THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

A Scalable Natural Language Generator

Tim Hunter
Bachelor of Engineering (Software Engineering)

June 2004

Supervisor: A/Professor Bill Wilson
Assessor: Dr. Mengistu Amberber

Acknowledgements

I would like to thank my co-supervisors for the guidance they have provided throughout this project: Bill Wilson from the perspective of other similar work in computer science, and Mengistu Amberber from that of theoretical linguistics. In addition I would like to thank Ryo Mihara for his help with the Japanese language, and Michael Potter and Peter Newcombe for proof reading drafts of this report.

Contents

Chapter 1

Introduction

1.1 Overview

The aim of this project is to build a natural language generator which not only generates syntactically well-formed sentences in the user's desired language, but also is constructed in such a way that the number of available languages can be easily increased. Specifically, assuming the system is capable of generating sentences in any of n languages, the work required to add the ability to generate in an $(n + 1)$th language should be small compared to the work required to construct the system as a whole.

To achieve this goal, the linguistic 'Principles and Parameters' (P&P) theory will be used to minimise the amount of language-specific information that is required. This theory proposes that knowledge of a particular human language consists of the settings of a finite number of formal, well-defined parameters, which constitute a small amount of information relative to the innate, universal principles shared by all human languages. (It should be noted that P&P is a theory of syntax, and thus 'knowledge of a language' in this context refers not to the knowledge of a language's vocabulary, but rather to the ability to combine known lexical items (words) to form meaningful phrases and sentences.) This theory is an ideal starting point for the design of a natural language generation system which will fulfil standard goals of software engineering, such as easy scalability and extensibility, and a minimum of code duplication; and fulfil these goals to an extent that would perhaps not otherwise seem possible.

The user's input will use the linguistic notion of semantic roles to specify the meaning of the sentence which is to be produced. Semantic roles describe the relationships between the entities participating in a sentence in a purely semantic, language-independent manner.

As well as obvious practical applications for natural language generation and as a step towards a full machine translation system, the attempt to implement such a system serves as a test of the parametric view of linguistic variation which has been proposed by theoretical linguistics.

1.2 Goals

1.2.1 Software Framework for Further Use

What it was hoped would be produced, from a software perspective, was not so much a program with certain functionality as a framework of source code, which allows programmers to easily describe the process of syntactic realisation for arbitrary human languages. To refine and demonstrate this framework, implementations of English, French and Japanese have been developed. The generation of correct sentences in these particular languages is not a primary goal in itself, but obviously is an important indicator of the viability and usefulness of the underlying framework.

1.2.2 A Test of the P&P Theory

If the parametric view of language variation is correct, then it follows logically that the scalable system that is being attempted here should be possible. Linguists have proposed a clear separation of the language-universal principles from the language-specific parameter settings, where the former is relatively large and the latter relatively small. The obvious deterministic nature of software makes this an ideal means of testing whether the formalisms which have been developed are really sufficient to account for what they claim to. If, in a software system, all the information needed for the generation of natural language sentences can be represented in the modular way suggested by P&P, this would provide further evidence for the P&P model as an answer to some of the questions surrounding language acquisition (see §2.2.1).

While the ability of such parameters as have been proposed by linguists to predict the characteristics of individual languages is impressive, it is unlikely that the syntax of human languages can be *completely* described by such formalisms. The system had to allow for the output of the parametric theory to be considered a first approximation, and thus for *some* genuinely language-specific modification, although clearly these modifications should be kept to a minimum.

1.3 Scope

1.3.1 The Lexicon

1. Only a partial lexicon was to be provided for each available language. A sample has been chosen for each language which covers as wide a range as possible of lexical phenomena. The structure of the system is such that adding new lexical items will not require changes to the syntactic realisation algorithms which are the system's main concern.

2. Whilst the syntactic information required to generate sentences in a given language consists of a small language-specific component and a large language-universal component, all lexical information was to be completely language-specific. If full lexicons were to provided for each language, the amount of lexical information in the system would thus be expected to grow linearly with the number of available languages, and to be very large relative to the syntactic information. Linguists such as de Saussure have

noted that the association between lexical items and meaning is arbitrary [27]; it is therefore unreasonable to expect any universal principles to predict lexical information. (A common objection to this claim is that onomatopoeic words have a logical connection with their meanings, but there is still some arbitrary variation across languages: for example, where English uses *woof*, French uses *ouaoua* and German *wauwau*. Thus even onomatopoeic words are not completely predictable.)

3. It was assumed that each language's lexicon provides a mapping from abstract concepts to lexical items. These abstract concepts are represented in small capitals throughout this report; for example WALK refers to the language-universal concept of walking, not to the English lexical item 'walk' (nor the lexical item any other language uses to represent this concept). See §5.2.2 for more details of this mapping.

4. Speed and memory efficiency in storing and searching lexicons were not of concern.

1.3.2 The Output

1. Given a semantic representation of the desired meaning, generating any one syntactically correct sentence in the chosen language which corresponds to that meaning is sufficient. If more than one possible sentence exists in the target language which expresses the input semantics, any one of them is acceptable.

2. It was more important to cover a range of languages than to cover a wide range of sentence types; because of the relatively short timespan, only a subset of sentence types is thus available, to ensure that the system's extensibility to a number of languages has been tested.

3. The output was to be a textual representation of the generated sentence. The phonological realisation of the sentence is of no significance.

1.4 Structure of this Report

Chapter 2 gives an overview of other related research and development which has been done, both applied linguistic work in computing and pure theoretical linguistic work.

Chapter 3 introduces the linguistic theory which the project uses, with a particular emphasis on concepts which provide a language-universal perspective. Chapters 4 and 5 describe how these concepts were implemented in the design and development of the program.

Chapter 6 evaluates the final program with respect to the major goals, as described in §1.2.

Note that Appendix A provides a brief glossary of some linguistic terms used throughout the report.

Chapter 2

Relation to Other Work

2.1 Computing Work

2.1.1 Overview

This project varies significantly from much of what has already been done in computational linguistics. It was originally conceived only from independent knowledge of theoretical linguistics and of computer science in general, without much familiarity with established methodologies in machine translation or natural language generation, and thus adopts what seems to be a relatively 'fresh' approach to how computers might best deal with natural language.

Steven Abney [1] has written of a widening 'gulf' between computational and theoretical linguists, due to fundamental differences in assumptions and goals: 'A large part of computational linguistics focuses on practical applications, and is little concerned with human language processing', while theoretical linguists argue that 'even if [computational techniques] make you oodles of money on speech recognizers and machine-translation programs ... they fail to advance our understanding.' This project however, though clearly computational in nature, is still based very firmly in theoretical linguistics, and in particular some of its proposals as to how knowledge of natural language might be represented in the human mind. It is a direct attempt to implement the P&P theory in a computer program, rather than an attempt to have a computer perform some particular manipulation of natural language using whatever techniques will work, appealing to linguistic theory when it is deemed to suit.

Nonetheless, implementation of the P&P theory is attempted in the hope that this *will* allow some useful manipulation of natural language (specifically, syntactic realisation), so some comparison with existing techniques is appropriate. The following subsections summarise these techniques, and describe how my project relates to them, with particular emphasis on multilinguality.

2.1.2 Natural Language Generation (NLG)

Stages of NLG

The complete process of NLG is commonly considered to comprise three stages: content planning (or discourse planning), sentence planning and syntactic real-

isation.

Content planning involves deciding what needs to be said by the text to be generated. From an existing knowledge base, which is not in a linguistic format, it must be decided what is important to include and what can be left out. The basis for this decision varies; in the case of 'question and answer' systems, for example, the query derived from the question posed by the user is the criterion for selecting what information is included in the output.

In the sentence planning stage, the information which was selected during the content planning phase is broken down into pieces, each of which will correspond to a sentence in the final output. Syntactic realisation is the final stage, where a natural language sentence is generated for each of the sentence-sized pieces which were the output of the previous stage.

It seems reasonable that the work to be done in the content planning and sentence planning stages is very nearly, or even entirely, language universal. It is not as obvious, though, how to design the syntactic realisation module in a manner that is easily adaptable to multiple languages, and the majority of NLG systems use totally language-specific syntactic realisation techniques. Clearly at least some of this stage must be language-specific, and this is likely to include all lexical information, but there is evidence from linguistic theory that there is a significant language-universal component to syntactic realisation.

My project involves only the syntactic realisation stage of NLG, because the input determines where sentences boundaries should be, but it attempts to achieve as much as possible of this task in a language-universal manner.

'True NLG' vs. Template Systems

Reiter [26] draws a distinction between text generation systems which work purely at the level of character strings ('template systems') and those which use a deeper representation of text ('true NLG').

Template systems simply concatenate predefined strings according to specific patterns, and use virtually no linguistic knowledge. Reiter gives as an example the Apple Macintosh Balloon Help system, which produced the text

(2.1) This is the kind of item displayed at left. This shows that test data is a(n) Microsoft Word document.

by inserting *test data* and *Microsoft Word* into template slots for 'file name' and 'application program'.

My project clearly falls into the category of 'true NLG', because it uses a significant amount of syntactic and morphological information in the realisation stage. In the tempate-based example above, even the simple agreement decision between *a* and *an* was not made, because the relationships between words which are not from the same predefined string are never considered. (This particular example is a phonological decision, but the same problem would prevent similar syntactic agreement decisions from being made.)

Amongst the advantages that Reiter gives for 'true NLG' over template-based generation are maintainability and multilinguality. Template-based generators can need major modifications to accomodate simple changes in requirements, for example, a preference that adverbial phrases be generated sentence-initially, or a preference that passive constructions be used wherever possible,

would require a modification of a large proportion of the (usually many) templates. A linguistics-based system is more likely to have the logic for adverbial phrases or for passive constructions centralised (rather than distributed across a large number of templates), allowing the program to be more efficiently modified.

Similarly, adapting a template-based system to multiple languages is likely to require a complete rewrite of the templates, because the linear order of constituents of a sentence is not constant across languages. To take the simplest of examples, English uses a subject-verb-object (SVO) consituent order in basic transitive sentences, whereas Japanese uses subject-object-verb (SOV) order (example from [29]):

(2.2) Taro saw the dog.

(2.3) *Taro ga inu o mita.*
Taro SUB dog OBJ saw.
Taro saw the dog.

Clearly a template used to produce (2.2) would have to be rewritten to allow generation of (2.3). If the linear order of constituents is analysed for more complex sentences, the divergence becomes more striking. When sentences are represented by deeper linguistic structures than strings of characters, however, these differences become manageable (see §3.2). A generator based on these structures is thus far more adaptable to multiple languages than a template-based system which is inherently linear in nature.

Reiter notes though that 'surprisingly little research has been done on the principles underlying multilinguality. For example, where can language independent modules be used in a multilingual system, and where is this impossible?' This project aims to make some progress towards answering these sorts of questions.

2.1.3 Machine Translation (MT)

Approaches to MT

The simplest approach to MT is known as the **direct** approach. An n-language translation system using the direct approach has one module for each possible language pair. The size of the system, and the effort required to construct it, therefore grows in the order of n^2; adding one new language requires translation modules to be written for the combination of this new language with every previously-existing language, and each of these modules requires knowledge of both languages. The direct approach therefore leads to systems which are relatively difficult to maintain.

The **transfer** approach provides better maintainability. The logic for analysing input sentences is only written once for each language. Each language's analysis module produces a language-specific machine-readable representation of the input sentence. This is passed on to a language-pair-specific transfer module, which generates a machine-readable representation of the same sentence, specific to the target language; and finally the target language's synthesis module converts this to the final target sentence. Just as each language has one analysis module which is applied to every source sentence in that language, irrespective

of the target language, each language has one synthesis module which is used to generate every output sentence in that language, irrespective of the source language. An n-language translation system then contains n^2 transfer modules, but only n analysis modules and n synthesis modules. This goes some way towards solving the direct approach's maintainability problems, but the system's size still has a quadratically-growing term.

The **interlingua** approach avoids this quadratic growth by completely eliminating all pairwise information. A system to translate amongst n languages comprises n analysis modules and n synthesis modules; all the analysis modules produce a representation of the source sentence in common interlingua, and all the synthesis modules convert this common interlingua representation to the output sentence in the corresponding language. In theory this allows a translation system to be built by a set of developers, each of whom is only familiar with one of the natural languages the system is required to work with (this is not possible using either the direct or transfer approach), while also ensuring that the size of the system grows only linearly with the number of languages.

Relation to NLG

At first it may appear that MT is the simple combination of NLG and its inverse, natural language understanding (NLU; the process of extracting useful information from natural language input). In reality, the questions usually addressed in the content planning and sentence planning stages of NLG have already been answered in the realisation (by a human or otherwise) of the source text, if we assume that information is to be included in the output text if and only if it is included in the source text, and that each sentence of source text should correspond to exactly one sentence of output text.

Thus the only stage of NLG which is necessary for MT is syntactic realisation. Similarly, the inverse stages of NLU corresponding to sentence planning and content planning are not needed, because the only aim is to transform one sentence in the source language to one sentence in the target language; only the inverse stage of syntactic realisation, which we might call syntactic analysis, is required.

Machine translation then, can be achieved by syntactic analysis immediately followed by syntactic realisation. An MT system which takes the interlingua approach would include two independent modules corresponding exactly to these two tasks. In theory these two modules could be useful parts of NLU and NLG systems. As discussed above, this project corresponds to the syntactic realisation stage of NLG, and is thus also potentially applicable to interlingua-based MT systems, with the same advantages in maintainability.

In the case of a transfer or direct approach though, the format of the input to the syntactic realisation stage depends on the source language. Thus the syntactic analysis and syntactic realisation functions cannot be modularised to the same extent as in an interlingua system, and it would not be possible to take the syntactic analysis/realisation module of a pure NLU/NLG system and use it directly for these types of MT.

2.1.4 Deterministic vs. Probabilistic Methods

In the last two decades, statistical or probabilistic methods have become dominant in computational linguistics; this project though deals only with deterministic rules. Its relationships with areas of the field where probabilistic methods are widely used are discussed here.

Grammatical Rules are Deterministic

Paul L. Garvin [15] noted that rules based on grammatical conditions, or the relationships within sentences, are primarily deterministic, but that probabilistic rules are more common when considering relationships across sentence boundaries. The implication of this for NLG is that the syntactic realisation stage will be largely deterministic, while probabilistic rules will be required to best address the tasks of content planning and sentence planning. Informally put, there are 'right' and 'wrong' ways to syntactically realise a sentence, but there are only qualitatively 'better' and 'worse' ways to group large amounts of information into multiple sentences.

Since this project works entirely at the syntactic realisation stage, no probabilistic methods are required.

Statistical Analysis of Linguistic Data

In corpus-based studies, large amounts of linguistic data are subjected to statistical analysis in order to discover characteristics of the language in question. The desired information might be lexical (i.e. what the lexical items of the language are and what they mean) or grammatical (i.e. how these lexical items are combined).

This project has little to say about the discovery of lexical items and their meaning, but it does attempt to isolate and minimise exactly what needs to be captured to represent the grammatical and syntactic knowledge of a language; this should allow these statistical analyses to reveal better descriptions of the languages they examine. For example, according to the P&P theory it is not necessary to separately investigate whether a language uses verb-object or object-verb order, and whether it uses prepositions or postpositions; these two characteristics are both manifestations of one parameter (the head-directionality parameter). This means that there is less statistical work to be done in such analyses (there are less questions to be answered), and that the derived rules should be more reliable (there are more data to base each rule on).

In this way 'distributional techniques do not so much compete with parameter setting as a model of acquisition, as much as complement it ... If parameter-setting views of syntax acquisition are correct, then learning the syntax ... is actually almost trivial.' [1].

2.1.5 Example of Language-Specific NLG

Allen [2] gives an example of a generation system which uses completely English-specific rules.

To represent the sentence

(2.4) Jack bought the red book from Sue.

an input semantic structure like the following is used:

(2.5)
- `b1` ϵ BUY
  AGENT: `jack1`
  FROM-POSS: `sue1`
  THEME: `bk2`
- `jack1` ϵ PERSON
  NAME: 'Jack'
- `sue1` ϵ PERSON
  NAME: 'Sue'
- `bk2` ϵ BOOK
  COLOR: `red1`

This states that the object `b1`, for example, which represents a particular buying event, is a member of the class BUY. `b1` has values assigned to each of three attributes, listed underneath it. The attribute AGENT is filled with a reference to the object `jack1`. `jack1` in turn is a member of the class PERSON, and has an attribute NAME. This value of this attribute is the string 'Jack'. (This input structure is similar to that used in this project, described in §5.2.1.)

The generation rules used to produce a sentence from this input are given below:

(2.6) (a) G(BUY) = G(>AGENT) bought G(>THEME) {from G(>FROM-POSS)} {for G(>CO-THEME)}

(b) G(PERSON) = G(>NAME) if node >NAME exists

(c) G(BOOK) = the G(>COLOUR) book {by G(>AUTHOR)}

(d) G(`red1`) = red

The rule in (2.6(a)), for example, describes how to realise a sentence based on an event of class BUY. The symbol > refers to the value of an attribute of the object on which the function G is currently being defined, so >AGENT refers to the value of the attribute AGENT. Braces indicate that a component is optional, and will be used only if the relevant attributes are defined.

The object `b1` in (2.5) is of class BUY, so we can apply (2.6(a)) to it:

(2.7) G(`jack1`) bought G(`bk2`) from G(`sue1`)

and applying the other rules in (2.6) to this result we obtain:

(2.8) Jack bought the G(`red1`) book from Sue

(2.9) Jack bought the red book from Sue.

This system has correctly produced the desired sentence, but the process for doing so is not easily extensible.

Firstly, other than the semantic input structure, nothing in Allen's system is applicable to any language other than English. To realise a description of an event of class BUY in some other language, rule (2.6(a)) would have to be rewritten completely. A Japanese version might look like this:

(2.10) G(BUY) = G(>AGENT) ga G(>THEME) o {G(>FROM-POSS) kara} katta

This has had to be completely rewritten to accomodate not only the new central lexical item (*katta* for *bought*), but also the change in constituent order and the Japanese use of case markers (*ga* and *o*). Every new language will require its own rule for describing a buying event.

Secondly, to produce variations of (2.9) resulting from syntactic transformations such as passivisation and negation, new rules again would have to be written, even for the same language:

(2.11) G(BUY, PASSIVE) = G(>THEME) was bought {from G(>FROM-POSS)} {for G(>CO-THEME)} {by G(>AGENT)}

(2.12) G(BUY, NEGATIVE) = G(>AGENT) did not buy G(>THEME) {from G(>FROM-POSS)} {for G(>CO-THEME)}

Since there is no syntactic structure to the output of the rules in (2.6), passivisation and negation cannot be implemented as operations on this output. This also means that sentences where both are present require still more rules:

(2.13) G(BUY, PASSIVE, NEGATIVE) = G(>THEME) was not bought {from G(>FROM-POSS)} {for G(>CO-THEME)} by G(>AGENT)

and this results in combinatorial growth of the number of rules.

The P&P theory provides solutions to these problems. My work does not require language-specific rules of the sort shown here; although it is not obvious from these surface structures, a large amount of what is captured in (2.6) and (2.10) can be 'factored out' and written just once. Similarly there is no need for construction-specific rules for passives, negatives, questions and so on; these particular forms are also results of more general, language-universal underlying principles.

2.2 Theoretical Linguistic Work

2.2.1 The Principles and Parameters Theory

Generative grammar is the study of 'what it is that a human being knows when he knows how to speak a language' [20], and in particular, which parts of this knowledge are language-specific and which parts are language-universal. The language-universal elements are attributed 'to the initial state of the language faculty, i.e. Universal Grammar (UG), thereby allowing the rules of the language to remain in the simplest form' [13]. In other words, this language-universal knowledge is considered to be innate to all human beings.

P&P theory is one version of generative grammar, the version which is used extensively in this project. Since formal syntactic theory's beginnings in the 1950s it has constantly been revised, resulting in a number of versions such as (in order) Transformational Grammar, Standard Theory, Government and Binding Theory, P&P and Minimalism; P&P is the name given to the version which developed around 1990. Other theories have also branched off this

sequence of revisions, such as Lexical-Functional Grammar and Head-Driven Phrase Structure Grammar [5].

As its name suggests, P&P emphasises the idea that the language-specific knowledge which is added to the innate principles of UG takes the form of the settings of a finite number of discrete parameters. These parameters should constitute a relatively small amount of information.

One reason this view is attractive is that it proposes a solution to one instance of what Chomsky calls 'Plato's Problem', a problem of 'poverty of the stimulus' [8]: the question of how human children appear to learn such a complex system of language in the first few years of life. What fluent speakers of a language appear to know about its syntactic system goes far beyond what they have been presented evidence for. If the language-specific information required is small, then it can be suggested that this is all that is actually learnt by children, and the rest was provided at birth. It appears that a lot of complex rules have been learnt for the same reason that human languages appear to differ dramatically on the surface, but the theory says that at a deeper level much of the syntactic 'mechanics' used to generate sentences are actually common to all languages.

In the same way that this model suggests that the language-learning process in children does not involve acquiring as much information as we might think at first, it suggests that it should be possible to construct a multilingual computer system to perform the same syntactic operations which is easily extensible to additional languages. The system should comprise a central 'core', which stores everything the theory attributes to 'the initial state of the language faculty', and one additional module for each language which the system 'knows'. The language-specific modules contain the information the theory says is learnt in the first few years of life about one's mother tongue; these are therefore small relative to the central core (putting aside lexical information; see §1.3.1, item 2). Adding a new language, then, is a relatively simple task, because only the 'distilled', genuinely language-specific information needs to be added, and nothing needs to be added which in fact applies to all languages; it is not obvious on the surface, but according to the theory this should be a significant saving. My system uses exactly the design described here.

2.2.2 Details of the Theory Used

The specific details of the formalisms which comprise the P&P theory are often contentious, and there are many linguistic phenomena for which there is no universally accepted approach. In these cases I have either chosen one from an array of approaches which have been proposed, or (for some minor points) independently developed one which is in keeping with the conventions of the theory.

My choices in these areas should not necessarily be considered meaningful contributions to the linguistic theory, but at the same time are minor enough not to invalidate the goal to test the P&P theory as described in §1.2.2. Those central ideas which are well-accepted have been used, and a project such as this can only test a theory to the extent that it is agreed upon by its proponents.

The following sections briefly point out the main pieces of theory I have used which are perhaps not universally accepted, but are not intended as complete explanations of the approaches I have adopted; these details are explained in due course later in the report.

X′ Theory as a Primitive

X′ theory is taken as a theoretical primitive and provides the framework in which the entire generation process is set. It is not possible for the system to build a structure which does not conform to its rules. The version used here is essentially that which was introduced by Chomsky in 1970, and was then considered a central part of generative grammar through to the mid-1990s [9]. See §3.2 for details.

Using X′ theory as a primitive is the most significant departure from the state of the art in syntactic theory. In the most recent revision of generative grammar, Minimalism, there have been moves away from the use of X′ theory as a fundamental. In 1995 Chomsky introduced Bare Phrase Structure, a simpler approach where the patterns defined by X′ theory, rather than pre-existing restrictions on what can be generated, are considered to be results of moving and merging operations motivated by other factors. Notions such as XP and X′ still exist, but they are read off the tree once it has been generated independently, rather than being part of the defining nature of the tree [5].

The main reason the pre-Minimalism approach was adopted was that the idea of a recurring phrase structure, from a programming point of view, elegantly allows one simple data structure to underpin a wide range of complex constructions. See chapter 4 for details.

The Functional Head Pred

To represent adjectival and nominal predicates, a functional head Pred is used in the position usually associated with a lexical head V [4]. How Pred is realised is a parameter; in English, for example, it is realised as a form of the verb 'to be'. If a predicate is to be constructed based on a verb, a VP is generated with that verb as its head; if a predicate is to be constructed based on an adjective or a noun, a PredP is generated with a maximal projection of that adjective or noun in its complement position. Throughout the rest of the syntactic realisation process, the exact same role traditionally played by the VP can be played by the PredP. See §5.3.3 for details.

The Functional Head Neg

To model negation, a function head Neg is introduced immediately above the VP; that is, a maximal projection NegP takes the VP's place as the complement of the TP, and the VP becomes the complement of the NegP. This is the only structural difference between a sentence and its corresponding negative version. The Neg head may block the verb-raising or affix-lowering movement which gives the verb its inflection in sentences which don't have an overt auxiliary; whether or not this blocking occurs is a parameter. See §5.3.1 for details.

Pronouns Realised as Determiners

Pronouns are realised in the head D of DPs, rather than the head N of the complement NP, as in Cook and Newson [9]. See §5.3.4 for details.

2.2.3 Limitations

I have abstracted away from some of the more radical traits found in some of the world's languages, notably non-configurational languages and polysynthetic languages.

In polysynthetic languages, such as Mohawk, each of a verb's arguments must be expressed on the verb, as in the following example (from Baker [3]):

(2.14) *Sak* *shako-nuhwe's* *ne* *owira's.*
Sak 3SING.3SING-like the baby.
Sak likes the baby.

Baker has labelled whether or not a particular language includes this constraint 'The Polysynthesis Parameter', but it is not clear what formal syntactic transformations of the style used in this project are performed to produce the correct surface realisations.

Polysynthetic languages also tend to be nonconfigurational, which means that word order is extremely free and need not even be hierarchically structured, as in the following Warlpiri example (Speas 1990, cited in [13]):

(2.15) *Wawirri* *kapi-rna* *panti-rni* *yalumpu.*
kangaroo AUX.1.PERSON.SUBJ spear.NONPAST that
I will spear that kangaroo.

Here two constituents which would usually be considered to form a DP, *wawirri* and *yalumpu*, appear at opposite ends of the sentence. My system will never generate a sentence with such a word order, but it may be worth noting that this is not the *only* grammatical word order for this sentence (cf. §1.3.2, item 1).

There have also been attempts to accomodate nonconfigurational languages in the P&P theory, for example Hale's 'configurationality parameter' [13]. Hale proposes that for each sentence a Lexical Structure (LS) and a Phrase Structure (PS) are a generated, and that the difference stems from the fact that while configurational languages (such as English) require that the Projection Principle hold of both LS and PS, nonconfigurational languages (such as Warlpiri) only require that it hold of LS. While formalising the observations somewhat, as with the Polysynthesis Parameter, this still does not describe the syntactic transformations sufficiently for them to be implemented in a project of this nature.

