Discovering Lexical Generalisations. A Supervised Machine Learning Approach to Inheritance Hierarchy Construction

Caroline Sporleder

Doctor of Philosophy
Institute for Communicating and Collaborative Systems
School of Informatics
University of Edinburgh
2004

Abstract

Grammar development over the last decades has seen a shift away from large inventories of grammar rules to richer lexical structures. Many modern grammar theories are highly lexicalised. But simply listing lexical entries typically results in an undesirable amount of redundancy. Lexical inheritance hierarchies, on the other hand, make it possible to capture linguistic generalisations and thereby reduce redundancy.

Inheritance hierarchies are usually constructed by hand but this is time-consuming and often impractical if a lexicon is very large. Constructing hierarchies automatically or semi-automatically facilitates a more systematic analysis of the lexical data. In addition, lexical data is often extracted automatically from corpora and this is likely to increase over the coming years. Therefore it makes sense to go a step further and automate the hierarchical organisation of lexical data too.

Previous approaches to automatic lexical inheritance hierarchy construction tended to focus on minimality criteria, aiming for hierarchies that minimised one or more criteria such as the number of path-value pairs, the number of nodes or the number of inheritance links (Petersen 2001, Barg 1996a, and in a slightly different context: Light 1994). Aiming for minimality is motivated by the fact that the conciseness of inheritance hierarchies is a main reason for their use. However, I will argue that there are several problems with minimality-based approaches. First, minimality is not well defined in the context of lexical inheritance hierarchies as there is a tension between different minimality criteria. Second, minimality-based approaches tend to underestimate the importance of linguistic plausibility. While such approaches start with a definition of minimal redundancy and then try to prove that this leads to plausible hierarchies, the approach suggested here takes the opposite direction. It starts with a manually built hierarchy to which a supervised machine learning algorithm is applied with the aim of finding a set of formal criteria that can guide the construction of plausible hierarchies. Taking this direction means that it is more likely that the selected criteria do in fact lead to plausible hierarchies. Using a machine learning technique also has the advantage that the set of criteria can be much larger than in hand-crafted definitions. Consequently, one can define conciseness in very broad terms, taking into account interdependencies in the data as well as simple minimality criteria. This leads to a more fine-grained model of hierarchy quality.

In practice, the method proposed here consists of two components: Galois lattices are used to define the search space as the set of all generalisations over the input lexicon. Maximum entropy models which have been trained on a manually built hierarchy are then applied to the

lattice of the input lexicon to distinguish between plausible and implausible generalisations based on the formal criteria that were found in the training step. An inheritance hierarchy is then derived by pruning implausible generalisations. The hierarchy is automatically evaluated by matching it to a manually built hierarchy for the input lexicon.

Automatically constructing lexical hierarchies is a hard task, partly because what is considered the best hierarchy for a lexicon is to some extent subjective. Supervised learning methods also suffer from a lack of suitable training data. Hence, a semi-automatic architecture may be best suited for the task. Therefore, the performance of the system has been tested using a semi-automatic as well as an automatic architecture and it has also been compared to the performance achieved by the pruning algorithm suggested by Petersen (2001). The findings show that the method proposed here is well suited for semi-automatic hierarchy construction.

Acknowledgements

I am grateful to my supervisors Alex Lascarides and Miles Osborne for their support and advice over the last three years. Alex was a continuous source of encouragement and also provided detailed and critical feedback. Miles provided useful help with the statistical aspects of this thesis. I would also like to thank my examiners David Weir and Ewan Klein for useful comments and feedback on this thesis.

Many thanks to Steve Finch for first introducing me to Galois lattices and giving a lot of helpful advice at the early stages of this thesis. I am also indebted to Ann Copestake and Jason Baldridge. Ann provided help with the LKB system and was always very responsive to my queries. Jason did the same for the *openNLP Maxent* package. I have also benefitted enormously from interesting discussions with Steve Clark, James Curran, Dan Flickinger, Julia Hockenmaier, Frank Keller, Mirella Lapata, Harald Lüngen and Wiebke Petersen. James also provided help on some implementational aspects and Steve, Mirella and Harald read individual chapters of this thesis and provided detailed feedback. Dziękuję bardzo to Kaska Porayska-Pomsta for sharing an office with me at a time when we were both writing up and providing moral support during the final phase of this dissertation. I am also grateful to the Informatics computing staff, who did their best to find sufficient computing resources for me to run my experiments on. Finally, I would like to thank my friends and family for their continuous encouragement and for providing welcome distraction from thinking about inheritance hierarchies.

The research reported in this thesis was supported by a University of Edinburgh Faculty of Science and Engineering scholarship. I am also grateful for the travel grants I received from the Informatics Graduate School and the Association for Computational Linguistics.

Thesis submitted: August 5, 2003
Thesis defended: January 9, 2004

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Caroline Sporleder*)

Contents

Chapter 1

Introduction

1.1 Motivation for Lexical Inheritance Hierarchies

The lexicon is of crucial importance for many natural language processing (NLP) applications. This is particularly true because modern grammar theories tend to be highly lexicalised. Traditionally, the lexicon was viewed as the "idiosyncratic part" of the grammar of a language (see Bloomfield (1933)) and every kind of linguistic behaviour that was regular enough to be predicted by rule was excluded from it. However, over the last three or four decades there has been a shift away from a sophisticated rule system to a more complex lexicon. Compared to non-lexicalist grammars, such as *Government and Binding Theory* (Chomsky, 1981), lexicalist grammars, such as *Head-Driven Phrase Structure Grammar* (Pollard and Sag, 1987, 1994), *Categorial Grammar* (Wood, 1993) or *Lexicalised Tree Adjoining Grammar* (Joshi and Shabes, 1991), only use a very basic system of highly abstract rules and put most of the linguistic information in the lexicon. This leads to highly complex lexicons, where each entry specifies details about several linguistic levels, such as phonology, orthography, how the entry behaves syntactically, what complements the entry takes, how it is inflected, what meaning is conveyed by it and how its meaning is combined with the meaning of its complements. It also means that the lexicon no longer contains only idiosyncratic information but also a high amount of regularity.

For example, the two verbs *continue* and *seem* behave in a syntactically similar way. They both take a noun phrase (NP) as their subject and a complementizer phrase (CP) introduced by *to* as their complement (see example (1.1)). They also assign similar semantic roles to their complements: the subject NP refers to the agent of the action described by the verb and it is

also the agent of the action described by the CP (e.g. it is the dog that chases the cat). They even follow a similar inflectional pattern, as both build their past tense by adding *-ed*.

(1.1) a. The dog continues to chase the cat.

b. The dog seems to chase the cat.

Representing the lexicon as a list of lexical entries does not account for these regularities. Consequently, there has been a lot of research into how lexicons can be represented in a way that captures generalisations. Morphological regularities are often captured by so-called *lexical rules* (see e.g. Flickinger (1987)). For example, the third singular rule states that verbs build their third singular form by adding the suffix *-s* to their root. This is true for most verbs. However, if a verb has an irregular third singular form, e.g. *be*, this form is explicitly stated in the lexicon. While lexical rules are good at capturing relations between different forms or different syntactic behaviours (e.g. active *vs.* passive) of one lexical entry, they are not so good at capturing generalisations that do not involve relating one form to another but involve similarities between sets of lexical entries (like *continue* and *seem*). They are also less suitable for capturing regularities which come with many subregularities (see the discussion below). For these kinds of regularity, lexical inheritance hierarchies are more suitable.

Inheritance hierarchies allow one to organise lexical entries into broader classes. Properties which are shared by all (or most) members of a class are *inherited* from the class node rather than listed individually for every single object. For example, the two verbs *seem* and *continue* both belong to the class *subject raising verb*. All subject raising verbs have in common that:

- they subcategorise for an NP subject
- they subcategorise for a CP complement
- their subject is also the subject of their complement

In *Head-Driven Phrase Structure Grammar* this can be represented by the following feature structure:[1]

$$\textit{subject raising verb}\left[\text{ARG-ST}: \left\langle \boxed{1}\left[\text{HEAD}\ \textit{noun}\right], \underset{\textit{compl. phrase}}{\left[\text{SPR}\ \boxed{1}\right]} \right\rangle\right]$$

[1] Section 2.1 gives a formal definition of the term 'feature structure', however a basic familiarity of Head-Driven Phrase Structure Grammar or other constraint-based grammars is presupposed.

The lexical inheritance hierarchy will then contain *subject raising verb* as a superclass from which verbs like *continue* and *seem* inherit the properties that subject raising verbs have in common (see Figure 1.1). The lexical entries of *continue* and *seem* only have to specify idiosyncratic properties, such as the orthography.

