmitt, Peter H. The KeY Tool. *Software and System Modeling*, 4:32–54, 2005.

Alter, Kai; Pirker, Hannes, and Finkler, Wolfgang. Concept to Speech Generation Systems. In *Proceedings of ACL97, Concept to speech generation systems workshop*, 1997.

Angelov, Krasimir. *GF Runtime System*. PhD thesis, University of Gothenburg, 2009.

Angelov, Krasimir and Enache, Ramona. Typeful Ontologies with Direct Multilingual Verbalization. *Controlled Natural Language*, pages 1–20, 2012.

Angelov, Krasimir; Bringert, Björn, and Ranta, Aarne. PGF: A Portable Runtime Format for Type-theoretical Grammars. *Journal of Logic, Language and Information*, 19(2):201–228, December 2009.

Beckman, Mary E. and Hirschberg, Julia. The ToBI Annotation Conventions, 1994. URL `http://www.cs.columbia.edu/~agus/tobi/tobi_convent.pdf`. Accessed 2013-11-28.

Bierner, Gann. TraumaTalk: Content-to-Speech Generation for Decision Support at Point of Care. In *Proceedings of the AMIA Symposium*, pages 698–702. American Medical Informatics Association, 1998.

Blackburn, Patrick and Bos, Johan. *Representation and Inference for Natural Language*. 1999.

Bringert, Björn. Speech Recognition Grammar Compilation in Grammatical Framework. In *Proceedings of the Workshop on Grammar-Based Approaches to Spoken Language Processing*, pages 1–8. Association for Computational Linguistics., 2007.

Bringert, Björn. Speech Translation with Grammatical Framework. In *Proceedings of the workshop on Speech Processing for Safety Critical Translation and Pervasive Applications*, pages 5–8, 2008. ISBN 9781905593521.

Bringert, Björn; Ljunglof, Peter; Cooper, Robin, and Ranta, Aarne. Multimodal Dialogue System Grammars. In *Proceedings of DIALOR05, Ninth Workshop on the Semantics and Pragmatics of Dialogue*, pages 53–60, 2005.

Büring, Daniel. Semantics, Intonation and Information Structure. In *The Oxford Handbook of Linguistic Interfaces*, pages 1–36. OUP, 2007.

Burnett, Daniel C.; Walker, Mark R., and Hunt, Andrew. Speech Synthesis Markup Language (SSML) Version 1.0, 2004. URL `http://www.w3.org/TR/speech-synthesis/`. Accessed 2013-10-18.

Burnett, Daniel C.; Shuang, Zhi Wei; Baggia, Paolo; Bagshaw, Paul; Bodell, Michael; Huang, De Zhi; Xiaoyan, Lou; McGlashan, Scott; Tao, Jianhua; Jun, Yan; Fang, Hu; Kang, Yongguo; Meng, Helen; Xia, Wang; Hairong, Xia, and Wu, Zhiyong. Speech Synthesis Markup Language (SSML) Version 1.1, 2010. URL `http://www.w3.org/TR/speech-synthesis11/`. Accessed 2013-10-22.

Calhoun, Sasha; Nissim, Malvina; Steedman, Mark, and Brenier, Jason. A Framework for Annotating Information Structure in Discourse. In *Pie in the Sky: Proceedings of the ACL workshop*, pages 45–52, 2005.

Caprotti, Olga. WebALT! Deliver Mathematics Everywhere. In *Society for Information Technology & Teacher Education International Conference*, volume 2006, pages 2164–2168, March 2006.

Dannélls, Dana; Damova, Mariana; Enache, Ramona, and Chechev, Milen. Multilingual online generation from semantic web ontologies. *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion*, page 239, 2012.

Davis, Brian; Enache, Ramona; Van Grondelle, Jeroen, and Pretorius, Laurette. Multilingual Verbalisation of Modular Ontologies using GF and lemon. *Controlled Natural Language*, pages 167–184, 2012.

Enache, Ramona. *Reasoning and Language Generation in the SUMO Ontology.* PhD thesis, Chalmers University of Technology and University of Gothenburg, 2010.

Enache, Ramona; Popov, Borislav, and Ranta, Aarne. Patent MT and Retrieval Prototype Beta. Technical report, MOLTO Project, 2012.

Feng, Junlan; Ramabhadran, Bhuvana; Hansen, John H L, and Williams, Jason D. Trends in Speech and Language Processing. *Signal Processing Magazine, IEEE 29, no. 1*, pages 2011–2013, 2012.

Fuchs, Norbert E; Kaljurand, Kaarel, and Kuhn, Tobias. Multilingual semantic wiki. Technical report, MOLTO Project, 2013.

Gundel, Jeanette K and Fretheim, Thorstein. Topic and Focus. In Horn, Laurence R and Ward, Gregory, editors, *The Handbook of Pragmatics*, pages 175–196. 2006.

Haji-Abdolhosseini, Mohammad. A Constraint-Based Approach to Information Structure and Prosody Correspondence. In *Proceedings of the HPSG-2003 Conference*, pages 143–162, 2003.

Hajičová, Eva; Sgall, Petr, and Skoumalová, Hana. An Automatic Procedure for Topic-Focus Identification. *Computational Linguistics*, 21(1):81–94, 1995.

Hausser, Roland R. *Surface Compositional Grammar*. Wilhelm Fink Verlag, München, 1984.