Research continues into how P&P theory can allow for these languages types. The more theoretically elegant and complete a proposal on this subject is, and the fewer changes to existing theory it requires, the easier it will be to incorporate into the system I have implemented.

Chapter 3

Language-Universal Linguistic Concepts

3.1 Semantic Relations

The notion of **semantic relations** (or **semantic roles**) allows a language-independent description of the relationships between the participants in a sentence. Consider the following sentence:

(3.1) The boy kicked the ball.

To describe what is happening here at a language-universal level, we can say that the participant (or noun phrase) *the boy* has the semantic role of **agent** and that *the ball* has the semantic role of **patient**. The role of agent applies to an entity which initiates an action; the role of patient applies to one which undergoes some change of state.

The traditional grammatical concepts of subject and object (or 'grammatical functions') are not sufficient for this purpose, because these are not necessarily consistent cross-linguistically. Consider the following examples from English and German:

(3.2) I like the picture.

(3.3) *Das Bild gefällt mir.*
the picture pleases to me
I like the picture.

These two sentences should be considered semantically equivalent. The noun phrase in (3.3) corresponding to *I* in (3.2) is *mir*, however *I* is a subject and *mir* is an indirect object. Similarly, *the picture* is an object, but *Das Bild* is a subject. Thus concepts of subject and object do not describe semantic relationships in a language-universal manner; they are defined by the syntax. When semantic roles are used though, the similarity between these two sentences becomes clear. Both *the picture* and *Das Bild* have the semantic role of **theme**, and *I* and *mir* share the semantic role of **experiencer**. Part of the meaning of the English verb *like* is that its subject is an experiencer and its object is a theme, and part of the

meaning of the German verb *gefallen* is that its subject is a theme and its indirect object is an experiencer; English and German speakers' knowledge of these verbs allows them to derive the correct semantics from these sentences. Semantic roles allow a universal representation of the intended meaning, irrespective of how any one language decides to construct a sentence to convey it.

Semantic roles are not tied to particular verbs or constructions. *The boy* has the agent role in all of the following examples:

(3.4) The boy broke the window.

(3.5) The boy ran.

(3.6) The ball was hit by the boy.

and *the window* in (3.4) and *the ball* in (3.6) both have the patient role. In the following sentence though, despite its similarities to (3.4), *the boy* has the patient role (discounting the interpretation where he deliberately harms himself):

(3.7) The boy broke his arm.

Furthermore, some noun phrases which have grammatical roles do not have semantic roles. In the sentence:

(3.8) It rained.

the subject *It* does not have a semantic role. It does not represent any participant in the act of raining, it simply appears for syntactic reasons, and is not relevant to the semantics of the sentence. Thus semantic roles and grammatical functions are effectively independent. The semantic roles present vary from one sentence to the next: not all sentences have an agent, not all sentences have an experiencer, and so on; but every sentence must have a grammatical subject (this condition forced the insertion of the subject *it* in (3.8)).

In P&P theory a distinction is sometimes made between semantic relations and **θ-roles** on the basis that a verb can assign more than one semantic relation to an argument. For example, *John* could be said to have the semantic roles of both agent and source in the following sentence:

(3.9) John gave the book to Mary.

Under this analysis, the semantic roles of agent and source would be grouped together in one θ-role, and this θ-role would be assigned to *John*. For the purposes of this project though, this distinction is not vital (see §5.2.1), so the term *θ-role* can be considered equivalent to *semantic relation*.

Also in contrast to grammatical functions, there is no universally accepted, exhaustive list of all possible semantic roles; they have 'clear central instances but fuzzy boundaries' [4]. From a syntactic point of view though, 'the existence of θ-roles is what is important rather than the differences between them' [9].

3.2 X′ Theory

X′ theory (pronounced 'X bar') is one module of the overall P&P system. Others include theta theory and case theory, for example; these are concerned with the creation and transformation of the formal structures which represent sentences. X′ theory underlies all these other components by providing a framework which defines the set of structures which the rest of the theory works with.

In particular, X′ theory proposes a common structure which is shared by all phrases of all categories (noun phrases, verb phrases, etc.), and all sentences and clauses, across all languages.

3.2.1 Evidence for X′ Phrase Structure

A fundamental fact of syntax is that sentences are not simply one-dimensional strings or sequences of words, even though this is the form they take when realised, either in written or spoken form. This notion is known as **structure-dependency**, and is evident when we consider the following pairs of utterances:

(3.10) (a) John has eaten the apple.
(b) Has John eaten the apple?

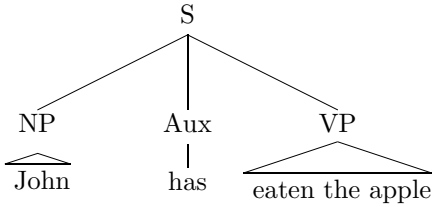
(3.11) (a) The man has eaten the apple.
(b) Has the man eaten the apple?

(3.12) (a) The man who has brown hair has eaten the apple.
(b) Has the man who has brown hair eaten the apple?

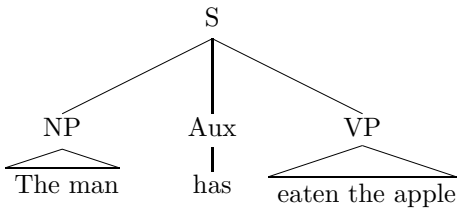
It is intuitive to any English speaker that in each case the sentence (a) has undergone the same transformation to produce the corresponding question (b), but in 3.10 it was the second word of the sentence which moved to the front, whereas in 3.11 it was the third word and in 3.12 it was the seventh word. If sentences were nothing more than linear sequences of words, no one rule could produce these three transformations. From 3.10 and 3.11 it could be suggested that the transformation involves searching for a particular word which should be moved, because in both these cases it is the auxiliary verb *has* which moves; but this rule would not work in 3.12, where the word *has* appears twice, and the correct instance of it must be moved for the derived question to be correct. Thus sentences must have some further underlying structure inside speakers' minds, and it is on this structure that syntactic transformations (such as question-forming) operate.

If the three (a) sentences above are represented by the following tree structures, the similarity in the three transformations which occurred to form the questions becomes clear. (These do not yet conform to X′ theory.)

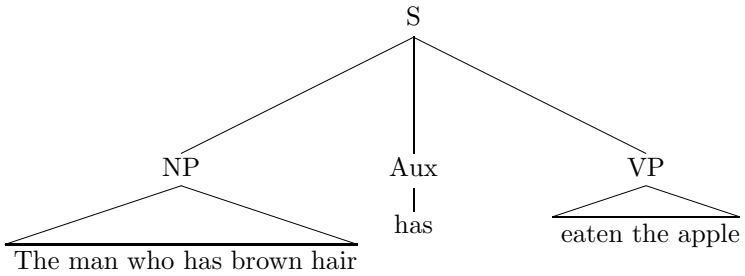
(3.13)



(3.14)



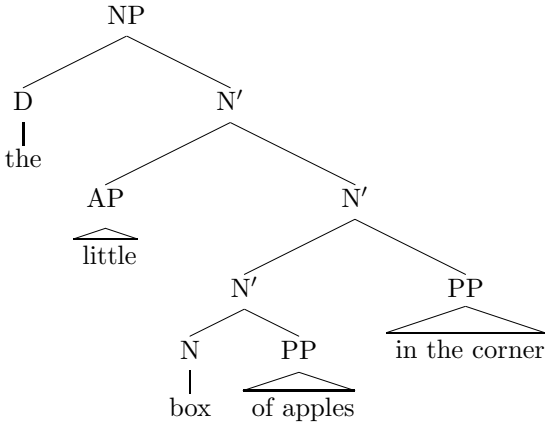
(3.15)



Given these structures, we can say that to form the corresponding question, the Aux node has to be moved to the front of the sentence. Thus it seems that such tree structures are a better representation of the underlying nature of sentences.

In the trees above, triangles were used to abstract away from internal details of the NP (noun phrase) and the VP (verb phrase) which were not relevant, but more detailed trees are possible. The following tree shows the internal structure of a relatively complex noun phrase, for example.

(3.16)



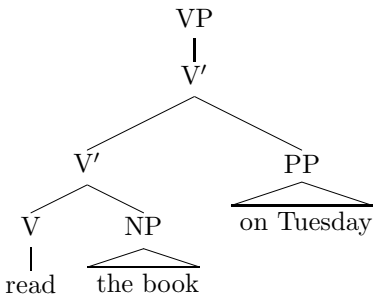
Evidence that this is the internal structure comes from tests for constituency: for example, if a string can be replaced by *one* then it should be a consituent, meaning that there should be a node of the tree which dominates precisely that string. Thus *box of apples* is a constituent because we can say

(3.17) I saw the little box of apples in the corner, but not the big one on the shelf.

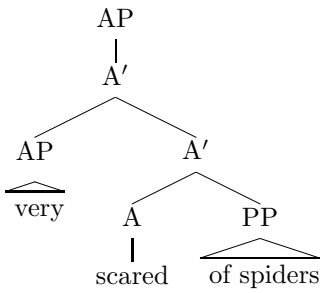
and there should be a node of the tree which dominates exactly that string. This is the lowest N′. No such replacement can be made for, say, *apples in the*, because it this not a constituent.

Using the same logic to expand sample verb phrases, adjectival phrases (APs) and prepositional phrases (PPs) leads to the following trees.

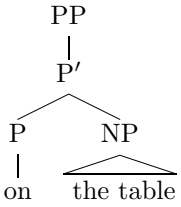
(3.18)



(3.19)



(3.20)



The structure of these trees can be expressed by the rewrite rules given below. Optional components are given in parantheses.

(3.21) (a) NP → (D) N′
(b) N′ → (AP) N′ or N′ (PP)
(c) N′ → N (PP)

(3.22) (a) VP → V′
(b) V′ → V′ (PP)
(c) V′ → V (NP)

(3.23) (a) AP → A′
(b) A′ → (AP) A′
(c) A′ → A (PP)

(3.24) (a) PP → P′
(b) P′ → P′
(c) P′ → P (NP)

Admittedly there does not seem to be any logic for the rules in (3.22(a)), (3.23(a)), (3.24(a)) and (3.24(b)) but these rules do no harm for now. Their use will become clear shortly.

Looking at these rules though, there are significant similarities in the internal structure of all these types of phrases. If we let a variable X stand for any of the lexical categories (N, V, A and P), then every XP consists of an X (known as the **head** of the XP), and some further optional constituents. The head is the only component which is ever compulsory.

Immediately we can unify the four (c) rules above in the following:

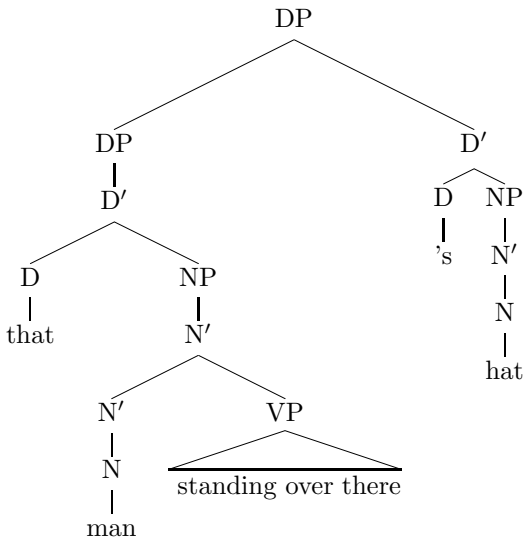
(3.25) $X' \rightarrow X$ (WP)

where X and W are variables which can take the value of any lexical category. The WP is said to be the **complement** of the parent XP.

A second similarity amongst the above rules is the fact that nearly all the optional constituents are phrases. The only exception is the D in (3.21(a)). To fix this, the D can be considered the head of a DP, and the NP the complement of this DP. As well as serving to further unify the rules in (3.21-3.24), evidence for this structure comes from the fact that the English possessive affix *-'s* can appear with any arbitrarily complex NP:

(3.26) that man standing over there's hat

(3.27)



It would not have been possible to construct a tree for this phrase using the rules above.

To reflect this new proposal, rewrite rules can be developed for DPs and must be modified slightly for NPs; these are shown below, along with those for VPs, APs and PPs reproduced as before.

(3.28) (a) DP → (DP) D′
(b) D′ → D′
(c) D′ → D (NP)

(3.29) (a) NP → N′
(b) N′ → (AP) N′ or N′ (PP)
(c) N′ → N (PP)

(3.30) (a) VP → V′
(b) V′ → V′ (PP)
(c) V′ → V (NP)

(3.31) (a) AP → A′
(b) A′ → (AP) A′
(c) A′ → A (PP)

(3.32) (a) PP → P′
(b) P′ → P′
(c) P′ → P (NP)

Note that the rule introduced in (3.25) still holds. Furthermore, now we have a set of rules where *every* optional (non-head) component is phrasal. This means that every phrase can be said to consist of exactly one head and (optionally) other phrases. Similarly to the rule in (3.25), we introduce the following to completely unify the system:

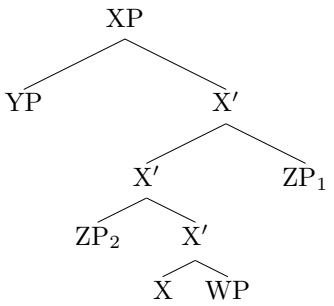
(3.33) XP → (YP) X′

(3.34) X′ → (ZP) X′ or X′ (ZP)

The YP in (3.33) is known as the **specifier** of the XP, and the ZP in (3.34) is known as an **adjunct** to the XP (since the rule is recursive, a phrase can have multiple adjuncts).

Now we have introduced a general, recursive structure applicable to all English phrases:

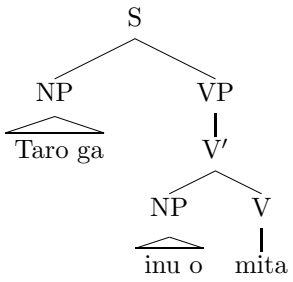
(3.35)



where any number of adjuncts (ZP_n) can occur on either side of the X′s. It can be seen that all phrases in the trees in (3.18-3.20) and (3.27) are indeed instances of this generalised structure.

Up to this point, though, all the data that have been used to derive these rules have been from English. We want a theory of phrase structure that applies to all languages, but these rules cannot generate, for example, the following typical Japanese sentence (also given in §2.1.2).

(3.36) (a)

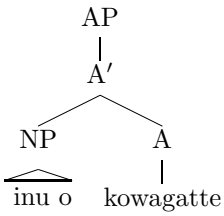


(b) *Taro ga inu o mita.*
Taro dog saw
Taro saw the dog.

The tree attempts to fit the sentence into the generalised rules above as much as possible, but the word order is fundamentally different. The rule in (3.25) states that the complement appears to the right of the phrase's head, but in this sentence the NP complement *inu o* is to the left of the head V *mita*. This appears to be a problem for X′ theory.

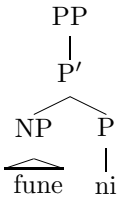
Consider though the tree structures below for some other typical Japanese phrases (from [9]).

(3.37) (a)



(b) *inu o kowagatte*
dog afraid
afraid of the dog

(3.38) (a)



(b) *fune ni*
boat on
on the boat

These also do not fit with the generalised X′ rules, but for exactly the same reason: the complement and the head (the A in (3.37) and the P in (3.38)) are the wrong way around. This turns out to be a consistent difference between Japanese and English. Instead of the rule given in (3.25), Japanese uses a variation:

(3.39) X′ → (WP) X

The precise ordering of heads and complements is an example of a **parameter** in the P&P theory. This is known as the **head-directionality parameter**; English is a head-initial language (along with French, Irish, and many others), and Japanese is a head-final language (along with Turkish, Basque, and many others again). Using traditional grammar, it would have been necessary to state separately that English uses SVO word order whereas Japanese uses SOV, that English uses prepositions whereas Japanese used postpositions, and so on; now, however, all these differences are captured in the one statement about head-directionality.

Thus the language-universal X′ rules are:

(3.40) XP → (YP) X′ or X′ (YP) (The Specifier Rule)

(3.41) X′ → (ZP) X′ or X′ (ZP) (The Adjunct Rule)

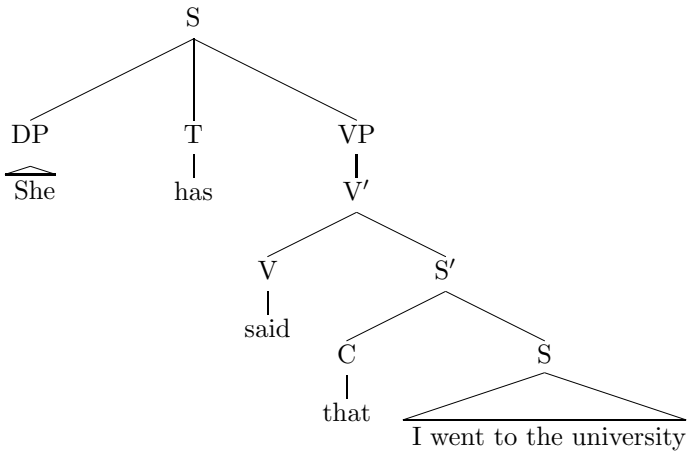
(3.42) X′ → (WP) X or X (WP) (The Complement Rule)

and the version of each rule which a particular language chooses (and then applies to every phrase type in that language) is a parameter of linguistic variation. The underlying structure is the same in every language, all that changes is the linear order in which these structures are realised. As will be shown below, a simple switch of this one parameter setting in a complex system of nested phrases can create variation in word order that suggests far more complicated underlying differences.

3.2.2 Extending X′ Structure to Sentences and Clauses

X′ theory now accounts for the internal structure of NPs, DPs, VPs, APs and PPs. These phrases come together to form larger, meaningful sentences. Based on terms from traditional English grammar, we might suggest that full English sentences are formed as in the following example, where C stands for complementiser and T roughly for any tense inflections and auxiliary verbs:

(3.43)



The rewrite rules describing this structure would be

(3.44) S′ → (C) S

(3.45) S → DP T VP

These clearly do not conform with X′ theory. In order to unify these with the theory, we can rename S′ to CP, with C as its head and S its complement (and an empty specifier position). Also we can rename S to TP, with DP as its specifier, T its head and VP its complement.

(3.46) CP → C′

(3.47) C′ → (C) TP

(3.48) TP → DP T′

(3.49) T′ → T VP

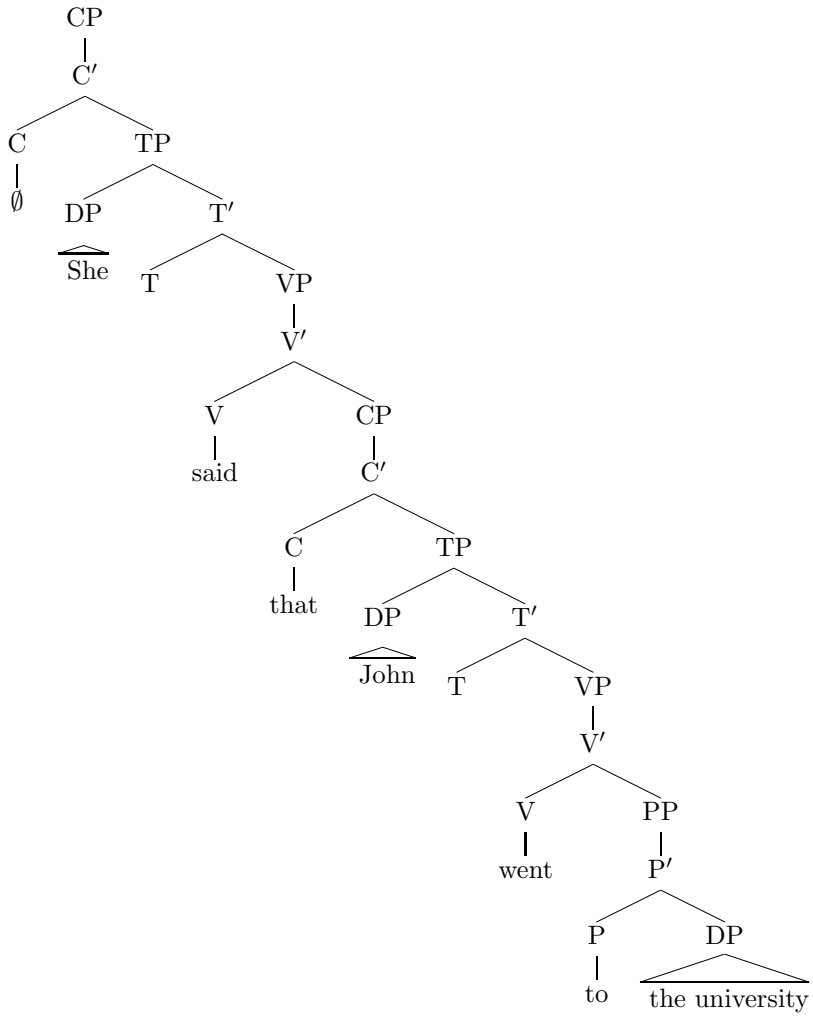
These are now almost consistent with rules (3.40-3.42) (and in fact with the English-specific rules (3.33-3.25), as would be hoped). The only differences concern which constituents are optional.

Firstly, in (3.44), the C was indicated as optional, based on the fact that English main clauses have no overt complementiser. This can be fixed by proposing that in fact there is a phonologically null complementiser in this position in these clauses, so the CP rule now meets the condition that every XP has a head X. (This proposal is supported by evidence based on the formation of questions; see §5.3.5.) Similarly, not all sentences have an overt auxiliary in the T position; where this is the case though, functional information about the tense of the sentence is considered to be stored in a T node (see §3.3.1).

Secondly, the TP, DP and VP in the rules above appear to be obligatory, even though these are in the place of specifiers and complements and therefore should be optional according to X′ theory. X′ theory is understood to 'overgenerate' though (it already allows, for example, PPs with a VP specifier and an AP complement, even though such an arrangement has not been needed), so it is not unreasonable to treat something which seems obligatory as optional. X′ theory itself thus allows CPs which do not have a complement; but X′ theory interacts with a number of other modules of P&P theory which restrict the possible range of outputs. It is important though that X′ theory does allow every occurring structure.

Having accounted for these differences, (3.46-3.49) now conform perfectly with the X′ rules; under this model, a sentence is nothing but a complementiser phrase (CP), with the standard X′ theory phrase structure. The sentence given in (3.43) can now be represented by the following tree:

(3.50)

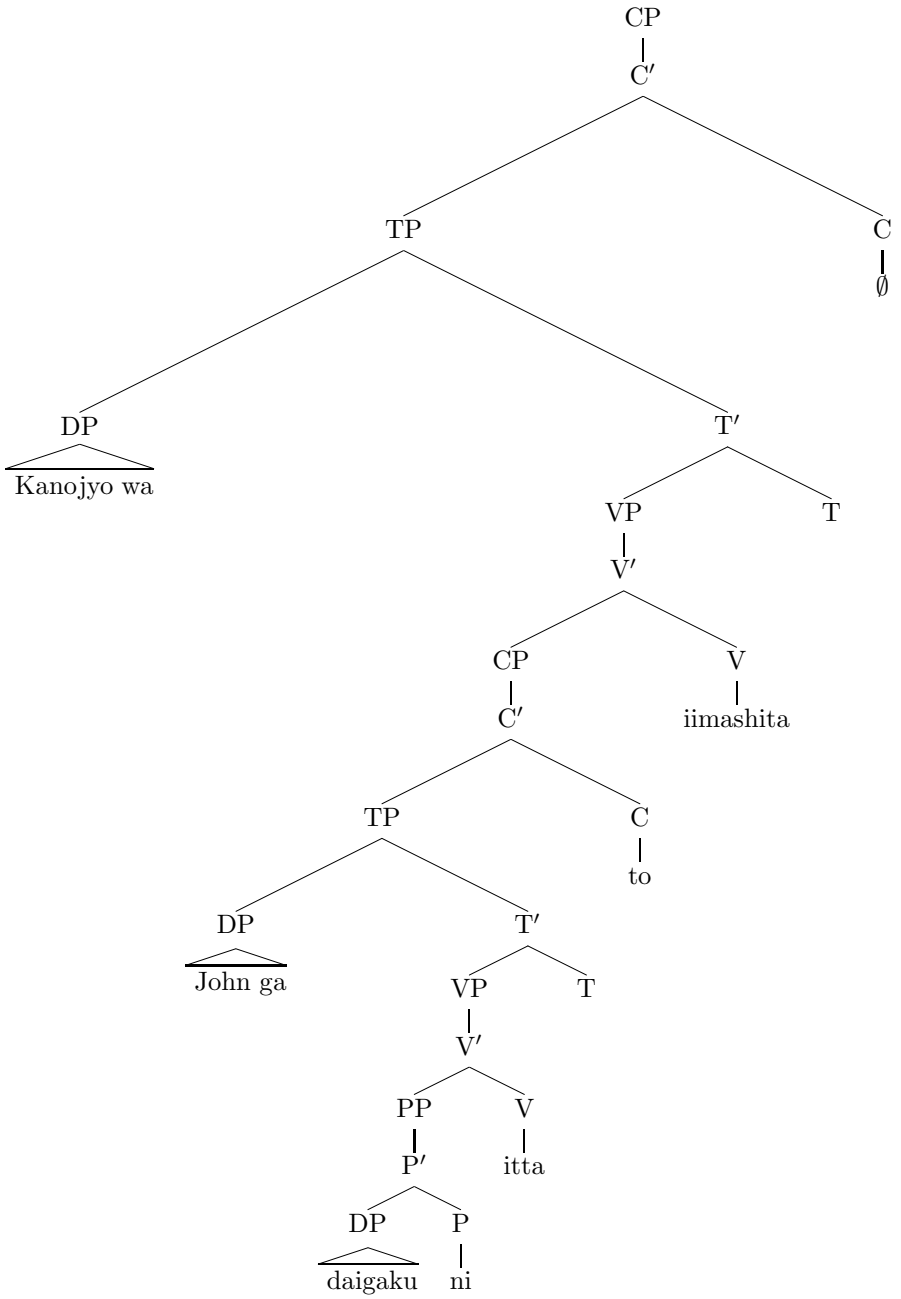


Consider now the equivalent Japanese sentence (from [30]):

(3.51) *Kanojyo wa* *John ga* *daigaku* *ni* *itta* *to* *iimashita.*
she John the university to went that said
She said that John went to the university.