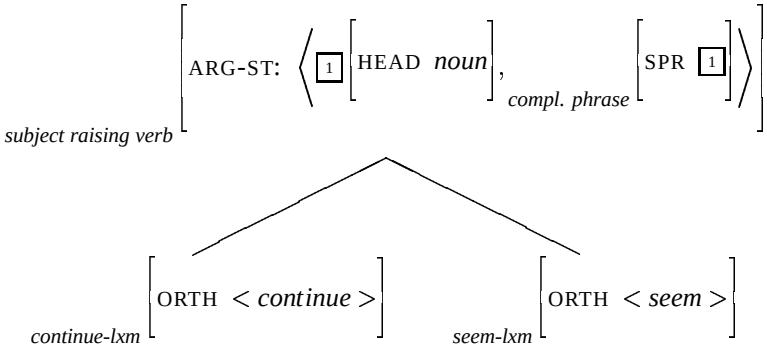


Figure 1.1: Fragment of an inheritance hierarchy

Linguistic regularities can occur in all areas of the lexicon. The similar syntactic behaviour of *subject raising verbs* is just one example. Other examples are morphological regularities, e.g. verbs containing the prefix *un-* usually describe a change of state (e.g. *unfasten*, cf. Light 1996), phonological regularities, e.g. in German voiced consonants are devoiced when they occur in a syllable final position, syntactico-semantic regularities, e.g. ergative-transitive subcategorisation alternations in cooking verbs (*Peter baked the bread* vs. *The bread baked*, cf. Levin 1993) etc.

Often regularities come with many sub-regularities and exceptions and these can be represented relatively well in an inheritance hierarchy but not so well by other means, like lexical rules. As an example of how hierarchies can deal with such cases consider Riehemann (1998), who discusses semi-regular productive affixation using German *bar*-adjectives. These adjectives are formed by attaching the suffix *-bar* ("-able") to transitive verbs. In the prototypical case, the direct object of the verb becomes the subject of the adjective (see Example 1.2). Furthermore, the semantics of the verb is usually constrained to have an intentional aspect for the *bar*-affixation to be permissible. Adding the suffix *-bar* typically results in the notion of "possibility" being added to the semantics of the verb.

(1.2) a. Sie bemerken die Veränderung.
They notice the change.

b. Die Veränderung ist bemerkbar.
The change is noticeable.

However, there are several cases of irregular or sub-regular formation of *bar*-adjectives. These can affect various areas:

- phonological: e.g. dropping of *-ig* in the stem (*entschuldigen* ⇒ *entschuldbar*, "excusable")
- semantic: e.g. no notion of possibility (*fruchtbar* "fruitful")
- syntactic: from non-transitive verbs, e.g. intransitive verbs or verbs with dative or prepositional objects (*entrinnen* (+ DAT) ⇒ *unentrinnbar*, "inescapable")

All of the above are sub-regularities rather than true exceptions, i.e. they do apply to classes of words. For example, several verbs which do not have a direct (i.e. accusative) object can form *bar*-adjectives in violation of the general rule that the verb has to be transitive (e.g. *unausweichbar* "inevitable", *unwiderstehbar* "irresistible"). Riehemann argues that these sub-regularities cannot be treated adequately by a lexical rule because it would have specify all the classes to which it applies, thereby potentially replicating class information that is specified elsewhere in the (hierarchical) lexicon. Instead she suggests to deal with those cases exclusively by inheritance. The hierarchy proposed by her is depicted in Figure 1.2. For reasons of space, it only shows the types and not the associated feature structures. The symbol "|" indicates disjunction of types, "&" indicates the boundary of a dimension (see page 31 for a definition of dimension) and has wide scope over "|", "…" indicates unspecified types. The leaf nodes of the hierarchy represent lexicalised *bar*-adjectives, with the exception of *reg-bar-adj*, which is underspecified for phonology and orthography and allows the productive creation of new *bar*-adjectives.[2] The type *trans-bar-adj* subsumes adjectives which are formed in a completely regular way. It inherits from *externalised*, which encodes the fact that the object of the verb becomes the subject of the adjective. The type *externalised* also subsumes passive constructions. Adjectives that are not derived from a transitive verb (such as *unentrinnbar*) do not inherit from *trans-bar-adj* but instead have their own supertypes specifying a different argument structure modification. For example, the type *prep-bar-adj* specifies that the head of

[2] Riehemann makes a closed-world assumption, i.e. it is not possible to add new subtypes to the lexicon but new forms can be created on the fly by applying the productive type schema *reg-bar-adj*.

the prepositional object of the verb becomes the subject of the adjective, the type *dative-bar-adj* specifies that the dative object of the verb becomes the subject of the adjective and the type *intr-bar-adj* specifies that the subject of the verb is also the subject of the adjective. The types *trans-bar-adj*, *dative-bar-adj*, *prep-bar-adj* and *intr-bar-adj* all inherit the notion of "possibility" from the type *possibility* via the type *poss-bar-adj*. Making *possibility* an individual type rather than specifying the introduction of "possibility" directly at *pos-bar-adj* is motivated by the fact that there are other affixes which also introduce the notion of "possibility" (such as *-lich*). Adding the type *bar-adj* between *poss-bar-adj* and *affixed* means that *bar*-adjectives which do not carry the notion of "possibility" (such as *fruchtbar*) can also be dealt with: these simply inherit directly from *bar-adj* rather than from *poss-bar-adj*.

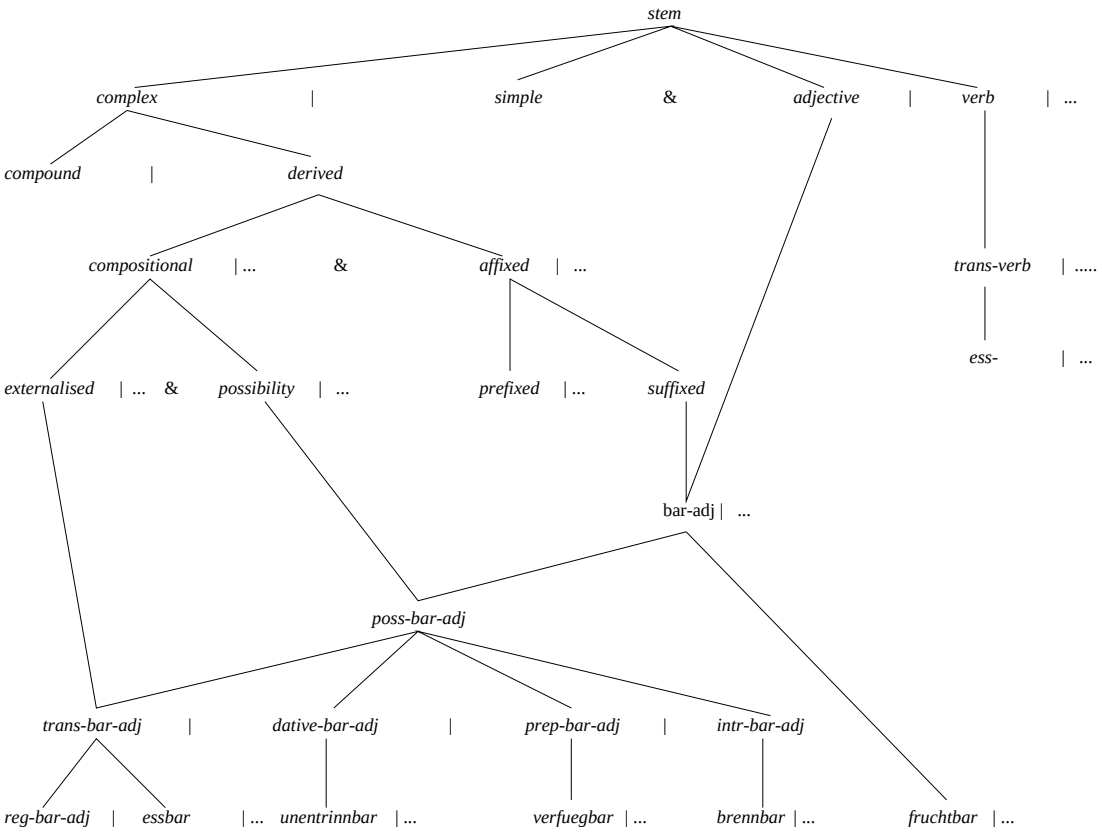


Figure 1.2: Riehemann's 1998 hierarchy for *bar*-adjectives

This example shows that regularities and sub-regularities can be dealt with efficiently in lexical inheritance hierarchies. Inheritance hierarchies are often a better tool for the representation of regularities than lexical rules. Compared with flat lists of lexical entries they have the

advantage that they can also make predictions about words not in the lexicon. For instance, the hierarchy in Figure 1.2 predicts that *bar*-adjectives can be formed productively from transitive verbs. For example, suppose an NLP system comes across the sentence in Example 1.3 and *faxbar* is unknown. Parsing the sentence would allow one to hypothesise that *faxbar* is an adjective. After identifying that *faxbar* ends in the suffix *-bar* one can assume that a new entry for *faxbar* should probably be inserted as a terminal node in the hierarchy in Figure 1.2. If the entry is inserted under *reg-bar-adj*, i.e. the type for newly formed *bar*-adjectives, it inherits several additional properties, like the fact that it carries a notion of possibility and that there is a corresponding verb *faxen*. This is information that could not be inferred directly if the lexicon was represented as a flat list of entries.[3]

(1.3) a. Dieses Document ist faxbar.
This document is faxable.

