Chapter 5

Spoken Language Generation

5.1 Introduction

Spoken language generation (SLG) concerns computer programs that transform non-linguistic input into intelligible speech, for a certain communicative goal, and using knowledge of the context and the language. Our research goal is to design, build, and test a linguistically-based SLG system that generates speech with pitch accents and intonational boundaries that most acceptable to experts (cf. Section 1.3). This objective gives rise to two types of research questions: those concerning the linguistic theory on which the system is based (how adequate is it, and how can it be improved?), and those concerning the architecture of an SLG system. The latter are the topic of the current chapter.

In Section 5.2, we discuss the general architecture of an SLG system. Since SLG is a combination of natural language generation (NLG) and speech synthesis, we start with summarising the architectures of conventional NLG systems (Section 5.2.1) and speech synthesisers (Section 5.2.2). Next, we address the questions of how to combine NLG and speech synthesis systems, and what additional information (besides text) must be generated by the NLG system to facilitate the control of intonation (Section 5.2.3). We continue with discussing the best place in the system for controlling the intonation (Section 5.2.4). Finally, we propose an architecture consisting of a domain-specific text planner and a domain-independent modular content-to-speech system called CONPAS, which forms the basis for our SLG systems.

In the next four sections, we describe the modules of our SLG system for text planning (Section 5.3), syntactic realization (Section 5.4), prosodic realization (Section 5.5), and speech synthesis (Section 5.6), and the intermediary representations that serve as the interface between them. The chapter ends with a summary (Section 5.7).

5.2 The architecture of an SLG system

5.2.1 Natural language generation

The first — and only so far — text book on natural language generation gives the following definition.

> "Natural Language Generation (NLG) is the subfield of artificial intelligence and computational linguistics that focuses on computer systems that can produce understandable texts in English or other human languages. Typically starting from some non-linguistic representation of information as input, NLG systems use knowledge about language and the application domain to automatically produce documents, reports, explanations, help messages, and other kinds of texts." (Reiter and Dale 2000, p. 1)

Traditionally, the process of language generation has been divided into two tasks: determining *what* to say (content), and determining *how* to say it (form) (Bateman and Hovy 1991; De Smedt, Horacek, and Zock 1996). The former became known as *strategic generation*, *deep generation*, *conceptualisation*, or *text planning*; the latter as *tactical generation*, *surface generation*, *formulation*, or *linguistic realization*. More recently, however, the most popular architecture for NLG systems involves three stages. We will first summarise this architecture, and later on investigate how it can be adapted and extended for current purposes.

Figure 5.1 shows the standard architecture of an NLG system following Reiter and Dale (1997, 2000), referred to as the 'consensus architecture' in Reiter (1994). Although it is generally acknowledged that it has certain drawbacks, it appears to be the best architecture available for a practical NLG system. The input is assumed to be a combination of four types of information:

1. **Knowledge source**
   The knowledge source is some structured form of information about the domain from which the basic content of the text is derived. Taking train travel descriptions as an example, the knowledge source is a table with information about time and place of departure, time and place of arrival, type of conveyance, etc.[1]

2. **Communicative goal**
   The communicative goal indicates the purpose of the text to be generated. For instance: to let the user know how to get from A to B by train.

[1] We assume that the NLG system receives only the relevant information from a preceding program. This is called *push* use of NLG technology in McDonald (1999). Alternatively, the NLG system may search a knowledge base itself, which is called *pull* use.

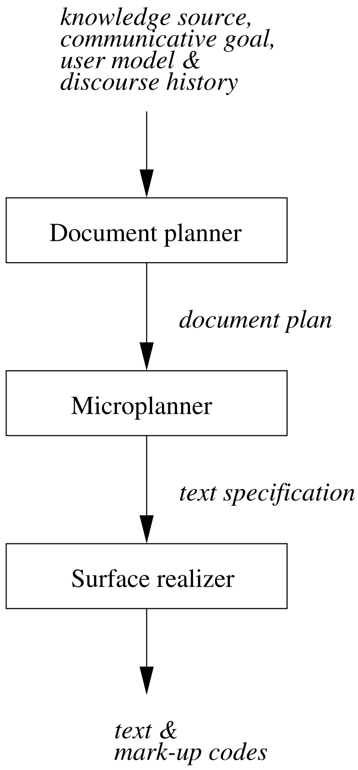

Figure 5.1: The standard architecture of an NLG system.

3. **User model**
   The user model contains a profile of the user. It may specify, for example, whether a particular user prefers a summary or extensive travel description.

4. **Discourse history**
   The discourse history is a structured record of what has been said so far in the discourse. It may inform the system, for example, that a particular station name has been recently mentioned.

The output of an NLG system can be either a plain text in electronic form, or a text enhanced with mark-up codes to be interpreted by a particular *document representation system* like a text formatting system (e.g. LaTeX) or a web browser.

The standard NLG system architecture consists of modules connected in a pipeline. Information flows from one module to the next, but feedback to a previous module is not allowed. There are three modules:

1. **Document planner**
   The document planner generates a global plan for the final document, which is called a *document plan*. It selects the relevant pieces of information from the knowledge source. Continuing with our example of train travel descriptions, it may decide – depending on the preferences stored in the user model – to either include the platform numbers or to omit them. It may derive new information by means of reasoning. For instance, it may calculate the waiting time, or the difference in time between two alternative routes. These tasks are known as *content determination*. In addition, the document planner is responsible for *document structuring*. This involves – among other things – a hierarchical structuring of the information based on coherence. As far as travel descriptions are concerned, the natural structure is dictated by the chronological order of the events, i.e. a sequence of departure-arrival pairs.

2. **Microplanner**
   The microplanner refines the global document plan, bringing it substantially closer to a linguistic form. It performs *lexical selection*, i.e. choosing content words, and *referring expression generation*, i.e. constructing appropriate expressions to refer to discourse entities. Here, for instance, the microplanner may choose to use the verb *depart*, and to refer to a station that was mentioned in the previous sentence by means of the anaphoric expression *there*. In addition, it performs *aggregation*, or parcelling of information that will be expressed in a single phrase or sentence. The result of microplanning is a *text specification* in the form of a tree. The non-terminal nodes are textual units like *section list*. The terminal elements are *phrase specifications*, which are detailed plans for the final form of phrases.

3. **Surface realizer**
   The surface realizer generates the final form of the text, carrying out two tasks. First, it maps the phrase specifications onto grammatical expressions in the target language. This is called *linguistic realization*, and commonly relies on a syntactic grammar to enforce grammatical constraints. It involves the insertion of function words, the inflection of content words, and the determination of the word order. Often, adding orthographic symbols is looked upon as a part of linguistic realization as well. Second, it adds mark-up codes for the document representation system, which is called *structure realization*. For instance, a textual element like *list* may be mapped onto the HTML tags for an (un)ordered list.

For a software system of such complexity as NLG and SLG systems, a modular design is to be preferred over a monolithic design, because when the information is concentrated and encapsulated in modules, it becomes easier to design, implement, understand, document, maintain, and adapt the system. In addition, it provides the opportunity to experiment with different versions of a module, and to reuse one or more modules in a new system. The same considerations argue in favour of using a pipeline architecture, in which the flow of information is strictly sequential, and no feedback is allowed. Although pipeline architectures have been criticised for being too restrictive (Inui, Tokunaga, and Tanaka 1992; Rubinoff 1992), alternative architectures are significantly harder to implement. For the time being, therefore, the best thing is to elaborate on the three-stage pipeline architecture when designing an SLG system.

5.2.2 Speech synthesis

> "Speech synthesis is the process which allows the transformation of a string of phonetic and prosodic symbols into a synthetic speech signal." (d'Alessandro and Liénard 1995, p. 196)

Traditionally, *speech synthesis* (SS) has been developed and applied in the context of *text-to-speech* (TTS). A TTS system takes as input a text consisting of words and punctuation marks, and attempts to read it aloud as well as possible.[2] Although SS and TTS are often used as synonyms, the former is actually a part of the latter. In general, TTS involves two tasks (see e.g. Klatt (1987), Dutoit (1997), and Sproat (1997)):

1. A linguistic analysis of the input, which involves subtasks such as text preprocessing, grapheme-to-phoneme conversion, and determination of the prosody, of which the intonation is an important part.[3] The result of this is usually an abstract specification of an utterance in terms of phonetic symbols (phonemes) and prosodic annotations (e.g. accent and pause markers).

2. Speech synthesis, which involves mapping this specification into a synthetic speech signal.

Admittedly, the precise boundary between the two tasks is somewhat arbitrary, and differs from one TTS system to another. However, all systems use one or more intermediate levels of representation between the input (text) and the output (speech). This means that, in principle, it is possible to specify input to the synthesiser in terms of one of these intermediate representations as well.

[2] Recent approaches investigate the use of HTML or XML documents as input, which allow for annotations in the form of mark-up codes that guide the conversion of text to speech (Sproat et al. 1998).

[3] *Prosody* is defined in Section 2.2.1; *intonation* in Section 1.1.

5.2.3 Combining NLG and SS

The trivial way to combine NLG and SS systems is by using the text output of an NLG system as input to a TTS system. As might be expected, this is not the best approach. By using text as the interface, we throw away valuable linguistic information produced by the NLG system in the process of language generation, and introduce a lot of redundant processing. One example of this concerns the syntactic structure. As explained above, NLG systems usually have a syntactic realizer, a module which takes care of the syntactic form of sentences. Typically, a syntactic grammar is used for building syntactic structures, which guarantees that only acceptable strings are generated. Then why should a TTS system attempt to parse a potentially ambiguous sentence, if it can get its syntactic structure for free from the generator? A much better alternative, therefore, is to have the NLG system output not only the text, but also the accompanying syntactic structure.