Hirschberg, Julia. Communication and prosody: Functional aspects of prosody. *Speech Communication*, 36(1-2):31–43, January 2002.

Hirschberg, Julia. Speech Synthesis: Prosody. In *Encyclopedia of Language & Linguistics*, number 7, pages 49–55. 2006a.

Hirschberg, Julia. Pragmatics and Intonation. In *The Handbook of Pragmatics*, pages 515–537. 2006b.

Hirschberg, Julia and Pierrehumbert, Janet. The intonational structuring of discourse. In *Proceedings of the 24th annual meeting on Association for Computational Linguistics*, pages 136–144. Association for Computational Linguistics, 1986.

Hitzeman, Janet; Black, Alan W; Taylor, Paul; Mellish, Chris, and Oberlander, Jon. On the use of automatically generated discourse-level information in a concept-to-speech synthesis system. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1–4, 1998.

Hitzeman, Janet; Black, Alan W; Mellish, Chris; Oberlander, Jon; Poesio, Massimo, and Taylor, Paul. An Annotation Scheme for Concept-to-Speech Synthesis, 1999.

Hiyakumoto, Laurie; Prevost, Scott, and Cassell, Justine. Semantic and Discourse Information for Text-to-Speech Intonation. *Concept to Speech Generation Systems*, pages 47–56, 1997.

Jurafsky, Daniel and Martin, James H. *Speech and Language Processing*. 2009.

Kaljurand, Kaarel and Kuhn, Tobias. A Multilingual Semantic Wiki Based on Attempto Controlled English and Grammatical Framework. In *The Semantic Web: Semantics and Big Data*, pages 427–441. Springer Berlin, Heidelberg, 2013.

Krifka, Manfred. Basic Notions of Information Structure. *Acta Linguistica Hungarica*, 55(3):243–276, 2008.

Kruijff-Korbayová, Ivana and Steedman, Mark. Discourse and Information Structure. *Journal of Logic, Language, and Information*, 12(3):249–259, 2003.

Kügler, Frank; Smolibocki, Bernadett, and Stede, Manfred. Evaluation of Information Structure in Speech Synthesis: The Case of Product Recommender Systems. In *Speech Communication; 10. ITG Symposium; Proceedings of. VDE*, pages 1–4, 2012.

Larsson, Staffan and Ljunglof, Peter. A Grammar Formalism for Specifying ISU-Based Dialogue Systems. *Advances in Natural Language Processing*, pages 303–314, 2008.

Li, Chunrong; Wu, Zhiyong; Meng, Fanbo; Meng, Helen, and Cai, Lianhong. Detection and emphatic realization of contrastive word pairs for expressive text-to-speech synthesis. *2012 8th International Symposium on Chinese Spoken Language Processing*, pages 93–97, December 2012.

Liberman, Isabelle Y.; Shankweiler, Donald, and Liberman, Alvin M. The alphabetic principle and learning to read. In *Meeting of the International Academy for Research on Learning Disabilities*, Evanston, IL, US, 1989.

Ljunglöf, Peter; Amores, Gabriel; Burden, Hå kan; Manchón, Pilar; Pérez, Guillermo, and Ranta, Aarne. Enhanced Multimodal Grammar Library, Talk and Look: Tools for Ambient Linguistic Knowledge. Technical report, 2006.

Mattingly, I. Reading, the linguistic process, and linguistic awareness. In Kavanagh, J and Mattingly, I, editors, *Language by ear and by eye: The relationships between speech and reading*, pages 133–147. MIT Press, 1972.

Mitankin, Petar; Ilchev, Atanas; Popov, Borislav; Popova, Reneta, and Petkova, Gergana. Knowledge Representation Infrastructure. Technical report, MOLTO Project, 2010.

Pan, Shimei; McKeown, Kathleen, and Hirschberg, Julia. Exploring features from natural language generation for prosody modeling. *Computer Speech & Language*, 16(3-4):457–490, July 2002.

Pérez, G; Amores, G; Manchón, P, and Maline, DG. Generating Multilingual Grammars from OWL Ontologies. *Research in Computing Science*, 18:3–14, 2006.

Prevost, Scott. An Information Structural Approach to Spoken Language Generation. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 294–301. Association for Computational Linguistics, 1995a.

Prevost, Scott. *A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation.* PhD thesis, University of Pennsylvania, 1995b.

Prevost, Scott and Steedman, Mark. Using Context to Specify Intonation in Speech Synthesis. Technical report, University of Pennsylvania, 1993.

Prevost, Scott and Steedman, Mark. Specifying intonation from context for speech synthesis. *Speech Communication*, 15(1-2):139–153, October 1994a.

Prevost, Scott and Steedman, Mark. Information Based Intonation Synthesis. In *Proceedings of the workshop on Human Language Technology*, pages 193–198. Association for Computational Linguistics, 1994b.

Ranta, Aarne. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189, 2003.

Ranta, Aarne. The GF Resource Grammar Library. *Linguistic Issues in Language Technology*, 2(2), 2009.

Ranta, Aarne. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI, Stanford, California, 2011.

Ranta, Aarne. GF - Grammatical Framework, 2012. URL `http://www.grammaticalframework.org/`. Accessed 2013-01-30.