This means that lexical inheritance hierarchies are not only linguistically insightful in a purely theoretical way but also of practical use for all robust NLP applications that make use of a lexicon. Because a lexicon can never be complete, robust applications should be able to extend the lexicon on the fly or at least be able to deal with unknown words. Well-structured lexical inheritance hierarchies make it possible to infer properties of a new word that cannot be inferred from context alone, therefore processing unknown words is easier than with flat lexicons. Since they can capture all kinds of lexical generalisations, inheritance hierarchies will be useful for various applications, such as parsing, natural language generation, speech processing etc. Research on how lexical inheritance hierarchies can be utilised to process unknown words has been undertaken by Zernik and Dyer (1986), Zernik (1987), Kilbury *et al.* (1994), Barg and Walther (1998) and Barg and Kilbury (2000).

Their ability to generalise beyond the lexicon is a major advantage of lexical inheritance hierarchies, however there are further advantages. Because they provide a high level of abstraction and modularity (Emele and Zajac, 1990) and an inference mechanism (namely inheritance), they are easier to maintain and update than flat lexicons. When adding a new lexical entry one need only specify its supertype(s) and the idiosyncrasies of the entry with respect to its supertypes. All non-idiosyncratic properties will be inherited. Because only idiosyncratic properties have to be listed for each lexical entry, a hierarchical lexicon is also less error-prone than a flat one and it is easier to maintain consistency.

[3]The approach outlined here bares some similarity with Light (1996). Light investigated how morphological cues can be used to infer the semantic properties of a word. He did not, however, use an inheritance hierarchy but rather explicitly coded rules to relate morphological cues to semantic properties.

For a decent sized lexicon the difference in size between an entry which lists all properties and one which lists only those that cannot be inherited can be huge. For example, the average entry size in a hierarchical lexicon like the LinGO ERG lexicon[4] is approximately 103 bytes. This includes the idiosyncratic properties for each entry and information about the node from which it inherits. However, if no use is made of inheritance and every entry lists all properties this rises to approximately 8,826 bytes per entry on average, i.e. more than 85 times as much.

1.2 Motivation for Automation

Lexical inheritance hierarchies are typically constructed manually, often parallel to the overall development of a lexicon. However, sometimes manual construction may not be feasible. Constructing a lexicon (flat or hierarchical) is commonly very time-consuming and often the bottleneck for NLP applications. As a result, there has been a lot of interest in the automatic extraction of lexical knowledge from corpora in recent years. Information that has been extracted automatically includes: subcategorisation frames (e.g. Briscoe and Carroll 1997), hyponymy relations (e.g. Hearst (1992)), and lexical semantics (e.g. Lapata (2000), Light (1996)).

Even for automatically extracted data it can be beneficial to choose a hierarchical representation rather than a flat one, as the former allows generalisations beyond the words contained in the lexicon. This means that the non-terminal classes of a hierarchy can, for example, be exploited to process unknown words. The previous section outlined how information about unknown *bar*-adjectives in German can be inferred from the manually built hierarchy provided by Riehemann (1998). While automatically extracted lexicons are usually not as detailed as manually built ones, it is still possible to extract information that leads to interesting generalisations. For example, consider the following mini-corpus:

(1.4)
a. The trustee believed the journalist would not attend the meeting.
b. The trustees approved the suggestion.
c. The workers did not trust their boss.
d. The new trainee introduced himself.
e. The company does not employ any trainees.
f. She trains as a doctor.
g. The amputee could leave hospital after five weeks.

[4] `http://lingo.stanford.edu/` (20.7.03)

h. Amputees often have to undergo extensive medical rehabilitation.

i. They had to amputate his hand.

For the words *trustee*, *trainee*, and *amputee* it is relatively easy to extract the following information:

- they are nouns (because they can occur directly after a determiner and seem to be able to take the suffix *-s*)
- they are count rather than mass nouns (because they seem to have singular as well as plural forms)
- they contain the suffix *-ee* and are derived from a verb (because there seems to be a corresponding verb form that does not end in *-ee*)
- they have animate referents (because they occur as subjects of verbs like *believed* and *introduce oneself* which usually take animate subjects)

Information about other words can be extracted in a similar way. Then a hierarchy can be built automatically for the extracted lexicon. A good hierarchy would contain a node *ee-noun* which inherits from a node *count-noun* and encodes the fact that there is a class of nouns which take the suffix *-ee* and have an animate referent (Figure 1.3). If this hierarchy is then used in a parsing system, the existence of the node *ee-noun* allows a generalisation to *ee*-nouns that are not contained in the lexicon, such as *blackmailee*. Hence the system can infer that the referent of *blackmailee* is animate even if this cannot be inferred from the context alone (e.g. *The police could not track down the blackmailee.* vs. *The police could not track down the bike.*)[5]

This shows that building a hierarchy can be beneficial even for an automatically extracted lexicon. If the lexical data itself is extracted automatically (or semi-automatically) it makes sense to go a step further and automate the construction of the hierarchical representation of the lexicon too.

Building lexical inheritance hierarchies automatically or semi-automatically has several advantages. Constructing hierarchies manually can be time-consuming, tedious and impractical.

[5]Note, however, that the ability to generalise beyond the current lexicon also bares the risk of *overgeneralisation*. For example, the system may assume that all instances of a class have a certain property because all instances it has seen so far did indeed have this property. But there is still a possibility that instances seen in future will not share the property. Consequently, a system which combines automatic extraction with automatic hierarchy building needs to incorporate some kind of backtracking mechanism which allows a modification of earlier decisions if a word occurs in a context which suggests that such a modification is necessary.

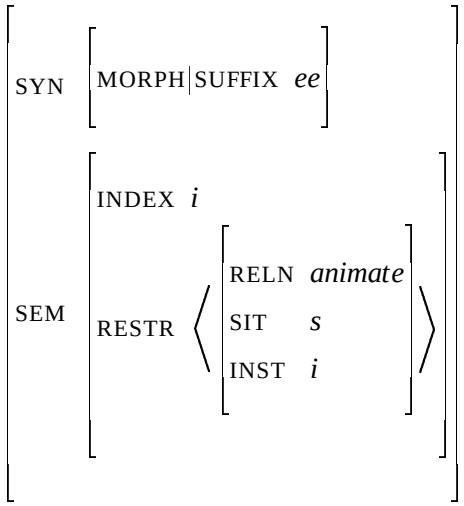


Figure 1.3: Feature structure for *ee-noun*

Also, with large data sets, the most plausible clustering of linguistic objects into classes is not always obvious at first glance.

Furthermore, manually constructed hierarchies for large data sets will often be based primarily on linguistic intuition and experience rather than on a systematic analysis of the data, since the data is often too big for close manual inspection. A computer program deriving hierarchies automatically, however, may take a more data-oriented approach, e.g. by systematically searching for the number of objects sharing a certain property. Ideally, the resultant hierarchy will reflect certain tendencies in the data set better. Therefore, it may also help to gain new insights into the data set, revealing interesting interrelations between lexical entries.

However, constructing inheritance hierarchies is to some degree subjective. Furthermore, no lexicon is 'complete' or provides an exhaustive analysis of its entries. Therefore a semi-automatic approach seems to be the best way forward as it can combine the best of both worlds: data orientation and linguistic insight supplied by the user.

1.3 Suggested Approach

So far there have been few attempts to construct lexical inheritance hierarchies automatically for flat lexicons (see Barg 1996a, Petersen 2001 and with a slightly different focus: Light 1994). All of them aimed to derive hierarchies that are in some sense "minimal", where minimality is

usually fairly narrowly defined. Possible criteria for minimality are, for example, the number of nodes, path-value pairs or inheritance links in the hierarchy. Sometimes such criteria are combined into one minimality criterion using simple or weighted summation (Barg 1996a, Light 1994). One motivation for using minimality criteria is that inheritance hierarchies are supposed to reduce the redundancy contained in the lexicon. A minimality-based approach also fits in with the logical principle of *Parsimony* (see page 78). However, focusing on minimal redundancy is problematic for several reasons. First, minimal redundancy is ill-defined in the context of lexical inheritance hierarchies. It could be the number of nodes, path-value pairs or inheritance links, or some (complex) combination of those. Furthermore, there is often a tension between different criteria. For example, a hierarchy which contains the minimal number of path-value pairs will not normally contain the minimal number of nodes and *vice versa* (see the example in Section 3.4).