The same goes for other types of linguistic information. In order to produce coherent pieces of text, the document planner in an NLG system relies on a discourse model that keeps track of the informational status of discourse referents. For instance, to use a pronoun, the generator needs to know that its antecedent is old, i.e. has been recently mentioned. This type of information, which is hard to extract automatically from a bare text, plays an important role in pitch accent placement (see Section 2.5.1). Likewise, generators often employ semantic categories such as predicates, arguments and modifiers. This kind of information is very useful for the correct accentuation of verbs (see Section 2.5.2). In conclusion, bare text is not the appropriate interface between NLG and SS, and a more elaborate representation is preferable. The same point has been made by a number of other researchers, e.g. Bateman (1997), Hess (1997), McKeown and Moore (1995), and Sproat (1997); in fact, the idea goes back as far as Young and Fallside (1979).

The interface proposed in the current work consists of three elements:

1. a phrase, i.e. a string of graphemic words;
2. its intonational structure, i.e. a phonological representation of the pitch accents, intonational boundaries, and tune;
3. part of its prosodic structure, i.e. a tree consisting of the higher-level prosodic constituent (U, π, φ).

This means that most of the tasks carried out during the linguistic analysis of the input in a conventional speech synthesis system, that is, a TTS system, become superfluous, as the desired information is already available in the input. The remaining tasks for the speech synthesiser involve grapheme-to-phoneme conversion, deriving the lower-level prosodic constituents, phonetic realization in terms of intonation and duration, and actual speech synthesis. These tasks will be described in Section 5.6. At the same time, the intonational and (part of the) prosodic structure is assumed to be generated before the speech synthesiser is reached. This raises the question where to generate prosody.

5.2.4 Where to generate prosody

Where does the generation of a phonological description of the prosody – including the intonational form – fit into the architecture of an SLG system? In fact, there is more than one plausible location: it can be seen as belonging to the microplanner, the surface realizer, or a separate module.

Recall that accent placement is highly correlated with the information status of discourse entities (cf. Section 2.5.1), that is, new information is usually accented, whereas old information is normally left unaccented. A similar correlation holds with respect to a definite versus an indefinite description, and between a full description versus an anaphoric one. Therefore, accent placement is akin to referring expression generation. Likewise, the structuring of information into sentences is akin to collecting information into intonational phrases, that is, aggregation is similar to intonational phrasing. Since both referring expression generation and aggregation belong to microplanning, it follows that prosodic realization must belong there as well.

However, there is a problem with this conception: certain information required for prosodic realization is not available when microplanning takes place. For instance, accent deletion rules like SAAR (cf. rule (20)) and the Rhythm Rule (cf. rule (22)) require the word order to be known. However, the word order is not known until after the next module has applied, that is, the surface realizer. Similarly, intonational phrasing requires information about the syntactic structure of the utterance, which is to be generated by the next module. Therefore, prosodic realization cannot be a part of the microplanner – or at least not entirely – because at that stage in the generation process the form of the text has not yet been sufficiently determined.

Alternatively, prosodic realization may be looked upon as a form of structure realization, which thus belongs to the surface realizer. Indeed, there are many parallels between printed, electronic, and spoken documents. According to this view, a speech synthesiser is just another type of document representation system, akin to a text formatting system or a web browser, and prosodic annotations are similar to mark-up codes or tags. For instance, the annotation for a pitch accent may be compared to a mark-up code for emphasis, which will ultimately be realized by the document representation system with the typographic means defined in its style sheets (e.g. in italics or underlined). Along the same lines, symbols for prosodic phrase boundaries may be compared to punctuation symbols or paragraph formatting commands. In addition, certain intonational tunes – question, exclamation, parenthetical remarks – appear to have direct orthographic counterparts. Thus according to this view, prosodic structuring takes place at an abstract level in the document planner and the microplanner, and its subsequent realization in the surface realizer is merely a straightforward mapping onto the prosodic symbols that are interpreted by the speech synthesiser.

Although this approach may be adequate for punctuation, orthography and layout, it is far less appropriate for speech, because it implies that the important decisions regarding prosody are made in the planning modules, leaving for the surface realizer only a more or less trivial mapping onto prosodic symbols. However, as was argued above, many decisions about prosodic realization cannot be made at the level of micro planning, let alone document planning, because at that stage in the generation process the final form of the text has not yet been sufficiently determined. Moreover, this approach fails to do justice to the fact that prosodic realization is a process of an inherently linguistic nature, rather than a rigid mapping from one structure to another. Accent placement, for instance, depends in a complex way on the focus distribution, the semantic structure, and the prosodic structure (which in turn depends on the syntactic structure). Therefore, even though there is an pervasive parallel between the prosodic structure and the visual structure of a document, the surface realizer is not the appropriate place for prosodic realization either.

A third option is to consider prosodic realization a separate task that is on a par with syntactic realization. This assumes that it is a linguistic rather than a purely formal mapping, and that it can be modelled in the form of a grammar, just like morphology, syntax, and segmental phonology. In addition, this approach provides the advantages of a modular architecture (cf. Section 5.2.1). We will therefore assume that prosodic realization requires an independent module that connects the surface realizer to the speech synthesiser. This means that the syntactic structure of an utterance, including its word order, is fully determined before its prosodic realization is taken into account, and that the problems mentioned earlier are avoided.

To sum up, prosodic realization cannot be reduced to either microplanning or structure realization, but is a separate task that requires an independent module. The description of prosodic realization in Chapter 7 will further support this view by showing that it enables a satisfactory implementation of the process.

5.2.5 A reusable content-to-speech system

Surface realization, and syntactic realization in particular, is probably the most well established branch of NLG. At least for English, several linguistic realization modules have been developed that are capable of mapping phrase specifications onto grammatically correct English phrases, e.g. SURGE (Elhadad and Robin 1996; Elhadad and Robin, to appear), KPML (Bateman 1994; Bateman 1997), RealPro (Lavoie and Rambow 1997; CoGenTex, Inc 1998), and Multex (Matthiesen, Licheng, Cross, Kobayashi, Teruya, and Canzhong 1998). These are freely or commercially available, reusable software packages that can be used to build new NLG systems. There are also surface realizer for Dutch (Appelo, Leermakers, and Rous 1993; Degand 1993; Degand 1996; De Smedt 1990; Kempen and Hoenkamp 1987; Marsi 1998b; Teunissen 1997), although their coverage is significantly less extensive than that of their English counterparts (see also Section 6.5.3). They allow developers of an NLG system to ignore the low-level syntactic details of the target language, and instead to focus on issues of document structuring and microplanning. Similar packages for document planning are not yet common, primarily because planning is intrinsically bound to a certain domain. For example, whereas the syntax used in a software user manual and a fairy tale is essentially the same, the planning processes for these texts are likely to be very different in nature (at least for most software).

Interestingly, prosodic realization appears to be closer to syntactic realization in this respect. That is, prosodic realization is fairly domain-independent; hence the existence of general-purpose TTS systems. This suggests that it is possible – at least to a certain degree – to develop a *reusable* 'content-to-speech' module that serves as the front-end for an SLG system, analogous to the reusable surface realizer in an NLG system. To carry the comparison further, this allows the developers of an SLG system to ignore the prosodic idiosyncrasies of the target language as long as they make sure their planning components deliver the required input. As a matter of fact, reusability provides an additional argument against the view of prosodic realization as a part of microplanning: since microplanning, in particular lexical choice and aggregation, is a fairly domain-dependent task, it would prevent the construction of a reusable module.

Following this idea, our SLG systems have two major parts. The first is a *domain-specific* text planner (PlanR) that is responsible for document structuring and microplanning. It generates the *content* of the text in the form of a text specification that includes phrase specifications. The second is a *domain-independent* front-end that is shared by all systems. This part is called CONPAS, which stands for 'CONtent to Prosodically Adequate Speech'. It is responsible for the *form* of the text, or rather, the speech. It translates content into a speech signal, using knowledge about the morpho-syntactic and prosodic properties of the target language. Figure 5.2 gives an overview of CONPAS as part of an SLG system.

Internally, CONPAS consists of three modules:

1. SynR: a module for syntactic realization
2. ProsR: a module for prosodic realization
3. SpeechS: a module for speech synthesis

Each of these modules will be described in the following sections, although SynR and ProsR are just introduced here, because they will be described in detail in the next two chapters.

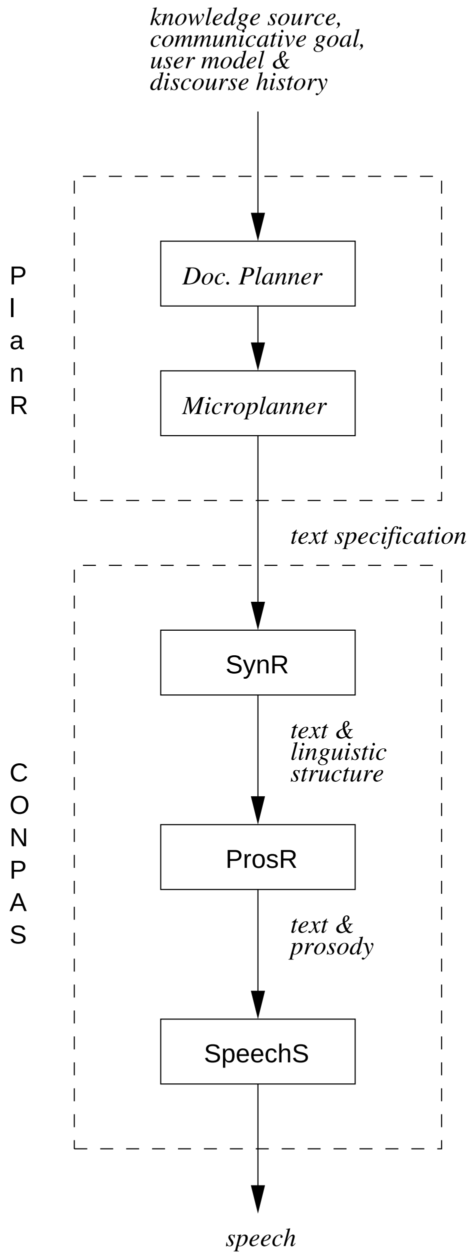

Figure 5.2: The architecture of an SLG system using CONPAS as its front-end.