On the surface, the variation in word order in a relatively complex sentence like this (compared to the simple SVO/SOV distinction) appears drastic and difficult to succinctly account for. However, if this sentence is analysed using the same rules as those used for (3.50), we derive the following tree:

(3.52)



Note that despite the significant differences in surface word order, the *only* difference between (3.50) and (3.52) is the head-directionality parameter. The same parameter as neatly accounted for Japanese VPs, APs and PPs in (3.36-3.38) is *still* all that is required to account for all the differences in word order between Japanese and English which have been presented.

3.2.3 Relation to Language Universals

The discussion above has shown that a consistent difference between English and Japanese is the ordering of phrases' heads and complements. In every English phrase given so far, the head precedes the complement; in every Japanese phrase, the head follows the complement.

It seems reasonable to say that every phrase of a given category (NP, DP, VP, etc.) in a given language will have the same head-directionality setting; but no evidence has been presented so far to suggest that a language must use the same head-directionality setting for every category of phrase. Could it not be the case that every language has one setting for each category of phrase, and English and Japanese just happen to have the same setting for all phrase types? (Seven phrase types have been presented so far, so around one in $2^6 = 64$ languages would be expected to do this.)

Evidence that there is really only one head-directionality setting for each language (or at the very least, one dominant one) comes from independent research into language universals and linguistic typology. Joseph Greenberg [16], based on a study of 30 languages from a wide range of genetic backgrounds, presented a number of universals concerning constituent order, including the following:

(3.53) 'Universal 2: In languages with prepositions, the genitive almost always follows the governing noun, while in languages with postpositions it almost always precedes.' (Of the 30 languages studied, 29 conformed to this rule and 'If anything, 1/30 is an overestimation of the proportion of exceptions on a world-wide basis'.)

(3.54) 'Universal 4: With overwhelmingly greater than chance frequency, languages with normal SOV order are postpositional.' (All 11 languages studied which use SOV order use postpositions.)

(3.55) 'Universal 9: With well more than chance frequency, when question particles or affixes are specified in position by reference to the sentence as a whole, if initial, such elements are found in prepositional languages, and, if final, in postpositional.' (Of the 12 languages with such particles, 10 conformed to this rule.)

If the genitive (or possessing) noun phrase is considered a complement of the governing (or possessed) noun phrase [24], then Universal 2 states that languages with head-initial PPs almost always have head-initial NPs, while languages with head-final PPs almost always have head-final NPs. Similarly, Universal 4 says that languages with head-final VPs tend to have head-final PPs. Question particles have not been discussed yet, but they will be analysed as complementisers (see §5.3.5). Observe the position of the empty complementisers in (3.50) and (3.52); Universal 9 therefore states that languages with head-initial CPs tend to have head-initial PPs, and languages with head-final CPs tend to have head-final PPs.

Thus independent typological findings support the suggestion that languages have a strong tendency to use one head-directionality setting at least for PPs, NPs, VPs and CPs; similar generalisations can be made for other phrase types too. It is not just by chance that English and Japanese use the same setting for every phrase type.

3.3 Theory of the Syntactic Realisation Process

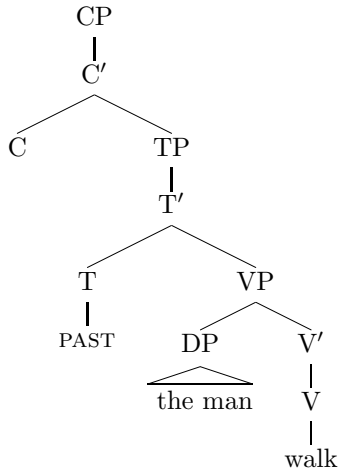
This section describes some of the basic procedures which comprise the syntactic realisation process. It is not intended to provide a detailed account of all the transformations which have been implemented, but rather to give the reader a general idea of the kind of operations which the theory says are performed on the structures made available by X′ theory. The implementation of the X′ framework (detailed in chapter 4) will need to allow for these operations.

Full details of the more complex transformations and the sentences they produce are given in chapter 5.

3.3.1 D-structure

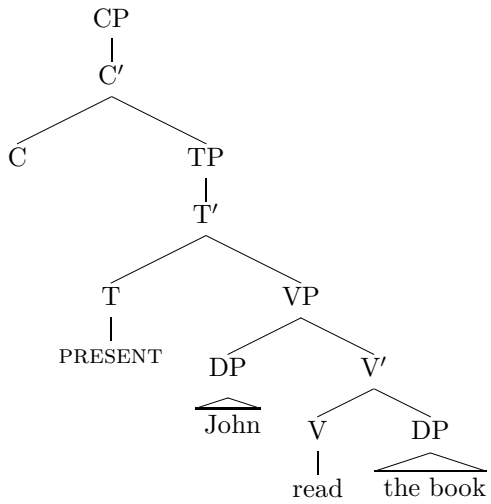
D-structure can be considered the input to the syntactic realisation process. It is derived from the desired semantics of the sentence to be generated, and from the target language's lexicon. Some sample D-structures, with the sentences they represent, are given below (order not important).

(3.56) (a)



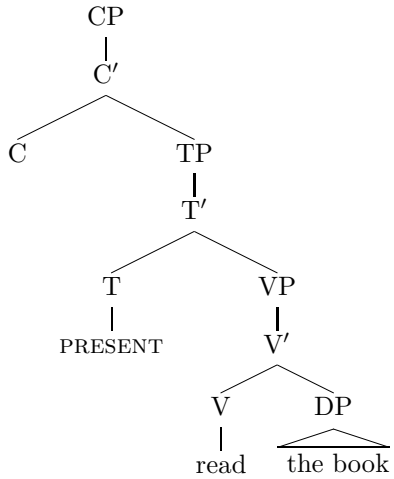
(b) The man walked.

(3.57) (a)



(b) John reads the book.

(3.58) (a)



(b) The book is read.

Note the significant similarities between the D-structures of all these sentences, reflecting their similar semantics, despite the obvious differences in the final constructions. It is the job of the syntactic realisation process to convert these D-structures to the linear forms listed below them.

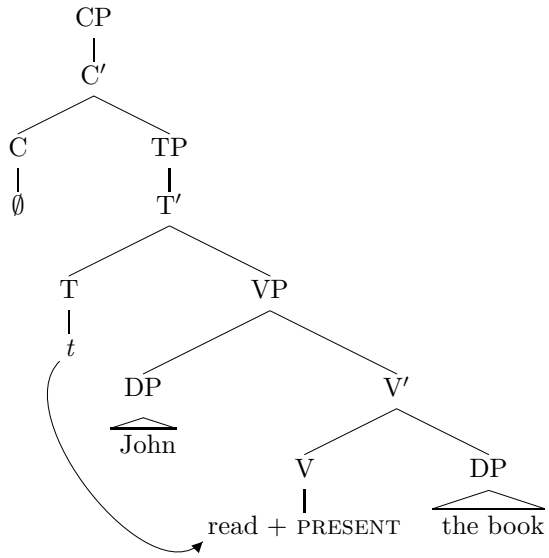
3.3.2 From D-structure to S-structure

A sentence's D-structure is converted by a series of transformations into what is known as S-structure. S-structure is very close to the final, realised form of a sentence. As noted above, D-structures are totally independent of the target language's syntax; it in the transformation from D-structure to S-structure that the (parametrised) syntactic differences amongst languages come into effect.

Producing Verb Inflections

One significant difference between the D-structure and the corresponding S-structure in, for example, (3.57), is that the D-structure contains an isolated abstract element PRESENT, whereas the S-structure indicates this by the inflection *-s* on the verb. The transformation which produces this result is, quite intuitively, a merging of the T and V heads. In English, the T is 'lowered' to the V position, represented as follows:

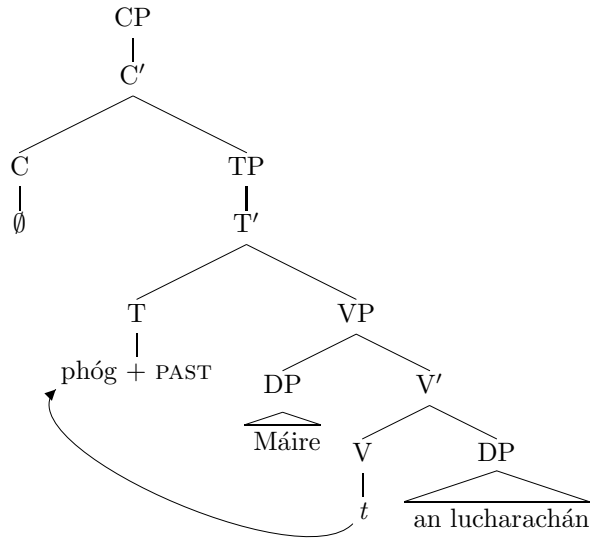
(3.59)



The t on this tree represents a trace, a phonologically empty element that is left behind when a constituent moves from one position to another. Reading the leaves off this tree, taking into consideration English's X′ parameter settings as a specifier-initial and head-initial language, now produces the correct linear word order.

The manner in which this merging of T and V occurs is another example of a parameter of language variation though. Irish, for example, uses the alternative method, and instead 'raises' the V to the T position, producing a VSO surface word order (data from [5]):

(3.60) (a)

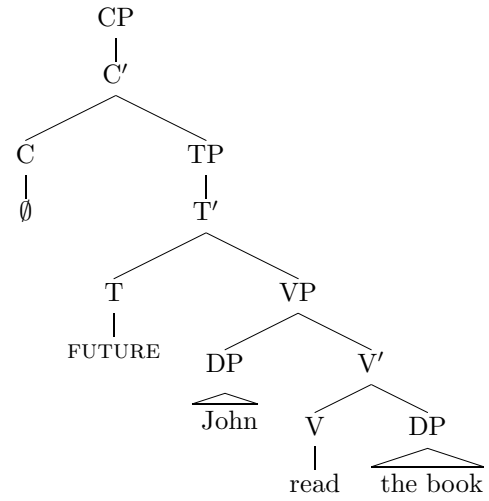


(b) *Phóg Máire an lucharachán.*
kissed Mary the leprechaun
Mary kissed the leprechaun.

Subject DP Movement

The merging of V and T is a logical way to produce inflected verb forms, but consider the following D-structure and desired output sentence:

(3.61) (a)

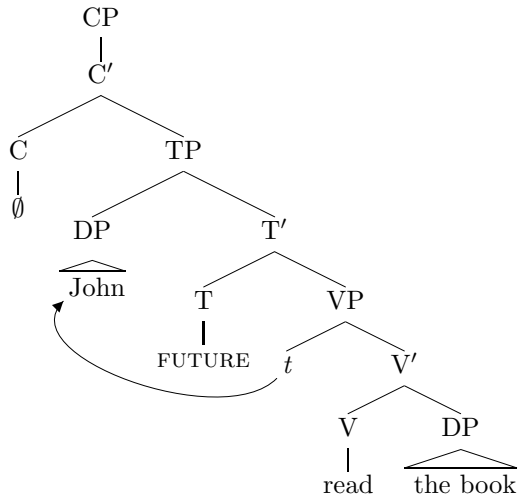


(b) John will read the book.

Here, no merging of V and T appears to have occurred. The abstract FUTURE element has been realised as the auxiliary *will*, and the verb has been realised as the uninflected *read*. However, the D-structure suggests an incorrect surface word order; the T node, which produces the auxiliary *will*, precedes the DP *John*. Some other movement is required to produce the correct word order.

The logical conclusion is that the subject DP moves to some higher position. Under X′ theory, there are only two possible positions, the specifier position of the TP and the specifier position of the CP. Both these positions are currently empty. It turns out that the specifier position of the CP has another use in the formation of *wh*-questions (see §5.3.5), so by elimination the subject DP must move up to the specifier position of the TP:

(3.62)



Note that in English sentences where the V and T nodes do merge, such as (3.59), whether or not this movement of the subject DP takes place will not affect the surface word order. For consistency, let us say that this DP movement takes place in every English sentence, irrespective of whether it is 'required' like it was in (3.62). (In fact, this claim can be justified by considering the position of adverbial adjuncts to the VP; see (3.64 below.)) The structure given in (3.59) is therefore not technically a correct final S-structure.

Clearly though, this movement of the subject DP does not occur in Irish, or the surface VSO word order would not be generated. Thus another parameter has been discovered.

3.3.3 A Note on the Distribution of Parameters

In this brief introduction to the theory, three parameters have been presented:

- the head-directionality parameter
- the verb-inflection parameter (either raise V to T, or lower T to V)
- the subject-movement parameter (either move the subject DP to the specifier of TP position, or leave it in the specifier of VP position)

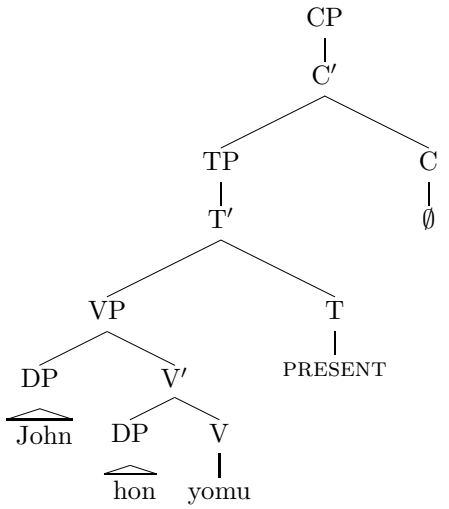
The settings for these parameters for three languages that have been used to demonstrate these ideas (English, Irish and Japanese), along with the basic word order of each, is shown in the following table:

	English	Irish	Japanese
head-directionality	head-initial	head-initial	head-final
verb-inflection	lower T	raise V	-
subject-movement	yes	no	-
basic word order	SVO	VSO	SOV

Table 3.1: Parameter settings for English, Irish and Japanese

Note that Japanese has not been given a setting for the verb-inflection or the subject-movement parameter. This is because these two parameters don't have a major effect on the word order of a language unless it is head-initial, like English and Irish. This becomes clear is we draw the D-structure of a simple transitive Japanese sentence, taking into account its head-final nature (from [30]):

(3.63) (a)



(b) *John ga hon o yomu.*
John SUB book OBJ reads
John reads the book.

Starting from this D-structure, no matter whether V raises to T or T lowers to V, the correct surface word order will be produced. Similarly, whether the subject DP stays in the specifier of the VP or moves up to the specifier of the TP will have no effect. Thus it is hard to say which way Japanese sets these parameters (and it is of little consequence). (Since these details do not affect the final Japanese word order, for simplicity, trees of Japanese examples in this report will often show subject DPs remaining in the specifier position of VP and will not explicitly indicate merging of V and T nodes; this merge does always occur though in Japanese, one way or the other.)

Consider now the following table showing the eight possible combinations of these three parameters, and the surface word order produced by each:

head-directionality	verb-inflection	subject-movement	basic word order
head-initial	lower T	yes	SVO
head-initial	lower T	no	SVO
head-initial	raise V	yes	SVO
head-initial	raise V	no	VSO
head-final	lower T	yes	SOV
head-final	lower T	no	SOV
head-final	raise V	yes	SOV
head-final	raise V	no	SOV

Table 3.2: Word orders resulting from parameter combinations

The last four rows of this table reflect the conclusion from above that head-final languages are not significantly affected by the verb-inflection and subject-movement parameters: all the combinations in these last four rows produce a basic SOV word order.

However the table also indicates that these two parameters only have a significant effect on head-initial languages in the particular combination in the

fourth row of the table. This is the arrangement used by Irish. Considering an English sentence such as that in (3.59), it is clear that any of the combinations in the first three rows of this table will produce the same surface word order. There are minor differences between the languages with these combinations of settings, but all are variations on the basic SVO word order. For example, English and French both use subject-movement, but English lowers T to V (first row of the table) whereas French raises V to T (third row), producing the minor difference in these two sentences:

(3.64) I often drink tea.

(3.65) *Je bois souvent du thé.*
I drink often tea
I often drink tea.

Here the adverbial phrase *often* is an adjunct to the VP; in English, the inflected verb is formed in the lower V position, and in French the inflected verb is formed in the higher T position. Both are essentially SVO languages though.

If this parametric view of languages is correct, and surface word order really is a result of more abstract, underlying, independent differences such as head-directionality and subject-movement, according to table 3.2 we would expect SOV to be the most common basic word order, followed closely by SVO, with VSO relatively rare. This is in fact the distribution of these basic word orders which has been found in typological studies of the world's languages [3]:

basic word order	percentage of languages	examples
SOV	45	Japanese, Turkish
SVO	42	English, Indonesian
VSO	9	Irish, Zapotec

Table 3.3: Distribution of basic word orders

The fact that the distribution shown in this table roughly follows that suggested by table 3.2 strongly suggests that basic word order is not a 'primitive' feature of a language. If each language 'chose' either SOV, SVO or VSO word order, then we would expect these three possibilities to occur in roughly equal numbers, but this is not the case. This is more strong evidence that these surface differences are brought about by the underlying parameters proposed here.

Furthermore, not only does the P&P view predict the right relative frequencies of these basic word orders, but, *without adding anything more to the theory*, it also correctly predicts differences *beyond* basic word order, such as adverb placement in English and French (as seen in (3.64-3.65)) and embedded clause word order in English and Japanese (as seen in (3.50) and (3.52)). This theory thus provides a more complete and powerful model of linguistic variation than can be derived from the more traditional grammatical concepts it underlies.

Chapter 4

Implementation of the X′ Theory Framework

This chapter describes the object-oriented design of the implementation of the X′ framework. The design goal was to accomodate the formal structures and operations proposed by the theory, as outlined in the previous chapter.

4.1 The `Phrase` Class

Taking advantage of the common structure proposed by X′ theory, there is only one `Phrase` class. This is used to represent any phrase of any type, from any language.

One attribute of the `Phrase` class is a pointer to an object of class `Head`. It is this head which determines the nature of the `Phrase` object which contains it, since in the theory an NP is an NP precisely because it is headed by an N, a CP is a CP precisely because it is headed by a C, and so on.

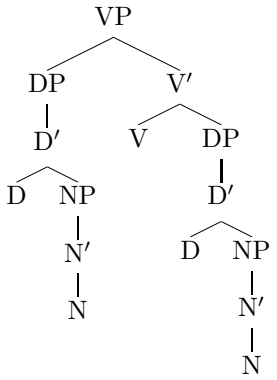
Additionally, every `Phrase` object has two pointers to other `Phrase`s, one for its complement and one for its specifier. Unlike a phrase's head, these are optional components of any phrase, so these pointers may be null. Using a single class `Phrase` allows complements and specifiers to be of any phrase type, as required by the theory. Similarly, to represent adjuncts, every `Phrase` object has a variable-length list of non-null pointers to other `Phrase`s, which may be empty because adjuncts are also optional.

As well as these 'downward' links from a phrase to its components, there are also 'upward' links; that is, each `Phrase` and `Head` object has a pointer to its parent `Phrase`. Although this makes the implementation slightly more complicated (and potentially allows for inconsistent states), it is absolutely necessary to be able to traverse the trees along any arbitrary path to perform a number of the transformations described in chapter 5.

Crucially, there is no notion of left and right branches or first and last constituents in the class `Phrase`. An X′ tree modelled with this class is an abstract network of heads, complements, specifiers and adjuncts, in no particular linear order. Some phrases can be said to be *within* other phrases, but the partial ordering derived from this relation has no bearing on the linear order of components in the final sentence; no component is *before* or *after* any other.

In order to represent the following linguistic tree, for example

(4.1)



a structure as indicated in the following object diagram would be constructed:

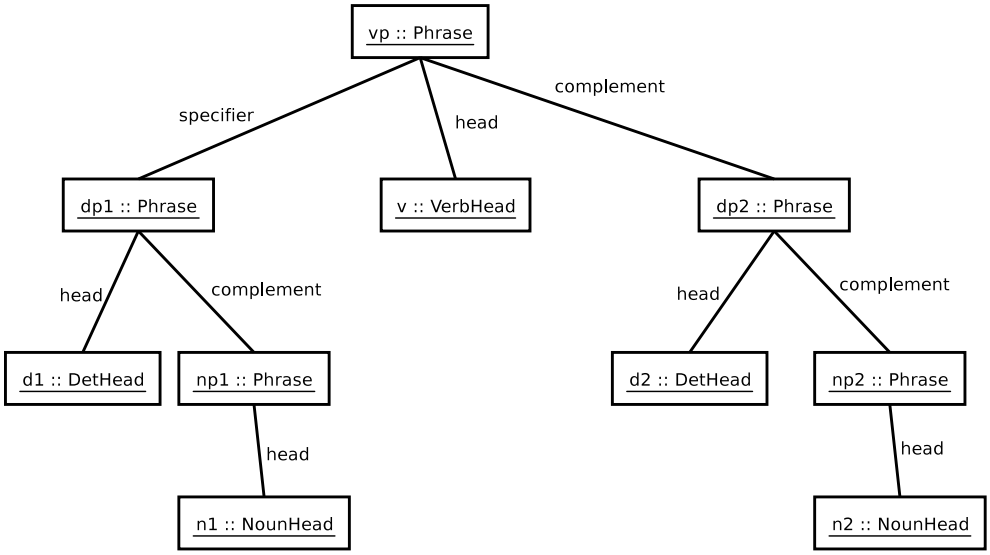


Figure 4.1: Sample object structure for a phrase

(`DetHead`, `NounHead` and `VerbHead` are subclasses of `Head`, as described below.) Even though only single links are shown on the diagram, as mentioned above they are all navigable in both directions.

Of course eventually though, a linear string must be produced for any phrase. This is done by the `getFinalForm` function, which accepts two arguments specifying the X′ directionality parameters of the desired language, and produces the appropriate string. Based on these parameters, the phrase decides which of its components appears first in the surface realisation, which appears second, and so on. In order to realise its component phrases, a phrase recursively calls the `getFinalForm` function on these component phrases; in order to realise its head, a phrase calls the `Head` class's `getInflectedForm` function, which is explained in §4.2 below.

Also recall from §3.2.2 that, under X′ theory, a sentence is nothing but a CP (complementiser phrase), with the same internal structure as all other phrase

types, so this `Phrase` class is sufficient to model linguistic trees of complete sentences.

4.2 The `Head` Class Hierarchy

There are a number of subclasses of the abstract class `Head`, and obviously an object from any of these can act as the head of a `Phrase` object. Thus a `Phrase` object represents a CP if and only if the runtime type of its head pointer is `CompHead`, for example.

The primary 'obligation' of a subclass of `Head` is to provide an implementation of the abstract function `getInflectedForm`, which is used, as mentioned above, to generate the final string realisations of each phrase. The discussion of the various subclasses which follows centres around how they achieve this.

The full hierarchy of subclasses is illustrated in the following class diagram:

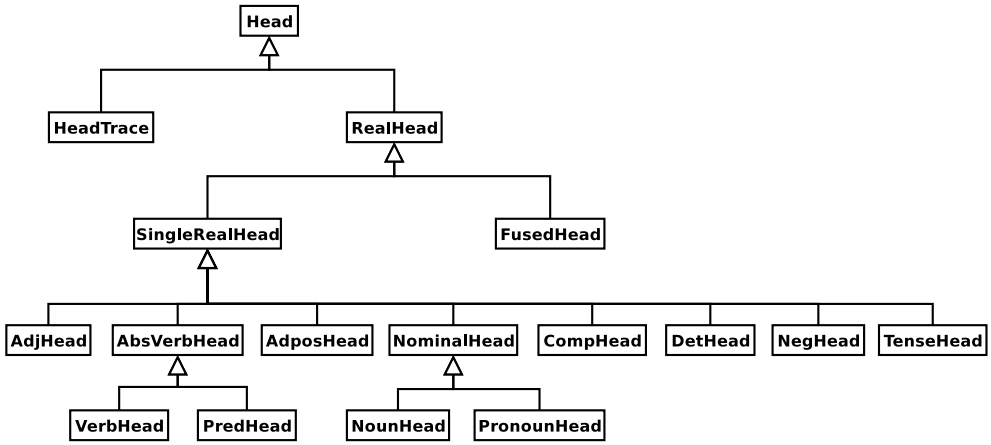


Figure 4.2: Hierarchy of subclasses of `Head`

4.2.1 Lexical Heads

There are four subclasses which represent heads of each of the four lexical categories: `NounHead`, `VerbHead`, `AdjHead` and `AdposHead`, representing nouns, verbs, adjectives and adpositions respectively. (The term **adposition** covers both prepositions and postpositions, a distinction which we have reduced to the head-directionality parameter; the linguistic properties of each are identical.)

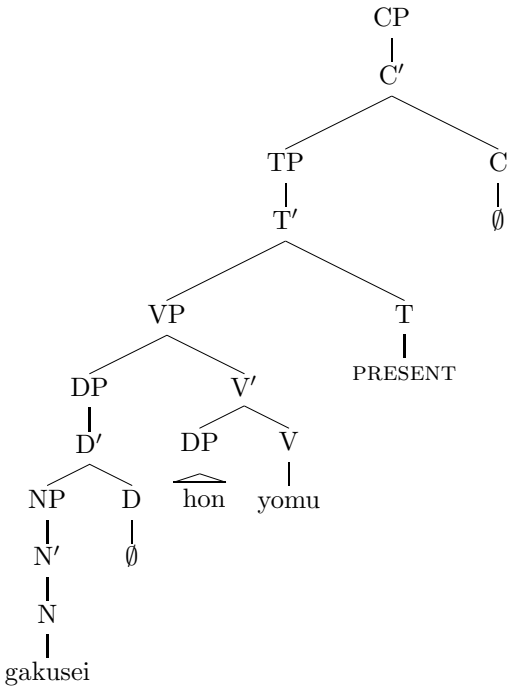
Clearly the surface realisations of these heads, and therefore these subclasses' implementations of `getInflectedForm`, depend on the lexicon of the target language. As such, an attribute of each of these subclasses is a pointer to an object representing a lexical item from the target language. There is a parallel set of classes `LexItemNoun`, `LexItemVerb`, `LexItemAdj` and `LexItemAdpos`, and each head type has a pointer to an object of the corresponding lexical item class, so an attribute of the `NounHead` class is a pointer to an object of type `LexItemNoun`, and so on. An object from one of these lexical item classes represents a particular lexical item in a particular language; for example, one `LexItemVerb` object might represent the English verb *to walk* (with its associated system of

inflections), and one `LexItemAdj` object might represent the French adjective *grand* (also with its associated system of inflections).