Second, lexical inheritance hierarchies should also be linguistically plausible and insightful. It is, of course, impossible to manually define exact rules which determine when a hierarchy is plausible and when it is not. One reason for this is that linguistic plausibility is to some extent a fuzzy and subjective concept. However, instead of using an *ad hoc* definition of conciseness which then may or may not lead to hierarchies which are not only concise but also plausible, one can go in the opposite direction by starting with a manually built hierarchy and extracting a definition of conciseness from it. This can be done by using the manually built hierarchy as training data in a supervised machine learning approach. The aim of the learning system is then to learn which formal criteria are especially useful in guiding an automatic construction algorithm towards hierarchies that are both, concise *and* plausible. This is the approach taken here.

The system I propose consists of two main parts. The search space is defined by a Galois lattice. Galois lattices will be discussed in more detail in Chapter 4. A Galois lattice is built by taking the upward closure over the set of lexical entries, i.e. it is a partial order of all non-empty intersections of sub-sets of entries. Each intersection of a sub-set of lexical entries can be viewed as a (potential) generalisation over this set of entries. Not all generalisations that can be drawn are linguistically meaningful, some may be accidental. Lexical inheritance hierarchies tend to express only meaningful generalisations. Consequently, the search for a good inheritance hierarchy can be re-defined as a search for a set of meaningful generalisations. Galois lattices allow one to do this by making the potential generalisations explicit. Viewing the problem in this way means that one does not have to enumerate all sound hierarchies for an

input lexicon (see Section 3.5.1).

The second part of the system is a machine learning component whose task it is to distinguish between generalisations that are linguistically meaningful or plausible and those that are not. The machine learning component takes the form of a maximum entropy model. The model is trained on an existing (i.e. manually built) hierarchy.

To derive an inheritance hierarchy for a new lexicon, the maximum entropy model is applied to the Galois lattice of the input lexicon to separate good generalisation from implausible generalisations. The former are retained, the latter pruned. Using a manually built hierarchy to decide which criteria are useful for distinguishing plausible from implausible generalisations means that it is more likely that these criteria do in fact lead to plausible hierarchies. Furthermore, the fact that the criteria can be combined automatically means that it is possible to include many more criteria than can be included in hand crafted definitions of minimality. Consequently, one can arrive at a more fine-grained definition of "conciseness"; one which does not only take simple minimality criteria into account but also other factors, such as interdependencies in the data.

This thesis presents an experimental system which makes use of this approach. The construction of a new hierarchy can be done both automatically and semi-automatically and both will be tested. But since Galois lattices are inherently monotonic the system is so far restricted to monotonic hierarchies. However, a possible extension to non-monotonic hierarchies is outlined in the final chapter of this thesis. The system includes an automatic evaluation method which assesses a derived hierarchy by matching it to the manually built hierarchy for the lexicon and determining how similar they are. The minimality-based pruning technique used by Petersen (2001) has also been tested and compared to the maximum entropy approach.

The system has been applied to real and relatively big lexicons rather than to purpose-built toy lexicons. Some points may be easier illustrated with a small lexicon but building a hierarchy for a real lexicon often involves deciding how generalisations in different areas (e.g. generalisations over verbs and generalisations over nouns) should be combined. Hence, it seems that the task of capturing a set of generalisations well cannot be reduced to a set of smaller tasks, where each tasks involves capturing a smaller set of generalisations in isolation.

1.4 Overview of the Thesis and Published Work

The thesis is organised as follows:

Chapter 2 introduces (Typed) Feature Logic and provides formal definitions for inheritance

hierarchies and related concepts. It also gives an overview of how inheritance hierarchies are used in linguistics and summarises previous research in the area of automatic inheritance hierarchy construction.

Chapter 3 describes the task in more detail and outlines some of the difficulties. It discusses which criteria of hierarchy quality have been suggested in the literature and why criteria that focus on a definition of minimality may not be the best way forward. The chapter concludes with a more detailed outline of the approach taken in this thesis.

Chapter 4 introduces Galois lattices. It provides the relevant definitions and discusses some advantages and problems of a lattice-based approach.

Chapter 5 introduces maximum entropy modelling. It starts by giving a formal outline of statistical modelling in general. Then maximum entropy models are formally defined. The chapter finishes by describing the maximum entropy features that are used to implement node context.

Chapter 6 gives some more details of the implementation of the system and discusses the lexicons that have been used for training and testing and how they are pre-processed. The chapter also deals with how training data can be sampled if the training lexicon turns out to be too big (and too imbalanced) to be used in its entirety. It finishes by outlining the advantages and disadvantages of different evaluation methods and gives a detailed description of the method adopted here.

Finally, Chapter 7 contains the experiments that have been conducted. It starts with a discussion of upper and lower bounds for the task and establishes two random baselines against which the results can be evaluated. The chapter finishes by discussing eight different experiments which investigate the performance of the maximum entropy approach in an automatic and two different semi-automatic architectures and compare it to the performance achieved by Petersen's (2001) approach.

Some of the material presented in this thesis has been published. This applies to Chapter 6, which contains material (evaluation by error-correcting matching) which also occurs in (Sporleder, 2002a,b,c).

1.5 A Note on Typography

Throughout this dissertation the following typographical conventions are used:

- Types in inheritance hierarchies and feature structures are set in *italics* (see Figure 1.1).

- In feature structures, types occur to the left of the feature structure.
- In inheritance hierarchies, types (or class/node names in untyped hierarchies) occur in *italics* above the feature structure to which they refer. Also, the surrounding square brackets of a feature structure in a hierarchy are usually omitted if the feature-structure is not nested.
- Technical terms that will be presupposed once they have been introduced are set in **bold face** when they are first defined.
- All other technical terms are set in *italics* when first introduced.
- Attribute names are set in SMALL CAPS (feature structures) or CAPITALS (hierarchies).
- Attributes in a path are separated by '|'.

Chapter 2

Lexical Inheritance Hierarchies

This chapter introduces lexical inheritance hierarchies, discusses their role in linguistics and outlines previous approaches to automatic hierarchy construction. Section 2.1 gives an overview of (Typed) Feature Logic, which was originally introduced by Kasper and Rounds (1986). Section 2.2 introduces the different types of inheritance hierarchies and Section 2.4 describes their use in linguistics. The chapter finishes with an overview of previous approaches to automatic hierarchy construction (Section 2.5).

2.1 Formal Definitions

Informally, a lexical inheritance hierarchy is a set of linguistic classes (the **nodes** of the hierarchy) ordered by specificity and via connections by **inheritance links** (or *edges*), such that more specific classes inherit properties from (compatible) less specific classes.

The linguistic classes in an inheritance hierarchy are usually represented by *feature structures* (*FSs*) or *typed feature structures* (*TFSs*). Feature structures are sometimes also called *attribute-value matrices*.

An **attribute-value pair** (*AVP*) is a partial function from an attribute to a value. A feature structure that consists of a single attribute-value pair (or a sequence of attributes together with the associate value) is called an **atomic feature structure**,[1] see Figure 2.1 below.[2]

Sometimes the term *feature* is used to refer to either attribute-value pairs or just to attributes. However, I will refrain from using the term in this context to avoid confusion with "features" in

[1] The term *atomic feature structure* will later be extended to feature structures that contain a single reentrancy.

[2] This use of the term *atomic feature-structure* deviates from Shieber's (1986) usage of the term. Shieber employs it to refer to simple values of attributes (i.e. values which are not themselves feature structures). Carpenter (1993) uses the term *atomic value* to refer to these values.

$$\begin{bmatrix} \text{ORTH:} & < cookie > \end{bmatrix}$$

Figure 2.1: Atomic feature structure

maximum entropy models. That is, attribute-value pairs will always be called *attribute-value pairs* and the term *feature* will be reserved for maximum entropy features.

A **complex feature structure** is a set of two or more attribute-value pairs (Figure 2.2).[3]

$$\begin{bmatrix} \text{ORTH:} & < cookie > \\ \text{PHON:} & /''kUki:/ \end{bmatrix}$$

Figure 2.2: Complex feature structure

The values of attributes can themselves be feature structures, i.e. feature structures can be embedded, as in Figure 2.3.

$$\begin{bmatrix} \text{ORTH:} & < cookie > \\ \text{PHON:} & /''kUki:/ \\ \text{AGR:} & \begin{bmatrix} \text{PER:} & 3 \\ \text{NUM:} & sg \end{bmatrix} \end{bmatrix}$$

Figure 2.3: Embedded feature structure

Following Carpenter (1993), I will use the term **atomic value** to distinguish simple values

[3]The machine-readable phonetic alphabet SAMPA has been used for the phonetic transcription: `http://www.phon.ucl.ac.uk/home/sampa/home.htm` (13.2.03). Note, double quotation marks (″) indicate primary stress.

like <*cookie*> (the value of ORTH) from values which are themselves feature structures.