5.3 Text Planning

5.3.1 Application domains

Unrestricted SLG is an achievement that is still beyond our reach, if only for the fact that a general-purpose knowledge representation formalism to model arbitrary domains is lacking. Restricting the domain of application of an SLG system is therefore unavoidable. Our work explored two application domains: the *Wild Flower Domain* and the *Public Transport Domain*.

The Wild Flower Domain (WFD) concerns botanical descriptions of wild flowers akin to those used in corpus-based work in the previous two chapters. These texts typically have long and complex utterances, which makes them challenging from a prosodic point of view. The choice for this domain allows us to reproduce the achieved results on phonological and intonational phrasing in an actual SLG system. The challenge is to show that the type of syntactic,semantic, and focus structures assumed, and manually assigned, in the corpus of flower descriptions can indeed be automatically generated by an SLG system. Example (1) is a short flower description generated by WFDGen, the CONPAS-based SLG system for this domain.

(1) a. *aardoorn is een *eenjarige pl*ant # met k*orte # d*ikke #
aardoorn is an annual plant with short thick
d*onkerbruine st*engels # en gr*ote # d*onkergekle*urde #
dark-brown stalks and big dark-colored
vl*ezige bl*aderen
fleshy leaves

b. de plant heeft r*ode # k*omvormige blo*emen die op een
the plant has red bowl-shaped flowers that at a
lanta*arn lijken
lantern resemble
'The plant has red, bowl-shaped flowers that look like a lantern.'

c. hij groeit in b*ossen en w*eilanden # in het no*orden van
he grows in forests and meadows in the north of
eur*opa
Europe
'It grows in forests and meadows in Northern Europe.'

The Public Transport Domain (PTD) concerns travel descriptions. The goal is to generate spoken travel descriptions that assist the user in travelling from one place to another using the public transport system, i.e. train, bus, or ferry.[4] In contrast to WFD, the utterances in this domain tend to be relatively

[4]Earlier work on PTD is reported in (Marsi 1995).

short and have a rather simple syntactic structure. For instance, subordinate clauses are rare, whereas complex noun phrases with multiple premodifiers do not occur at all. However, one of the interesting aspects of PTD is the occurrence of numbers and times. Also, unfocused information occurs more frequently than in WFD. We will refer to the CONPAS-based SLG system for this domain as PTDGen. Example (2) is a short travel description generated by PTDGen.

(2) a. eerst neemt u om $\overset{*}{a}$chtien uur dri$\overset{*}{e}$ # de sn$\overset{*}{e}$ltrein
first take you at eighteen hour three the fast-train
die richting $\overset{*}{a}$rnhem en den h$\overset{*}{a}$ag gaat
that towards arnhem and the hague goes
'First, you take the fast train to Arnhem and Den Haag at 18.03'

b. die vertrekt in nijmegen van perron $\overset{*}{e}$en $\overset{*}{a}$
that departs in nijmegen from platform one a
'Which leaves in Nijmegen from platform 1a'

c. dan komt u om n$\overset{*}{e}$gentien $\overset{*}{u}$ur in $\overset{*}{u}$trecht aan
then arrive you at nineteen hour in utrecht `prt`
'Which gets you to Utrecht at 19.00.'

d. vervolgens stapt u op de interc$\overset{*}{i}$ty richting amsterd$\overset{*}{a}$m en
next step you on the intercity towards amsterdam and
h$\overset{*}{a}$arlem # die ger$\overset{*}{e}$edstaat op perron z$\overset{*}{e}$ven $\overset{*}{a}$ $\overset{*}{b}$
haarlem that stands-ready at platform seven a b
'Next, take the intercity to Amsterdam and Haarlem, which is waiting at platform 7a/b.'

e. die vertr$\overset{*}{e}$kt in utrecht om n$\overset{*}{e}$gentien uur tw$\overset{*}{e}$e # en
that leaves from utrecht at nineteen hour two and
die komt in amsterdam $\overset{*}{a}$an om n$\overset{*}{e}$gentien uur
that arrives in amsterdam `prt` at nineteen hour
v$\overset{*}{i}$erend$\overset{*}{e}$rtig
thirty-four
'Which leaves from Utrecht at 19.02 and arrives in Amsterdam at 19.34.'

5.3.2 On the approach to text planning

Recall that the focus of the current research is on prosodic realization, in particular pitch accent placement and intonational phrasing. We argued earlier that this task should not be a part of microplanning, let alone document planning. As a consequence, both types of planning are less relevant for our purpose, allowing us to opt for a rather basic approach. The most important issue in planning is what information should be produced to facilitate syntactic and prosodic realization later on. In fact, in some respects it would be more appropriate to talk about our approach as *simulating* a real planning process.

To begin with, we will not maintain a distinction between document planning and microplanning. Although we fully subscribe to the arguments for distinguishing these two stages in NLG, there is no reason for it in our simple approach. Instead, we will refer to them collectively as *text planning*. As explained above, we assume that text planning is domain-specific, whereas the domain-independent realization is taken care of by CONPAS. As a result, there is a specialised text planner for each domain, i.e. a text planner for PTD called PTDPlanR, and one for WFD called WFDPlanR. Whenever the domain is evident or irrelevant, we use the term *text planner* (PlanR).

There are roughly two approaches to text planning found in NLG. *Relation-based planning*, which originates from the work of by Hovy (1988, 1991), relies on an AI-based planner that constructs hierarchical text specifications by combining rhetorical relations in order to satisfy communicative goals. *Schema-based planning*, which goes back to McKeown (1985b, 1985a), employs text schemas that describe the structure of a text, comparable to the way a syntactic grammar describes the syntax of a language. Although schema-based planning is less flexible, it it easier to implement, fast, and well-suited for generating the PTD and WFD type of texts. Hence, both PTDPlanR and WFDPlanR are schema-based text planners. They are discussed in more detail in the next section.

A more dynamic, constraint-based method of text planning for PTD is described in Marsi (1998a). In this method, the packaging and linear ordering of information, the focus distribution, the use of anaphoric expressions, and that of discourse markers result from explicit calculation. It is implemented as a functional unification grammar in FUF (Elhadad 1993). For reasons of space, we will not go into it here.

In contrast to surface realization modules, which are often implemented in a special-purpose grammar formalism or programming language, most text planners are coded directly in a general-purpose programming language, because the domain-dependent nature of the process has prevented the development of sufficiently general methods. The same goes for PTDPlanR and WFDPlanR, which are both implemented in Icon, a very high-level programming language with extensive facilities for processing strings and structures (Griswold and Griswold 1993).[5]

The purpose of our text planners is somewhat different than that of most text planners. Text planners are usually designed to generate a text specification that is optimally suited to achieve the communicative goal aimed for. This means that they are deterministic: with identical inputs, they will generate identical outputs. However, since we intend to use SLG systems as a test bed for prosodic theory, we are interested in generating as much variation as

[5] According to present terminology, Icon qualifies as a *scripting language*, akin to languages like Perl and Python.

possible. After all, the more varied the texts are, the better these theories can be put to the test. Hence, our text planners emphasise generating variation.

Recall from Section 5.2.1 that, according to the standard architecture, the input to an NLG system consists of a knowledge source, a communicative goal, a user model, and a discourse history. Actually, most systems take less input, because part of it is hard-wired in the program. For instance, many systems do not employ a user model, as assumptions about the user are implicit in the code. Of course, this precludes intelligent behaviour like tailoring output to different users (Paris 1988). However, since we are less interested in planning issues here, the input to our text planner is rather limited: PTDPlanR takes as input a knowledge source and a communicative goal, whereas WFDPlanR takes no input at all, as all of it is implicitly assumed in the code, which means that texts are randomly generated.

To sum up, text planning, which in our case covers both document planning and microplanning, is accomplished by a straightforward, schema-based approach, which is implemented in a general-purpose programming language. It aims at generating maximal variation on the basis of limited input.

5.3.3 Abstract Phrase Specifications

Prior to describing the details of the text planner, we have to address its output, which also is the input to CONPAS. According to the standard architecture of an NLG system described earlier, the result of text planning[6] is a tree-shaped text specification. The nonterminal nodes represent textual units – e.g. *section*, *paragraph* or *list* – which will ultimately be mapped to punctuation symbols and/or mark-up codes during structure realization. The terminal nodes are phrase specifications that tell the surface realizer what kind of phrase it should generate. Phrase specifications can be supplied at different levels of abstraction. They can take the form of canned text, which means that they are very close to the surface form, and the task of the surface realizer is limited to string concatenation and some orthographic processing. Close to the other extreme, they can be abstract specifications of the semantic structure in terms of semantic roles and features, often referred to as *lexicalised case frames* (Bateman and Hovy 1991; Reiter and Dale 2000). These require the surface realizer to have detailed knowledge about the syntax and semantics of the target language. The main advantage is that the more abstract a phrase specification is, the more freedom the planner has in constructing phrase specifications that optimally serve the communicative goal aimed for.

Clearly, canned text is not an option in an SLG system, because the corresponding surface realizer fails to provide the linguistic structure of phrases required for prosodic realization. In contrast, lexicalised case frames appear to hold too much information for our purpose. For instance, although the difference between arguments and modifiers is important for pitch accent placement, the subclassification of arguments into semantic roles like *agent*, *theme* and *ben-*

[6] Actually, the result of document planning followed by microplanning.

eficiary is assumed to be irrelevant (Gussenhoven 1992b). Therefore, the type of phrase specification used by CONPAS is somewhere in the middle between these extremes. It is called an *Abstract Phrase Specification* (APS), and is akin to a dependency structure. It describes a phrase in terms of content words with features, and their dominance relations. The format of APS's will be described in detail in the next chapter. For now, it is sufficient to have some idea about the form and meaning of APS's, so we can continue the discussion of text planning. Some examples with their corresponding strings are given in (3).

Example (3-a) is the most basic phrase specification for an NP. By default, the NP gets an indefinite determiner. This is a common thread in all APS's: the assumption of reasonable defaults allows for concise specifications. The feature def in (3-b) makes the NP definite. A modifier is added in (3-c), where P: indicates the syntactic category of the modifier's head, i.e. Preposition, which is different from the default, i.e. Noun. In addition, prop indicates a proper noun, which normally does not take a determiner. Finally, (3-d) shows the specification for a full sentence. Arguments of the verb are enclosed in square brackets instead of parentheses. [<listen>] generates a noun without content, i.e. a 'pronoun', where the feature for 'listener' (2nd person) helps select the correct form.