Ranta, Aarne. GF Resource Grammar Library, 2013. URL `http://www.grammaticalframework.org/download/release-3.5.html`. Accessed 2013-11-22.

Ranta, Aarne and Cooper, Robin. Dialogue Systems as Proof Editors. *Journal of Logic, Language and Information*, 13(2):225–240, 2004.

Ranta, Aarne; Enache, Ramona; Chechev, Milen, and Damova, Mariana. Multilingual Online Translation. Technical report, MOLTO Project, 2012.

Rooth, Mats. A theory of focus interpretation. *Natural Language Semantics*, 1 (1):75–116, February 1992.

Schröder, M. and Trouvain, J. The German Text-to-Speech Synthesis System MARY: A Tool for Research, Development and Teaching. *International Journal of Speech Technology*, 6:365–377, 2003.

Spyns, P.; Deprez, F.; Van Tichelen, L., and Van Coile, B. Message-to-Speech: high quality speech generation for messaging and dialogue systems. In *Proceedings of ACL97, Concept to speech generation systems workshop*, number 1, pages 11–16, 1997.

Steedman, Mark. Structure and Intonation. *Language*, 67(2):260–296, 1991.

Steedman, Mark. Representing discourse information for spoken dialogue generation. In *Proceedings of the International Symposium on Spoken Dialogue, ICSLP*, 1996.

Steedman, Mark. Information Structure and the Syntax-Phonology Interface. *Linguistic Inquiry*, 31(4):649–689, 2000.

Steedman, Mark. *The Syntactic Process*. MIT Press, London, 2001.

Steedman, Mark. *The Surface-Compositional Semantics of English Intonation*. Manuscript, University of Edinburgh, draft 5, 2012.

Taylor, PA. Concept-to-speech synthesis by phonological structure matching. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 358(1769):1403–1417, April 2000.

Taylor, Paul. *Text-to-Speech Synthesis*. Cambridge University Press, Cambridge, 2009.

Theune, Mariët. Contrastive accent in a data-to-speech system. In *ACL '98 Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 519–521, 1998.

Theune, Mariët. Contrast in concept-to-speech generation. *Computer Speech & Language*, 16(3-4):491–531, July 2002.

Vallduvi, Enric. *The Informational Component*. PhD thesis, University of Pennsylvania, 1993.

Veilleux, Nanette M. Probabilistic Model of Acoustic/Prosody/ Concept Relationships for Speech Synthesis. In *Proceedings of ACL97, Concept to speech generation systems workshop*, pages 1–10, 1997.

Walker, Marilyn and Rambow, Owen. Spoken language generation. *Computer Speech & Language*, 16(3-4):273–281, July 2002.

Ward, Gregory and Birner, Betty. Information Structure and Non-canonical Syntax. In Horn, Laurence R and Ward, Gregory, editors, *The Handbook of Pragmatics*, chapter 7, pages 153–174. 2006.

Xu, Yi. Speech prosody: a methodological review. *Journal of Speech Sciences*, 1(1):85–115, 2012.

Yagi, Y.; Hirose, K.; Takada, S., and Minematsu, N. Improved concept-to-speech generation in a dialogue system on road guidance. *2005 International Conference on Cyberworlds (CW'05)*, pages 8 pp.–436, 2005.

Yule, George. New, Current and Displaced Entity Reference. *Lingua*, 55(1): 41–52, 1981.

Appendix B

Data

B.1 Test set of sentences

B.1.1 Sentences

1. whom did Barcelona sign today
2. when did Boateng score the third goal
3. when did Ajax pay Matheus twenty thousand pounds
4. how much did Chelsea offer Boateng
5. who lost the game against Ajax
6. whom did Barcelona pay twenty thousand pounds
7. whom did Ajax buy today
8. when did Muller score the first goal
9. whom did Chelsea lose to
10. when did Chelsea sign Muller
11. how many goals did Matheus score yesterday
12. how much did Chelsea offer Boateng yesterday
13. when did Matheus score the first goal
14. when did Matheus score the second goal
15. who scored the first goal
16. who won the game against Barcelona

17. who played
18. which game did Ajax lose
19. whom did Ajax sign
20. which goal did Muller score
21. whom did Ajax lose to yesterday
22. whom did Ajax beat today
23. which game did Arsenal lose
24. when did Chelsea pay Boateng twenty thousand pounds
25. how much did Ajax offer Boateng
26. whom did Ajax buy
27. whom did Arsenal buy
28. who lost
29. how many goals did Muller score
30. whom did Chelsea offer thirty thousand pounds
31. which game did Chelsea lose yesterday
32. how much did Chelsea offer Muller today
33. whom did Arsenal beat
34. how much did Barcelona pay Boateng
35. when did Matheus score the first goal
36. which goal did Matheus score today
37. which game did Arsenal win
38. who scored the second goal
39. whom did Arsenal offer twenty thousand euros
40. which goal did Muller score

B.1.2 Parses

1. `QUtt (WhomPlayerQ (mkClSlashTPlayer barcelona (addTimeVPSTPlayer (mkVPSTPlayer sign) today)))`
2. `QUtt (WhenQ (mkClauseP boateng (mkVPPlayerPoint score2 (ord_goal third))))`