When the `getInflectedForm` method is called on a lexical head object, the head in turn calls the `getInflectedForm` method *of the associated lexical item object*, which will return the appropriately-inflected string. In order to determine which inflections are required though, the lexical item object needs to know about the structure of the sentence, and where the head being realised fits into it. For this reason, the head object passes a pointer to itself as a parameter to the lexical item. Thus the lexical item can examine the head which needs to be realised, in relation to its parent nodes, sibling nodes, etc., in order to determine any necessary inflections. Sometimes the correct realisation will also depend on functional information which is encoded in the `Head` object itself, for example number and gender in the case of `NounHead` objects.

Suppose for example that the following Japanese S-structure tree has been generated, and now needs to be realised as a string. Note the case markings which need to be inserted to produce the correct result (data from [30]).

(4.2) (a)



(b) *Gakusei ga hon o yomu.*
student SUB book OBJ reads
The student reads the book.

When the `getInflectedForm` method is called on the `NounHead` object representing the N node *gakusei*, the `NounHead` object will in turn call the `getInflectedForm` method of the `LexItemNoun` object representing the Japanese noun *gakusei* (*student*), passing a pointer to itself as an argument. The `LexItemNoun` object will examine the `NounHead` object it is being asked to provide a realisation for, and its position in the tree, and will discover that the DP containing this `NounHead` object is in the specifier position of the TP. This is

the position for a noun which is known in traditional grammar as the subject position. Thus the Japanese lexical item object knows that it should append the nominate case marker (or subject marker) *ga* to the stem form *gakusei*, and the string *gakusei ga* is returned, first by the `LexItemNoun` and then by the `NounHead`.

The same process allows for verb inflections (and all other syntactic markings), although verb inflections are complicated slightly by the fact that they involve the merging of two heads (as described in §3.3.2); this is explained in §5.2.4.

4.2.2 Functional Heads

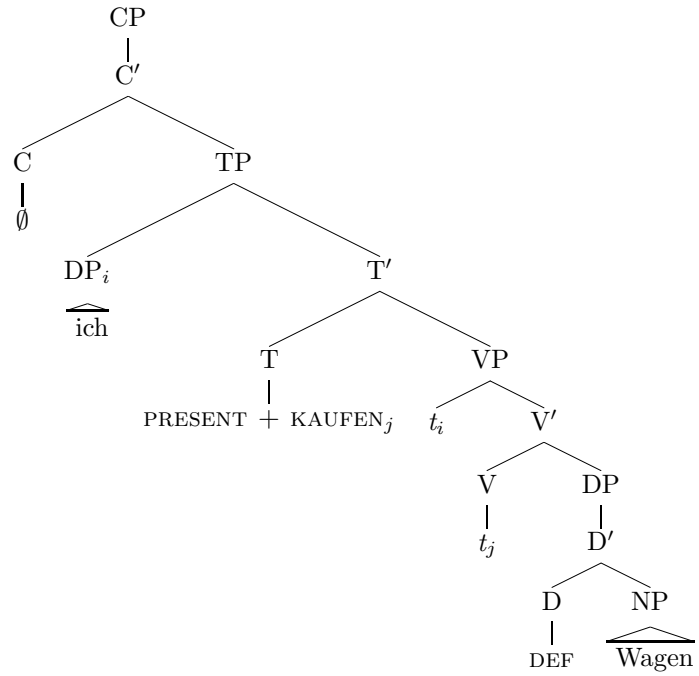
Other subclasses of `Head` are used to represent what the theory calls 'functional heads', such a complementisers (`CompHead`), determiners (`DetHead`) and tense inflections (`TenseHead`). Rather than being associated with particular lexical items, the realisations of these heads constitute the 'closed word classes', such as those words which are traditionally known as articles and auxiliary verbs, for example.

Instead of a pointer to a `LexItem`, then, these heads have a pointer to a `Generator` object, which essentially represents a language. There are subclasses of the abstract class `Generator` such as `EnglishGenerator`, `FrenchGenerator` and so on (see §5.1). When the `getInflectedForm` function is called for these heads, the head calls an abstract function of the `Generator` class. The `DetHead` class, for example, calls the `getDeterminer` function, and the `CompHead` class calls the `getComp` function. Each subclass of `Generator` implements these functions so as to return the correct 'function words' for its particular language.

As in the case of the lexical heads, the correct realisation may depend on the head's position in the tree, so the functional head also passes a pointer to itself as a parameter to the corresponding `Generator` function. It is also common for functional heads to have features encoded on them such as definiteness in the case of `DetHead`s and tense and voice in the case of `TenseHead`s, and these features will also be needed by the `Generator` function to determine the correct resulting string.

For example, consider the realisation of the definite article *den* in the following German example:

(4.3) (a)



(b) *Ich* *kaufe* *den* *Wagen.*
I buy the.MASC.ACC car
I buy the car.

When the `getInflectedForm` function is called on the `DetHead` object representing the definite article, the `DetHead` object will call the `getDeterminer` function of the `Generator` object with which it has been associated; in this instance, it will be an implementation of the `Generator` interface for the German language, most likely from a subclass called `GermanGenerator`. (Note that German has not currently been implemented, but German provides a good example for this discussion.) German determiners depend on definiteness, the gender of the associated noun, and case. The `getDeterminer` function receives as an argument a pointer to the `DetHead` object, from which it can discover the definiteness of the required determiner. It can also examine the determiner's position in the tree, and, finding that its parent DP is in the complement position of a VP, will decide that an accusative case determiner is required; and finally, it can look at head noun of the D's complement NP to find the required gender. Based on these three properties, the `getDeterminer` function can then return the correct string, *den*.

4.2.3 Movement and Traces

The implementation needs to allow for the movement of heads from one location to another, as in §3.3.2 for example. Clearly the phrase whose head moves away still needs to be represented by a `Phrase` object, and a `Phrase` object necessarily contains a non-null pointer to a `Head` object. Therefore a subclass of `Head`, `HeadTrace`, is provided which acts as a placeholder. A `HeadTrace` object corresponds to the t which indicates a trace on linguistic trees. In the

linguistic theory, traces are phonologically null elements with no realisation, so the `HeadTrace` class's implementation of `getInflectedForm` simply returns an empty string. The head which has moved will be realised as usual in its new position.

A link needs to be maintained though between a head and its original location. Sometimes it is important, in making syntactic decisions, to know what a trace has been left by, and where the original head went. In particular, when, for example, a `VerbHead` object is moved, the phrase which contained it originally needs to maintain its status as a VP, even though its head position is now occupied by an object of class `HeadTrace`. For this reason, there is also a two-way link between a `HeadTrace` object and the `Head` object (now in its new position) which it replaced; each has a pointer to the other.

Note that since `HeadTrace` is a subclass of `Head`, the pointer in a `HeadTrace` which indicates the new location of the moved head may point to another `HeadTrace` object. Thus a head can move any number of times, leaving behind it a trail of `HeadTrace` objects which essentially form a doubly-linked list.

A similar mechanism is provided for the movement of entire phrases (as is required for the DP-movement introduced in §3.3.2, for example) using a class `PhraseTrace`.

4.2.4 Merging of Heads

The implementation also needs to allow for two heads to merge together. The merge operation involves one head moving to fuse with another, and the two then jointly occupy the position previously occupied by the latter. As with traces, the `Phrase` object which then contains this new joint head must have a valid non-null pointer to a `Head` object, so another subclass of `Head`, `FusedHead`, is provided. A `FusedHead` object essentially consists of a set of pointers to other `Head` objects, and fills the head position of the phrase. The head which originally filled that position is detached from its parent phrase and attached to the `FusedHead` object instead.

In the same way that heads and phrases have links to their parent phrases, heads which have merged have links to their parent `FusedHead`s. Thus, in the same way that a head's position relative to its surrounding nodes may be analysed in order to decide how it should be realised, any heads it has merged with may also contribute to this decision.

The `FusedHead` class's implementation of `getInflectedForm` simply calls the `getInflectedForm` method of each of its component heads and concatenates the results. The order in which the component heads appear in the resulting string is arbitrary, but in any set of heads which have merged into one, at most one is ever realised, so order is irrelevant. For example, if a `TenseHead` has merged with a `VerbHead`, it will not be realised (i.e. the `TenseHead`'s `getInflectedForm` method will return an empty string), but, in a language with verb agreement, the fact that these two heads have merged will mean that the `VerbHead`'s `getInflectedForm` function returns a correctly inflected result.

The relationship between the component heads of a `FusedHead` is not totally symmetrical though, because, as in the case of traces above, a phrase which is now headed by the `FusedHead` must maintain its status as a phrase of a particular category. For this reason one of the pointers in a `FusedHead` is specified to be the 'original head' in that position.

To illustrate the use of `HeadTrace` and `FusedHead`, the following object diagram shows the transformation which would take place to represent the merging of a V with a T, as in (3.60):

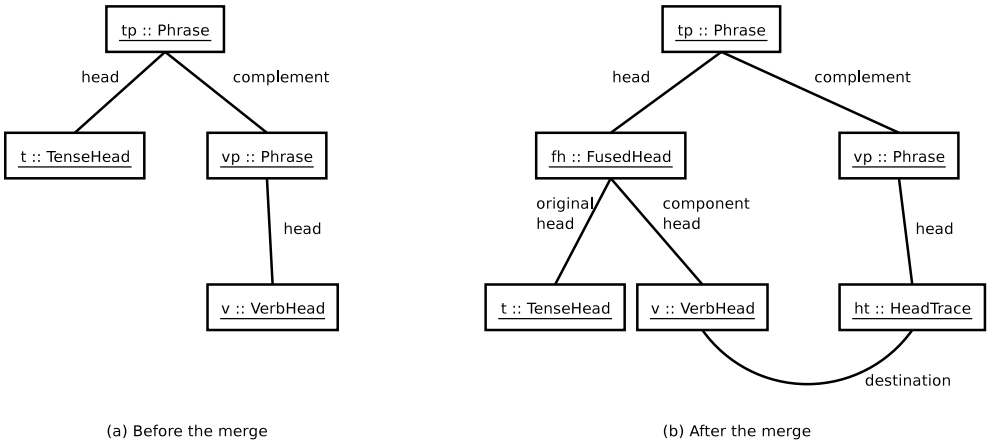


Figure 4.3: Transformation on the object structure to represent the merging of two heads

4.2.5 `PronounHead` and `PredHead`

These two categories of heads share some properties of both the lexical heads and the functional heads. (Pred is the head which represents what is traditionally known as the 'verb' in sentences such as 'Chris is a teacher' and 'Chris is tall'.)

The realisations of these heads do not depend on any particular lexical items, so in this respect they ressemble functional heads. However in many ways they behave in the same way as certain lexical categories. `PronounHead`s occur in exactly the same positions as `NounHead`s do, and can have the same properties encoded on them, such as number, person and gender. `PredHead`s occur in exactly the same positions as `VerbHead`s, and interact in the same way with `TenseHead`s, for example.

Thus the classes `NominalHead` and `AbsVerbHead` (for 'abstract verb head') were introduced. `NominalHead` is an abstract subclass of `Head` which captures the linguistic properties shared by nouns and pronouns; an object of this class can thus have features such as number and person encoded on it. Both `NounHead` and `PronounHead` share these properties, so these are concrete subclasses of `NominalHead`. They differ primarily in their implementations of `getInflectedForm`; `NounHead` relies on a lexical item, as described above, while `PronounHead` acts more like a functional head in this respect. For more detail on the use of `PronounHead`, see §5.3.4.

Similarly, `AbsVerbHead` is an abstract subclass of `Head` which captures the linguistic behaviour associated with all surface forms which are known in traditional grammar as 'verbs', such as the ability to merge with a `TenseHead`. `VerbHead` and `PredHead` are both derived from it, and again their main difference is in their implementation of `getInflectedForm`. `VerbHead` relies on a lexical item, whereas `PredHead`, in exactly the same way as `DetHead`, for exam-

ple, calls `getDeterminer`, calls the `getPred` abstract function of the `Generator` class. For more detail on the use of `PredHead`, see §5.3.3.

4.3 Design Issues

This implementation of the X′ framework has given rise to two features of the code which are often considered potential signs of bad design. I believe that in this instance though, these techniques were justified.

Firstly, the class `SingleRealHead` has a method `getCategory()`, which returns NOUN in the case of a `NounHead`, COMP in the case of a `CompHead`, and so on. This is mostly used to determine the category of a phrase via the 'wrapper' function `getCategory()` of the `Phrase` class. Strict object-oriented design suggests that polymorphism is a better alternative to type-revealing methods like this, because it avoids the need to add additional cases to large groups of `if` statements when a new class is added to the hierarchy.

The `getCategory()` method is never used to choose between a large number of alternatives like this though; it is only used to check for one specific category of head (or, very rarely, one of two categories). For example, in *wh*-questions, a DP may need to move to the specifier position of CP. To do this, clearly the CP must be found, so the tree is traversed and the category of each phrase is examined, until a phrase with head of category COMP is found. The precise category is not important in this search, all that is important is whether the phrase's head is of category COMP or not, so introducing another class to the hierarchy of heads would not require any modification of this code.

Also, it should be noted that there is not a one-to-one relationship between the categories which are returned from `getCategory()` and the subclasses of `SingleRealHead`. 'True' polymorphism is used in the relationship between `NominalHead` and its subclasses `NounHead` and `PronounHead`. There are abstract methods which the two subclasses implement differently, but, significantly, the `getCategory()` method is implemented in the base class, `NominalHead`, and not overridden. Syntactically, nouns and pronouns are in a sense equivalent, and belong to the same category (known as NOUN, and represented by N on tree diagrams).

Thus similarities in the linguistic properties of nouns and pronouns are captured by the relationships between `NounHead`, `PronounHead` and `NominalHead`. However, there are no such similarities between the other subclasses of `SingleRealHead`. The reason that `VerbHead`, `AdposHead`, `CompHead`, etc. are also derived from `SingleRealHead` is purely to allow any of them to be embedded in a `Phrase` object. (Exactly the same argument can also be applied to `AbsVerbHead`, `VerbHead`, `PredHead` and the category VERB, since these share exactly the same relationships as exist between `NominalHead`, `NounHead`, `PronounHead` and the category NOUN, respectively.)

To deal with the example of searching for a CP above, one could instead use an abstract method of the `SingleRealHead` class called, say, `moveToSpecForWh` which returns a boolean value indicating whether or not this phrase is the appropriate position to move to; a collection of methods like this would avoid the need for `getCategory`. It is just as simple, though, to consider the fact that it is the target position for *wh*-movement to be one of the properties represented by the linguistic category COMP, return COMP from `getCategory`, and let the

caller interpret this. In fact, this is simpler than using a collection of abstract methods, because there is no need to add a method to the class hierarchy every time a new such distinction needs to be made between categories.

In summary, the category to which a (phrase's) head belongs is a genuine piece of linguistic information; it is not an implementation detail.

Secondly, the `dynamic_cast` operator is used relatively frequently; this allows safe downcasting from a base class to a derived class, but the very need to do this is also sometimes considered a sign of bad design. I have used it when a `Phrase` or `Head` object of the desired category has been found, using the `getCategory` method, and then access to methods specific to that category is needed. For example, to decide on a correct determiner, a `getDeterminer` function may browse to the appropriate position in the tree and find the `NominalHead` representing the corresponding noun. That a head of the correct category has been found can be checked using `getCategory`, but then methods specific to the `NominalHead` subclass need to be called, such as `getNumber` perhaps. Thus the `Head` pointer must be downcast to a `NominalHead` pointer, because the `getNumber` method is not available in the abstract `Head` class.

Significantly though, `dynamic_cast` is never used without first using `getCategory` to check that the cast can be performed successfully; and it is never used to downcast to one of the 'sub-category' types (i.e. `NounHead`, `PronounHead`, `VerbHead` and `PredHead`), only ever to the superclasses representing their true linguistic categories, `NominalHead` and `AbsVerbHead`.

The need to use `dynamic_cast` is thus another result of the fact that inheritance from the class `Head` has been used only to capture common structure, not common functional linguistic properties.

Chapter 5

Implementation of the Realisation Process

This chapter draws together the theory from chapter 3, and the underlying framework implemented as described in chapter 4, to present the overall method I have developed to produce the correct syntactic realisation of the input semantics.

5.1 The `Generator` Class

The abstract class `Generator` represents an entity which is capable of performing syntactic realisation in a particular language. For each language available to the system, a concrete subclass of `Generator` is provided, called, for example, `EnglishGenerator` or `JapaneseGenerator`.

The 'obligations' of these subclasses include, firstly, as was mentioned in §4.2.2, providing implementations of the functions which provide the 'closed word classes' such as determiners, pronouns and auxiliaries.

Secondly, a language must provide an implementation of the `initLexicon` function. This function must initialise the 'dictionary' representing the language's lexicon. It takes the form of a mapping from abstract semantic concepts to lexical items.

Thirdly, there are a set of abstract functions representing the discrete parameters of syntactic variation, such as the X′ directionality parameters, the verb-inflection parameter and the subject-movement parameter, as discussed in chapter 3. For example, an implementation of the abstract function `getHeadDirection` must return one of the symbolic constants HEAD_FIRST or HEAD_LAST; each language generally only requires trivial one-line implementations of such functions.

The crucial part of the `Generator` class is the `generate` function, which accepts as an argument the semantic representation of the desired sentence, and returns the final sentence as in the desired language as a string. This function is not abstract, but is implemented in the base `Generator` class. It defines the skeleton of an algorithm which applies to syntactic realisation of any sentence in any language. The parts of this algorithm which are language-specific are

performed by calling the abstract 'hook' methods which each language has provided an implementation of. This is an instance of the 'template method' design pattern. (The fact that `generate` is a template method is somewhat obscured by the fact that many of the calls to `Generator`'s hook functions are actually made from inside the various `Head` classes, in the process of performing the movement transformations which occur between D-structure and S-structure; since this process is driven by the `generate` function though, this is only a notional difference.)

5.2 Basic Procedure

This section traces through the syntactic realisation process from start to finish. Starting from a common semantic input, the corresponding sentence will be generated in two languages, English and French, and the two will be explained in parallel, to emphasise the general nature of the overall algorithm. Once this process has been explained, the next section will introduce a number of variations on it which allow more complex sentences to be realised.

The English sentence which will be generated here and its French equivalent are

(5.1) A man missed the boy.

(5.2) *Le garçon a manqué à un homme.*
the boy has lacked to a man
A man missed the boy.

These sentences have a somewhat artificial ring to them, but will demonstrate the relevant points well (and are, in any case, undoubtedly grammatical).

5.2.1 Semantic Input

The input from the user which describes the semantics of the desired sentences can be represented as follows:

(5.3)

MISS	
EXPERIENCER	`literal`(MAN, SINGULAR, INDEFINITE)
THEME	`literal`(BOY, SINGULAR, DEFINITE)
`tense`	PAST
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

Recall that labels such as MISS, MAN and BOY do not refer to particular lexical items, but rather to abstract concepts, in a language-independent manner. The labels used in this report have obviously been derived from English, but the structure represented above is truly language-universal.

The first line of this structure is the concept which constitutes the predicate, or event, which is being described. In this case, the sentence has to do with the concept of 'missing'.

The next two lines describe the arguments to this predicate, or the participants in the event. (There can be any number of these; in this example

there are two.) Each is associated with a semantic role. Both of the arguments in this example are `literal` arguments; this means that they refer to simple real world objects (other types of arguments, such as pronouns and *wh*-question arguments, will be introduced later), in this case, the objects associated with the abstract labels MAN and BOY. Both are singular in number; one is definite and the other is indefinite. The possibile alternatives to SINGULAR here are DUAL and PLURAL; English (as well as French) happens to treat dual and plural nominals the same way, but this range of inputs allows for a language-universal representation, because in some languages that distinction is important. (It would also be trivial to add more alternatives to the system, such as paucal, representing a small number greater than two.)

The fourth line indicates that the event being described should be considered to have occurring 'in the past' relative to when the sentence is being generated.

The last three lines of the structure state that the sentence should not be negated, should not be a question, and should be expressed in the active voice. These features will be explained in more detail later, and are included here mainly for completeness.

The class `AbstractSentence` is used to represent structures of this sort.

5.2.2 Building the D-structure

In this example, the user has requested that the above structure be realised in both English and French, so a pointer to an `AbstractSentence` object with this information is passed as an argument to the `generate` function of an `English-Generator` object and a `FrenchGenerator` object.

The first step of the `generate` function is to search in the language's lexicon for a lexical item which can be used to represent the desired predicate concept, MISS. In the case of English the verb *miss* will be found; in the case of French, the verb *manquer*. Associated with each of these verbs will be the following information (which corresponds roughly to what is known in linguistics as a 'theta-grid'; and is represented in the program by the class `ThetaGrid`):

(5.4)

miss	
EXPERIENCER	SPEC_VP
THEME	COMP_VP

(5.5)

manquer	
EXPERIENCER	COMP_COMP_PP, *à*
THEME	SPEC_VP

Recall from §3.1 that part of the 'meaning' of a verb is the way it links itself syntactically with its arguments; that is what is being expressed here. For each argument to a predicate, such as those given in (5.3), a DP will be generated (see below). Stored with the English verb *miss* is the fact that the DP representing an argument with the experiencer semantic role should be generated in the specifier position of the VP at D-structure (represented by SPEC_VP), and that that for a theme argument should be generated in the complement position (COMP_PP). Similarly the DP for a theme argument to the French verb *manquer* should be placed in the specifier position of the VP; and the DP for an experiencer should be placed in the complement position of a PP headed by the adposition

à, and this PP should in turn be placed in the complement position of the VP (COMP_COMP_PP, *à*). (The fourth and final possible position in which a verb can specify a DP should be placed is COMP_ADJ_PP, meaning the complement position of a PP which is an adjunct to the VP ('complement of an adjunct PP'). Like COMP_COMP_PP, an adposition must be supplied with it. The system does not allow for ditransitives using the idea from linguistics of vP shells.)

Now, for each argument in the `AbstractSentence` representing (5.3), the generator looks for a line in these verbs' theta grids which desribes where to place an argument with the appropriate semantic role. Thus the arguments can be associated with a position in the D-structure where they need to be placed:

(5.6)

miss	
`literal`(MAN, SINGULAR, INDEFINITE)	SPEC_VP
`literal`(BOY, SINGULAR, DEFINITE)	COMP_VP

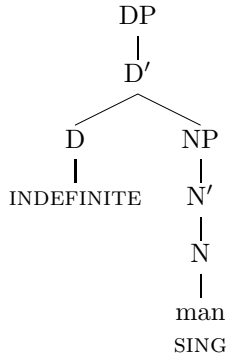
(5.7)

manquer	
`literal`(MAN, SINGULAR, INDEFINITE)	COMP_COMP_PP, *à*
`literal`(BOY, SINGULAR, DEFINITE)	SPEC_VP

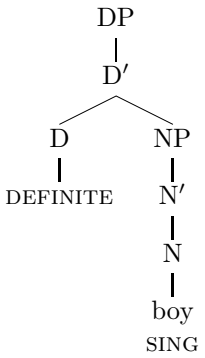
Next, a DP must be generated for each of these arguments. In the case of `literal` arguments, the lexicon is searched for a lexical item corresponding to the required concept. A search in the English lexicon for the concepts MAN and BOY will yield the nouns *man* and *boy*; French will find *homme* and *garçon*. (Currently the system requires that the lexical items found here be nouns (i.e. objects of the class `LexItemNoun`); potentially it could be modified to allow systematic nominalisation of lexical items from other categories.)

In each case, an NP is constructed with a `NounHead` object associated to the lexical item (as in §4.2.1) as its head. The number (singular for both arguments in this example) is encoded on this `NounHead` object. The NP is then placed in the complement position of a DP, and the type of determiner (for now, either DEFINITE or INDEFINITE) is encoded on the `DetHead` of this DP. This produces the following four DPs:

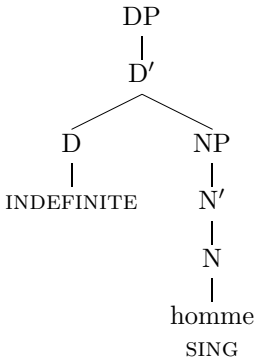
(5.8)



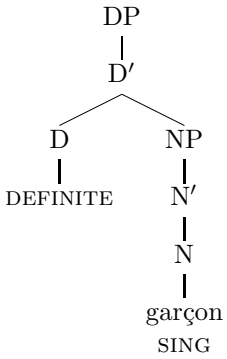
(5.9)



(5.10)



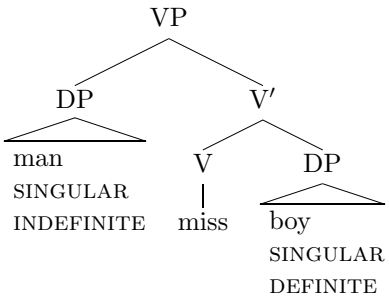
(5.11)



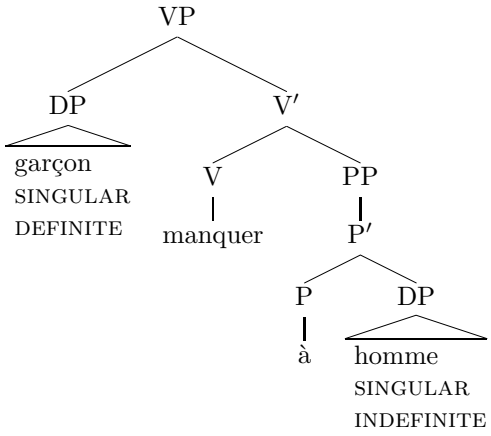
Note that once a noun lexical item in the target language has been found, the process for building the corresponding DP is *completely independent* of the target language and of the lexical item involved. This is the sort of procedure which is implemented just once, in the `Generator` class, and is used to generate in any language.