(3) a. APS: (sneltrein)
String: een sneltrein
a fast-train

b. APS: (sneltrein<def>)
String: de sneltrein
the fast-train

c. APS: (sneltrein<def>(P:naar(Amsterdam<prop>)))
String: de sneltrein naar Amsterdam
the fast-train to Amsterdam

d. APS: (V:nemen[<listen>]
[sneltrein<def>(P:naar(Amsterdam<prop>))])
String: u neemt de sneltrein naar Amsterdam
you take the fast-train to Amsterdam

The goal of text planning in PTD and WFD is to generate a list of APS's for a travel or flower descriptions respectively.

```
From Nijmegen to Amsterdam CS
10th of January 2000, Monday

Station          Time   Track  Type        Direction/Destination

Nijmegen     D   18:03  1a     Fast train  Arnhem, Den Haag CS
Utrecht CS   A   19:00
Utrecht CS   D   19:02  7a/b   Intercity   Amsterdam CS, Haarlem
Amsterdam CS A   19:34

Travel time: 1:31, with 1 transfer
```

Figure 5.3: Output of the travel information system that serves as the knowledge source for PTDPlanR.

5.3.4 Text planning in PTD

To start PTDGen, the user queries the *NS Reisplanner*, a public transport information system, on how to get from one place to another at a certain date and time. Alternatively, a query may be generated randomly. The output of the information system is an ASCII file with tabular data regarding place and time of departure, place and time of arrival, and type of conveyance. Normally, the direction of the conveyance, its number, and the platform it leaves from are given as well. See Figure 5.3 for an example.[7]

This tabular information serves as the knowledge source for PTDPlanR. The file is parsed and for each *section* (a part from station to station) the travel parameters are stored in the form of a record. The record corresponding to the first section in Figure 5.3 is shown in Figure 5.4, where `sec` is the name of the record, and `sec.dp` refers to the field storing the departure place, `sec.dt` is the the departure time, etc.[8]

In addition, the planner accepts a communicative goal that specifies the type of description that is required. This can be BRIEF DESCRIPTION, NORMAL DESCRIPTION, or ELABORATE DESCRIPTION. For a BRIEF DESCRIPTION, it includes only the essential information, i.e. time of departure and destination, sometimes accompanied by the place of departure and type of conveyance, all in a single sentence. In a NORMAL DESCRIPTION, the place of arrival and the type of conveyance are always mentioned, while direction and platform are included when available. The information is divided over more than one sentence. Finally, an ELABORATE DESCRIPTION includes all information available

[7]The original output is in Dutch. For convenience, the English translation is given here. Moreover, there is an additional column called *Opmerkingen* 'Remarks' that is ignored by the planner.

[8]Sometimes the platform or direction are lacking, and the corresponding fields in the record remain empty.

```
sec.dp = "Nijmegen"
sec.dt = "18:03"
sec.ap = "Utrecht CS"
sec.at = "19:00"
sec.co = "Sneltrein"
sec.di = ["Arnhem", "Den Haag CS"]
sec.pf = "1a"
```

Figure 5.4: Record corresponding to the first section of the journey in Figure 5.3.

in the knowledge source. In addition, it will sometimes mention calculated waiting times as well. The information is divided over a number of sentences, and repetition of placenames is preferred over the use of anaphoric expressions. However, complex placenames are reduced when unfocused; for instance *Amsterdam* instead of *Amsterdam Centraal*.

As mentioned earlier, text planning is based on a text schema that describes the hierarchical structure of the texts. In the original work on schema-based planning (McKeown 1985b), this structure consists of rhetorical predicates that are claimed to capture general, domain-independent textual relation. Without intending to criticise this approach, we found it hard to apply these predicates to our domains, and instead opted for text schemas stated in terms of domain-dependent concepts. With respect to PTD, it is assumed that a description of a *route* consists of the descriptions of one or more *sections*, and that each section is in turn described by one or more APS's. Taking into account that there are brief, normal and elaborate descriptions, we arrive at the following text schema.[9]

(4) **Text schema for travel descriptions in PTD**

$$\begin{aligned}
&TravelDescription \longrightarrow\\
&\quad BriefTravelDescription^{+};\\
&\quad NormalTravelDescription^{+};\\
&\quad ElaborateTravelDescription^{+}.
\end{aligned}$$

$$\begin{aligned}
&BriefTravelDescription \longrightarrow\\
&\quad BriefSectionDescription^{+}.
\end{aligned}$$

$$\begin{aligned}
&NormalTravelDescription \longrightarrow\\
&\quad NormalSectionDescription^{+}.
\end{aligned}$$

$$\begin{aligned}
&ElaborateTravelDescription \longrightarrow\\
&\quad ElaborateSectionDescription^{+}.
\end{aligned}$$

[9] The superscript + in the regular expression means 'one or more times'.

$$BriefSectionDescription \longrightarrow$$
$$APS^+.$$

$$NormalSectionDescription \longrightarrow$$
$$APS^+.$$

$$ElaborateSectionDescription \longrightarrow$$
$$APS^+.$$

For each type of *SectionDescription*, there are several sets of APS templates available in order to promote variation in the output. The process of text planning is basically a matter of randomly choosing one of the appropriate sets, and substituting the travel parameters from the record in the slots of the template. This action is then repeated for each section of the route. The result is a list of APS's to be passed on to the syntactic realizer.

As an example, Figure 5.5 shows a text schema consisting of three templates implemented as an Icon procedure called `NormalDescription1`. It is called with three parameters: the record with the section's travel parameters (`sec`), the number of this section (`num`), and the total number of sections the route consists of (`tot`). It returns three APS's in the form of a list (`[...]`) of three strings (`"..."`). The templates themselves are simply concatenations of strings: everything between double quotes is a string, and `||` is the concatenation operator. A hash (`#`) indicates a comment, which is ignored by Icon.

The result of the call `Normal1(sec,1,2)`, where `sec` is the record in Figure 5.4, is the list of strings shown in Figure 5.6.[10] The lists resulting from planning subsequent sections are appended to obtain the full text specification. Its ultimate realization as a text was given in (2).

Actually, the travel parameters are not substituted directly, but are formatted by specialised functions first. These functions not only translate the knowledge source format to APS format, but also add significant pieces of content. For instance, the call `Place("Nijmegen","back")` in the first template returns the string `"(Nijmegen<back, prop>)"`, which specifies an NP with a backgrounded (unfocused) proper noun. Similarly, the call `Platform("1a")` returns `"(NUM:(1)(LET:a)(perron))"`.

The function `SectionDiscourseMarker` randomly generates an appropriate discourse marker on the basis of the number of the section and the total number of sections. With respect to Figure 5.6, the call `SectionDiscourse-Marker(1,2)` resulted in the adverb *eerst* 'first'.

The actual implementation of most schemas and templates is slightly more complicated. First, some checks are required to make sure that the application of a particular schema is allowed in the current context, which may be tested by means of preconditions. For instance, the template in Figure 5.5 can only be used if both the direction and platform number are available in the knowledge

[10] The layout is for convenience only. The actual output has no white space.

```
procedure NormalDescription1(sec,num,tot)
    return [

        # APS-TEMPLATE-1:
        # eerst/vervolgens neemt u om <DT> de <CO> richting <DI>
        # first/next take you at <DT> the <CO> towards <DI>
        "(V:nemen" ||
            "[<listen>]" ||
            "(P:om" || Time(sec.dt) || ")" ||
            "[" || Conveyance(sec.co) ||
                "(P:(N:richting<nonref,back>(P:" ||
                Direction(sec.di) || ")))]"
            SectionDiscourseMarker(num,tot) || ")",

        # APS-TEMPLATE-2:
        # die vertrekt in <dp> van <PF>
        # which leaves in <dp> from <PF>
        "(V:vertrekken" ||
            "[<spat>]" ||
            "(P:in(" || Place(sec.dp,"back") || "))" ||
            "(P:van" || Platform(sec.pf) || "))",

        # APS-TEMPLATE-3:
        # dan komt u in <AP> om <AT> aan
        # then arrive you in <AP> at <AT> +part
        "(V:aankomen" ||
            "[<listen>]" ||
            "[P:in(" || Place(sec.ap) || ")]" ||
            "(P:om" || Time(sec.at) || ")" ||
            "(ADV:dan<top,back>))"
        ]
end
```

Figure 5.5: A text schema for (a part of) a travel description implemented as an Icon procedure that returns a list of APS's.

```
[   "(V:nemen
        [<listen>]
        (P:om
            (TIME:(1)(8)(0)(3)))
        [sneltrein<def>
            (P:
                (N:richting<nonref,back>
                    (P:
                        (N:{(Arnhem<prop>)
                            (Den_Haag<prop>)})))))]
        (ADV:eerst<top,back>))",

    "(V:vertrekken
        [<spat>]
        (P:in
             (Nijmegen<prop,back>))
        (P:van
            (NUM:(1)(LET:a)(perron))))",

    "(V:aankomen
        [<listen>]
        [P:in
            (Utrecht<prop>)]
        (P:om
            (TIME:(1)(9)(0)(0)))
        (ADV:dan<top,back>))"
    ]
```

Figure 5.6: List of APS's that forms the the first part of the text specification for the travel description in (2). It is the result of applying the record in Figure 5.4 to the text schema in Figure 5.5.

source. Another example is a template using the verb *overstappen* 'transfer', which is appropriate only for non-initial sections. Second, some templates contain random choices to either include or omit pieces of information.

Notice that most of the tasks that belong to document planning and microplanning according to the standard architecture are not separately implemented in PTDPlanR. Instead, they are compiled in the schemas and templates. For example, the use of a pronoun is not the result of applying a referring expression algorithm, but is hard-wired in the template. Although this straight-forward, template-based approach may have its drawbacks (Reiter 1995), in our case there were no particular benefits to be obtained from using a dynamic approach.