3. QUtt (WhenQ (mkClauseT ajax (mkVPTPA pay matheus (pounds twenty_thousand))))
4. QUtt (HowMuchQ (mkClSlash3 chelsea (mkVPS2P offer boateng)))
5. QUtt (WhoTeamQ (mkVPTeamContest lose2 (game_someone ajax)))
6. QUtt (WhomPlayerQ2 (mkClSlash2 barcelona (mkVPS2A pay (pounds twenty_thousand))))
7. QUtt (WhomPlayerQ (mkClSlashTPlayer ajax (addTimeVPSTPlayer (mkVPSTPlayer buy) today)))
8. QUtt (WhenQ (mkClauseP muller (mkVPPlayerPoint score2 (ord_goal first))))
9. QUtt (WhomTeamQ (mkClSlashTT chelsea (mkVPSTT lose_to)))
10. QUtt (WhenQ (mkClauseT chelsea (mkVPTeamPlayer sign muller)))
11. QUtt (HowManyQ goal (mkClSlashPP matheus (addTimeVPSPP (mkVPSPP score2) yesterday)))
12. QUtt (HowMuchQ (mkClSlash3 chelsea (addTimeVPS2A (mkVPS2P offer boateng) yesterday)))
13. QUtt (WhenQ (mkClauseP matheus (mkVPPlayerPoint score2 (ord_goal first))))
14. QUtt (WhenQ (mkClauseP matheus (mkVPPlayerPoint score2 (ord_goal second))))
15. QUtt (WhoPlayerQ (mkVPPlayerPoint score2 (ord_goal first)))
16. QUtt (WhoTeamQ (mkVPTeamContest win2 (game_someone barcelona)))
17. QUtt (WhoTeamQ (mkVPTeam play1))
18. QUtt (WhichContestQ game (mkClSlashTContest ajax (mkVPSTContest lose2)))
19. QUtt (WhomPlayerQ (mkClSlashTPlayer ajax (mkVPSTPlayer sign)))
20. QUtt (WhichPointQ goal (mkClSlashPP muller (mkVPSPP score2)))
21. QUtt (WhomTeamQ (mkClSlashTT ajax (addTimeVPSTT (mkVPSTT lose_to) yesterday)))
22. QUtt (WhomTeamQ (mkClSlashTT ajax (addTimeVPSTT (mkVPSTT beat) today)))
23. QUtt (WhichContestQ game (mkClSlashTContest arsenal (mkVPSTContest lose2)))
24. QUtt (WhenQ (mkClauseT chelsea (mkVPTPA pay boateng (pounds twenty_thousand))))

25. QUtt (HowMuchQ (mkClSlash3 ajax (mkVPS2P offer boateng)))
26. QUtt (WhomPlayerQ (mkClSlashTPlayer ajax (mkVPSTPlayer buy)))
27. QUtt (WhomPlayerQ (mkClSlashTPlayer ajax (mkVPSTPlayer buy)))
28. QUtt (WhoTeamQ (mkVPTeam play1))
29. QUtt (HowManyQ goal (mkClSlashPP muller (mkVPSPP score2)))
30. QUtt (WhomPlayerQ2 (mkClSlash2 chelsea (mkVPS2A offer (pounds thirty_thousand))))
31. QUtt (WhichContestQ game (mkClSlashTContest chelsea (addTimeVPSTContest (mkVPSTContest lose2) yesterday)))
32. QUtt (HowMuchQ (mkClSlash3 chelsea (addTimeVPS2A (mkVPS2P offer muller) today)))
33. QUtt (WhomTeamQ (mkClSlashTT arsenal (mkVPSTT beat)))
34. QUtt (HowMuchQ (mkClSlash3 barcelona (mkVPS2P pay boateng)))
35. QUtt (WhenQ (mkClauseP matheus (mkVPPlayerPoint score2 (ord_goal first))))
36. QUtt (WhichPointQ goal (mkClSlashPP matheus (addTimeVPSPP (mkVPSPP score2) today)))
37. QUtt (WhichContestQ game (mkClSlashTContest arsenal (mkVPSTContest win2)))
38. QUtt (WhoPlayerQ (mkVPPlayerPoint score2 (ord_goal second)))
39. QUtt (WhomPlayerQ2 (mkClSlash2 arsenal (mkVPS2A offer (euros twenty_thousand))))
40. QUtt (WhichPointQ goal (mkClSlashPP muller (mkVPSPP score2)))

B.2 Questions

Table B.1 shows the 14 answer sentences with their corresponding possible questions that were used in the perceptual experiment.

Barcelona beat Chelsea	Whom did Barcelona beat? Who beat Chelsea?
Chelsea beat Ajax today	Whom did Chelsea beat today? Who beat Ajax today? When did Chelsea beat Ajax?
Watson scored three goals	Who scored three goals? How many goals did Watson score?
Fischer scored the second goal	Which goal did Fischer score? Who scored the second goal?
Wigan offered Fischer twenty thousand euros	How much did Wigan offer Fischer? Who offered Fischer twenty thousand euros? Whom did Wigan offer twenty thousand euros?
Wigan offered Daniels thirty thousand euros	Whom did Wigan offer thirty thousand euros? How much did Wigan offer Daniels? Who offered Daniels thirty thousand euros?
Wigan offered Giggs thirty thousand euros yesterday	When did Wigan offer Giggs thirty thousand euros? Whom did Wigan offer thirty thousand euros yesterday? Who offered Giggs thirty thousand euros yesterday? How much did Wigan offer Giggs yesterday?
Barcelona signed Fischer	Who signed Fischer? Who did Barcelona sign?
Barcelona won yesterday's game	Which game did Barcelona win? Who won yesterday's game?
Wigan lost the game against Chelsea	Which game did Wigan lose? Who lost the game against Chelsea?
Chelsea beat Ajax today	When did Wigan beat Ajax? Whom did Chelsea beat today? Who beat Ajax today?
Watson scored three goals	How many goals did Watson score? Who scored three goals?
Barcelona won yesterday's game	Who won yesterday's game? Which game did Barcelona win?
Fischer scored the second goal	Who scored the second goal? Which goal did Fischer score?