Putting these DPs in their appropriate positions, as per (5.6) and (5.7), we can construct complete VPs.

(5.12)

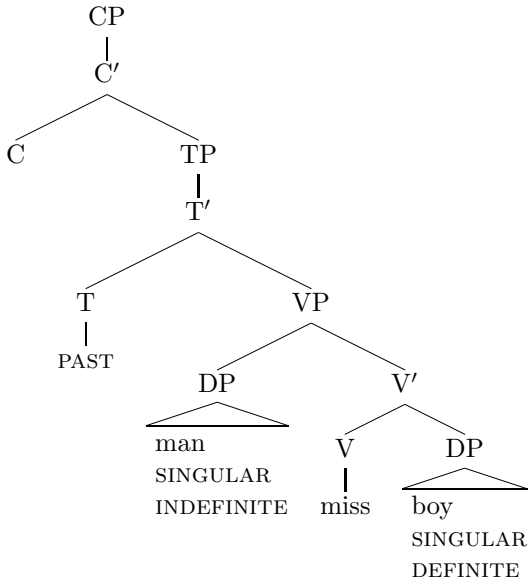


(5.13)

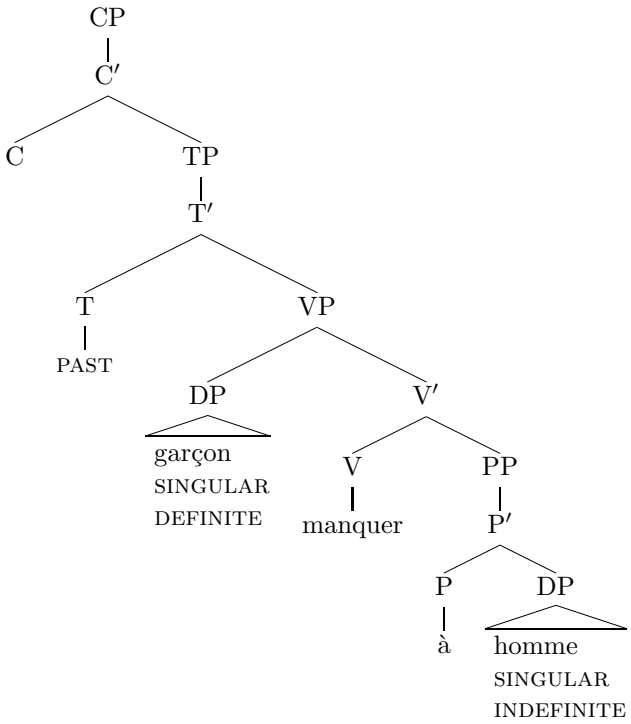


Finally, these VPs are 'wrapped up' in a TP and a CP. The PAST feature from (5.3) is encoded on the `TenseHead` object.

(5.14)



(5.15)



These are now complete D-structures. All the relevant semantic information has been encoded on these trees, and the structure in (5.3) can now be 'thrown away'; syntactic transformations will do everything required to convert the D-structures to S-structures and then to realised sentences.

5.2.3 Converting D-structure to S-structure

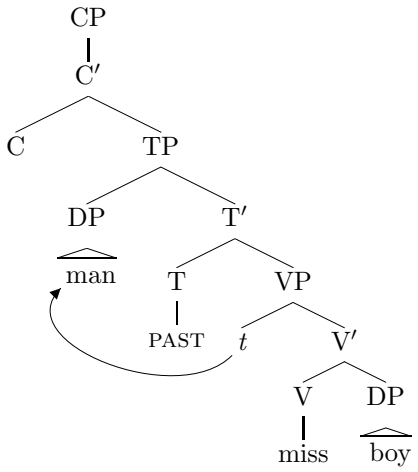
The 'subject movement parameter' introduced in §3.3.2 was a slight simplification of part of case theory. In more general terms, every DP must be assigned an abstract property known as Case in order for it to 'surface' and be realised. Case is assigned in certain positions: it is assigned to the complement position of a VP, to the complement position of a PP, and one other position. This other position is subject to variation; in some languages, it is the specifier position of a TP, in others, it is the specifier position of a VP. If a DP is not assigned Case in its D-structure position, it must move to a position where it can be assigned Case, or it can not be realised. Thus the 'subject movement parameter' reduces to whether Case is assigned to the specifier position of VP (in which case no movement is required) or to the specifier position of TP (in which case a DP in the specifier position of VP will have to move).

Movement of DPs to ensure they have case (if required) is the first transformation performed by the `generate` function on D-structures. Each DP which was generated from an argument is examined to check whether it is assigned Case in its current position, if not, it is moved to a position where it will be. Note again that the skeleton algorithm for this process (looping through each of the generated DPs, checking, and moving) can be specified in some detail in the language-universal `Generator` class itself, parametrised just by which position a language assigns Case in. The value of this parameter is determined by

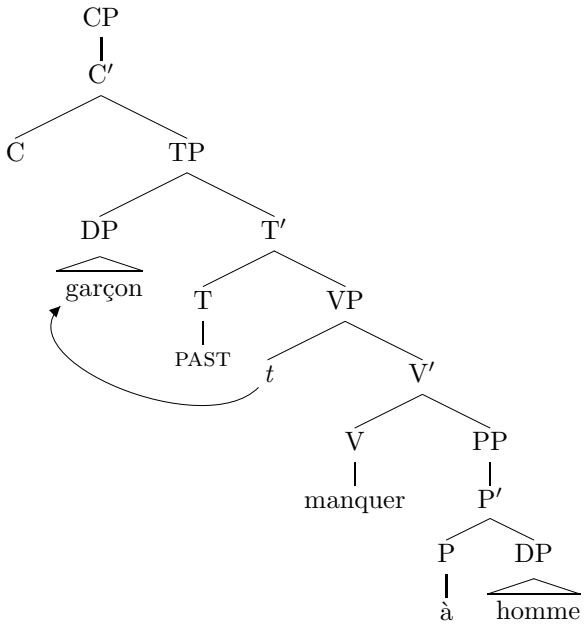
calling the abstract `getNomCaseParam` function, which is implemented by each language to simply return a value indicating whether TPs or VPs assign Case to their specifiers.

In both English and French, the TP assigns Case to its specifier rather than the VP (as seen in 3.3.2), so the DPs which currently occupy the specifier of VP position must move:

(5.16)



(5.17)



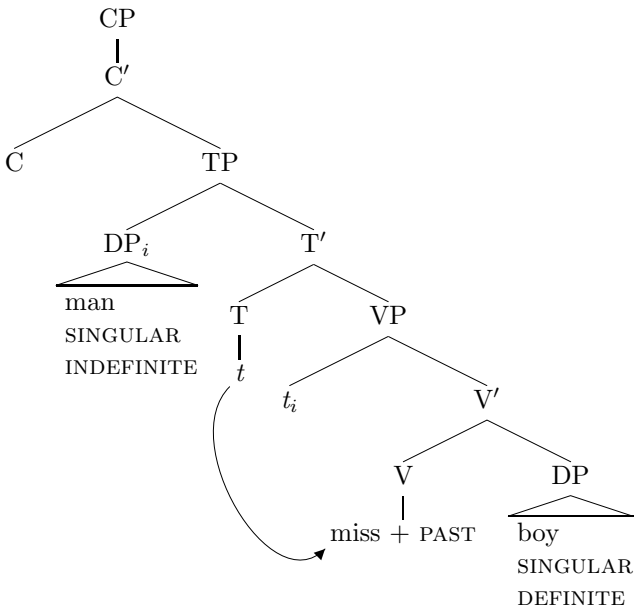
The other DPs, being in the complement positions of a VP and a PP, are assigned Case in their current positions, and so do not need to move.

The next general step in the process is to ensure that the features encoded on the `TenseHead` will be realised in some way. (If they are not, then not all the semantic information will be expressed.) A `TenseHead` can realise its features either by being overtly realised itself as an auxiliary verb, or by merging with a `VerbHead` and having the verb's inflection realise its features. If the

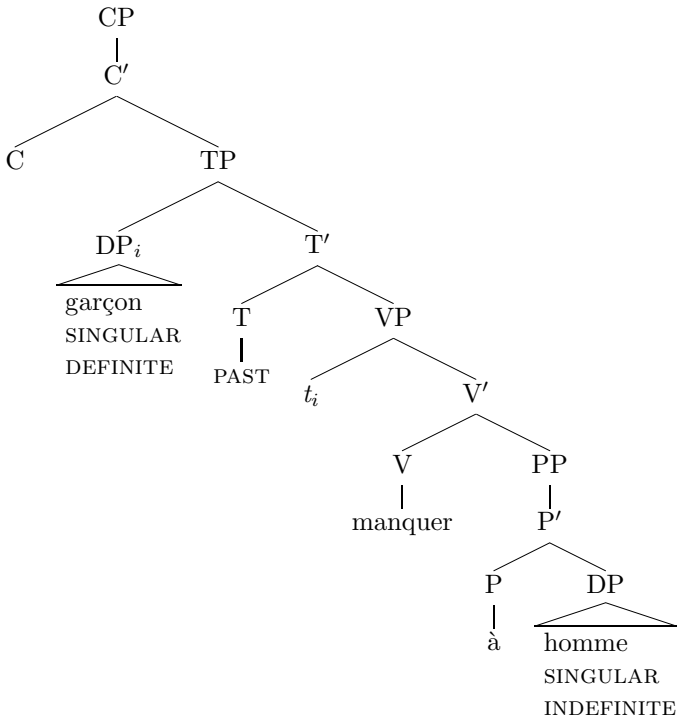
tense/voice/etc. of the sentence is such that no auxiliary verb will be used, the `TenseHead` will have to merge with the `VerbHead`. Which of these two heads moves to join the other is a parameter, as seen in §3.3.2.

The only English past tense which is implemented is the 'simple past', which does not involve an auxiliary, and the only French past tense which is implemented is the 'perfect', which does involve an auxiliary. (This selection of past tenses was chosen in part to demonstrate this divergence in constructions.) Therefore, in the French example, no merging of the V and T is required. In the English example, however, the merge is needed; the abstract `getVerbInflParams` function of the `Generator` class is called to check whether the V is raised or the T is lowered, and the `EnglishGenerator` implementation of this function returns a value indicating that the T should be lowered. After ensuring that the `TenseHead`'s features will be realised, we thus have the following two trees:

(5.18)



(5.19)

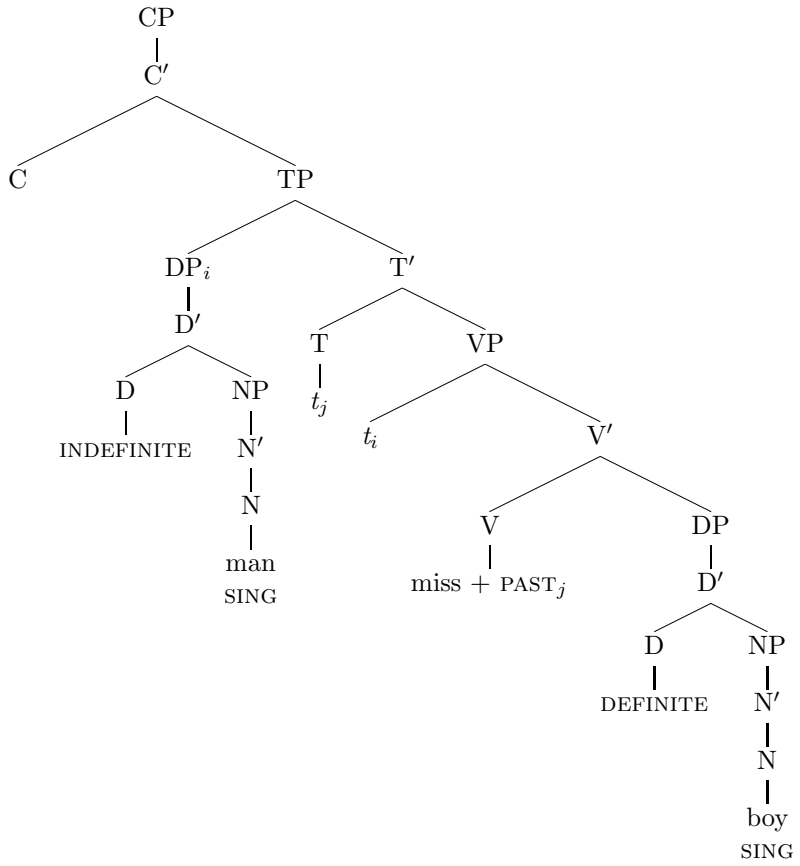


Again it should be emphasised that these transformations are motivated by language-universal factors such as the need for DPs to be assigned case and the need for heads to have all their features realised; non-trivial skeleton algorithms for performing these transformations, calling the language-specific hooks when necessary, have been implemented in the `Generator` class or in the X′ module classes, and can be used by every language.

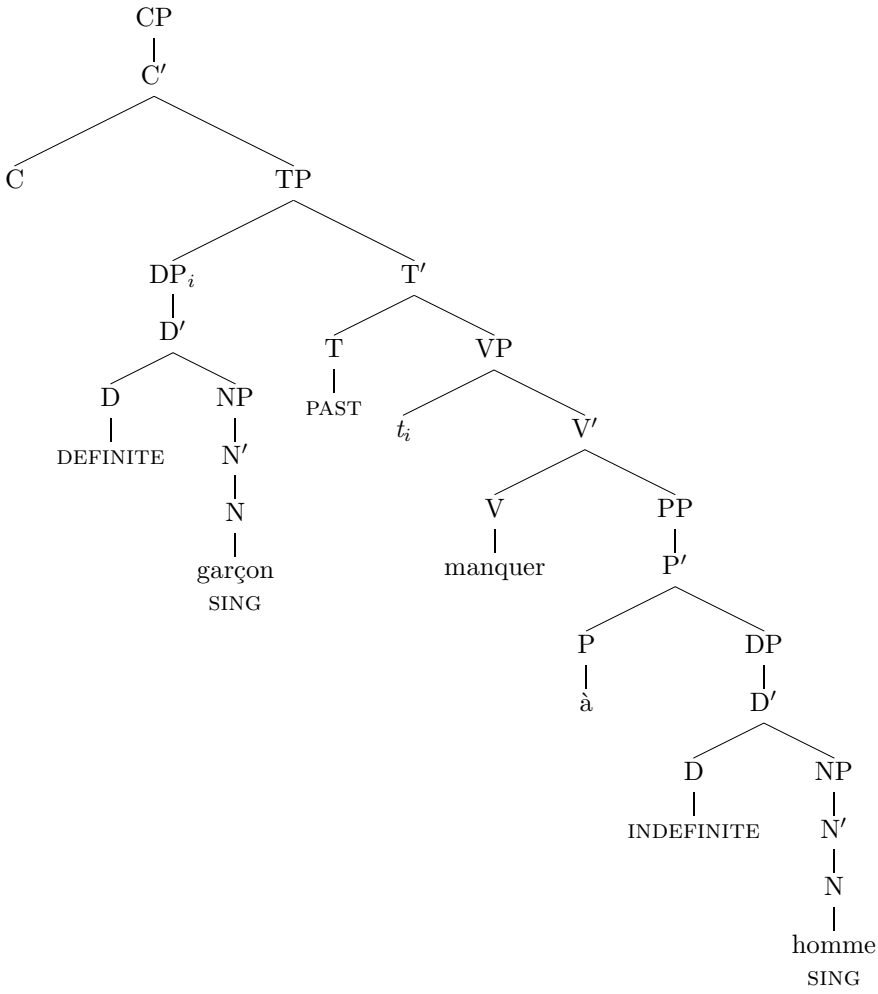
5.2.4 Surface Realisation from S-structure

After applying the necessary transformations to the two D-structures, we have derived the following S-structures. Internals of the DPs have been expanded, to show all relevant details.

(5.20)


CP
C′
C
TP
DP$_i$
T′
D′
T
VP
D
NP
t_j
INDEFINITE
N′
t_i
V′
N
V
DP
man
SING
miss + PAST$_j$
D′
D
NP
DEFINITE
N′
N
boy
SING

(5.21)



To produce the surface realisations of these structures, `generate` calls the `getFinalForm` function of the `Phrase` object representing the 'root' CP. The X′ directionality parameters for the target language are retrieved using the abstract functions `getHeadDirection` and `getSpecDirection`, and these are passed as arguments to `getFinalForm`, as described in §4.1.

English and French share the same X′ directionality parameters; they are both head-initial and specifier-initial. Thus in both languages, when the root CP is realised, it first realises anything in its specifier position; in both the structures above, the CP's specifier position is empty though, so nothing is generated. Next the head C is realised; this is done by calling the `getInflectedForm` function on the `CompHead` object representing it, which, as seen in §4.2.2, will call back to the abstract `getComp` method of the `Generator` object. Neither English nor French has overt complementisers in main clauses though, so both `EnglishGenerator` and `FrenchGenerator` will return an empty string from `getComp` in this case. The third and final component of the CP to be realised is its complement, the TP, so the `getFinalForm` function is called on this `Phrase` object, with the same X′ directionality parameters as were passed to the CP.

Again the first component to be realised is the specifier, and in the case of the TP this is non-null, so `getFinalForm` is called on the `Phrase` object representing the TP's specifier DP. These recursive calls to `getFinalForm`, always passing the same X′ directionality parameters at each level, continue throughout the tree structure.

In realising this DP, a form must be found for the head D. As with the head C above, the `getInflectedForm` function is called on the `DetHead`, which calls the abstract `getDeterminer` function of the `Generator` class. Recall that the `DetHead` object is passed as an argument. The `EnglishGenerator` implementation of `getDeterminer` will examine the definiteness encoded on the `DetHead`, and, finding it to be INDEFINITE, will return the definite article *a*. The `FrenchGenerator` version is more complicated. It will also take the definiteness into account, but in addition depends on the gender of the French noun it is modifying, *garçon*. To find this, it calls the `getNounClass` function of the `NominalHead` which heads the complement NP, and this function will essentially return the French lexical item for the noun *garçon* (see §5.3.4 for more details of the `getNounClass` function). This noun belongs to the class of masculine nouns in French, so the masculine definite article will be returned *le*.

Next the DP's complement NP must be realised; its only component is the head N. When the `getInflectedForm` function of the `NounHead` object is called, recall from §4.2.1 that it will call the `getInflectedForm` function of the `LexItemNoun` with which it is associated, passing itself as an argument. Both the `LexItemNoun` objects, representing the nouns *garçon* and *man*, will examine the number encoded on the `NounHead` object and, finding it to be SINGULAR, will return the appropriate singular forms *garçon* and *man*.

Thus so far the two 'subject' DPs have been realised as *a man* and *le garçon*.

In the English structure, the next head to be realised is the `FusedHead` object resulting from the merged V and T nodes. (Recall that traces are not overtly realised.) This `FusedHead` will call the `getInflectedForm` function of each of its component heads and concatenate the results. The `TenseHead` object will call the `getAuxiliary` function of the `Generator` class; this function will examine the calling `TenseHead` object and, finding that it has merged with a V (an object of class `AbsVerbHead`), will return the empty string. The `VerbHead` object, however, will call the `getInflectedForm` of its associated `LexItemVerb` object; this function will examine the calling `VerbHead` object and, finding that it has merged with a T, will return a form of the verb which reflects the tense features encoded on the `TenseHead`, namely the simple past *missed*.

In the French structure, on the other hand, the next head to be realised is the isolated `TenseHead` object. `FrenchGenerator`'s implementation of `getAuxiliary` will discover that the T has not merged and is to be used in a past tense construction; therefore an auxiliary derived from the verb *avoir* is required. The required form depends on the person and number of the 'subject', so the tree will be traversed to find the DP in specifier position of TP (this being the 'subject' position in French), and then the `NominalHead` object that heads its complement NP. The person and number features encoded on this `NominalHead` are retrieved (third person singular; details in §5.3.4), and the correct auxiliary, *a*, is then returned.

The next component of the French structure to be realised is the isolated `VerbHead` object. Since the calling `VerbHead` object has not merged with a T, the lexical item's `getInflectedForm` function will decide that a non-finite

form is required. The `TenseHead` which dominates the V will be examined and based on the PAST feature encoded on it, the lexical item decides that the past participle *manqué* is required.

Thus now we have *a man missed* and *le garçon a manqué*; in each case what remains to be realised is the complement of the VP.

In the French structure the next head to be realised is the `AdposHead`. As for nouns and verbs as shown above, this head will call the `getInflectedForm` function of the `LexItemAdpos` object representing the preposition *à*. Since adpositions in French (as in most languages) are not subject to any further inflections, this function simply returns the string *à*.

Finally the remaining DPs must be realised; the process exactly the same manner as for the 'subject' DPs. The English head D is realised as *the*, and the English N is the singular *boy*; the French head D results in the masculine indefinite article *un*, and the head N in the singular *homme*.

The call to `getFinalForm` on the `Phrase` object representing the root CP thus returns the following complete sentences, as required, having concatenated the results of each recursively-realised component:

(5.22) A man missed the boy.

(5.23) Le garçon a manqué à un homme.

5.3 Technicalities/Variations

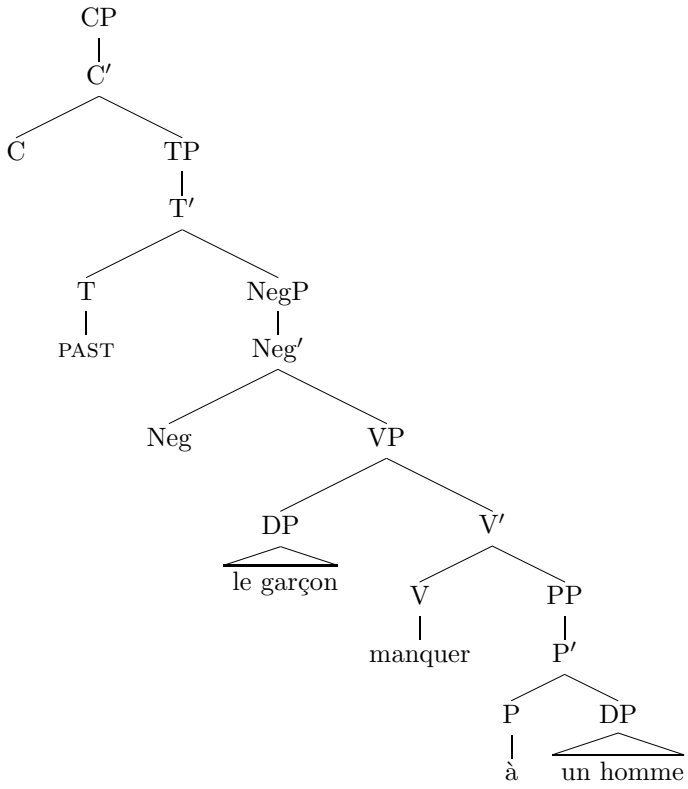
The previous section described in detail the complete process which is used to generate a relatively simple sentence. This section presents some variations on this basic process which produce other, more complex sentences.

Significantly, most of the variations illustrated here occur independently. There is no special theory for passive questions, for example; these just involve the standard passive transformations and the standard question transformations. This means that during the implementation of the system, each of these variations could be considered and coded independently, and generally, as soon as they worked in isolation, combined without any extra effort with all other features of the system (cf. §2.1.5). (Two exceptions are the interaction between negation and V/T merging and between T-to-C movement and V/T merging, but these are well-documented and researched; see §5.3.1 and §5.3.5 respectively.)

5.3.1 Negation

A sentence can be negated by setting the negative 'flag' on the input, indicated on the fifth line of the sample input in (5.3), to TRUE. This causes a phrase headed by the functional head Neg to be inserted 'between' the TP and the VP when the D-structure is generated. For an input otherwise the same as (5.3) but with the negative attribute set to TRUE, the French D-structure generated would be

(5.24)



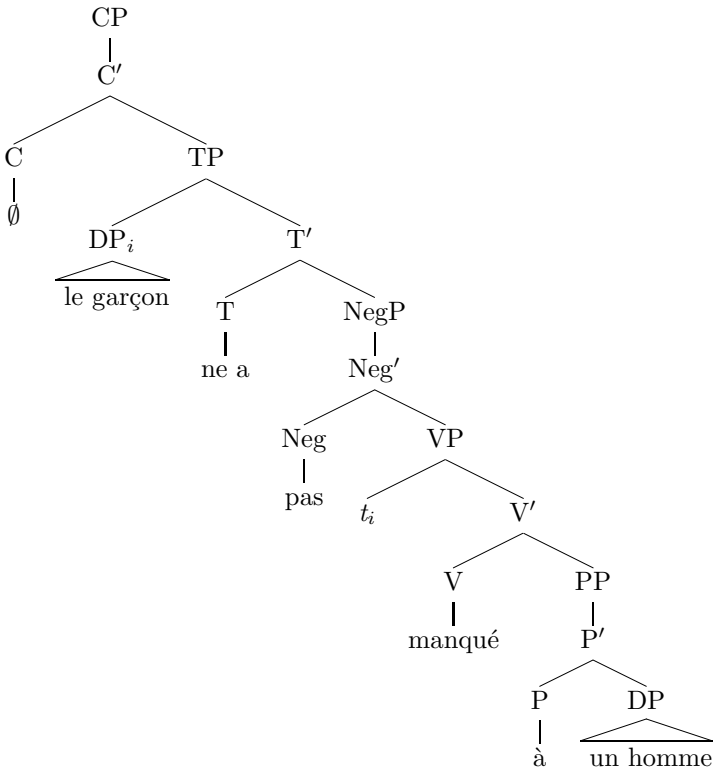
and the required output sentence is

(5.25) Le garçon ne a pas manqué à un homme.

(In fact *ne a* would be abbreviated to *n'a* for purely phonological reasons, but the two are separated here to better show the syntactic operations involved.)