5.3.5 Text planning in WFD

The goal of text planning in WFD is to deliver text specifications for the generation of flower descriptions similar to those found in popular botanical guidebooks. If we were to follow the standard architecture, this would require a text planner that is far from trivial. It has to start from a communicative goal supplied by the user, something like DESCRIBE(FLOWER X). Its knowledge source must be a large database about wild flowers. It has to select the relevant information, and possibly infer additional information. Next, it has to organise this information into coherent units, and present them in an order that makes sense. It might also need a user model to tailor the text to a layman or an expert, and a discourse history to make cross-references to previously generated flower descriptions. With respect to microplanning, it should be able to perform complex forms of lexical choice and aggregation.

Clearly, development of such a state-of-the-art text planner is beyond the the scope of the current work. Still, our goal is generating non-trivial text specifications to test CONPAS. The way we obtain them is by simulating the output of a text planner. For this purpose, we use a method of progressive expansion of APS templates. The approach is largely similar to that in PTD, with the difference that both the communicative goal and the knowledge source are hard-wired in the text planner.

Figure 5.7 shows a text schema for a flower description implemented as an Icon procedure. It returns a text specification as a list of three APS's (of which only the first APS is shown here). Each APS is responsible for describing one or more aspects of the plant such as:

- name;
- parts such as the flower, the leaves, and the stalk;
- geographical location, i.e. in which (parts of) countries it occurs (or does not occur);
- geological location, e.g, in which type of landscape it grows, on what type of soil, and with what density;

```
procedure Schema2()
    return [

        # APS-TEMPLATE-2a:
        # plant name, properties of stalk,
        # and properties of leave
        "(V:zijn" ||
            Plantname() ||
            "[plant" ||
                 "(" || PlantProperties() || ")
                 "(P:met" ||
                     "({" ||
                         "(stengel<plur>" ||
                              "(" || StalkProperties() || "))" ||
                         "(blad<plur>" ||
                             "(" LeaveProperties() || "))}))])",

        # APS-TEMPLATE-2b:
        # flower properties and flower relations
        ...

        # APS-TEMPLATE-2c:
        # geographical and geological location of occurrence
        ...

        ]
end
```

Figure 5.7: A text schema for a flower description implemented as a Icon procedure that returns a list of APS's (... indicate code that is not shown here).

- remarks, e.g., that it is useful for a certain purpose, or that it is similar to another flower.

The schema encodes the upper structure of the APS's, but relies on external procedures to fill in the details. For instance, APS template 2a in Figure 5.7 contains a call to the procedure `StalkProperties` in Figure 5.8, which generates an APS describing a flower's stalk. This procedure returns a randomly chosen element from a list of APS's that ranges from simple to complex ones.[11] The schemas were extracted from the flower descriptions in Forey (1992), and cover the most frequently observed patterns of presentation.

Procedures like `StalkProperties` may contain subsequent calls to other

[11] The question mark before the list, i.e. `?[ ...]`, is a unary Icon operator that produces a random element from the list.

```
procedure StalkProperties()
    return ? [

         # 1 property

            # 1a:
            # lang
            # long
            "A:lang",

            ...

        # 2 properties

            # 2a:
            # rechtopstaand, tot 40 centimeter hoog
            # erect, up to 10 centimeters high
            "A:{(V:rechtopstaan<part>)(A:hoog(P:tot
             (centimeter<nonref>(NUM:(4)(0)))))}",

            ...

        # 3 properties

            # 3a:
            # kort, dik , donkerbruin
            # short, thick, darkbrown
            "A:{(A:kort)(A:dik)(A:donkerbruin)}"

            ...
        ]
end
```

Figure 5.8: An Icon procedure that randomly generates an APS for describing a flower's stalk. It is invoked by the text schema in Figure 5.7.

```
[   "(V:zijn
        [aardoorn<prop>]
        [plant
            (A:eenjarig)
            (P:met
                ({
                    (stengel<plur>
                        (A:{
                            (A:kort)
                            (A:dik)
                            (A:donkerbruin)}))
                    (blad<plur>
                        (A:{
                            (A:groot)
                            (A:donkergekleurd)
                            (A:vlezig)}))}))])",

    ... ]
```

Figure 5.9: First part of the list of APS's that forms the text specification for the flower description in (1). It is the result of expanding the text schema in Figure 5.7.

ones. Of course, the more descriptions are chopped up into randomly chosen parts, the more varied they become. However, not all variation is interesting for our purpose. Most variation in lexical choice (e.g. with respect to a plant's name) is of limited interest with respect to prosodic realization, whereas variation in the syntactic structure and the length of phrases is far more interesting. Furthermore, concatenating smaller and smaller APS's increases the risk of generating nonsensical content. Even at the current level of granularity, the content of procedures must sometimes be chosen carefully. For instance, a geological and geographical location can be mutually exclusive: there is no way a flower is going to occur along the Swiss coast.

Figure 5.9 shows the first part of the list of APS's that forms the text specification for the flower description in (1). It is the result of expanding the text schema in Figure 5.7.

5.4 Syntactic realization

The APS's generated by the text planner have to be expanded to full-blown syntactic structures. This process of syntactic realization takes place in SynR, and will be the topic of the next chapter. Therefore, we only summarise the process here by means of an example. Figure 5.10 and Figure 5.11 show the syntactic tree corresponding to the first APS in Figure 5.6. The contribution of SynR covers the following areas:

- **Insertion of function words**
  Most function words do not possess an immediate semantic content of their own, but rather serve a grammatical function that depends on the context. SynR inserts all determiners, auxiliaries, pronouns[12], and pronominal adverbs (e.g. *daarvoor* 'to that end'). In addition, it inserts complementizers, prepositions and conjunctions not specified in an APS. Examples are the personal pronoun *u* 'you', the determiner *de* 'the' Figure 5.10, and the relative pronoun *die* 'that', and the conjunction *en* 'and' Figure 5.11.

- **Inflection of content words**
  Content words are usually supplied in the APS, but their morphological form depends on the context. For instance, *nemen* 'take' gets the form *neemt*, i.e. finite, singular, third person, and present tense. Likewise, nouns can be inflected as plural or possessive, and adjectives may receive the inflection suffix *-e*.

- **Distribution of focus**
  Unless something is explicitly focused or backgrounded by PlanR, SynR applies default rules to determine the focus distribution. For instance, it focuses the PP *om achttien uur drie*, but denies focus to the pronoun *u*. The focus distribution is instrumental in word ordering and pitch accent placement.

- **Expansion of numerals, times, dates, and abbreviations**
  An example of expansion the mapping of `(TIME:(1)(8)(0)(3)))` to the syntactic structure for *achttien uur drie*. Notice that whereas this type of expansion may be optional in NLG, it is obligatory in SLG.

- **Determination of word order**
  The word order is highly restricted by the syntax. For instance, the finite verb *neemt* is fronted in the main clause, whereas the finite verb *gaat* remains in final position in the embedded relative clause. Within the limits of syntax, the word order is also affected by the information structure, which includes the focus distribution, and by the weight or length of constituents.

[12] It was argued earlier that pronouns are actually determiners (Section 2.6.3).

The above tasks presuppose additional syntactic processing: syntactic roles like subject and object have to be determined, grammatical case must be assigned, and tense, mood, and voice have to be accounted for. In addition, mechanisms for feature percolation have to enforce subject-verb agreement and other syntactic dependencies. It will be clear that syntactic realization is a non-trivial task that involves much more than a few string manipulations: it requires the implementation of a fair amount of syntactic knowledge of the target language. Therefore, SynR is implemented as syntactic *grammar* that takes APS strings as input and produces labelled bracket strings as output.

Since CONPAS outputs speech rather than written text, it may seem superfluous to generate graphemic words. Nevertheless, it generates the graphemic forms first and leaves it to the speech synthesiser to convert these to their phonemic form. The alternative is more economical, and perhaps also more attractive from a theoretical point of view. However, the speech synthesiser has a large grapheme-to-phoneme lexicon at its disposal, as well as algorithms for deriving lower-level prosodic constituents (Section 5.6). Given the focus of the current project on the prosodic rather than the segmental level, we chose to take advantage of these existing resources. In addition, graphemic output allows SynR to be reused in future linguistically-based NLG systems.[13]

[13] Currently, SynR has a provisional implementation of structure realization. It can capitalise the first letter of sentence, as well as that of proper nouns. It can also insert commas and periods at the trivial locations.

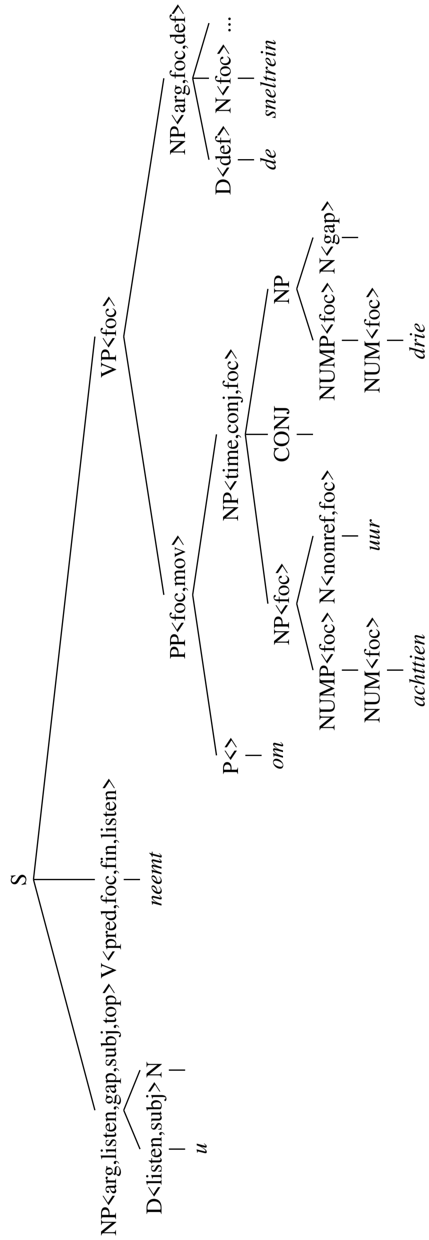

Figure 5.10: First part of the syntactic tree generated by SynR *corresponding to the first APS in Figure 5.6; the second part – shown in the next Figure – is attached at the three dots (...); see (2-a) for a gloss.*