Table B.1: Experiment answers with possible questions

Appendix C

Code

C.1 English RGL extension

C.1.1 MarkupEng.gf

```
resource MarkupEng = open ExtraEng,
                          ParadigmsEng,
                          SentenceEng,
                          ResEng,
                          Prelude,
                          Predef in
{

  oper

mark = overload
{ -- NP
mark : Str -> lincat NP {s : NPCase => Str ; a : Agr} ->
              lincat NP {s : NPCase => Str ; a : Agr} =
\m,np -> lin NP
{ s = \\npcase => "<"+m+">" ++ np.s!npcase ++ "</"+m+">" ;
       a = np.a } ;

  -- N
mark : Str -> lincat N {s : Number => Case => Str ; g : Gender} ->
              lincat N {s : Number => Case => Str ; g : Gender} =
\m,n -> lin N
{ s = \\x,y => "<"+m+">" ++ n.s!x!y ++ "</"+m+">" ;
  g = n.g } ;

      -- A
```

```
mark : Str -> lincat A {s : AForm => Str} ->
              lincat A {s : AForm => Str} =
\m,a -> lin A
{ s = \\aform => "<"+m+">" ++ a.s!aform ++ "</"+m+">" } ;

 -- Num
mark : Str ->
lincat Num {s : Case => Str ; n : Number ; hasCard : Bool} ->
lincat Num {s : Case => Str ; n : Number ; hasCard : Bool} =
\m,x -> lin Num
{ s = \\c => "<"+m+">" ++ x.s!c ++ "</"+m+">" ;
  hasCard = x.hasCard ;
  n = x.n } ;

 -- Card
mark : Str -> lincat Card {s : Case => Str ; n : Number} ->
              lincat Card {s : Case => Str ; n : Number} =
\m,x -> lin Card
{ s = \\c => "<"+m+">" ++ x.s!c ++ "</"+m+">" ;
 n = x.n } ;

      -- AP
mark : Str -> {s : Agr => Str ; isPre : Bool} ->
              {s : Agr => Str ; isPre : Bool} =
\m,ap ->
{ s = \\agr => "<"+m+">" ++ ap.s!agr ++ "</"+m+">" ;
  isPre = ap.isPre ; } ;

      -- AdV, Adv, Predet
mark : Str -> Str -> Str =
\m,s -> "<"+m+">" ++ s ++ "</"+m+">" ;

      -- Prep, Subj
mark : Str -> { s : Str } -> { s : Str } =
\m,x ->
{ s = "<"+m+">" ++ x.s ++ "</"+m+">" } ;

      -- Conj
mark : Str -> {s1,s2 : Str ; n : Number} ->
              {s1,s2 : Str ; n : Number} =
\m,c ->
{ s1 = "<"+m+">" ++ c.s1 ++ "</"+m+">" ;
  s2 = "<"+m+">" ++ c.s2 ++ "</"+m+">" ;
  n = c.n ; } ;

      -- Det
mark : Str ->
{s : Str ; sp : NPCase => Str ; n : Number ; hasNum : Bool} ->
{s : Str ; sp : NPCase => Str ; n : Number ; hasNum : Bool} =
\m,det ->
```

```
{ s = "<"+m+">" ++ det.s ++ "</"+m+">";
  sp = \\npcase => det.sp ! npcase;
  n = det.n ;
  hasNum = det.hasNum ; } ;

      -- Comp
mark : Str -> {s : Agr => Str ;} -> {s : Agr => Str ;} =
\m,ap ->
{ s = \\agr => "<"+m+">" ++ ap.s!agr ++ "</"+m+">" ; } ;

      -- ClSlash
mark : Str -> ClSlash -> ClSlash =
\m,cls -> lin ClSlash
{ s = \\t,ant,c,o => "<"+m+">" ++ cls.s!t!ant!c!o ++ "</"+m+">" ;
  c2 = "<"+m+">" ++ cls.c2  ++ "</"+m+">" } ;
  --a = cls.a ;
  --gapInMiddle = cls.gapInMiddle } ;

  -- ClSlash
mark : Str -> Str -> ClSlash -> ClSlash =
\m,n,cls -> lin ClSlash
{ s = \\t,ant,c,o => "<"+m+">" ++ cls.s!t!ant!c!o ++ "</"+m+">" ;
  c2 = "<"+n+">" ++ cls.c2  ++ "</"+n+">" } ; --;
  --a = cls.a } --;
  --gapInMiddle = cls.gapInMiddle } ;

  -- Cl
mark : Str -> Cl -> Cl =
\m,cl -> lin Cl
{ s = \\t,ant,c,o => "<"+m+">" ++ cl.s!t!ant!c!o ++ "</"+m+">"  } ;

  -- ClSlashVP
mark : Str -> ClSlashVP -> ClSlashVP =
\m,sl -> lin ClSlashVP
{ s = markNP m sl.s ;
  s2 = "<"+m+">" ++ sl.s2 ++ "</"+m+">" ;
  a = sl.a } ;

  -- ClSlashVP (mark tail individually)
mark : Str -> Str -> ClSlashVP -> ClSlashVP =
\m,n,sl -> lin ClSlashVP
{ s = markNP m sl.s ;
  s2 = "<"+n+">" ++ sl.s2 ++ "</"+n+">" ;
  a = sl.a } ;

  -- VPSlash
mark : Str -> VPSlash -> VPSlash =
\m,vp -> lin VPSlash
{ s   = \\t,ant,cp,o,agr =>
let
```

```
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = "<"+m+">" ++ verb.aux ;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf } ;
  p   = vp.p  ++ "</"+m+">";   -- verb particle