As with all functional heads, the `NegHead` is realised by an abstract function of the `Generator` class, `getNegative`. In French, the head Neg is always realised as *pas*. The only missing component now is the *ne*. This is generated as part of the head T; the French `getAuxiliary` prepends *ne* to whatever otherwise would have been generated if there is a NegP in the complement position of the TP, so it returns the complete string *ne a*. (Similarly, in French sentences where the V and T heads have merged, the inflected verb form resulting from the `VerbHead` is prepended with *ne*.) Everything else (movement of the subject DP to the specifier position of the TP, for example) happens as in §5.2, and the correct result is produced as follows:

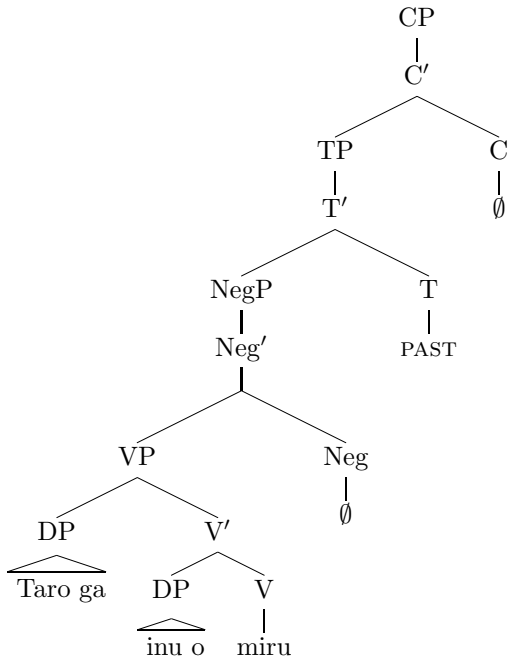
(5.26)



French has a relatively complicated negation system. Many languages do not realise the Neg head overtly, but only use a marking on the (finite) verb; that is, they have something that corresponds roughly to the French *ne*, but nothing corresponding to the French *pas*. For example, Japanese modifies the inflection on the verb (data from [30]):

(5.27) *Taro ga inu o mita.*
Taro SUB dog OBJ saw.
Taro saw the dog.

(5.28) (a)



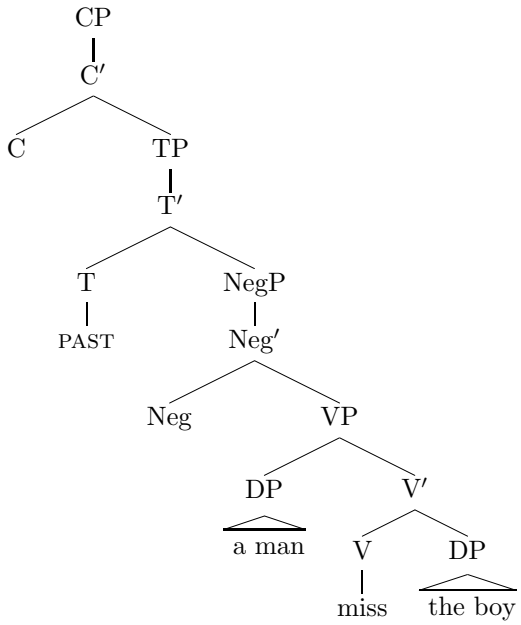
(b) *Taro ga inu o minakatta.*
Taro SUB dog OBJ see.NEG.PAST
Taro did not see the dog.

The graph in (5.28(a)) is a D-structure (no movement has taken place yet). Note that, in X′ terms, the tree is virtually identical to the D-structure in (5.24); the relationship between the D-structure of a sentence and that of its negative is language-universal, the only difference being the presence of the NegP.

After (5.28(a)) has been converted to an S-structure (ie. DP movement for case and the merging of V and T has taken place), the merged V/T head will be realised as the past negative form *minakatta*. A negative form of the verb was chosen because of the presence of the NegP. In Japanese, the Neg head is not overtly realised (that is, `JapaneseGenerator`'s implementation of `getNegative` returns the empty string).

English, roughly speaking, uses the 'other part' of French's complex negation system: it has an element which corresponds to *pas*, but not to *ne*. This is the word *not*. There is an extra complicating factor in English though; consider the following D-structure (again constructed according to the universal rule of simply inserting the NegP):

(5.29)



We would expect the subject DP to move to the specifier position of the TP and the T to lower and merge with the V, as usual in English; notice that this, though, assuming that Neg is realised as *not*, would produce

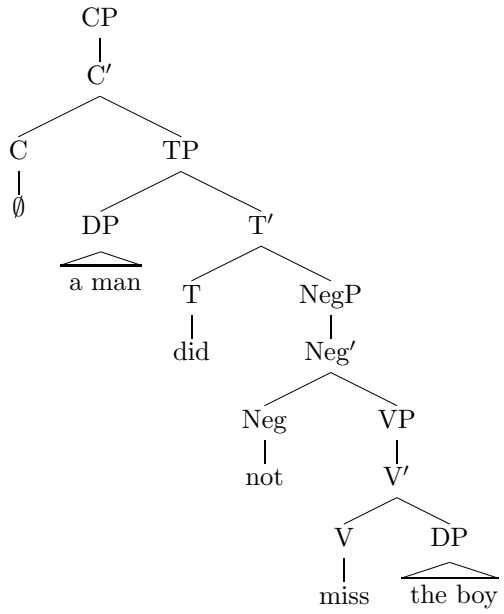
(5.30) *A man not missed the boy.

but the required result is

(5.31) A man did not miss the boy.

The theory here is that the head Neg has 'blocked' the merging of the T and the V, so the verb *miss* is an uninflected form (i.e. it does not reflect tense features) which has been generated by the isolated V. The T, however, is also isolated, but its features (PAST) must be realised in some way. Since it can't merge with the head V, English inserts a 'dummy' verb *do*, which appears in the T position and can reflect its features; in this case it reflects the PAST feature by surfacing as the past form *did*. (If the T was marked PRESENT, for example, it would surface as *does*.) This technique is known as '*do*-insertion'. Thus the S-structure, with the realisation of each head shown, is as follows:

(5.32)



Whether or not the Neg head blocks the usual merging of V and T is another parameter of linguistic variation.

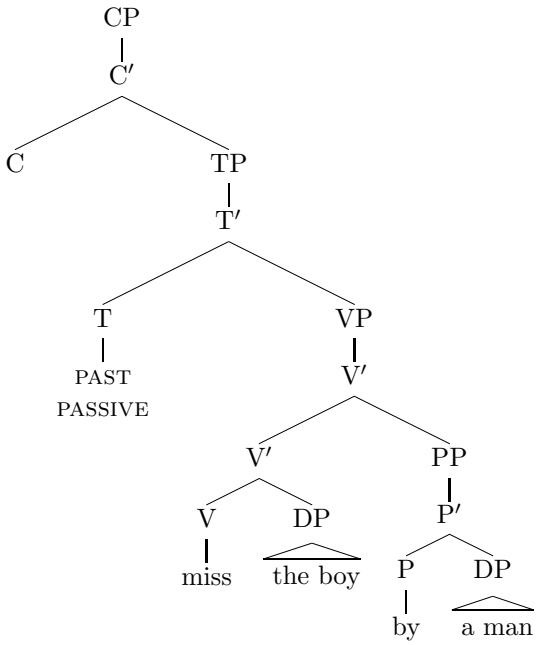
5.3.2 Passives

The voice attribute of the semantic input can be set to PASSIVE in order to generate sentences in the passive voice. DPs are constructed for each argument, exactly as in §5.2, but the process varies slightly when these need to be inserted into the VP.

Any arguments which are usually placed in the specifier position of the VP are instead placed in the complement position of a PP, and this PP is connected as an *adjunct* to the VP. The head of this PP is an adposition which the language specifies as its 'passive adposition'. In English, for example, this is *by*; in French it is *par*; in Japanese it is *ni*. Other arguments are generated as usual, as dictated by the theta-grid of the verb being used.

Thus, for the passive version of the English sentence represented by (5.14), the D-structure is

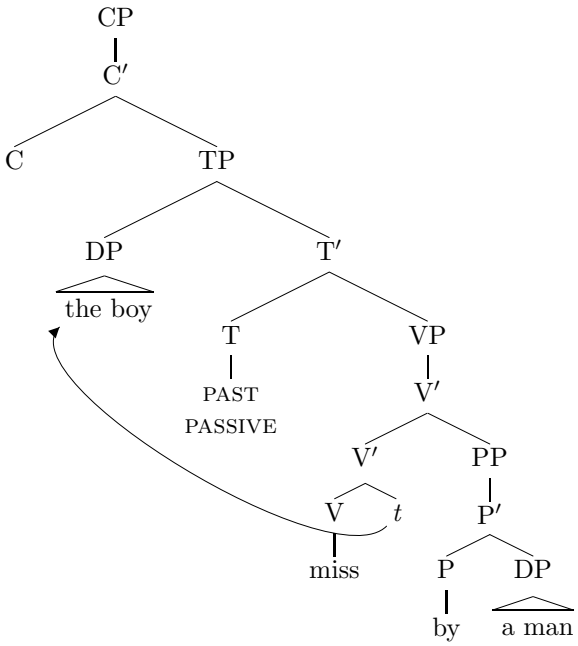
(5.33)



The PASSIVE feature has also been encoded on the head T.

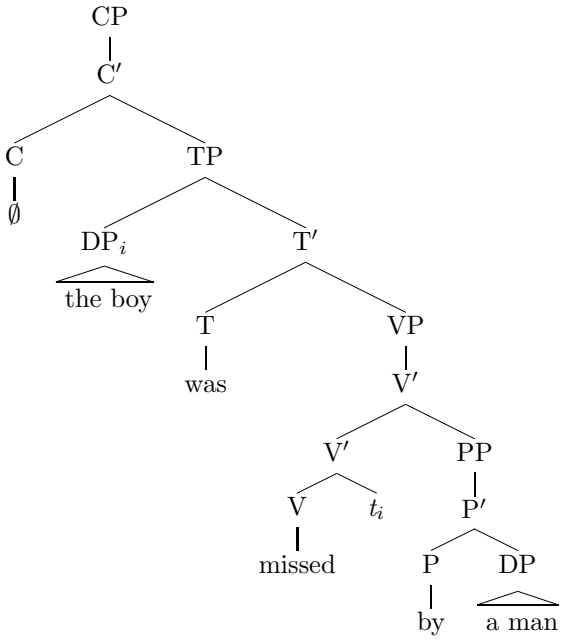
Recall that the phenomenon of subject movement is actually attributed to the requirement that every DP be assigned the abstract feature Case; DPs are not assigned Case in the specifier position of a VP in English, for example, so they move to the specifier position of a TP, where Case is assigned. In all previous examples, DPs in the complement position of VPs have not needed to move because this is another position where Case is assigned. In passive sentences though, the VP's ability to assign Case to its complement disappears. Therefore, in the tree above, the DP *the boy* is not currently assigned Case, so it must move to a position where it can be. (The DP *a man* is assigned Case by the adposition *by*; the complement position of a PP is always assigned Case.) The available position where Case is assigned, as before, is the specifier position of the TP.

(5.34)



The head T is realised as auxiliary because of its PASSIVE feature. This means that, like in the French sentence in §5.2, there is no need to merge the T and the V. When the V is realised, it uses a passive participle form because of the PASSIVE feature encoded on the T head. Thus the final S-structure, with the realisation of each head shown, is as follows:

(5.35)



There are no modifications to this process required to generate passives in French or Japanese. The only variation is in the adposition used in the place of *by*.

Anti-passives and other 'valance-changing devices' have not been directly attempted, but the design of the mechanisms which produce passives is intended to be general enough to easily allow these other transformations to be added to the system. Specifically, this involves an implementation of the `Predicate` interface, which wraps around the theta grid associated with a verb and modifies the associations between semantic relations and argument positions; `ActiveVerbPredicate`, for example, does not make any modifications, whereas `PassiveVerbPredicate` overrides the theta grid and replaces SPEC_VP with a COMP_ADJ_PP position.

5.3.3 Adjectival and Nominal Predicates

Adjectical Predicates

In §5.2, the lexical items associated with the predicate concept (MISS) were verbs. Predicates can also be constructed around adjectives though. This allows for a language-universal representation of sentences which may have different forms in different languages; what is expressed by an adjective in one language may be expressed by an intransitive stative verb in another. Consider the following input structure:

(5.36)

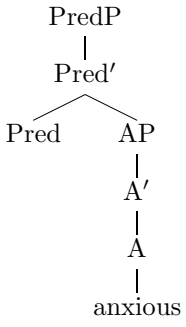
WORRY	
EXPERIENCER	`literal`(BOY, SINGULAR, DEFINITE)
`tense`	PRESENT
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

To generate a sentence from this input, the concept WORRY could be used to retrieve the English verb *worry*, the argument placed in subject position (specifier of the VP), and following the usual process, the result would be

(5.37) The boy worries.

Suppose though that the verb *worry* did not exist in English, and instead the concept had to be expressed as an adjective, say the adjective *anxious*. In this case, when searching for a lexical item associated to the concept WORRY, the adjective *anxious* would be retrieved. Instead of constructing a VP with its head V associated to a lexical verb, this time an AP is constructed with its head A associated to the adjective. This AP is placed in the complement position of a PredP:

(5.38)



Once this structure has been built, the PredP acts like a VP with a lexical head would throughout the rest of the realisation process. The fact that it was derived from an adjective rather than a verb can now be forgotten.

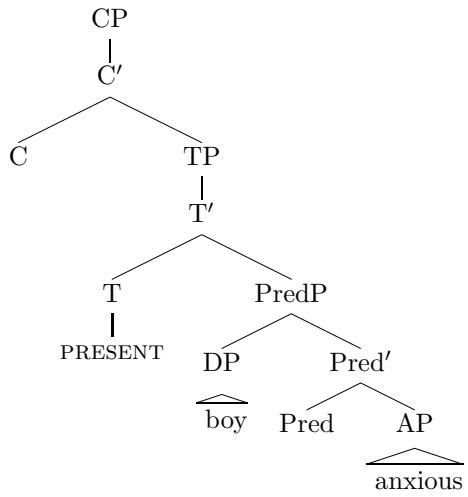
Stored with each adjective in a language's lexicon is the semantic role of its (one and only) argument; the semantic role of the argument to *anxious* is that of experiencer. Arguments to adjectival predicates are *always* placed in the specifier position of the VP (note that 'VP' here refers a phrase with the logical linguistic category of V, one headed by an `AbsVerbHead`; see §4.2.5); that is, what could be called the 'theta-grid' of an adjective always consists of exactly one line, the second column of which is always SPEC_VP:

(5.39)

anxious	
EXPERIENCER	SPEC_VP

From this point the construction of the D-structure proceeds exactly as in §5.2. A DP is generated corresponding to the argument and inserted into the position indicated by the 'theta-grid', and the PredP is placed inside a TP and a CP as usual:

(5.40)



The conversion from D-structure is also essentially the same as above, with the subject DP moving to the specifier position of the TP and the Pred merging with the T. Each language specifies its own way of realising Pred, by implementing the `Generator` class's abstract function `getPred`; English (like French and many other languages) uses a form of the verb *to be*; in this case, *is*:

(5.41) The boy is anxious.

In English, though, there is a slight difference in the behaviour of Pred heads and V heads, as shown in the following examples:

(5.42) The boy is often anxious.

(5.43) The boy often reads books.

The positions of the Pred (realised as *is*) and the V (*reads*) are different when an adverbial adjunct to the VP (or PredP is present). Recall from §3.3.3 that French differs from English in this respect:

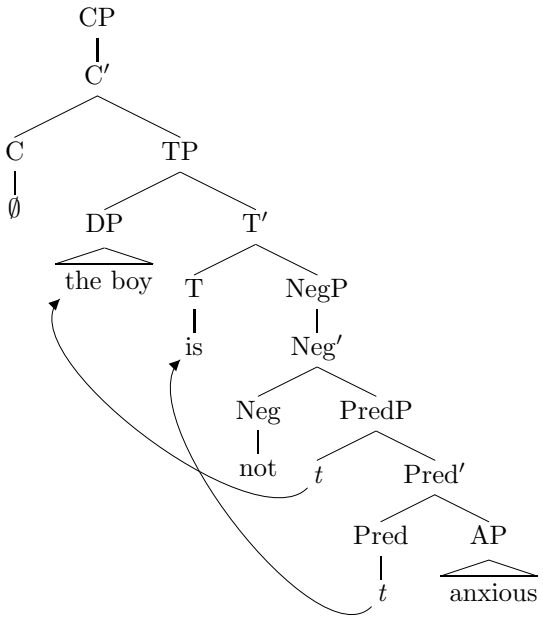
(5.44) *Le garçon lit souvent des livres.*
the boy reads often books
The boy often reads books.

This difference was attributed to the 'verb-inflection' parameter; French raises the V to T position, whereas English lowers the T to V position. The position of the verb *lit* is the same as that of *is* in (5.42). Thus is appears that English acts like French and raises the head Pred to merge with the T; English uses one setting of the parameter for lexical verbs, and another for the functional head Pred.

This difference in the behaviour of lexical verbs compared to Pred may appear to conflict with the argument above that PredP can be treated in the same way as a VP, resulting in a unified process. To a small extent this is true, but it is significant that the behaviour of Pred in English is not something new altogether, which has never been encountered with lexical verbs; on the contrary, Pred behaves exactly as lexical verb heads do in other languages (such as French and Irish). Thus the unified realisation process already has the ability to raise verbs rather than lower tense inflections if the language dictates that this is necessary. The only change is that what was thought to be one parameter has been separated into two finer-grained parameters, one for lexical V heads and one for Pred.

Analysis of negative sentences constructed from adjectival predicates provides further evidence that Pred raises to the T position. In addition, it reveals that English similarly 'splits' the parameter which allows the Neg head to block the merging process; the merging of V and T is blocked by Neg, as seen in §5.3.1, but the merging of Pred and T is not:

(5.45)



Nominal Predicates

As well as from verbs and adjectives, predicates can be constructed from nouns. Nouns in the lexicon do not have any semantic relations associated with them;

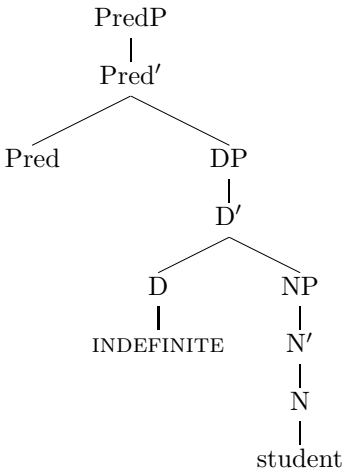
the one and only argument to a nominal predicate must have the semantic role I have labelled 'copula', and is always placed in the specifier position of the 'VP'. (This is the only occasion where the system assigns a special meaning to a particular semantic role; elsewhere they are purely used to 'match' arguments with positions as specified by verbs and adjectives.) Thus nouns have an even more restricted form of 'theta-grid' than adjectives; the grid used to place arguments to a nominal predicate is always as follows (using the English noun *student* as an example):

(5.46)

student	
COPULA	SPEC_VP

If the concept used to indicate the predicate in the input maps to a noun in the target language, an NP headed by an N associated to this noun is generated. This NP is placed in the complement position of a DP, which is placed in the complement position of the PredP:

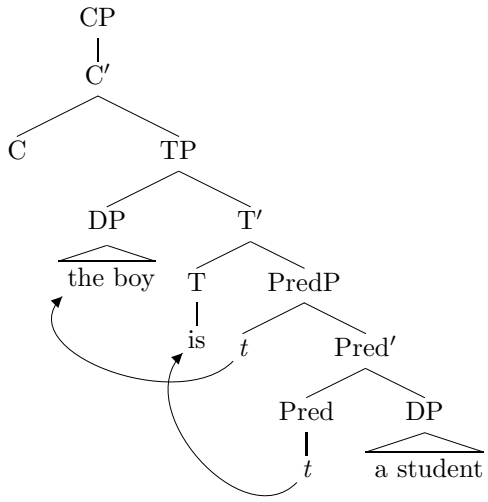
(5.47)



(The system currently always sets the D at the head of this DP to be indefinite.)

Using again the sample argument from above, represented by `literal(`BOY, SINGULAR, DEFINITE`)`, assuming that it was given the semantic role 'copula', after construction of the D-structure and movement exactly as for adjectival predicates, we derive the following:

(5.48)



5.3.4 Pronouns

Personal Pronouns

The example in §5.2 used only `literal` arguments, represented in the input by, for example, `literal`(MAN, SINGULAR, INDEFINITE). To generate pronouns such as *he* and *they*, a different type of argument is required; these are known as `conversant` arguments, and refer to human beings. For example, to generate the sentence

(5.49) A man missed her.

the following input would be required:

(5.50)

MISS	
EXPERIENCER	`literal`(MAN, SINGULAR, INDEFINITE)
THEME	`conversant`(SINGULAR, -SPEAKER, -LISTENER, HUMAN_FEM)
`tense`	PAST
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

The first parameter of a `conversant` argument is its number.

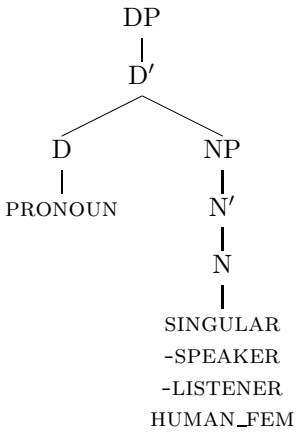
The second and third parameters indicate whether the speaker is included in the entity/entities being described, and whether the listener is included, respectively. In this example, the argument being described is in the third person (*her*), so neither is included and we have (-SPEAKER, -LISTENER). These two parameters allow for the complete range of person systems used in the world's languages. The pair (-SPEAKER, +LISTENER) corresponds to second person; the speaker is not included, and the listener is. First person in English may correspond to either (+SPEAKER, +LISTENER) or (+SPEAKER, -LISTENER); it may include the listener, or it may not. There are many languages though where a distinction is made between the 'inclusive *we*' and the 'exclusive *we*'; the former can be represented by (+SPEAKER, +LISTENER), and the latter can be represented by (+SPEAKER, -LISTENER).

The fourth parameter of a `conversant` argument is the gender of the human(s) being described. This may be either HUMAN_MASC, HUMAN_FEM or (in the case of arguments with non-singular number) HUMAN_MIXED. This is clearly necessary to distinguish between, for example, *he* and *she*.

There is a language-universal process for producing a DP from each type of argument. (At the level of implementation, the different types of arguments are represented by subclasses of an abstract class `Argument`, such as `LiteralArgument` and `ConversantArgument`.) The process for constructing a DP from a `literal` argument was described in §5.2.

In order to build a DP to represent a `conversant` argument, a `DetHead` object must be constructed. The two types of determiners introduced above were DEFINITE and INDEFINITE, but for `conversant` arguments a different type is required, PRONOUN; this is encoded on the `DetHead` object. The complement of the DP is headed by a `PronounHead`, as opposed to a `NounHead` which was used with `literal` arguments. The person, number and gender features are encoded on the `PronounHead`. Thus the DP derived from the `conversant` argument in (5.50) would be

(5.51)



Once the DP has been constructed, the type of argument which was used to generate it is no longer of any significance. It is inserted into the D-structure in exactly the same way as described it §5.2, and is subject to exactly the same syntactic transformations.

The process thus continues exactly as in §5.2 through to the point where the DP comes to be realised. To realise the above DP in English, first the `getInflectedForm` of the `DetHead` object will be called. As usual, this will call the `getDeterminer` function of the `EnglishGenerator` class, which will examine the `DetHead` and find its type to be PRONOUN. Now the `EnglishGenerator` needs to know the person, number and so on of the required pronoun; for this it browses to the head of the complement NP, the `PronounHead` object, and examines the features encoded there. (The functions for retrieving these are part of the interface of the abstract `NominalHead` class; see §4.2.5.) A third person, singular, feminine pronoun is required. The exact form will also depend on the DP's position in the tree, because English pronouns are affected by case; based on this, the `EnglishGenerator` will return either *she* (for nominative case) or *her* (for accusative case). This pronoun appears in the D position. When the N node is realised, the `PronounHead` object returns an empty string. (`PronounHead`'s

`getInflectedForm` function *always* returns an empty string; this class is only used in the complement of a DP with determiner type PRONOUN, so the pronoun is always realised in the D node.)

Impersonal Pronouns

Consider now though the English pronoun *it* and the French pronoun *il* in the following sentences, referring to something which is not explicitly stated but is considered to be clear from the context.

(5.52) I read it yesterday.

(5.53) *Il est dans mon sac.*
it is in my bag
It is in my bag.

These pronouns do not refer to humans, and can not be represented by the concept of a `conversant` argument as outlined above. They refer to the same objects that the `literal` arguments used in §5.2 might. Some other mechanism is needed to represent this sort of pronoun in the semantic input.

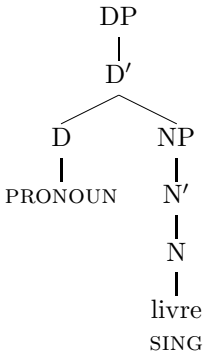
Note that in French (and many other European languages), the pronoun required in sentences like (5.53) depends on the 'gender' of the object being referred to. The pronoun in (5.53) might refer to a book (because the corresponding French noun, *livre*, is masculine), it might refer to a pen (because the French *stylo* is masculine); but it can not refer to a letter, for example, because the French noun *lettre* is feminine, and so the pronoun *elle* would be required instead. In order for the system to generate the correct pronoun in that French sentence, it *must* know the 'gender' of the French noun which could be used to refer to the object. This French 'gender' can't be part of the semantic input to the system though, because the semantic input needs to be language-universal.

The solution is to use abstract concepts, like MAN and BOY, to specify what these impersonal pronouns refer to. Supposing that the pronoun in (5.53) refers to a book, this argument would be specified at the input as

(5.54) `literal(`BOOK, SINGULAR, PRONOUN`)`

and the resulting DP would be

(5.55)



Note that the algorithm for building this DP from the input is exactly the same as that used for the `literal` arguments in §5.2; the concept BOOK has been mapped to the French lexical item *livre*, and so on.

When realising the `DetHead` object, the same process as for personal pronouns will be used; the `getDeterminer` function will look at the head of the DP's complement NP to find the gender, person and so on of the required pronoun. (Note that `NounHead`s are *always* third person; only `PronounHead`s can represent first and second person.) This time, instead of finding HUMAN_MASC, HUMAN_FEM or HUMAN_MIXED, it finds a French lexical noun; every French lexical noun can be queried for its gender, so `getDeterminer` does this, and then has all the information it needs to return a masculine pronoun. When the `NounHead` representing the N in the above tree is realised, it will return an empty string, like `PronounHead`s do, because its associated `DetHead` has determiner type PRONOUN.