The above representation of a feature structure is called an **AVM (attribute-value matrix) diagram** but feature structures can also be viewed and represented as (rooted) *directed acyclic graphs* (**DAGs**), with attributes labelling the arcs and values labelling the terminal nodes of the graph. For example, the feature structure in Figure 2.3 corresponds to the DAG in Figure 2.4.

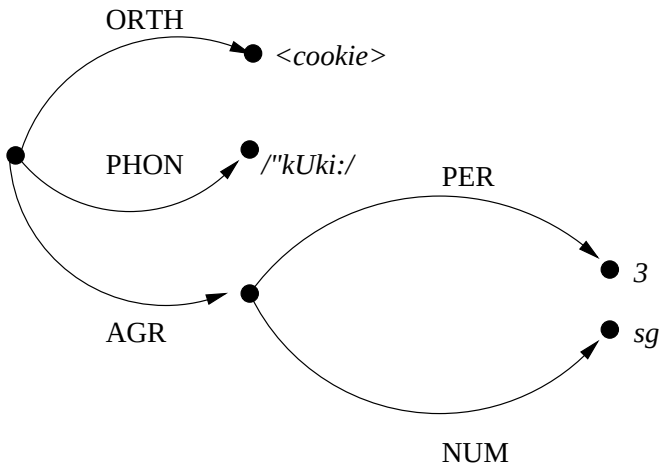


Figure 2.4: Feature structure as a DAG

A sequence of arcs/attributes that can be followed from the root node is called a **path** into the feature structure. For example, the attribute AGR is also a path in the feature structure in Figure 2.3. I will use the term **complete path** to refer to a path from the root node to a terminal node (i.e. an atomic value). The path AGR|NUM:*sg*[4] is, in fact, a complete path in Figures 2.3 and 2.4. Every complete path is also an atomic feature structure, i.e. each complex feature structure is formally a set of atomic feature structures.

Different paths do not have to lead to different nodes. The two DAGs in Figure 2.5 are perfectly acceptable. A situation where two paths lead to the same node is called a **reentrancy** (sometimes also called *structure-sharing* or *co-indexation*). Reentrancies play an important part in feature value grammars as many linguistic facts are best modelled in this way. A reentrancy expresses the fact that two values are equal. The values themselves, however, can be unknown (as in Figure 1.1). In AVM diagrams, reentrancies are represented by boxed integers. If two (or more) paths in a feature structure lead to the same boxed integer it means that they are reentrant. Figure 2.6 shows the AVM diagrams of the two reentrant DAGs in Figure 2.5.

[4]The symbol "|" is commonly used in the HPSG literature as a delimiter between the attributes of a path.

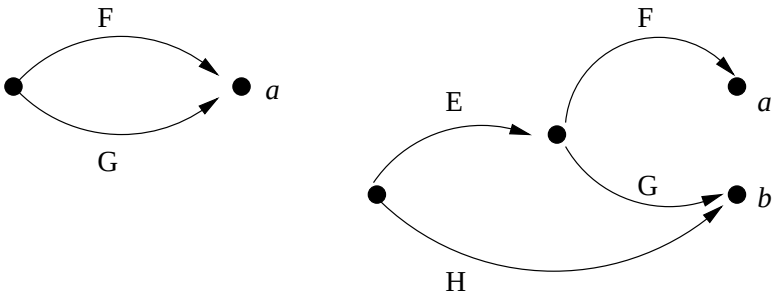


Figure 2.5: Reentrancies in DAGs

Figure 2.6: Reentrancies in feature structures

Note, that it is important to distinguish node-identity (i.e token identity) from value identity (i.e. type identity). Two paths can have identical values and still lead to different nodes as in Figure 2.7 below.

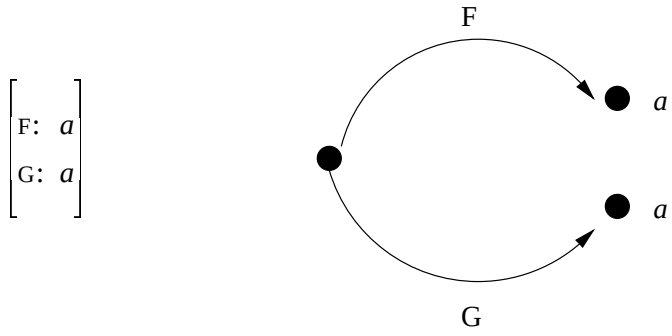


Figure 2.7: Value identity (no reentrancy)

The term *atomic feature structure* is usually extended to include a feature structure consisting of a single reentrancy between two paths but without a value being specified for the reentrant paths (Carpenter, 1993, p. 20), as in Figure 2.8.

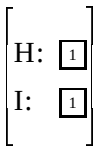


Figure 2.8: Atomic reentrant feature structure

The DAG view of feature structures gives rise to the following formal definitions (cf. Keller 1993, p. 22; Carpenter 1993, pp 17 & 20):

Definition 2.1 (Feature Structure) *A* feature structure *(over a set of atomic values Val and a set of attributes Attr) is a tuple:*

$A =< Q, q_0, \delta, \pi >$ *where* :

- Q : *a finite set of nodes rooted at* q_0
- $q_0 \in Q$: *the root node*
- $\delta : (Q \times Attr) \rightarrow Q$: *the partial* attribute value *function, where* δ *may be extended to a partial function* $\delta* : (Q \times Attr*) \rightarrow Q$ *from node-path pairs to nodes in the following way:*
  1. $\delta*(q, \varepsilon) = q$, *where* ε *is the empty path*
  2. $\delta(q, p) = \delta*(\delta(q, a), p')$, *for* $a \in Attr$
- $\pi : Q \rightarrow Val$: *the partial* atomic value *function, such that* $\forall q \in Q, \forall a \in Attr$ *if* $\pi(q)$ *is defined, then* $\delta(q, a)$ *is undefined.*

subject to the additional constraints:

- *Rootedness: there is no node q or attribute a such that* $\delta(q, a) = q_0$.
- *Connectedness: for every node q in Q there is a path p such that* $\delta*(q_0, p) = q$.
- *Acyclicity: the resulting graph is acyclic in that there is no node q or (non-empty) path p such that* $\delta*(q, p) = q$

Definition 2.2 (Atomic Feature Structure) *A feature structure is* atomic *if it is one of the following two forms:*

- Path Value*: the feature structure contains a single path assigned to an atomic value.*
- Path Sharing*: the feature structure contains only a pair of (possibly identical) paths which are shared.*

Two feature structures A and A' can differ with respect their information content, i.e. one can be more informative than the other. If all path-value pairs contained in A are also contained in A' then A is said to **subsume** A' ($A \sqsubseteq A'$) and A' is said to **extend** A:

Definition 2.3 (Subsumption (FS)) [5] *Let $A = (Q, q_0, \delta, \pi)$ and $A' = (Q', q_0', \delta', \pi')$ be two feature structures. Then A* subsumes *A' ($A \sqsubseteq A'$) just in case there exists a mapping h: $Q \rightarrow Q'$ which meets the following conditions:*

- $h(q_0) = q_0'$
- *if* $\delta(q,a)$ *is defined, then* $\delta'(h(q),a) = h(\delta(q,a))$
- *if* $\pi(q)$ *is defined, then* $\pi'(h(q)) = \pi(q)$

For example, the feature structure in Figure 2.2 subsumes the feature structure in Figure 2.3 (see Figure 2.9). It is often assumed that there is a most general class in the subsumption order of feature structures, which subsumes every other feature structure. This most general class is called **top** and often written as $\top$.

$$\begin{bmatrix} \text{ORTH:} & <cookie> \\ \text{PHON:} & /''kUki:/ \end{bmatrix} \quad \sqsubseteq \quad \begin{bmatrix} \text{ORTH:} & <cookie> \\ \text{PHON:} & /''kUki:/ \\ \text{AGR:} & \begin{bmatrix} \text{PER:} & 3 \\ \text{NUM:} & sg \end{bmatrix} \end{bmatrix}$$

Figure 2.9: Subsumption (FS)

[5] (Keller, 1993, p. 24).

Given this definition of subsumption, one can define what it means if two feature structures are **incomparable**, **equivalent** or **compatible**.[6]

Definition 2.4 (Equivalence) *Two feature structures A and A' are* equivalent *if and only if* $A \sqsubseteq A'$ *and* $A' \sqsubseteq A$

Definition 2.5 (Incomparability) *Two feature structures A and A' are* incomparable *if and only if neither* $A \sqsubseteq A'$ *nor* $A' \sqsubseteq A$

Definition 2.6 (Compatibility) *Two feature structures A and A' are* compatible *if and only if there exists a third feature structure B such that* $A \sqsubseteq B$ *and* $A' \sqsubseteq B$

B is called a *unifier* of A and A'. If there is no other unifier B' such that $B' \sqsubseteq B$ then B is called a *minimal unifier* of A and A'. Sometimes, especially in connection with inheritance hierarchies, the term **lower bound** is used for *unifier* and **greatest lower bound** for *minimal unifier*. The greatest lower bound of a set of feature structures S can also be written as $\sqcup S$. The lowest upper bound of S can be written as $\sqcap S$. **Unification** is defined as follows:[7]

Definition 2.7 (Unification) *The* unification *of two feature structures A and A' (*$A \sqcup A'$*) is taken to be the greatest lower bound of A and A' in the collection of feature structures ordered by subsumption.*

For example, in Figure 2.10, the feature structure on the left can be unified with the feature structure in the middle to the feature structure on the right.