  prp = vp.prp ;   -- present participle
  ptp = vp.ptp ;   -- past participle
  inf = vp.inf ;   -- the infinitive form
  ad  = vp.ad ;   -- sentence adverb
  s2  = \\agr => vp.s2!agr ; -- complement
  c2 = "<"+m+">" ++ vp.c2 ++ "</"+m+">" ;
  gapInMiddle = vp.gapInMiddle
     } ;

-- marks everything, assumes the VP is linearised contiguously
mark : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = "<"+m+">" ++ verb.aux ;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf } ;
  p   = vp.p ;   -- verb particle
  prp = vp.prp ;   -- present participle
  ptp = vp.ptp ;   -- past participle
  inf = vp.inf ;   -- the infinitive form
  ad  = vp.ad ;   -- sentence adverb
  s2  = \\agr => vp.s2!agr ++ "</"+m+">" -- complement
     } ;

} ;

-- recursive call of an overloaded oper gives some issues
markNP : Str -> NP -> NP =
\m,np -> lin NP
{ s = \\npcase => "<"+m+">" ++ np.s!npcase ++ "</"+m+">" ;
       a = np.a ; } ;

-- marks everything, using the infinitive form
markInfVP : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
```

```
{aux = verb.aux ;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf } ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = "<"+m+">" ++ vp.ad ;
  s2  = \\agr => vp.s2!agr  ++ "</"+m+">"
    } ;

-- marks everything, using the imperative form
markImpVP : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux ;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf } ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = "<"+m+">" ++ vp.ad ;
  s2  = \\agr => vp.s2!agr ++ "</"+m+">"
    } ;

-- marks everything, taking into account
-- the split between aux and the rest
markQVP : Str -> VP -> VP = \m,vp ->
            markVPMain m (markVPAux m vp) ;

-- the following should typically be used to mark focus,
-- instead of phrase-level information structure

-- marks everything except the aux
markVPMain : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux ;
 adv = "<"+m+">" ++ verb.adv ;
 fin = verb.fin ;
 inf = verb.inf
```

```
} ;
  p   = vp.p;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = vp.ad ;
  s2  = \\agr => vp.s2!agr ++ "</"+m+">"
     } ;

-- marks only the auxiliary verb
markVPAux : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = "<"+m+">" ++ verb.aux ++ "</"+m+">";
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf
} ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = vp.ad ;
  s2  = \\agr => vp.s2!agr
     } ;

-- marks only the finite/infinitive main verb
markVPFin: Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux;
 adv = verb.adv ;
 fin = "<"+m+">" ++ verb.fin ++ "</"+m+">" ;
 inf = verb.inf
} ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = vp.ad ;
  s2  = \\agr => vp.s2!agr
     } ;

-- marks only the infinitive form
```

```
markVPInf: Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = "<"+m+">" ++ verb.inf ++ "</"+m+">"
} ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = "<"+m+">" ++ vp.inf ++ "</"+m+">" ;
  ad  = vp.ad ;
  s2  = \\agr => vp.s2!agr
     } ;

-- marks only "not" in the verb phrase
markVPNot : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux ;
 adv = "<"+m+">" ++ verb.adv ++ "</"+m+">" ;
 fin = verb.fin ;
 inf = verb.inf
} ;
  p   = vp.p ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = vp.ad ;
  s2  = \\agr => vp.s2!agr
    } ;

-- marks only the verb parts used in imperatives
markVPImp : Str -> VP -> VP =
\m,vp -> lin VP
{ s   = \\t,ant,cp,o,agr =>
let
verb  = vp.s ! t ! ant ! cp ! o ! agr ;
in
{aux = verb.aux ;
 adv = verb.adv ;
 fin = verb.fin ;
 inf = verb.inf
```

```
    } ;
  p   = vp.p  ++ "</"+m+">" ;
  prp = vp.prp ;
  ptp = vp.ptp ;
  inf = vp.inf ;
  ad  = "<"+m+">" ++ vp.ad ;
  s2  = \\agr => vp.s2!agr
    } ;

}
```