This system for impersonal pronouns has unified the idea of noun classes for lexical nouns (which is not present in, for example, English), and the idea of different personal pronouns corresponding to different genders. The concept of 'noun class' in my system covers both these notions. The possible noun classes in a particular language are HUMAN_MASC, HUMAN_FEM and HUMAN_MIXED, *plus* all the classes of lexical nouns in that language. In English, the classes of lexical nouns (in fact there is only one of these classes) do not overlap with HUMAN_MASC, HUMAN_FEM and HUMAN_MIXED. In French, the classes of lexical nouns, which we might label FRENCH_MASC and FRENCH_FEM, do overlap with the logical, language-universal classes of personal pronouns; in fact, FRENCH_MASC acts exactly the same way as HUMAN_MASC, and FRENCH_FEM acts exactly the same way as HUMAN_FEM. In German the system is slightly different again: the language-specific classes which we might label GERMAN_MASC and GERMAN_FEM correspond exactly to HUMAN_MASC and HUMAN_FEM respectively, as in French, but there is also a third class of lexical nouns, GERMAN_NEUTER, which does not overlap at all with personal pronouns.

Regarding the implementation of this idea, the `NominalHead` class has an abstract method `getNounClass`; `PronounHead` implements this by returning the 'human noun class' which is encoded on it (HUMAN_MASC etc.), and `NounHead` implements it by returning the lexical item it is associated with. (The apparent difference in return types is handled by a wrapper class, `NounClass`, which can encapsulate either type.) This lexical item, if from a language which uses lexical noun classes, will have a language-specific method which will reveal the lexical noun class to which it belongs.

5.3.5 Questions

Yes/No Questions

Another 'flag' in the semantic input indicates whether or not the output should be a question. Consider the following input, for example.

(5.56)

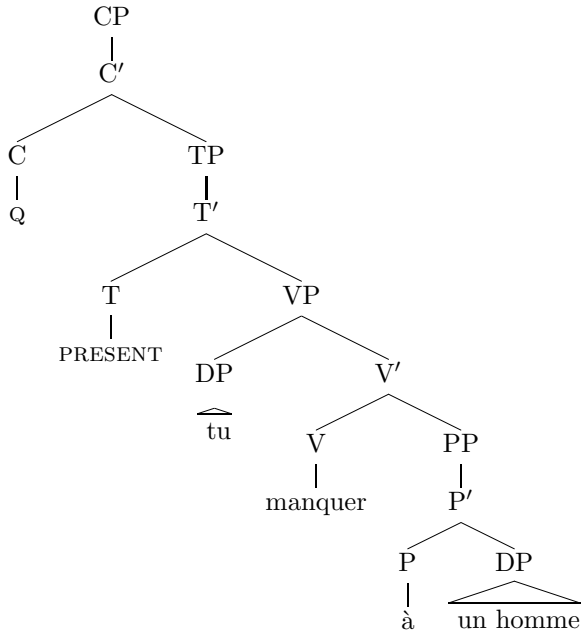
MISS	
EXPERIENCER	`literal`(MAN, SINGULAR, INDEFINITE)
THEME	`conversant`(SINGULAR, -SPEAKER, +LISTENER, HUMAN_FEM)
`tense`	PRESENT
`negative`	FALSE
`question`	TRUE
`voice`	ACTIVE

The desired French output for this input is

(5.57) *Manques-tu à un homme?*
lack-you to a man
Does a man miss you?

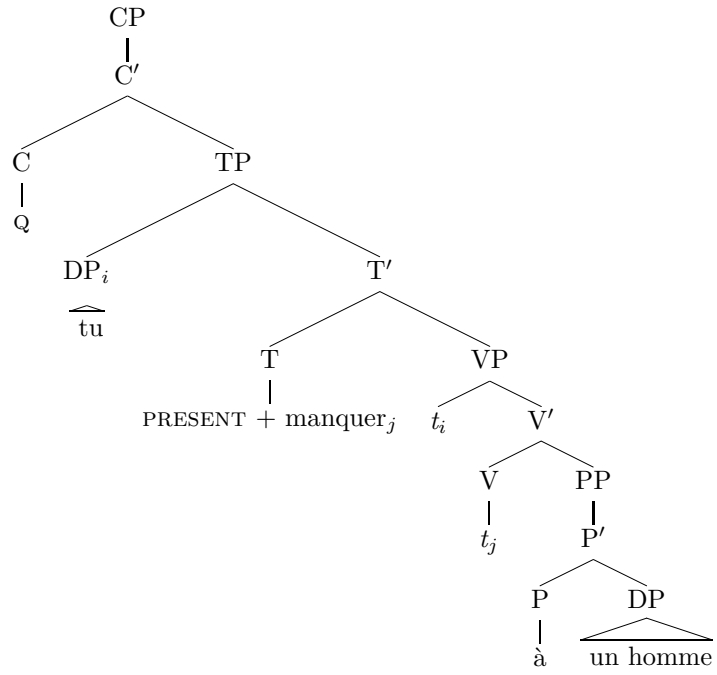
The D-structure is generated exactly as in previous examples, the only extra point being the Q feature encoded on the head C:

(5.58)



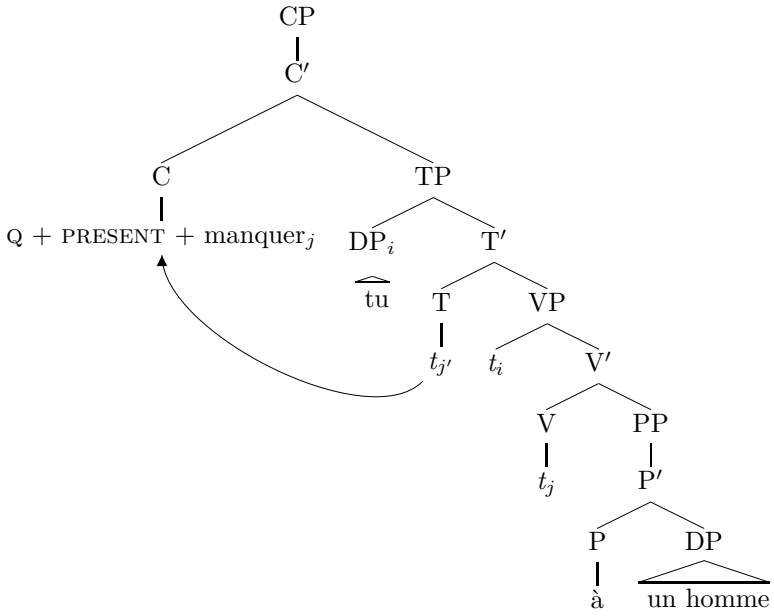
The usual movement (subject movement and V/T merging) now takes place, resulting in the following:

(5.59)



The Q feature on the C triggers some additional movement in French though; specifically, it forces the T to move and merge with the C. The T in this example has also merged with a V, which gets dragged along to the C position as well:

(5.60)



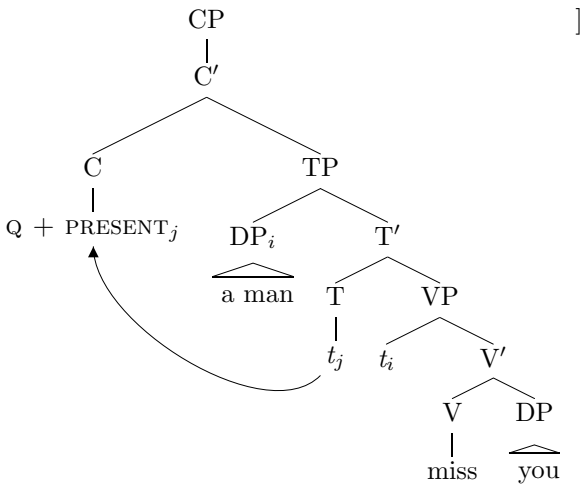
This structure now produces the correct surface word order:

(5.61) Manques-tu à un homme?

The process for English questions is almost identical. However, whereas French raises the V to the T position, English lowers the T to the V position. In

questions though, C attracts the T, which conflicts with English's usual method of merging the V and T; the T can't move in both directions, up towards the T and down towards the V. The result is that the T is attracted to the C position, and does not merge with the V (T-to-C movement 'overrides' the usual merging):

(5.62)

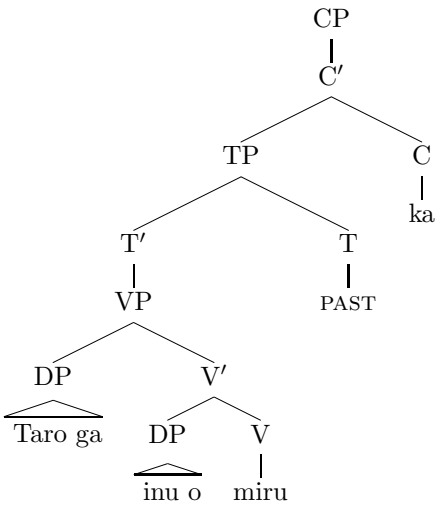


The same problem has arisen here as did in negatives though; the tense features have become isolated, without a verb to express them. Exactly the same strategy is used here as before to overcome this: a dummy verb *do* is inserted to allow the T head's features to be realised. Thus the merged C/T head is realised as *does* (reflecting the PRESENT feature and agreeing with the nominal in its subject position), and the correct final word order is produced:

(5.63) Does a man miss you?

The attraction of the T to the C is subject to parametric variation though; Japanese does not do this. Instead Japanese simply realises complementisers which have the Q feature as the 'question particle' *ka*, without triggering any additional movement (data from [30]):

(5.64)



(5.65) *Taro* *ga* *inu* *o* *mita* *ka?*
Taro SUB dog OBJ see.PAST QUESTION
Did Taro see the dog?

wh-questions

The procedure for generating *wh*-questions is largely the same as for yes/no questions, with one extra transformation. Firstly, a new type of argument is required in the semantic input, to indicate the argument that is being questioned. For example, to represent the sentence

(5.66) What did you see?

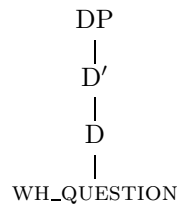
the user would input a semantic structure like the following:

(5.67)

SEE	
EXPERIENCER	`conversant`(SINGULAR, -SPEAKER, +LISTENER, HUMAN_FEM)
THEME	`unknown`
`tense`	PAST
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

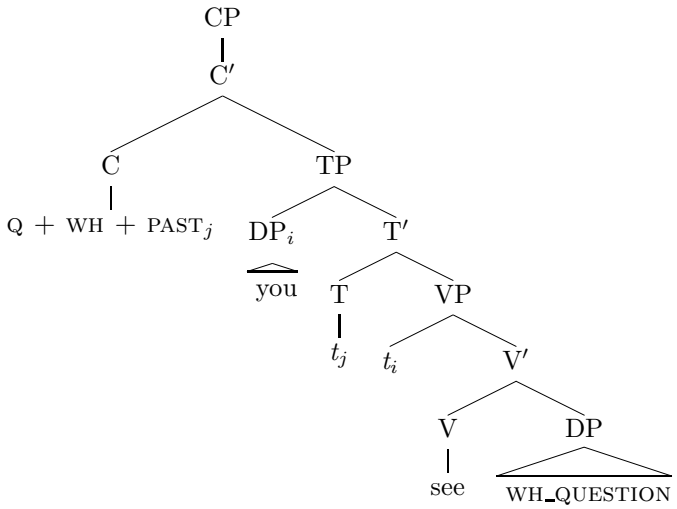
When one of the arguments is `unknown`, the value of the `question` attribute is ignored; the result is necessarily a question.

Like other argument types, `unknown` arguments must be converted to a DP which will be placed in the D-structure in the appropriate position. The head D of these DPs are of a new determiner type: instead of DEFINITE, INDEFINITE or PRONOUN they are WH_QUESTION. DPs generated for `unknown` arguments also have no complement NP:

(5.68)


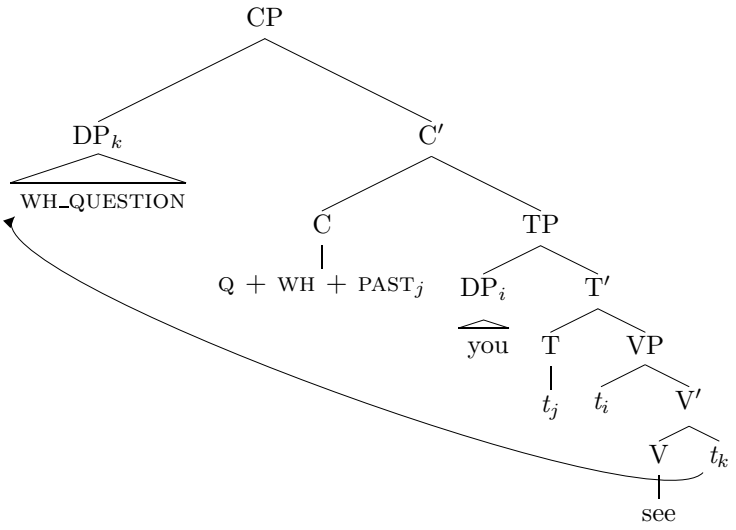
The presence of an `unknown` argument also causes a WH feature to be encoded on the complementiser, as well as the same Q feature as was used to generate yes/no questions. This Q feature causes the same T-to-C movement as was seen above; thus the following structure is derived:

(5.69)



In addition though, the WH feature of the complementiser requires that a DP with the WH_QUESTION feature move to the specifier position of the CP:

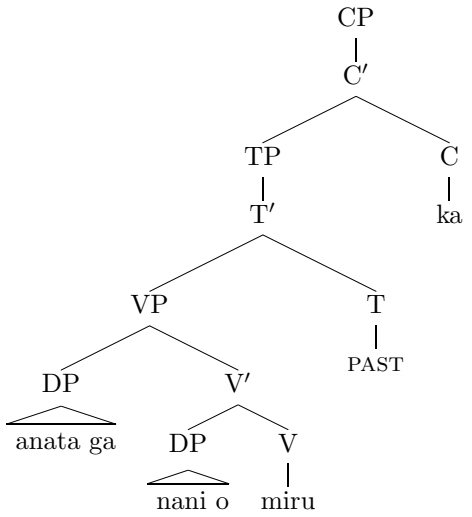
(5.70)



If the WH_QUESTION DP is realised as *what*, and *do*-insertion occurs as previously to realise the T's features, the correct word order is now derived.

Movement of a DP with the WH_QUESTION feature is also subject to parametric variation. English and French both do this movement as above, but Japanese does not. This means that the word order in questions is essentially the same as in statements (SOV), with the WH_QUESTION DP being realised as *nani* (plus usual case markings), but the Q feature on the complementiser again causes it to be realised as *ka*, as in yes/no questions (data from [30]):

(5.71)



(5.72) *Anata* *ga* *nani* *o* *mita* *ka?*
you SUB what OBJ see.PAST QUESTION
What did you see?

Another associated parameter concerns 'adposition stranding'. If a DP must be moved to the specifier position of CP but is generated in the complement position of a PP, in some languages, such as French, the entire PP must be moved, to avoid 'stranding' the adposition. Adposition stranding is common in spoken English, but the entire PP is sometimes moved in formal use; the difference is illustrated in the following two sentences:

(5.73) (a) Who(m) did you give it to?
(b) To who(m) did you give it?

In (5.73(a)) the DP *who(m)* has been moved out of the PP, 'stranding' the adposition *to*. In (5.73(b)), the entire PP *to who(m)* has been moved to the specifier position of the CP. French and Japanese do not allow adposition stranding, so in these languages the entire PP is moved to the specifier position of the CP.

One complicating factor is the fact that T-to-C movement does not occur in English when a WH_QUESTION DP moves from subject position:

(5.74) Who sees you?

In this example, the DP *who* has moved from the specifier position of TP to the specifier position of CP, as usual; but unlike in other questions, the tense features on the T have been realised on the verb itself (as the agreement suffix *-s* in this case). This means that the T has lowered to merge with the V, as it does in normal statements, instead of raising to merge with the C, as it does in most questions. This happens only when the DP undergoing *wh*-movement is has passed through the specifier position of TP (it appears that this movement from the TP to the CP is in some sense sufficient to achieve what T-to-C movement usually does, but the more theoretical details are not the concern here).

5.3.6 Relative Clauses

In order to represent sentences with relative clauses, two semantic input structures of the type which have been shown previously are required: one for the main clause and one for the relative clause. Suppose we want to generate the English sentence

(5.75) I eat the apples which the men bought.

First, one input is needed for the relative clause, *which the men bought*:

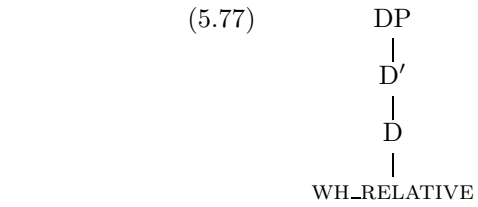
(5.76)

BUY	
AGENT	`literal`(MAN, PLURAL, DEFINITE)
THEME	`relativewh`
`tense`	PAST
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

A new type of argument has been introduced, `relativewh`. This goes in the position of the nominal which the relative clause is modifying; this relative clause is modifying *the apples*, and *the apples* have the theme role in the buying 'event'.

The DPs generated for `relativewh` arguments are similar to those for `unknown` arguments, but with a new type of determiner:

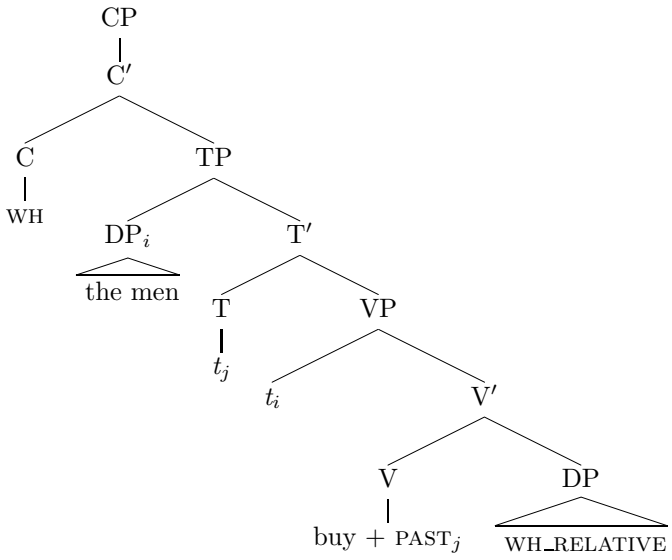
(5.77)



This DP is inserted in the position dictated by the theta grid of the English verb *buy* (i.e. the complement position of the VP), and the presence of a WH_RELATIVE argument also causes a WH feature (the same as is used in *wh*-questions) to be encoded on the C.

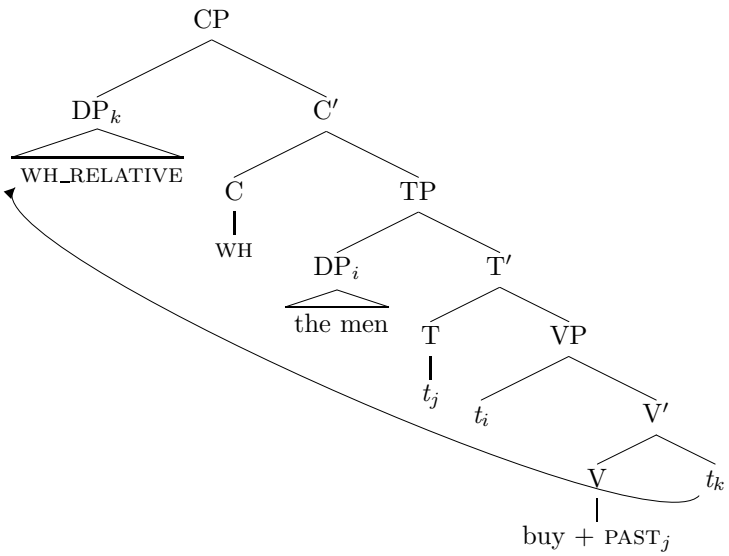
The process continues as usual until the following is generated:

(5.78)



In questions, the WH feature on the C forced a DP with the WH_QUESTION feature to move to its specifier position, but the C's WH feature can also be satisfied (or 'checked') in the same way by a DP with the WH_RELATIVE feature:

(5.79)



(The two features WH_RELATIVE and WH_QUESTION are identical for the purposes of movement, but need to be distinguished because their DPs are realised differently in some languages, such as English.)

English realises determiners with the WH_RELATIVE feature as *which*, so now the correct form for the relative clause has been produced:

(5.80) which the men bought

Note that the C has not itself been overtly realised, but the WH feature encoded on it has triggered an important transformation.

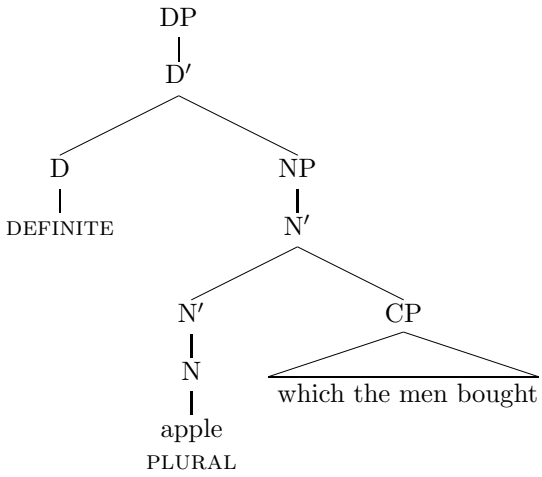
Now we turn to the main clause of (5.75). Clearly the two clauses are linked in some way; to achieve this, suppose there is an identifier `subclause0` which refers to the structure in (5.76). (See appendix C for details of how this is actually achieved when executing the program.) Then the input to represent the main clause would be

(5.81)

EAT	
AGENT	`conversant`(SINGULAR, +SPEAKER, -LISTENER, HUMAN_MASC)
PATIENT	`relative`(APPLE, PLURAL, DEFINITE, `subclause0`)
`tense`	PRESENT
`negative`	FALSE
`question`	FALSE
`voice`	ACTIVE

Yet another type of argument type has been introduced here, `relative`. This is used for arguments in a main clause which are modified by a relative clause. The first three pieces of information listed for the `relative` argument are the same as those used for `literal` arguments, but the final part links the argument to the semantic input for the relative clause which modifies it. The DPs constructed for `relative` arguments are similar to `literal` arguments, but the CP generated from the semantics of the relative clause is added as an adjunct to the NP. Thus for the `relative` argument above, the DP generated is

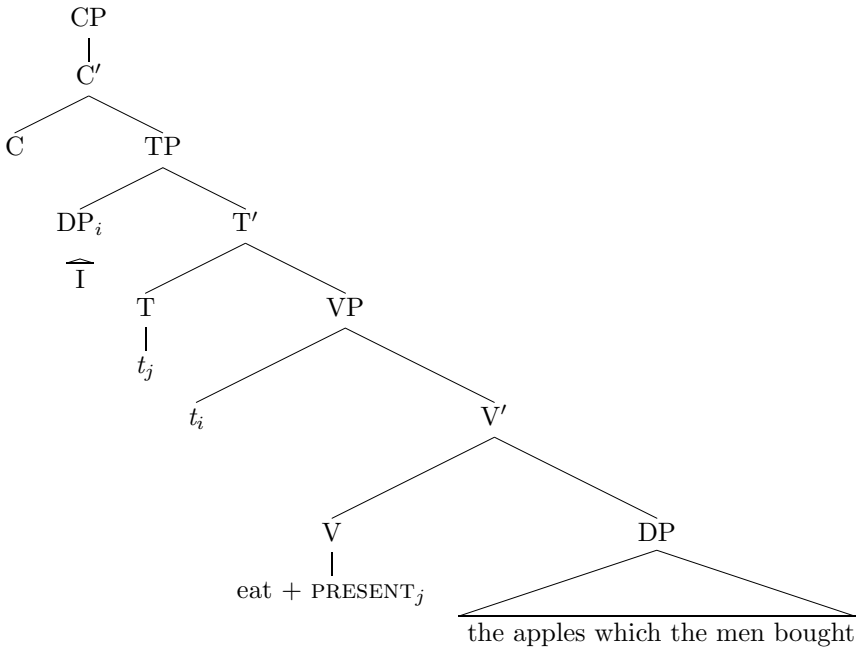
(5.82)



Note that the identifier `subclause0` does not refer to the generated English CP in (5.79), but to the language-universal input in (5.76). The CP for the relative clause is generated as part of the process of generating the DP for the `relative` argument in the main clause.

Now that a DP has been built for that argument, the rest of the generation process continues as usual for the main clause; this DP is placed in the complement position of the VP, as specified by the theta grid of the verb *eat*, and so on.

(5.83)



This can now be realised as usual to form the correct final result:

(5.84) I eat the apples which the men bought.

The transformations which produce French relative clauses are identical to those for English:

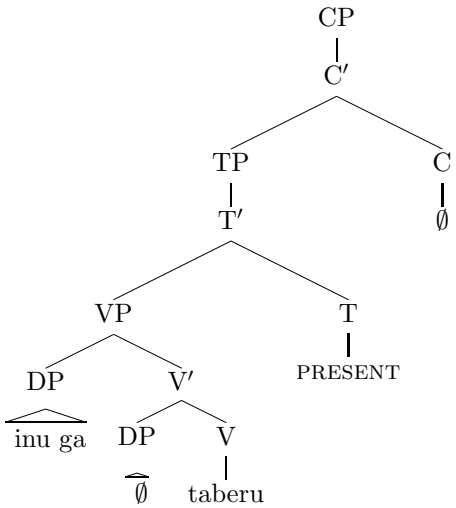
(5.85) *Je mange les pommes que les hommes ont achetées.*
I eat the apples which the men have bought
I eat the apples which the men (have) bought.