A unification between two feature structures A and A' is said to **fail** if no lower bound for the two feature structures exists. This happens if the two feature structures contain *conflicting* attribute-value pairs. To indicate unification failure the symbol $\perp$ (**bottom**) is often used.

[6] Cf. (Keller, 1993, p. 25).

[7] Cf. Carpenter (1993). There is some disagreement between different research traditions as to whether type hierarchies should be viewed with the root (i.e. the most general) node at the top or at the bottom. In Carpenter's type hierarchies the most general level occurs at the bottom whereas in the standard HPSG literature it occurs at the top. This means that Carpenter defines unification in terms of the least upper bound between two feature structures whereas standard HPSG usually defines it in terms of the greatest lower bound. Similarly there is disagreement about the orientation of symbols for unification and subsumption. Carpenter (1992), Sag and Wasow (1999) and Shieber (1986) use $\sqsubseteq$ for subsumption and $\sqcup$ for unification while Pollard and Sag (1987) and Copestake (2002) use $\sqsupseteq$ and $\sqcap$ for subsumption and unification, respectively. I follow Sag and Wasow (1999) here, i.e. I will view lexical inheritance hierarchies with the most general class at the top and I will use the symbols $\sqsubseteq$ for subsumption and $\sqcup$ for unification. I believe that $\sqsubseteq$ and $\sqcup$ are more mnemonic than $\sqsupseteq$ and $\sqcap$ because the former resemble the symbols for subset ($\subseteq$) and union ($\cup$) thus capturing the intuition that subsumption can be viewed (at least for untyped feature structures) in terms of one feature structure's attribute-value set being a subset of another feature structure's attribute-value set and unification can be viewed in terms of the union between two attribute-value sets.

$$\begin{bmatrix} \text{ORTH}: & <cookie> \\ \text{PHON}: & /''kUki:/ \end{bmatrix} \sqcup \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{PER}: & 3 \\ \text{NUM}: & sg \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{ORTH}: & <cookie> \\ \text{PHON}: & /''kUki:/ \\ \text{AGR}: & \begin{bmatrix} \text{PER}: & 3 \\ \text{NUM}: & sg \end{bmatrix} \end{bmatrix}$$

Figure 2.10: Unification (FS)

For example the two feature structures in Figure 2.11 cannot be unified because the values $<cookie>$ and $<chocolate>$ do not have a unifier (in an untyped system no two different atomic values have a unifier). The same is true for */"kUki:/* and */"tSQkl@t/*.

$$\begin{bmatrix} \text{ORTH}: & <cookie> \\ \text{PHON}: & /''kUki:/ \end{bmatrix} \sqcup \begin{bmatrix} \text{ORTH}: & <chocolate> \\ \text{PHON}: & /''tSQkl@t/ \\ \text{AGR}: & \begin{bmatrix} \text{PER}: & 3 \\ \text{NUM}: & sg \end{bmatrix} \end{bmatrix} = \quad \bot$$

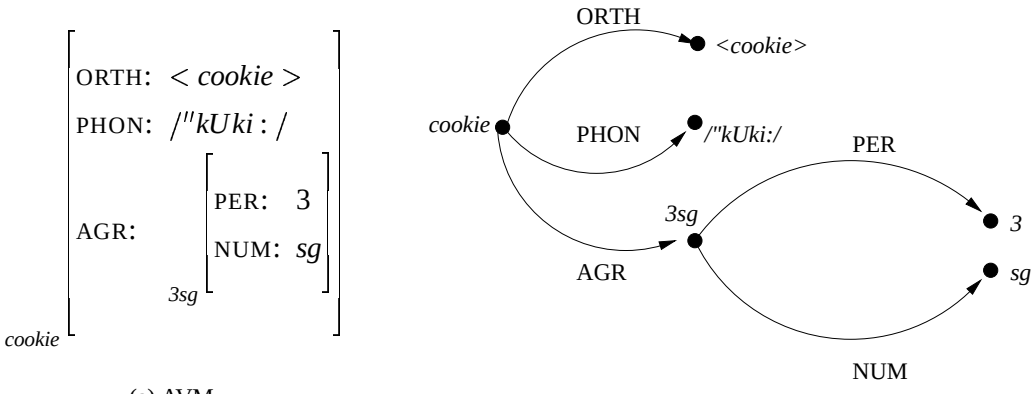
Figure 2.11: Unification failure (FS)

Typed feature structures are a special kind of feature structure. All nodes in the DAG representation of typed feature structures are interpreted to be **types**. Types permit an explicit organisation of feature structures into natural classes.

In AVM representations, the type of a feature structure occurs usually next to the left bracket of the feature structure, either on the top right or on the bottom left. I will use the bottom left version. In lexical inheritance hierarchies, I will usually write types above the associated feature structures. Types are also usually displayed in italics.

The typed version of the feature structure in Figure 2.3 is shown in Figure 2.12(a). There

are six types in this feature structure: *cookie* (the type of the whole feature structure), *<cookie>* (the value type of the attribute ORTH), */"kUki:/* (the value type of the attribute PHON), *3sg* (the type of the embedded feature structure, i.e. the value type of the attribute AGR), *3* (the value type of the attribute PER), and *sg* (the value type of the attribute NUM). In DAG representation this is equivalent to labelling all nodes with types (Figure 2.12(b)).



(a) AVM (b) DAG

Figure 2.12: Typed feature structure

Typed feature structures presuppose the existence of a **type hierarchy** in which all value types are ordered according to specificity. For example, the type hierarchy in Figure 2.13 might underlie the typed feature structures in Figure 2.12 (and others).

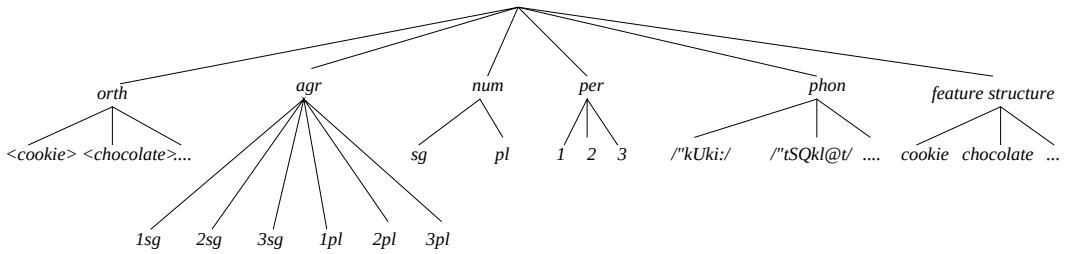


Figure 2.13: Type hierarchy underlying a set of typed features structures

The formal definition of typed feature structures is then given as follows (cf. Keller 1993, p. 36, Carpenter 1992, p. 36):

Definition 2.8 (Typed Feature Structure) *A* typed feature structure *is a directed acyclic graph defined on a set of attributes Attr, and a* type hierarchy $<\mathit{Type},\sqsubseteq>$

$A =< Q, q_0, \delta, \theta >$ *where* :

- Q : *a finite set of nodes rooted at* q_0
- $q_0 \in Q$: *the root node*
- $\delta : Q \times \mathit{Attr} \rightarrow Q$: *the partial* feature value *function*
- $\theta : Q \rightarrow \mathit{Type}$*: a total* node typing *function*

subject to the three constraints stated in the definition of feature structure *above.*

Thus in the definition of typed feature structures the typing function θ replaces the atomic value function π in the definition of untyped feature structures (Definition 2.1). Note also that θ is a total function, i.e. it assigns types to *all* nodes, whereas π is partial and only assigns values to terminal nodes.

It is often convenient to specify which attributes are necessary for a given type. Therefore, type feature logic allows the specification of **appropriateness conditions** for types. For example, one could state that the attributes PER and NUM are appropriate for type *agr*. This would mean that all feature structures of type *agr* have to contain these two attributes. Appropriateness conditions can be written into the type hierarchy. A type hierarchy that is annotated in this way is called a *signature*. Appropriateness can be formally defined as follows (Keller, 1993, p. 37):[8]

Definition 2.9 (Appropriateness) *An* appropriateness specification *is a partial function* Approp$:(\mathit{Attr} \times \mathit{Type}) \rightarrow \mathit{Type}$ *from attribute-type pairs to types which meets (at least) the following conditions:*

- Minimal Introduction*: for every attribute* $a \in \mathit{Attr}$*, there is a most general type* $\sigma \in \mathit{Type}$ *such that* $\mathrm{Approp}(a,\sigma)$ *is defined*
- Upward Closure/Right Monotonicity*: if* $\mathrm{Approp}(a,\sigma)$ *is defined and* $\sigma \sqsubseteq \tau$ *then* $\mathrm{Approp}(a,\tau)$ *is defined and* $\mathrm{Approp}(a,\sigma) \sqsubseteq \mathrm{Approp}(a,\tau)$

[8] Instead of *Minimal Introduction*, Copestake (2002) and Carpenter (1992) use the term *Maximal Introduction*.