C.1.2 Lines added to ExtraEngAbs.gf

```
fun
    -- ways to build clause types
    ComplClSlash : ClSlash -> NP -> Cl ;
    AddAdjunct : Cl -> Adv -> Cl ;
    QuestSlashAlt : IP -> ClSlash -> QCl ;

    -- additional ways to build VPSlash's;
    VPSlashAdv : VPSlash -> Adv -> VPSlash ;
    VPSlashAdV : Adv -> VPSlash -> VPSlash ;
    SlashV3a    : V3 -> VPSlash ;
    Slash3V3a   : V3 -> NP -> VPSlash ;
```

C.1.3 Lines added to ExtraEng.gf

```
lin

ComplClSlash cl np =
  {s = \\t,a,p,o =>
    cl.s ! t ! a ! p ! o ++ np.s!NPAcc ++ cl.c2 } ;

AddAdjunct cl adv =
  { s = \\t,a,p,o =>
    cl.s ! t ! a ! p ! o ++ adv.s } ;

QuestSlashAlt ip slash =
  mkQuestionAlt (ss (ip.s!NPAcc)) slash ;

VPSlashAdv vp adv = {
s = vp.s ;
p = vp.p ;
prp = vp.prp ;
ptp = vp.ptp ;
```

```
inf = vp.inf ;
ad = vp.ad ;
s2 = \\a => vp.s2 ! a ;
c2 = vp.c2 ++ adv.s ;
gapInMiddle = vp.gapInMiddle
    } ;

VPSlashAdV adv vp = {
s = vp.s ;
p = vp.p ;
prp = vp.prp ;
ptp = vp.ptp ;
inf = vp.inf ;
ad = vp.ad ++ adv.s ;
s2 = \\a => vp.s2 ! a ;
c2 = vp.c2 ;
gapInMiddle = vp.gapInMiddle
    } ;

SlashV3a v = predVc v ;
Slash3V3a v np = predV v **
  {c2 = np.s!NPAcc ; gapInMiddle = True } ;


  oper
agrStr : Str -> (Agr => Str) = \obj -> \\_ => obj ;

mkQuestionAlt :
      {s : Str} -> ClSlash -> QCl = \wh,cl -> lin QCl
      { s = \\t,a,p =>
            let
              cls = cl.s ! t ! a ! p ;
              why = wh.s
            in table {
              QDir   => why ++ cls ! OQuest ++ cl.c2 ;
              QIndir => why ++ cls ! ODir ++ cl.c2
              }
      } ;
```