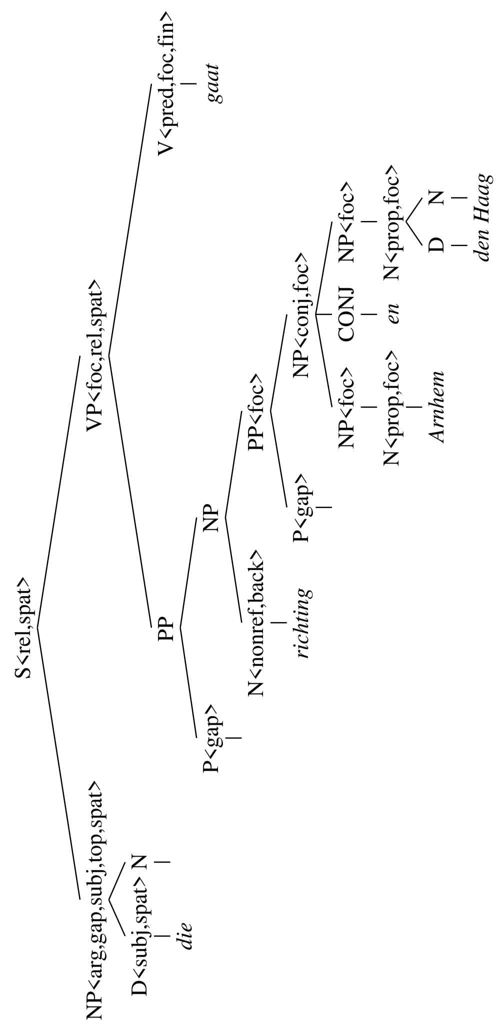

Figure 5.11: Second part of the syntactic tree generated by SynR *corresponding to the APS in Figure 5.6.*

5.5 Prosodic realization

The output of SynR is a list of utterances with their syntactic tree and focus distribution. On the basis of this information, ProsR has to generate an adequate prosodic form.

Recall that according to prosodic phonology, the prosodic structure of an utterance is a tree consisting of prosodic constituents (Section 2.2). The first part of prosodic realization in ProsR involves the derivation of the higher-level prosodic constituents, i.e. U, π, and φ. The prosodic constituents derived by ProsR for the tree in Figure 5.10 and Figure 5.11 is shown in Figure 5.12.

The second part of prosodic realization is generating the intonational form. As explained earlier (Section 2.4), this involves:

- pitch accent placement
- intonational phrasing (or the derivation of association domains)
- tune choice

The intonational form that ProsR generates for the tree in Figure 5.10 and Figure 5.11 is shown in Figure 5.13. Notice that there are two AD's, of which the second one has a Downstep tone morpheme (!).

The output of ProsR is a list of word strings interspersed with mark-up codes for pitch accents, AD-boundaries, tone attributes, and prosodic constituents. This phonological description is then translated into a phonetic description and a speech signal by SpeechS. Prosodic realization will be fully described in Chapter 7.

5.6 Speech synthesis

The final step in the generation process is speech synthesis in the module SpeechS. For this purpose, CONPAS incorporates the KUNTTS system developed at the University of Nijmegen (Kerkhoff and Rietveld 1994; Kerkhoff, Rietveld, and van Bergem 1998). Since a number of the tasks of a TTS system are superfluous when it is embedded in an SLG system, a stripped version of the KUNTTS system was used from which the modules for text preprocessing, tagging of word classes, parsing of the syntactic structure, deriving phonological and intonational phrases, assigning pitch accents, and choosing a tune were removed. The remaining tasks can be divided into three areas: phonology, phonetics, and actual synthesis.

5.6.1 Phonology

The process of mapping the graphemic words in the output of ProsR to their phonemic form is known as *grapheme-to-phoneme conversion*.[14] This is accom-

[14] KUNTTS uses combinations of ASCII characters to represent phonemes, similar to CPA (Lennig and Brassard 1984) and ESPRIT's SAMPA (ESPRIT 1992). See Kerkhoff, Rietveld,

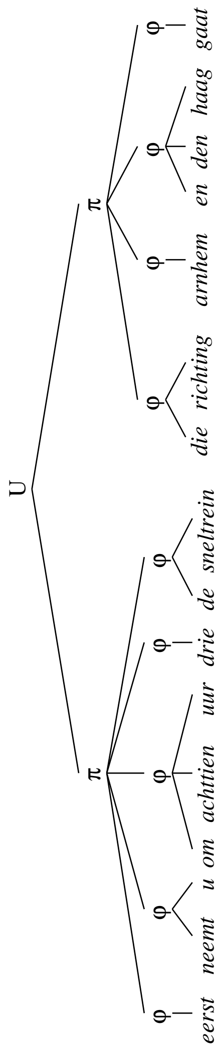

Figure 5.12: Higher-level prosodic constituents generated by ProsR *for the tree in Figure 5.10 and Figure 5.11; see (2-a) for a gloss.*

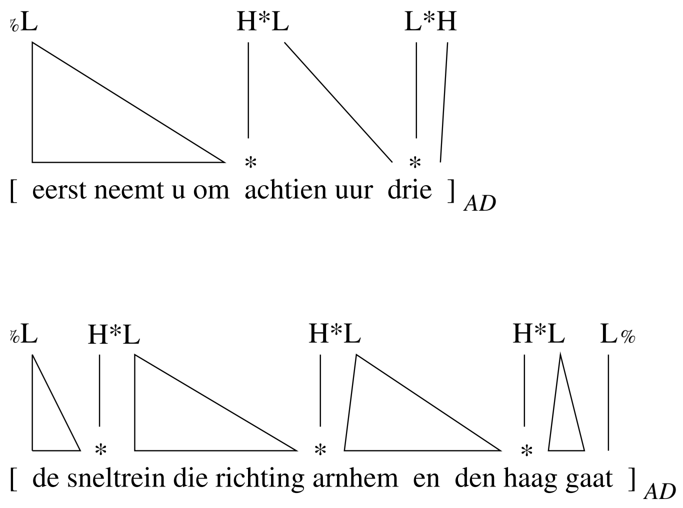

Figure 5.13: Intonational form generated by ProsR *for the tree in Figure 5.10 and Figure 5.11; see (2-a) for a gloss.*

plished by looking up the words in a lexicon (Gulikers and Willemse 1992). For instance, (5) shows the phonemic strings corresponding to the words *achttien* 'eighteen' and sneltrein 'fast-train'.

(5) a. `achttien` $\longrightarrow$ `'Axt-tin`
    b. `sneltrein` $\longrightarrow$ `'snEl-+->trEIn`

Notice that the phonemic form retrieved from the lexicon does not only contain phonemes, but also information about the prosodic and morphological structure of the word: syllable boundaries are indicated by a single dash (`-`), compound boundaries by a dash-plus-dash sequence (`-+-`), the primary stressed syllable is preceded by a single right quote (`'`), and the secondary stressed syllable by greater-than sign (`>`). Thus, a word's syllabic structure and stress pattern are encoded in the lexicon.[15]

The information about stress, syllabification and compounding enables the derivation of the prosodic constituents below the φ: Prosodic Words (ω), Feet (Σ), and Syllables (σ) (cf. Table 2.1). The derivation of ω's is provisional: it

and van Bergem (1998, Appendix IV) for a translation table.

[15] KUNTTS allows for a user-defined lexicon, which overrules the main lexicon. Domain-dependent lexicons were defined for PTD and WFD to add words without affecting the main lexicon. When a word is not found in the lexicon, KUNTTS falls back on linguistic rules for grapheme-to-phoneme conversion and stress assignment. However, since all words generated in PTD and WFD are included in the domain-dependent lexicons, these rules are never used in our SLG systems.

is assumed that the ω's are identical to the graphemic words. Although this is a reasonable approximation, the relation between the morphological structure and prosodic structure of a word is often non-isomorphic – see e.g. Booij (1995, p.47). The implementation of ω-derivation in KUNTTS is the topic of current research.

In contrast, KUNTTS contains well-founded algorithm for Σ-derivation by Kerkhoff, Rietveld, and van Bergem (1998), based on the phonological description in Gussenhoven (1993), and supported by experimental evidence from Rietveld, Kerkhoff, and Gussenhoven (1999, to appear). A ω consists of one or more Σ's. The relative prominence of the Σ's is indicated by a *strong-weak* labelling. The Σ that contains the stressed syllable is marked as *strong* (Σ_s), and all others as *weak* (Σ_w). Each Σ consists in turn of one or more σ's, of which the first one is *strong* (σ_s) and all others are *weak* (σ_w).[16]

At this stage, a complete prosodic constituent structure of the utterance is available: the lower constituents – σ, Σ, and ω – are produced by KUNTTS, while the upper ones – U (Utterance), φ (Phonological Phrase), and π (Intonational Phrase) – are provided by ProsR. Figure 5.14 shows the prosodic structure of corresponding to the tree in Figure 5.10 and Figure 5.11.

Next, the phonemic string – the *underlying form* in the phonological sense – is modified by postlexical segmental rules that apply within – or at the juncture of – prosodic constituents of a certain rank.[17] As a matter of fact, these rules constitute a substantial part of the evidence for existence of prosodic constituents (cf. Section 2.2). For instance, the rule *Regressive Voice Assimilation* causes voiceless obstruents to become voiced before a voiced stop (Booij 1995). In (6), it changes the final /f/ of *elf* into /v/.

(6) [$_\pi$ die Amsterdam om v*eertien uur *elf bereikt]
that Amsterdam at fourteen hour eleven reaches
'which reaches Amsterdam at 14.11 hours'

However, the rule cannot apply across π-boundaries. In (7), where the non-restrictive relative clause forms a separate π[18], the final /f/ of *elf* is unaffected, even though it is followed by the voiced stop /d/.