A typed feature structure is said to be **well-typed** if each of its attributes is appropriate and takes an appropriate value. A typed feature structure is said to be **totally well-typed** if the converse is also true and every attribute that is appropriate for the feature structure is also present (Keller, 1993, p. 37).

Definition 2.10 (Well-typing) *A typed feature structure* $A = (Q, q_0, \delta, \theta)$ *is* well-typed *if whenever* $\delta(q,a)$ *is defined, then* $\text{Approp}(a, \theta(q))$ *is defined and* $\text{Approp}(a, \theta(q)) \sqsubseteq \theta(\delta(q,a))$

Definition 2.11 (Total Well-typing) *A typed feature structure* $A = (Q, q_0, \delta, \theta)$ *is* totally well-typed *if*

- *it is well-typed and*
- *whenever* $\text{Approp}(a, \theta(q))$ *is defined, then* $\delta(q,a)$ *is also defined*

For typed feature structures, the definition of subsumption deviates slightly from Definition 2.1. Values in typed feature structures are typed so for $A \sqsubseteq A'$ to hold it is required that the value type of every attribute in A subsumes the value type of the equivalent attribute in A'. An (atomic) type t subsumes an (atomic) type t' if t occurs higher in the type hierarchy and there is a path between t and t'. The formal definition of subsumption for typed feature structures is thus:[9]

Definition 2.12 (Subsumption (TFSs)) *A typed feature structure* $A = (Q, q_0, \delta, \theta)$ *subsumes a typed feature structure* $A' = (Q', q_0', \delta', \theta')$ *if and only if there is a total function* $h : Q \to Q'$*, called a* morphism *such that:*

- $h(q_0) = q_0$
- $\theta(q) \sqsubseteq \theta'(h(q))$ *for every* $q \in Q$
- $h(\delta(q,a)) = \delta'(h(q),a)$ *for every* $q \in Q$ *and attribute* $a \in Attr$ *such that* $\delta(q,a)$ *is defined*

For example, in Figure 2.14, the feature structure on the left subsumes the feature structure on the right, given the type hierarchy in Figure 2.13.

The definition of unification for typed feature structures is essentially the same as the one given above for feature structures (see Definition 2.7). The only difference is that, whereas atomic values in untyped feature structures cannot be unified, two atomic types τ and τ' can

[9] Cf. (Carpenter, 1992, p. 41).

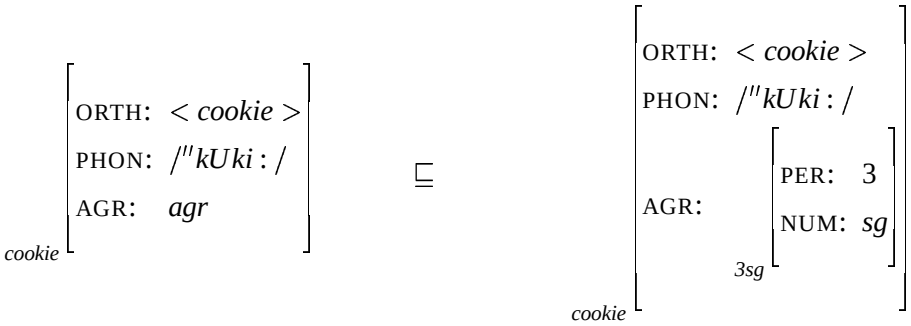


Figure 2.14: Subsumption (TFS)

be unified to σ if $\tau \sqsubseteq \sigma$ and $\tau' \sqsubseteq \sigma$ in the type hierarchy and if there does not exist a type σ' such that $\tau \sqsubseteq \sigma'$ and $\tau' \sqsubseteq \sigma'$ and $\sigma' \sqsubseteq \sigma$. Thus, given the type hierarchy in Figure 2.15, the feature structure in Figure 2.16 (left) can be unified with the feature structure in Figure 2.16 (middle), to the feature structure in Figure 2.16 (right), because the types *w* and *v* have a common subtype, namely *b*.

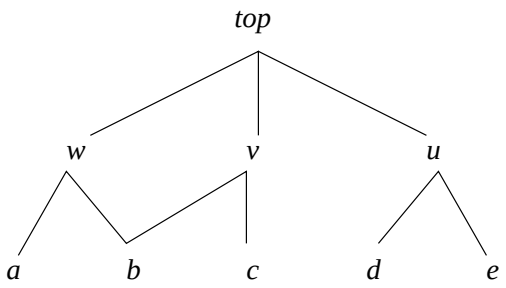


Figure 2.15: Type Hierarchy

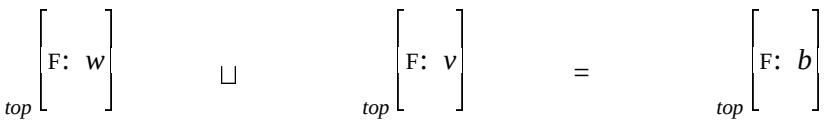


Figure 2.16: Unification (TFS)

On the other hand, the feature structure in Figure 2.17 (left) cannot be unified with the feature structure 2.17 (middle) because *w* and *u* do not have a common subtype in the hierarchy in Figure 2.15, i.e. *w* and *u* are incompatible.

Reentrancies pose no further problem for unification. Thus, given the hierarchy in Figure

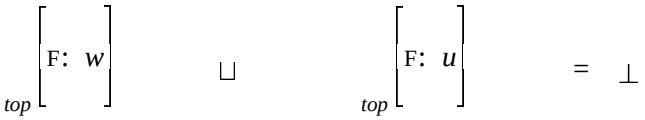


Figure 2.17: Unification failure (TFS)

2.15, the feature structure in Figure 2.18 (left) can be unified with the feature structure in Figure 2.18 (middle) to the feature structure in Figure 2.18 (right). Note, that the result is different if mere type identity rather than reentrancy is involved (Figure 2.19). Figure 2.20 gives an example of unification failure.

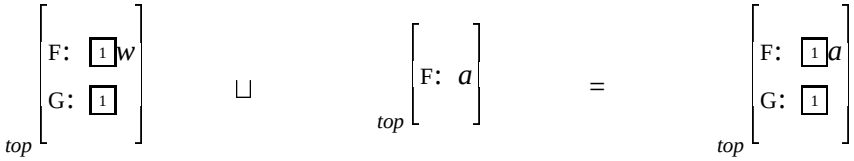


Figure 2.18: Unification (TFS), reentrancy

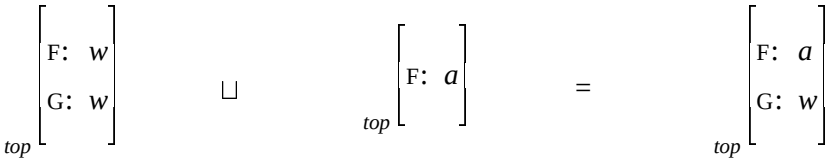


Figure 2.19: Unification (TFS), no reentrancy

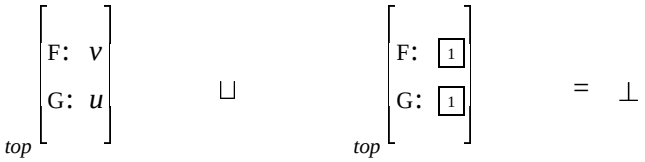


Figure 2.20: Unification failure (TFS), reentrancy

The definitions for *equivalence*, *incomparability* and *compatibility* for typed feature structures are the same as for untyped feature structures.

An **inheritance hierarchy** can be formally defined as:

Definition 2.13 (Inheritance Hierarchy) *An* inheritance hierarchy *is a* finite partial order *over feature structures* $\langle FS, \sqsubseteq \rangle$ *or over typed feature structures* $\langle TFS, \sqsubseteq \rangle$. *In the latter case the inheritance hierarchy is usually called a* type hierarchy.