C.1.4 Question-answering system

```
class FootballQASystem:

  def __init__(self, football_grammar_filename, language):

    self.football_grammar = pgf.readPGF(football_grammar_filename)
    self.lang_grammar = self.football_grammar.languages[language]
```

```
  self.salients_categories = []

  self.teamV1 = [ x for x in \
      self.football_grammar.functionsByCat("TeamV1") \
      if x.find("focus") == -1]
  self.teamteamv2 = [ x for x in \
      self.football_grammar.functionsByCat("TeamTeamV2") \
      if x.find("focus") == -1]
  # ... etc ...

  self.players = [ x for x in \
      self.football_grammar.functionsByCat("Player") \
      if x.find("focus") == -1]
  self.teams = [ x for x in \
      self.football_grammar.functionsByCat("Team") \
      if x.find("focus") == -1]
  # ... etc ...

  self.players_salient = []
  self.teams_salient = []
  # ... etc ...

  self.salients_categories.append(('Player',
                                   self.players,
                                   self.players_salient))
  self.salients_categories.append(('Team',
                                  self.teams,
                                  self.teams_salient))
  # ... etc ...

  self.updated = []

# main entry point
def reply(self, input):
  self.updated = []

  try:
    utt_iter = self.lang_grammar.parse(input)
    (probability,utt_expr) = utt_iter.next() # get first parse
    (utt_type, utt_children) = utt_expr.unpack()

    utt_types = {'QUtt' : self.answer,
                 'DeclUtt' : lambda x: "I suppose that's true..."}

    func = utt_types.get(utt_type, lambda x: None)
    answer = func(utt_children)
    if answer:
      tree = "DeclUtt (" + answer + ")"
    else:
      return "Error: could not compute an answer"
```

```
    self.decay_salients()

    return self.lang_grammar.linearize(pgf.readExpr(tree))

  except pgf.ParseError:
    return "Error: could not parse"
  except NoParseException:
return "Error: could not parse"
  except StopIteration:
    return "Error: no parse"

# determine question type and call appropriate method
# to construct answer tree string
def answer(self, utt_children):

  question_expr = utt_children[0]
  (q_type,q_children) = question_expr.unpack()

  q_types = { # traditional structures
              'WhoPlayerQ' : self.whoXA,
              'WhoTeamQ' : self.whoXA,

              # traditional with adverb
              'WhenQ' : self.whenA,

              # simple vpslash from v2
              'WhomTeamQ' : self.whomXA,
              'WhomPlayerQ' : self.whomXA,

              # simple vpslash from v3
              'WhomPlayerQ2' : self.whomXA,

              # vpslash from v3 with complex answer
              'HowMuchQ' : self.whomXA,

              # vpslash from v2 with common nouns
              'HowManyQ' : self.howManyA,
              'WhichPointQ' : self.whichPointA,
              'WhichContestQ' : self.whichContestA
            }

  func = q_types.get(q_type, lambda x: None)
  answer = func(q_type,q_children)
  return answer

# builds answer tree string from (focus marked) theme and rheme
def whoXA(self, q_type, q_children):
  vp_expr = q_children[0]
  (vp_type, vp_children) = vp_expr.unpack()
```

```
  vp_types = { 'addTimeVPT' : self.addTimeVPT,
               'addTimeVPP' : self.addTimeVPP,
               'mkVPPlayerPoint' : self.mkVPPlayerPoint,
               'mkVPTPA' : self.mkVPTPA,
               'mkVPTeam' : self.mkVPTeam,
               'mkVPTeamContest' : self.mkVPTeamContest,
               'mkVPTeamPlayer' : self.mkVPTeamPlayer,
               'mkVPTeamTeam' : self.mkVPTeamTeam
             }

  func = vp_types.get(vp_type, lambda x: None)
  (answer,theme_string) =  func(vp_children)
  theme_string = self.mark_focus_theme(theme_string)
  rheme_string = self.mark_focus_rheme(answer)
  return self.answer_type(q_type) +" "+ rheme_string \
         +" "+ theme_string

# theme string and query are constructed
# - query is used to compute answer element
# - answer element and theme string are returned
def mkVPTeamTeam(self,vp_children, time="_"):
  v2_expr = vp_children[0]
  v2_type = v2_expr.unpack()[0]
  team_expr = vp_children[1]
  team_type = team_expr.unpack()[0]
  vp_string = "(mkVPTeamTeam "+v2_type+" "+team_type+")"

  answer_team = self.query(v2_type, team_type, time)
  theme_string = "(mkVPTeamTeam "+v2_type+" "+team_type+")"

  return (answer_team, theme_string)

# marks the complete subtree with root c as focused
  def mark_subtree_focus(self, c):
      if len(c.split()) > 0:
          head = c.split()[0]
      else:
          head = c
      if head in self.teams:
          return "(focusTeam "+c+")"
      elif head in self.players:
          return "(focusPlayer "+c+")"
      elif head in self.timesA:
          return "(focusTimeA "+c+")"
      elif head in self.timesN:
          return "(focusTimeN "+c+")"
      elif head in self.amounts:
          return "(focusACard "+c+")"
      elif head in self.fcards:
```

```
            return "(focusFCard "+c+")"
        elif head in self.fords:
            return "(focusFOrd "+c+")"
        else:
            return c

# entry point for marking theme focus
  def mark_focus_theme(self,theme_string):
      theme_expr = pgf.readExpr(theme_string)
      return self.mark_theme_nodes_focus(theme_expr)

# recurively marks as focused all nodes
#   (and their subtrees) as defined in the
#   alternative sets
# makes them salient in the discourse
def mark_theme_nodes_focus(self, expr):
  (type,children) = expr.unpack()
  if type in self.currencies:
    amount = children[0].unpack()[0]
    if self.is_contrastive(amount):
      marked = "("+type+" "+self.mark_ct_focus(amount)+")"
    else:
      marked = "("+expr.__str__()+")"
      self.make_salient(amount)
  else:
    if len(children) == 0:
      if self.is_contrastive(type):
        marked = self.mark_ct_focus(expr.__str__())
      else:
        marked = "("+expr.__str__()+")"
    else:
      child_string = ""
        for c in children:
          child_string = child_string +" "+ \
                         self.mark_theme_nodes_focus(c)
        if self.is_contrastive(type):
          marked = "("+self.mark_ct_focus(type+" "+child_string)+")"
        else:
          marked = "("+type +" "+child_string+")"

  self.make_salient(type)
  return marked

# marks "composite" rhemes with focus
# makes all mentioned types salient in the discourse
def mark_focus_rheme(self,rheme_string):

  rheme_expr = pgf.readExpr(rheme_string)
  (rheme_type, rheme_children) = rheme_expr.unpack()
```

```
  a = rheme_type in self.currencies
  b = rheme_type in self.points
  c = rheme_type in self.contests

  if a or b or c:
    child_type = rheme_children[0].unpack()[0]
    if not self.is_salient(child_type):
      rheme_string = "("+rheme_type +" "+ \
                     self.mark_ct_focus(child_type)+")"
    elif  self.is_salient(child_type):
      rheme_string = "("+self.mark_ct_focus(rheme_expr.__str__())+")"
    self.make_salient(child_type)

  self.make_salient(rheme_type)
  return "("+rheme_string+")"

# helper function for computing default answer types
def answer_type(self,q_type):
  q_index = q_type.rfind('Q')
  q_code = q_type[q_index:]
  a_code = q_code.replace('Q','A')
  return q_type[:q_index]+a_code
```

Appendix D

List of Acronyms

Acronym	Meaning
AM	Autosegmental-metric
CCG	Combinatory Categorial Grammar
CTS	Context-to-speech
GF	Grammatical Framework
GPSG	Generalized phrase structure grammar
HMM	Hidden Markov model
HTK SLF	Hidden Markov Model Toolkit Standard Lattice Format
ISU	Information State Update
JSGF	JSpeech Grammar Format
Nuance GSL	Nuance Grammar Specification Format
OWL	Web Ontology Language
PGF	Portable Grammar Format
QA	Question-answering
RGL	Resource Grammar Library
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
SUMO	Suggested Upper Merged Ontology
ToBI	Tones and Break Indices
TTS	Text-to-speech

Table D.1: List of acronyms