[16] As a matter of fact, some words have an exceptional Σ-structure. In *perron* /pE-rOn/ 'platform', for instance, the first syllable is not dominated by a Σ, but dominated directly by a ω. The second syllable forms a monosyllabic Σ. This violation of the Strict Layer Hypothesis is assumed to be necessary for prosodic constituents below the φ (Booij 1995, p.144-145). See also (Gussenhoven 1993).

[17] In TTS, *lexical* rules, like *Final Devoicing*, apply to words that have been derived by grapheme-to-phoneme conversion. However, since we have chosen to include all words used in PTD and WFD in the lexicon, lexical rules are never applied.

[18] The derivation of π's will be described in Section 7.2.2.

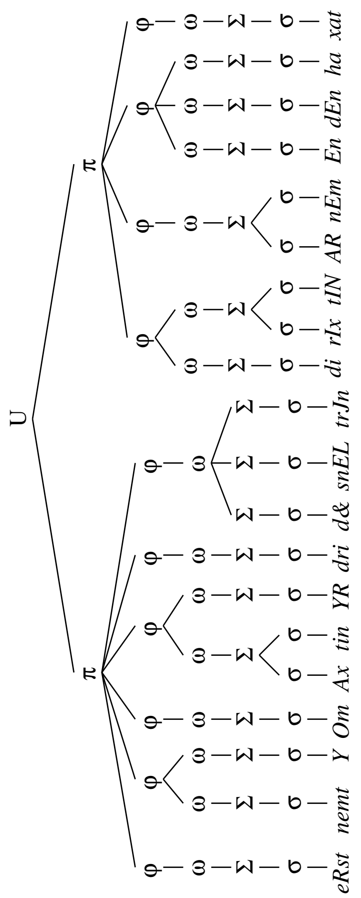

Figure 5.14: Prosodic structure derived by ProsR *and* SpeechS *corresponding to the tree in Figure 5.10 and Figure 5.11; see (2-a) for a gloss.*

(7) [$_\pi$ de trein van veertien uur elf] [$_\pi$ die naar Amsterdam
the train of fourteen hour eleven which to Amsterdam
gaat]
goes
'the 14.11 train, which goes to Amsterdam'

Although some interesting results for Dutch are available (Loots 1983; Menert 1994; Ernestus 2000), there is probably much more to be learned about the relation between segmental rules, the prosodic constituent structure, and performance factors like speaking rate and speech style. The current framework with KUNTTS embedded in CONPAS is well-suited for implementing, testing and improving our knowledge concerning this topic. Our informal experience so far indicates that the obtained agreement between the prosodic and the segmental structure of an utterance contributes to the perceived naturalness of its synthetic rendering.

5.6.2 Phonetics

The phonological structure consisting of a phoneme string, a tone string, and a prosodic constituent structure is passed on to the phonetic implementation module, which computes the phonetic details. Two dimensions of the phonetic structure can be distinguished: duration and intonation.

5.6.2.1 Duration

Creation of the durational structure involves two tasks. First, each phoneme must be assigned a duration in milliseconds. Second, pauses of various length must be inserted at appropriate positions.

Initially, vowels (with the exception of diphthongs) are assigned the same duration. No distinction is made between so-called 'long', 'half-long', and 'short' vowels. This basic duration corresponds to the mean duration found in vowels in the weakest prosodic position, that is, a weak syllable in a non-peripheral weak foot (Rietveld, Kerkhoff, and Gussenhoven 1999). The consonants are assigned specific, intrinsic values. Next, segments are lengthened according to their prosodic position in terms of headedness of foot and syllable, and vowel type. High lax vowels (/i, y, u/) are lengthened less than the other vowels. Finally, phoneme durations are modified by context-sensitive rules. The prosodic structure plays an important role here, as the right-side boundaries of prosodic constituents are subject to *preboundary lengthening*, i.e. the duration of the phonemes preceding the boundary is increased. It is assumed that the amount of lengthening is proportional to the rank of the prosodic constituent: the higher the rank, the more lengthening occurs (Hofhuis, Gussenhoven, and Rietveld 1995; Gussenhoven and Rietveld 1992a; Cambier-Langeveld 2000). In a similar way, strong Σ's and strong σ's are lengthened (Rietveld, Kerkhoff, and Gussenhoven 1999). Which phonemes are involved and how much they are lengthened depends on the phonemic context. For instance, the vowel in a strong σ that is the head of strong Σ is lengthened by 20%. These rules and their interactions form a delicate system for shaping the rhythmic structure of an utterance; for a more elaborate description, see Kerkhoff, Rietveld, and van Bergem (1998). The nett result is that the hierarchical and strong-weak relations in the prosodic structure of an utterance become mirrored in its durational structure.

Besides determining the length of phonemes, creating the durational structure also involves the insertion of pauses at appropriate positions. In analogy to preboundary lengthening, prosodic phonology predicts that the chance of pause insertion, as well as the length of the pause, is proportional to the rank of the prosodic constituent. In SpeechS, a φ-boundary triggers a small pause (by default 15ms), whereas a π-boundary triggers a larger pause (by default 50ms).

As might be expected, the durational structure is related to the intonational structure. Accented syllables receive additional lengthening. Similarly, the right-side boundaries of AD's force additional preboundary lengthening (Gussenhoven and Rietveld 1992a).

5.6.2.2 Intonation

The second aspect of the phonetic implementation concerns the details of the intonational structure. Recall that the phonological description of the intonation is a tone string that consists of pitch accents (H*L, L*H and H*LH), initial boundary tones (H% and L%), final boundary tones (%H and %L), and tone morphemes (e.g. ! for Downstep). Every starred tone is associated with a stressed syllable, and every boundary tone with an AD-boundary. This phonological description has to be translated to a sequence of F_0 targets describing the corresponding pitch contour. An F_0 target is a pitch value (in Hz) at a point in time (in ms). Determining its point in time is called *timing* (or *alignment*); determining its pitch value is called *scaling*. The pitch contour is obtained by interpolation through the targets. This *target-interpolation model* is described more extensively in van den Berg, Gussenhoven, and Rietveld (1992), Gussenhoven and Rietveld (1992b), and Kerkhoff and Rietveld (1995).

Timing Each H or L tone is phonetically realized by at least one F_0 target. We will take the tone string %L H*L H% as an example. A graphic representation of the timing of its targets is given in Figure 5.15. The appropriate domain for timing these targets is the AD. The timing of initial (t1) and final (t6) boundary tones is provided by, respectively, the (start of the) first and (end of the) last voiced phoneme of the AD. The starred tone of a pitch accent (t3) is timed at STARTIME. By default, it is timed at 30% of the duration of the vowel in the accented syllable, counting from the start of the vowel. However, the STARTIME is shifted as a function of the segmental content of a syllable (Rietveld and Gussenhoven 1995).

The remaining tones are timed relative to these reference points. The variables TOTIME and FROMTIME are used to locate them to the left or the

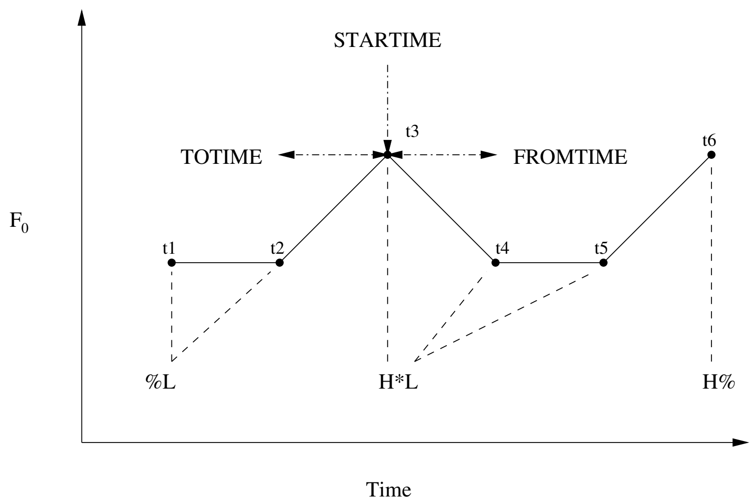

*Figure 5.15: Graphic representation of target timing for the tone string %L H*L H%. The initial (t1) and final (t6) boundary tones are timed at respectively the start and end of the AD. The starred tone of a pitch accent (t3) is timed at STARTIME, which is expressed as a percentage of the duration of the vowel in the accented syllable. The second target of the (spreading) initial boundary tone (t2) is located at TOTIME before the starred tone. The first target of the (spreading) non-starred tone (t4) is located at FROMTIME after the starred tone, while its second target (t5) is located at TOTIME before the final boundary tone.*

right of a reference point, respectively. Some tones control the pitch of longer stretches of speech, and yield two targets. This is called *spreading* of tones. An initial boundary tone always spreads. Its second target (t2) is located at TOTIME before the first starred tone. The second tone of an AD-final pitch accent spreads as well. Its first target (t4) is located at FROMTIME after the starred tone, while its second target (t5) is located at TOTIME before the final boundary tone.[19]

The time required by the variables FROMTIME and TOTIME is sometimes more than the time actually available in the phoneme string. To accommodate for such cases, overlap functions are used to merge targets.

Scaling The scaling of targets is based on the model described in Ladd (1987), Ladd (1990), and van den Berg, Gussenhoven, and Rietveld (1992). At its core is a scaling function $F(T, n, m)$ that yields a pitch value for the starred tone T (H* or L*) that is the nth accent in the mth AD. It has the following parameters:

1. *Frequency reference* (Fr) defines the bottom of the speaker's range. It represents speaker-dependent differences in range, e.g. between male and female speakers.
2. *Normalisation* (N) defines the degree of raising of the register. It models the overall emphasis or prominence of the pitch.
3. *Width* (W) defines the width of the register, i.e. the difference between the pitch value for L^* and H^* targets. It is related to the 'liveliness' of the pitch.
4. *da* (downstep accentual) defines the step size for lowering of H^* targets in an AD with accentual downstep.
5. *dp* (downstep phrasal) defines the step size for lowering of AD's in an SD with phrasal downstep.