In Japanese, the overall structure is the same again, but the DP with the WH_RELATIVE feature is not overtly realised. So for example, in the sentence (from [30]):

(5.86) *Watashi ga inu ga taberu ringo o miru.*
I SUB dog SUB eats apple OBJ see
I see the apple which the dog eats.

the relative clause *inu ga taberu* is constructed as follows:

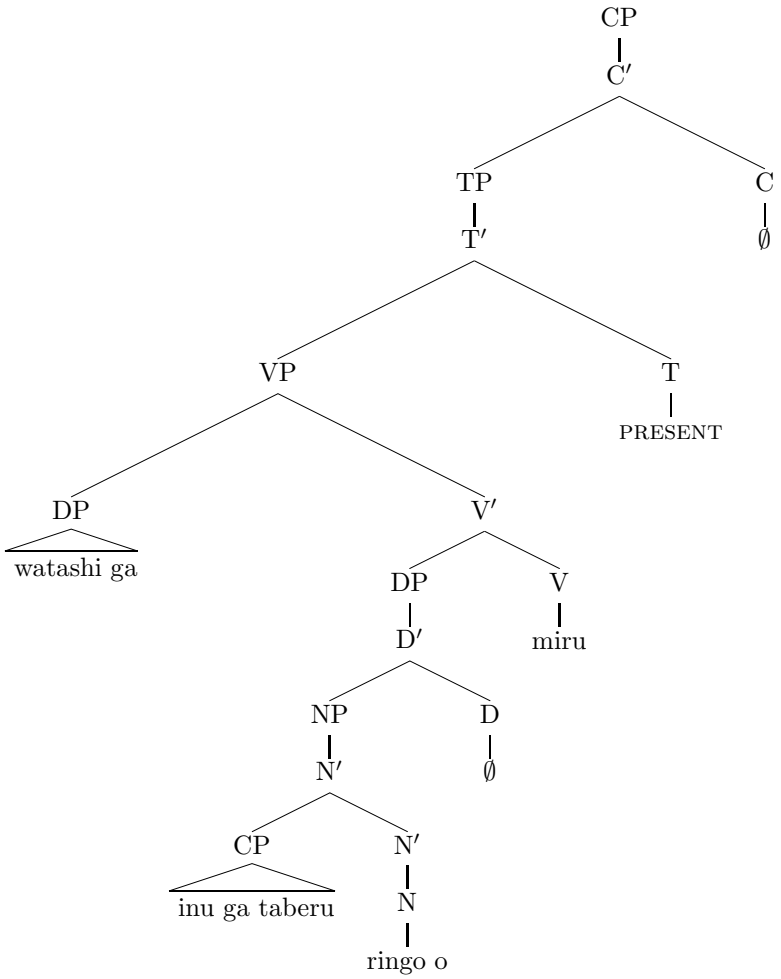
(5.87)



The DP in the complement of VP position is that which corresponds to *which* in English, but Japanese does not have an equivalent and this DP has not been overtly realised. (This tree assumes that, since *wh*-movement does not occur in Japanese questions, that it does not occur here either, but in fact it would not matter since the DP which would move is not realised.)

Also, instead of being adjoined to its NP on the right, as in English and French, the relative clause's CP is adjoined on the left. Thus the structure of the complete sentence is:

(5.88) (a)



(b) *Watashi ga inu ga taberu ringo o miru.*
I SUB dog SUB eats apple OBJ see
I see the apple which the dog eats.

(Recall that Japanese does not have any overt determiners.)

Note that the word order in this sentence is drastically different from that of its English equivalent, but the differences were brought about by simple variations on the overall procedure for generating relative clauses: whether the WH_RELATIVE argument is realised, which side the subordinate clause is adjoined to the NP, and, of course, the head-directionality parameter.

Chapter 6

Evaluation

6.1 Collected Data

The following table shows how the source code of the system is distributed amongst the language-universal 'core' and the added modules containing the information specific to each of the (partially) implemented languages.

component	lines of code	percentage of total
language-universal	3559	64
English	557	10
French	964	17
Japanese	520	9
entire system	5600	100

Table 6.1: Distribution of source code over system's modules

The information relevant to syntactic realisation in a particular language can be considered to be that in the language-universal core plus that in the appropriate language-specific module. From the above, we then derive this second table showing the distribution of language-specific and language-universal information used for each implemented language.

	language-universal		language-specific	
language	lines of code	percentage	lines of code	percentage
English	3559	86	557	14
French	3559	79	964	21
Japanese	3559	87	520	13

Table 6.2: Distribution of source code for each language

Lexical information about the nouns, verbs, adjectives and adpositions of the implemented languages does not contribute to the figures here. I am only attempting to analyse the syntactic properties of languages; as mentioned in 1.3.1 item 2, there was no expectation that lexical information could be predicted by any universal principles.

6.1.1 Completeness

Clearly the 'implementations' of these languages are nowhere near complete, and there are many sentences which can not even be expressed as input to the system; but there are also some cases, within the scope of the implementation described in chapter 5, where the correct output has not been achieved. Details are in appendix B. They are all minor language-specific problems though, and, while many other language-specific peculiarities have successfully been accommodated, instead of spending time fixing these I felt it was more important to ensure that a reasonable range of syntactic phenomena was covered.

However I believe that what has been done is definitely significant enough to indicate that the approach could be extended to full 'implementations' of these languages, and does provide meaningful data with respect to the goals set out in §1.2.

6.2 Analysis in Relation to Initial Goals

6.2.1 Software Framework for Further Use (§1.2.1)

It was hoped that the framework used to model the language-universal information would be 'large' and comprehensive enough to make the process of adding a new language to the system relatively easy. From the data above, this does seem to have been achieved to some reasonable extent: to add a fourth language, based on averages from the three existing languages, one would expect to have to add approximately 680 lines of code, and the other 3559 of the total 4239 lines of code used to generate sentences in the new language would already be provided.

The obvious alternative to the structure of my system is a design consisting of nothing but complete language-specific realisation modules, one for each available language. The data above do not necessarily indicate that around 4239 lines of new code would be required to add a new language to a system built this way, because the nature of the information to be added would be fundamentally different, but it seems very likely that more than 680 would be needed.

Measuring the 'ease' with which a new language can be added using numbers of lines of code is a fairly crude technique though; it is worth considering the nature of the code in the language-specific parts of what has been implemented. While, as mentioned above, information about the open lexical classes does not contribute to the tables, the code which describes the inflection systems applied to these lexical items and the closed word class systems (i.e. systems of determiners, auxiliary verbs, etc.) *is* included. In fact a very large proportion of the code found in the language-specific modules resides in the functions which decide on the realisations for the functional heads, such as `getDeterminer`, `getAuxiliary` and so on; these tend to take the form of large groups of nested `if` statements, covering all the combination of features (case, person, number, gender, tense, etc.) which are factors in deciding the correct realisation. While this is clearly necessary information for the syntactic realisation process, writing these sections of the code is a fairly simple task; the programming is mundane, and the relevant grammatical information can easily be read off traditional tables. Implementing these functions, although they contribute significantly to line counts, was very simple compared to implementing the mechanics of the X′

framework and the transformations performed upon it, for example; and it is these more challenging components of the system which need only be written once for all languages.

Therefore I believe that the process of adding a new language, relative to the process of building the system as a whole, is even easier than is suggested by the favourable ratios in the tables above.

(Note that no quantitative targets were set regarding these ratios because the other primary aim of the project is to adhere strictly to the theory which has been proposed by linguists; to implement this theory to see what it yielded. A project where a particular ratio of language-specific to language-universal information was attempted could become a significant theoretical linguistic work, requiring modifications to the theory if the desired ratio was not achieved using existing theory.)

6.2.2 A Test of the P&P Theory (§1.2.2)

The ability of the system to generate correctly correct sentences, given the modular structure as represented in the tables above, supports the possibility that all characteristics of all human languages (and possibly our minds' representation of language) might be structured the same way. When P&P is considered as a solution to Plato's problem, the data could indicate that between 79 (in the case of French) and 87 (in the case of Japanese) percent of what a fluent speaker of a language 'knows' about the language's syntax is innate and does not have to be learnt.

As mentioned above though, a large part (when measured by lines of code) of the language-specific modules concerns the realisation of functional heads such as T and D. The numerous choices encoded for these are not really part of what P&P claims to be able to predict though. Morphology and syntax are closely dependent on each other, and the fact that English pronouns, for example, depend on person, number, gender and case may be considered as related to the syntax; but the fact that the third person singular accusative masculine form is *him*, whereas the third person singular accusative feminine form is *her*, is arbitrary, just like the links between lexical items from the open word classes and their meanings (see §1.3.1, item 2). Thus the figures in the above tables are distorted somewhat by the fact that this information, which is closer in nature to the lexical information which has deliberately omitted than it is to the syntactic information which is the main concern, contributes to the size of the language-specific modules.

The syntactic information specific to each language which produces the different surface constituent orders, the real focus of P&P theory, has successfully been distilled down to a handful of binary parameters, exactly as the theory suggests should be possible.

6.3 Other Observations

6.3.1 Different Sizes of Language Modules

Table 6.1 reveals that the language-specific module for French is significantly larger than those for English and Japanese. This is due mainly to French's

more detailed inflection systems; according to P&P theory, there is no reason to believe that the syntax of French is any more 'complex' than that of English or Japanese.

A language with detailed inflection systems in known as a synthetic language, and the opposite is known as an isolating language. There exist far more synthetic languages than French though, and if one of these were added to the system its module would be likely to be very large if measured by the simple lines-of-code metric used above. In general the more synthetic a language is, the less rigid its word order is [29]; this is a logical consequence of the fact that if inflections already indicate the relationships between the elements of a sentence, there is less need for the word order to do so. In theory then, French will have a slightly less rigid word order than English or Japanese, although in the context of the world's languages the difference between these languages in this respect is relatively minor.

The polysynthetic (i.e. very synthetic) and nonconfigurational languages mentioned in §2.2.3 are extreme cases of this phenomenon, with highly detailed inflection systems and very free word order. Since the focus of my system and the theory underlying it is to predict constituent orders across the world's languages, it is perhaps fundamentally better suited to isolating languages anyway, where word order is relatively rigid.

6.3.2 Potential for NLU/MT Systems

This system has separated the information required for syntactic realisation in a number of languages into a relatively large language-universal component and relatively small language-specific components. It seems likely that all the information required for the understanding of textual sentences should be able to be modularised in the same scalable way. Note that Chomsky's instance of Plato's problem is not restricted to the productive side of language; it applies equally to the receptive tasks.

If this parametric model of language variation were used to underpin a multilingual NLU system, the language-universal skeleton algorithms in the 'core' would involve vastly different computing techniques (eg. parsing of the input string, and perhaps heuristics to deal with potential ambiguities), but the details which the language-specific 'hook' methods were used to retrieve would be essentially the same as in my system.

6.3.3 Which Sentences to Produce?

Some of the sentences which the system generates may not actually be the most common way for speakers to express the desired semantics. A notable example is French questions; in spoken French, a question is often formed simply by adding *est-ce que* to the front of a sentence:

(6.1) *Il joue au football?*
he plays soccer
He plays soccer?

(6.2) *Est-ce qu' il joue au football?*
is-it that he plays soccer
Does he play soccer?

However, the alternative version, that the system will generate, is:

(6.3) *Joue-t-il au football?*
plays-he soccer
Does he play soccer?

The reason for this is that P&P theory focusses on the syntactic transformations which, for example, convert statements to questions. (6.1) and (6.3) have been shown to be related by a syntactic transformation, and produced from the same D-structure (excluding abstract features encoded on heads).

The relationship between (6.1) and (6.2), though, is of a different nature; (6.2) contains a subordinate clause, so syntactically these two sentences differ at a very deep level, in spite of the similar semantics. The sentence related to (6.2) in the same way as (6.1) is related to (6.3) would be

(6.4) *C' est que il joue au football.*
it is that he plays soccer
He plays soccer.

but this is not a commonly used form at all.

The reason it is more interesting in the context of this project to produce (6.3) rather than (6.2) is that it is generated from (6.1) by a syntactic transformation which can be modelled by a language-universal skeleton algorithm.

Perhaps the reason that French speakers prefer to produce (6.2) is that, while conforming to the structural rules of French syntax, it can be generated by a simple linear operation, the concatenation of *est-ce que* to the front of a statement.

Appendix A

Linguistics Glossary

This glossary provides definitions of a short list of general linguistic terms. It is not intended to cover concepts specific to the P&P theory; all the relevant P&P-specific terminology is explained in body of the report, but assuming knowledge of the general terms here.

active see **voice**

adposition a general term covering both prepositions and postpositions

auxiliary verb in complex verb constructions, the part which has a functional rather than semantic value, for example indicating tense; eg. *has* in *He has arrived*, and *are* in *They are playing football*

article a word equivalent to either the English *the* (definite article) or the English *a(n)* (indefinite article)

case a property of a nominal indicating its relationship with the verb in the clause; subjects are said to have 'nominative' case, objects are said to have 'accusative' case, for example

clause a part of a sentence (possibly a whole sentence) containing a subject and a finite verb

dual a possible value for the number of a nominal in some languages, indicating exactly two

lexical item an entry in the lexicon; essentially, a word

lexicon the vocabulary of a language; the elements available to it to combine to form phrases and sentences

main clause a clause which can stand alone and form a complete sentence

number a property of a nominal indicating how many entities are present; examples are singular and plural

passive see **voice**

paucal a possible value for the number of a nominal in some languages, indicating a small number larger than two

postposition a word equivalent to English prepositions (*in*, *on*, *at*, etc.) but which is placed after the nominal it is associated with

relative clause a clause which serves to modify one of the nominals in another clause; eg. *which I read* in *The book which I read was long*

semantic concerning the meaning of a word, phrase or sentence, rather than its structure

voice a property of a clause which determines the participant being emphasised; *John ate the cake* uses the active voice and emphasises *John*, whereas *The cake was eaten (by John)* uses the passive voice and emphasises *The cake*

Appendix B

Errors/Omissions

This appendix describes cases which are included in the scope of the implementation detailed in chapter 5, but for which the correct output has not been achieved. See §6.1.1 for more information.

Particularly in Japanese, there are probably other subtleties which I am not aware of; modelling the 'coarse-grain' features of this language was my main aim though.

B.1 French Perfect Tense Auxiliaries

The auxiliary verbs used in the perfect tense in French are not constant, but depend on the main verb being used.

(B.1) *Il a travaillé.*
he has worked
He (has) worked.

(B.2) *Il est arrivé.*
he is arrived
He (has) arrived.

In (B.1), the auxiliary *a* is derived from the verb *avoir* (*to have*), whereas in (B.2) *est* is from the verb *être* (*to be*). The choice is dictated purely by the main verb; *avoir* must be used with *travailler*, and *être* must be used with *arriver*.

The current implementation of French does not allow for this information to be stored with the lexical verbs of the language, and always uses an auxiliary derived from *avoir* (which is far more common) in the perfect tense.

B.2 Dislocation in French Questions

T-to-C movement in French questions moves the verb to a pre-subject position, but when the subject is not pronominal, it is left-dislocated:

(B.3) *Le garçon, joue-t-il au football?*
the boy plays-he soccer
Does the boy play soccer?

Here the pronoun *il* is in the normal subject position (specifier of TP) across which the verb *joue* has moved. (The *t* is inserted between them only for phonological reasons.) However, since the subject is not a pronoun, it is placed at the front of the sentence, and a 'dummy' pronoun is put in its usual position. Note that this is not movement in the sense often presented in this report, leaving behind only a phonologically null trace, because the original subject position is not empty.

The system does not handle this dislocation, and generates the subject in its usual position, with the verb in front of it:

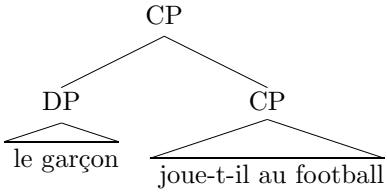
(B.4) **Joue le garçon au football?*
plays the boy soccer
Does the boy play soccer?

Note that if the subject really is a pronoun, no such dislocation occurs, and the system generates the correct sentence:

(B.5) *Joue-t-il au football?*
plays-he soccer
Does he play soccer?

The theory to account for this dislocation is that the 'duplicated' subject appears in a position adjoined to the CP:

(B.6)



However this clearly does not conform to the version of X′ theory presented in §3.2 and taken as fundamental to this project.

B.3 Japanese Adjectives

Japanese has two classes of adjectives, and the way in which inflections for tense and negation are applied differ dramatically between the two. Only one of these classes, those sometimes called the *-i* adjectives, have been accommodated.

Appendix C

Input Format

The sentences which the program generates are determined by input from the standard input stream; the most convenient way to run the program is to redirect the contents of a file from the command line. A sample input file is included here:

```
mainclause
predicate:  SEE
experiencer:  conversant singular +listener
theme:  unknown
.
subclause 0
theme:  relativewh
agent:  literal MAN plural definite
predicate:  BUY past
.
mainclause
patient:  relative APPLE 0 plural
agent:  conversant singular +speaker -listener masc
predicate:  EAT present
.
languages
french
english
.
```

From this input, the program would generate the following sentences:

(C.1) Que regardes-tu?

(C.2) What do you see?

(C.3) Je mange les pommes que les hommes ont achetées.

(C.4) I eat the apples which the men bought.

Any lines in the input which begin with the # character are ignored. Any blank lines are also ignored.

The input consists of a number of **blocks**. Each block ends with a line containing only a full stop. The first line of a block determines which type of block it is.

C.1 `mainclause` Blocks

A block beginning with the line `mainclause` specifies a main clause which should be generated by the program as a sentence.

For the rest of the `mainclause` block, the token before the colon on each line determines what type of line it is. A `mainclause` block consists of exactly one `predicate` line and a number of argument lines (in any order).

C.2 `subclause` Blocks

A block beginning with a line beginning with `subclause` specifies a subordinate clause which may be referred to in `mainclause` blocks, but is not to be generated as an independent sentence.

After `subclause` there must be a token which will act as the **subordinate clause ID** for this clause. In the example above, this token is `0`.

The body of a `subclause` block is identical to that of a `mainclause` block.

C.3 The `languages` Block

The `languages` block specifies the list of languages which the sentences should be generated in. One language is specified per line (after the first line). The values which may appear in these lines, for the languages currently implemented, are `english`, `french` and `japanese`.

The program generates the sentence corresponding to the first `mainclause` block found, in each of the languages in the `languages` block, in the order given; then the sentence corresponding to the second `mainclause` block found, in each of these languages; and so on.

(The input can include more than one `languages` block, in which case the contents of each are effectively concatenated.)

C.4 Lines Comprising `mainclause` and `subclause` Blocks

C.4.1 `predicate` Lines

After `predicate:` there must be a token specifying the 'concept' which the predicate of the clause is based on. The examples in the above are `SEE`, `BUY` and `EAT`; these correspond to the 'concepts' indicated in small caps in the body of the report (eg. SEE).

After the 'concept' any number of the following optional tokens may be added, in any order:

- `past` indicates that the clause is in the past
- `present` indicates that the clause is in the present
- `negative` indicates that the clause is negated
- `positive` indicates that the clause is not negated
- `question` indicates that the clause is a question
- `notquestion` indicates that the clause is not a question
- `active` indicates that the clause is in the active voice
- `passive` indicates that the clause is in the passive voice

These are all optional. By default a clause is in the present tense, is not negated, is not a question and is in the active voice.

Note that if a clause includes an `unknown` argument, it will be constructed as a question, regardless of the presence (or absence) of `question` or `notquestion`.

C.4.2 Argument Lines

Any lines in a `mainclause` block which are not `predicate` lines are argument lines. The token before the colon on an argument line is the semantic role of the argument. The next token on the line indicates which type of argument it is.

`literal` Argument Lines

After the argument type `literal` there must be a token specifying the 'concept' which the argument is based on. (As above, these correspond to the items represented in small caps in the body of the report.)

After the 'concept' any number of the following optional tokens may be added, in any order:

- `singular` indicates singular number
- `dual` indicates dual number
- `plural` indicates plural number
- `definite` indicates that the determiner is DEFINITE
- `indefinite` indicates that the determiner is INDEFINITE
- `pronoun` indicates that the argument is a PRONOUN

These are all optional. By default a `literal` argument is singular and definite.

`conversant` Argument Lines

After the argument type `conversant`, any number of the following optional tokens may be added, in any order:

- `singular` indicates singular number
- `dual` indicates dual number
- `plural` indicates plural number
- `+speaker` indicates that the speaker is included
- `-speaker` indicates that the speaker is not included
- `+listener` indicates that the listener is included
- `-listener` indicates that the listener is not included
- `masc` indicates HUMAN_MASC gender
- `fem` indicates HUMAN_FEM gender
- `mixed` indicates HUMAN_MIXED gender

These are all optional. By default a `conversant` argument is singular, includes neither the listener nor the speaker (i.e. is third person), and is masculine.

`unknown` Argument Lines

These are indicated by the token `unknown` after the colon.

`relativewh` Argument Lines

These are indicated by the token `relativewh` after the colon.

`relative` Argument Lines

After the argument type `relative` there must be a token specifying the 'concept' which the argument is based on. (As above, these correspond to the items represented in small caps in the body of the report.)

After the 'concept' there must be a token specifying the subordinate clause ID of the relative clause which will modify this argument. In the example above, this token is `0`. The `subclause` block with this subordinate clause ID *must occur before* the `relative` argument which refers to it.

After the subordinate clause ID, any number of the following optional tokens may be added, in any order:

- `singular` indicates singular number
- `dual` indicates dual number
- `plural` indicates plural number
- `definite` indicates that the determiner is DEFINITE
- `indefinite` indicates that the determiner is INDEFINITE

These are all optional. By default a `relative` argument is singular and definite.

C.5 A Note on 'Concepts' and Semantic Roles

The set of possible values for 'concepts' and semantic roles in this input is simply the set of all strings. There are no further restrictions. These tokens are simply matched with the tokens in the files storing the lexicon for each language, which are also read at runtime. This allows new 'concepts', and even new semantic roles, to be created and used at will, without any modification to the program.

The files which contain the lexicons for each language are named `lexicon.txt` and are stored in the directory corresponding to that language, eg. `english/lexicon.txt`, `french/lexicon.txt`.

The only instances where special meanings are assigned to concepts or semantic roles is the use of the semantic role 'copula', as described in §5.3.3, and the assumption that each language has an adposition associated with the concept BY for the construction of passives.

Bibliography

[1] Steven Abney, 'Statistical Methods and Linguistics'. pp. 1-26 in J. Klavans and P. Resnik (eds) *The Balancing Act*. MIT Press, Cambridge MA, 1996.

[2] James Allen, *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Menlo Park, California, 1987.

[3] Mark C. Baker, *The Atoms of Language: The mind's hidden rules of grammar*. Basic Books, New York, 2001.

[4] Mark C. Baker, *Lexical Categories: Verbs, Nouns and Adjectives*. Cambridge University Press, Cambridge, 2003.

[5] Andrew Carnie, *Syntax: A generative introduction*. Blackwell Publishers, Oxford, 2002.

[6] John J. Chew, *A transformational analysis of modern colloquial Japanese*. Mouton & Co., The Hague, 1973.

[7] Noam Chomsky, *Lectures on government and binding*. Foris Publications, Dordrecht, Holland, 1981.

[8] Noam Chomsky, *Knowledge of Language: Its nature, origin, and use*. Praeger, New York, 1986.

[9] V.J. Cook and Mark Newson, *Chomsky's Universal Grammar: An introduction*. Blackwell, Oxford, 1996, 2nd ed.

[10] Peter W. Culicover, *Principles and parameters : an introduction to syntactic theory*. Oxford University Press, New York, 1997.

[11] Evans. R., P. Piwek and L.J. Cahill, 'What is NLG?', *Proceedings of INLG 2002*. New York, USA, 2002.

[12] Edward Finegan, David Blair and Peter Collins, *Language: Its structure and use*. Harcourt, Sydney, 1997, 2nd ed.

[13] Naoki Fukui, 'The Principles-and-Parameters Approach: A comparative syntax of English and Japanese'. p. 327-369 in Masayoshi Shibatani and Theodora Bynon (eds) *Approaches to Language Typology*. Oxford University Press, Oxford, 1999.

[14] Nathalie Garric, *Introduction à la Linguistique*. Hachette, Paris, 2001.

[15] Paul L. Garvin, 'A Linguist's View of Language-Data Processing'. pp. 109-127 in Paul L. Garvin (ed) *Natural Language and the Computer*. McGraw-Hill, New York, 1963.

[16] Joseph H. Greenberg, 'Some universals of grammar with particular reference to the order of meaningful elements'. pp. 58-90 in Joseph H. Greenberg (ed) *Universals of Language*. MIT Press, Cambridge MA, 1963.

[17] Liliane Haegeman, *The Syntax of Negation*. Cambridge University Press, Cambridge, 1995.

[18] Liliane Haegeman and Jacqueline Guron, *English Grammar: A generative perspective*. Blackwell Publishers, Malden MA, 1999.

[19] Heidi Harley and Elizabeth Ritter, 1Person and Number in Pronouns: A feature-geometric analysis'. p. 482-526 in *Language*, vol. 78, number 3, 2002.

[20] Ray Jackendoff, *X′ Syntax: A Study of Phrase Structure*. MIT Press, Cambridge MA, 1977.

[21] Karen Jensen, George E. Heidorn and Stephen D. Richardson (eds) *Natural Language Processing: The PLNLP Approach*. Kluwer Academic Publishers, Dordrecht, Holland, 1993.

[22] Mìcheál Ó Siadhail, *Modern Irish: Grammatical structure and dialectic variation*. Cambridge University Press, Cambridge, 1989.

[23] Steven Pinker, *The Language Instinct*. Penguin Books, London, 1994.

[24] Andrew Radford, *Transformational Grammar: A first course*. Oxford University Press, Oxford, 1988.

[25] Andrew Radford, *Syntax: A Minimalist Introduction*. Cambridge University Press, Cambridge, 1997.

[26] Ehud Reiter, 'NLG vs. Templates', *Proceedings of the 5th European Workshop on Natural Language Generation*. Leiden, Netherlands, 1995.

[27] Ferdinand de Saussure, *Cours de Linguistique Générale*. Payot, Paris, 1972.

[28] Keith, Smillie, *Some Notes on Japanese Grammar*. [online] accessed May-June 2004, http://www.csse.monash.edu.au/~jwb/s_jgrammar.html

[29] Lindsay J. Whaley, *Introduction to Typology: The unity and diversity of language*. Sage Publications, Thousand Oaks, California, 1997.

[30] Personal communication with Ryo Mihara, May-June 2004.