We will take the downstepped tone string !%L H*L H*L H*L L% as an example. A graphic representation of the scaling of its targets is given in Figure 5.16. The target for the first starred tone (t1) is scaled at frequency f1, which equals the reference frequency (Fr) plus the frequency normalisation (N) plus half of the register width (W). The second starred tone (t3) is scaled at f2, which equals the accentual downstep factor (da) times the frequency of the preceding starred tone (f1). Similarly, the third starred tone (t5 at f3) is scaled slightly lower again than the previous one. The low targets (t2, t4, and t6) are unaffected by downstep.

Scaling of nonstarred tones and boundary tones depends on the context. The pitch value for an H is somewhat lower than that of an H^*, whereas the pitch value of an L is somewhat higher than that of an L^* (cf. t2, t4, and

[19] Some tone attributes trigger the spreading of starred tones as well (Gussenhoven 1988).

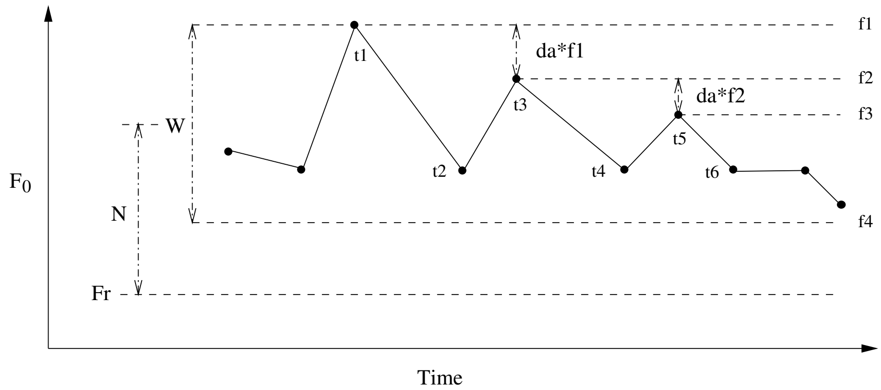

*Figure 5.16: Graphic representation of target scaling for the downstepped tone string !%L H*L H*L H*L L%. The target for the first starred tone (t1) is scaled at frequency f1, which equals the reference frequency (Fr) plus the frequency normalisation (N) plus half of the register width (W). The second starred tone (t3) is scaled at f2, which equals the accentual downstep factor (da) times the frequency of the preceding starred tone (f1). Similarly, the third starred tone (t5 at f3) is scaled slightly lower again than the previous one. The low targets (t2, t2, and t6) are unaffected by downstep.*

t6 in Figure 5.16); the difference is roughly one-third of the register width. The pitch value for a final boundary tone is obtained by multiplying the pitch value of the preceding tone by a factor less than one (for L%) or greater than one (for H%). Finally, utterance final H*L pitch accents are implemented with a certain amount of *final lowering* (Ladd 1984; Liberman and Pierrehumbert 1984; Pierrehumbert and Beckman 1988).

In addition to Accentual Downstep, there is another form of downstep that affects subsequent AD's, which is called *Phrasal Downstep.* It is similar to Accentual Downstep, but rather than lowering the next H*, the next AD is lowered, causing all the H* targets it contains to be lowered by same amount. This amount of phrasal downstep is controlled by the value of the parameter dp.

The pitch values of the targets depend on the values that are chosen for the scaling parameters mentioned above. It is assumed that these remain constant across a *scaling domain* (SD), which normally corresponds to a whole utterance. In principle, it is possible to vary the scaling parameters from one utterance to the next according to the global discourse structure. For instance, to obtain a gradual lowering of the average pitch level in a paragraph, or a temporary narrowing of the register in the case of a parenthetical remark. Although we experimented with this idea, it is has not been implemented in CONPAS, as it is beyond the scope of our research questions. We will further discuss this issue in Section 9.2.2.

5.6.3 Synthesis

With all phonetic details implemented, the only step left is the actual synthesis of speech. For this, SpeechS transfers the output of KUNTTS to the MBROLA speech synthesiser (Dutoit and Leich 1993; Dutoit, Pagel, Pierret, van der Vreken, and Bataille 1996). It makes use of *diphones*, units that start in the middle of a phoneme's steady state and end in the middle of the next one. Carefully selected diphones for all phoneme transitions occurring in the target language are stored in an efficiently coded database. We used the *nl2* diphone database developed by Arthur Dirksen.

The first step is to retrieve the diphones that match the phonemes in the input. Figure 5.17 shows the input format.[20] Each line starts with a phoneme symbol and its duration, optionally followed by one or more F_0 targets, which are specified in terms of a percentage of the phoneme's duration and an F_0 value in Hz.

Next, the pitch and duration of the diphones are modified to fit the parameters in the input. This is accomplished using a variant of the *pitch-synchronous overlap-add* (PSOLA) algorithm (Charpentier and Stella 1986). Finally, the diphones are concatenated to a speech signal, which can be played directly, or saved in some audio file format. As for any normal speech signal, we can

[20] These phonemes are slightly different from those used internally by KUNTTS, because they depend on the particular diphone database that is used.

```
_  20   0 109
I: 110  0 109
R  45
s  65
t  60
_  15
n  80
e  115
m  60
t  60
y  105
_  15
O  80   87 98
m  60
A  100  30 158
x  75
t  55
i  75
n  70
y  125
R  60
d  65
r  65   52 105  98 83
i  140  42 83  71 158
_  50   0 158
.
.
.
```

Figure 5.17: Input to the MBROLA synthesiser for the first part of utterance (2-a). The first and second column contain the phoneme labels and their duration in ms. The (optional) following columns specify F_0 targets as a percentage of the phoneme's duration and a frequency value in Hz. For instance, the 4th target occurs at 30% of the duration of the phoneme A, at 158 Hz.

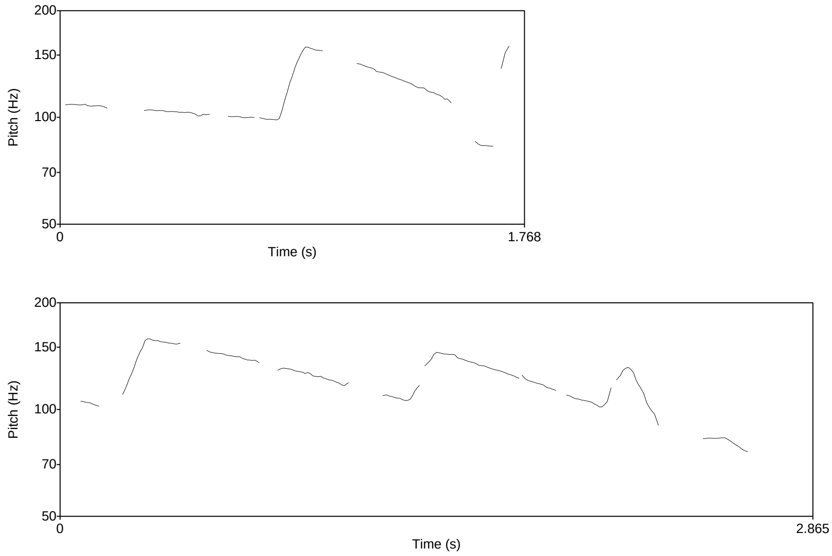

Figure 5.18: Pitch contour of the synthesised version of utterance (2-a).

extract the pitch contour with a signal processing program. Figure 5.18 shows the pitch contour for our example utterance.

5.7 Summary

This first part of this chapter addressed questions regarding the general architecture of an SLG system. We suggested to conceive an SLG systems as an extension of the standard three-module pipeline architecture for NLG systems. With regard to combining NLG and SS, it was argued that text is not a good interface. Instead, a phonological representation was proposed consisting of a sentence, its intonational structure (pitch accents, intonational boundaries, and tune), and its higher-level prosodic constituents. With respect to the proper place for controlling the intonation, it was argued that prosodic realization, which includes generating the intonational form, cannot be reduced to either microplanning or structure realization, but is a complex, inherently linguistic task that deserves an independent module.

We proposed to use a domain-independent, reusable content-to-speech system, analogous to the reusable surface realizers that are common in NLG. Accordingly, the architecture of our SLG systems has two major parts. First, a domain-specific text planner (PlanR) that generates the *content* as a text specification consisting of phrase specifications. Second, a domain-independent front-end called CONPAS ('CONtent to Prosodically Adequate Speech') that is responsible for the *form* of the spoken output. It translates the content to a speech signal, using knowledge about the morpho-syntactic, phonological, and phonetic properties of the target language. Internally, it has modules for syntactic realization (SynR), prosodic realization (ProsR), and speech synthesis (SpeechS).

Two application domains to explore prosody in SLG were introduced: the Public Transport Domain (PTD) and the Wild Flower Domain (WFD). An approach to text planning for these domains was described that is straightforward, schema-based, and directed towards generating maximal variation on the basis of limited input. We briefly introduced the abstract phrase specifications (APS's) that serves as the intermediary representation between PlanR and CONPAS, and argued that these representation have the desired level of abstraction. Next, domain-specific text planners for PTD and WFD were described. It was shown how text schemas and APS templates are implemented in Icon.

An overview of syntactic and prosodic realization was provided. The former involves the insertion of function words, the inflection of content words, the distribution of focus, the expansion of numerals, times dates, and abbreviations, and the determination of word order. The latter is concerned with pitch accent placement, intonational phrasing, tune choice, and derivation of higher-level prosodic constituents. Both forms of realization will be described extensively in the next two chapters.

Finally, speech synthesis was described in terms of three areas. The first area is phonology, which involves grapheme-to-phoneme conversion, the derivation of lower-level prosodic constituents, and the application of lexical and post-lexical segmental rules. The second area is phonetics, which deals with the durational and intonational details of the utterance. It was shown how the availability of a full prosodic constituent structure facilitates preboundary lengthening and pause insertion. It was also described how the phonological description of the intonation is translated into a sequence of F_0-targets for a pitch contour by a target-interpolation model that performs timing and scaling of targets. The third area is the actual synthesis, for which the MBROLA diphone speech synthesiser is used.