Figure 2.21 shows a set of feature structures which is partially ordered by subsumption. The following terms are defined relative to a given partial order. For a node *n* the nodes that subsume *n* are called **ancestors** of *n*. The set of all of *n*'s ancestors is also called the **upward closure** of *n*. The nodes that are subsumed by a node *n* are called its **descendants**. The set of all nodes subsumed by *n* is called *n*'s **downward closure**. If there is no other node between a node *n* and its ancestor *m* then *m* is called the **immediate ancestor** (or **parent**) of *n* and *n* is called the **immediate descendant** (or **child**) of *m*. I will use the term **terminal descendants** of node *n* for those descendants of *n* that are terminal nodes in the partial order, i.e. nodes that do not subsume any other nodes. For example, the terminal descendants of *Node 2* are *Node 6* and *Node 9*.

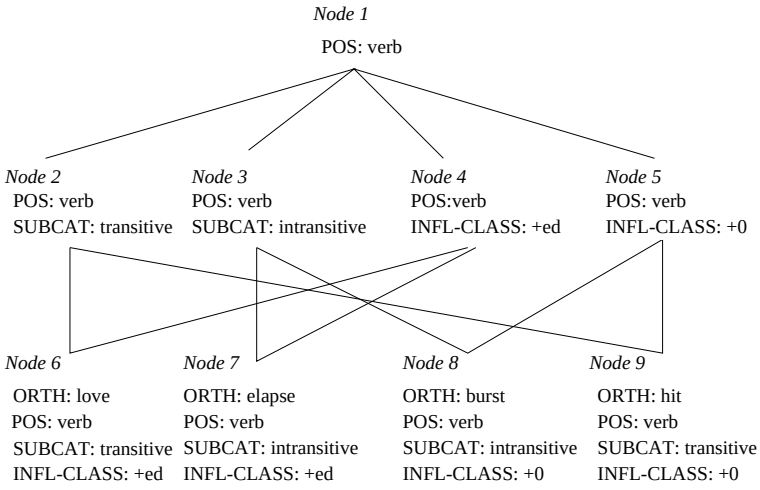


Figure 2.21: Partial order over feature structures

For reasons that will become clearer in Chapter 4, I will called the set of attribute-value pairs of a node *n* the **intension** of node *n*. For instance, the intension of *Node 3* is:

$\{$ POS:*verb*, SUBCAT:*intransitive* $\}$

Inheritance hierarchies usually omit attribute-value pairs from the intension of a node *n* if they are also part of the intension of an immediate ancestor of *n*. In this case it is said that *n* **inherits** the attribute-value pair from its ancestor. The inheritance hierarchy which corresponds to the partial order in Figure 2.21 is shown in Figure 2.22.

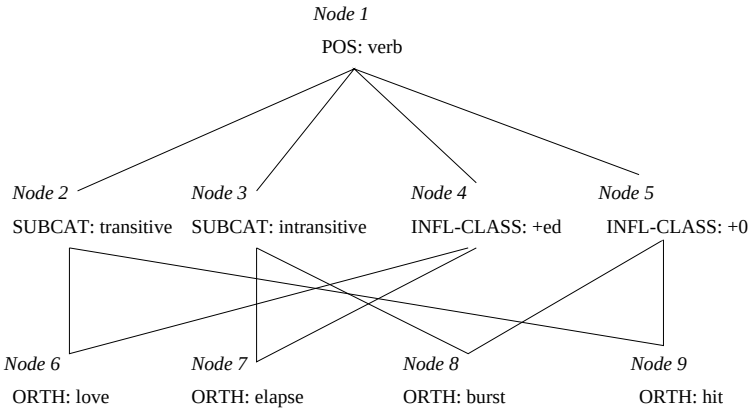


Figure 2.22: Inheritance hierarchy

Note, that the omission of attribute-value pairs which can be inherited is a matter of representation; it does not *change* the original intension of a given node, which can be retrieved by traversing an inheritance hierarchy top-down adding those attribute-value pairs to the attribute-value pair set of a node which can be inherited from an ancestor. This method is called **compiling out** a hierarchy.

It is useful to distinguish between the attribute-value pair set of a node as it occurs in a hierarchy and the attribute-value pair set of a node as it occurs in the compiled out hierarchy. I will therefore refer to the former as **overt intension** and to the latter as **underlying intension** (or simply as **intension**). For example, the overt intension of *Node 3* in Figure 2.22 is $\{\text{SUBCAT}:\textit{intransitive}\}$ while the underlying intension is $\{\text{SUBCAT}:\textit{intransitive}, \text{POS}:\textit{verb}\}$.

The terminal nodes in a lexical inheritance hierarchy usually correspond to the entries of the lexicon that is represented by the hierarchy. Each entry (partially) describes a linguistic entity. For example, the terminal nodes of the hierarchy in Figure 2.22 describe the four lexemes *love*, *elapse*, *burst*, and *hit*. I will call the linguistic entities described by a lexicon (a lexical inheritance hierarchy) the **domain** of the lexicon (the hierarchy). Not only terminal nodes provide (partial) descriptions of linguistic entities, non-terminal nodes do to. For example, *Node 2* partially describes the two lexems *love* and *hit*. The set of linguistic entities partially described by a node *n* is called *n*'s **extension**. Note that the term *extension* is defined relative to

the domain of a hierarchy, i.e. while there are other lexemes that are partially described by the feature structure [SUBCAT:*transitive*], the extension of *Node 2* in the above hierarchy contains only the two lexemes *love* and *hit*.

Sometimes there are further constraints on inheritance hierarchies. For type hierarchies it is often required that they are not only a finite partial order but a *finite bounded complete partial order* (*BCPO*) (cf. Carpenter 1992, p. 13). A partial order is said to be **bounded complete** if for every compatible set of (typed) feature structures there is a unique greatest lower bound. A unique greatest lower bound is referred to as a **meet** in lattice theory and a hierarchy which ensures unique greatest lower bounds is called a *meet semi-lattice*. A least upper bound is called a **join**.

Commonly, lexical inheritance hierarchies are also required to be **rooted**, i.e. there has to be a unique root node from which all other nodes inherit, and *acyclic*, i.e. a node cannot inherit from itself and if *A* inherits from *B* then *B* cannot inherit from *A* or from any of *A*'s ancestors.

2.2 Types of Inheritance Hierarchies

There are two main dimensions along which inheritance hierarchies can be classified. *Monotonicity* describes whether an inheritance is monotonic or non-monotonic. *Multiplicity*[10] describes whether single or multiple inheritance is used.

Single inheritance hierarchies are tree-structured graphs, i.e. no node has more than one parent, while nodes in **multiple inheritance** hierarchies can have several parents (see Figure 2.22). Thus, multiple inheritance hierarchies are no longer trees but fall in the more general class of directed acyclic graphs. Multiple inheritance hierarchies are usually more adequate for modelling linguistic properties than single inheritance hierarchies, since linguistic generalisations are often independent of each other. For example, whether or not a verb in English exhibits regular inflection does not depend on its subcategorisation class. Neither does its subcategorisation class determine its inflection. Independencies like these are difficult to express in a tree-like structure where they usually require the duplication of information. One either has to duplicate inflectional information (Figure 2.23) or one has to duplicate subcategorisation information (Figure 2.24). This leads to unintuitive hierarchies, e.g. the hierarchy in Figure 2.23 contains nodes which are identical with respect to their feature structures (e.g. *reg. trans. verb* and *reg. intrans. verb*) but are nonetheless *incomparable* in the hierarchy.

[10] The term *multiplicity* is used in work on object-oriented programming to refer to the dimension that deals with single *vs.* multiple inheritance, cf. Wegner (1990), Ewing (1997).

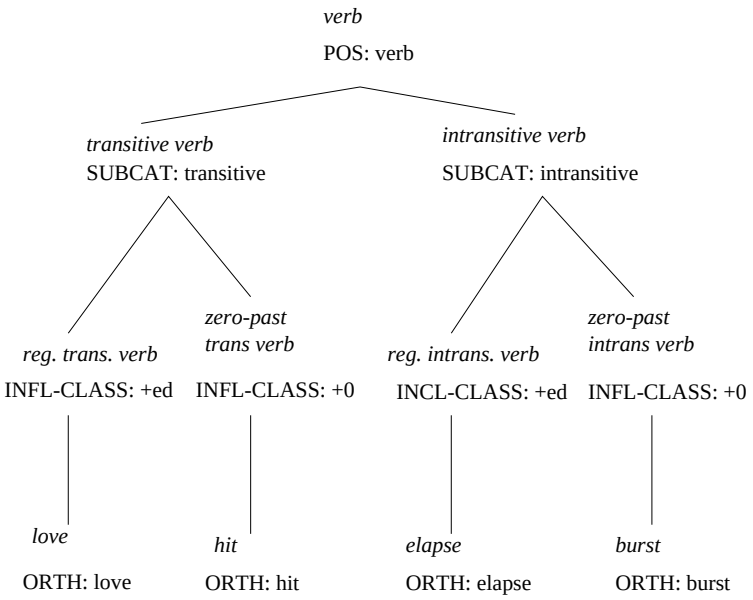


Figure 2.23: Monotonic single inheritance: inflectional information is duplicated

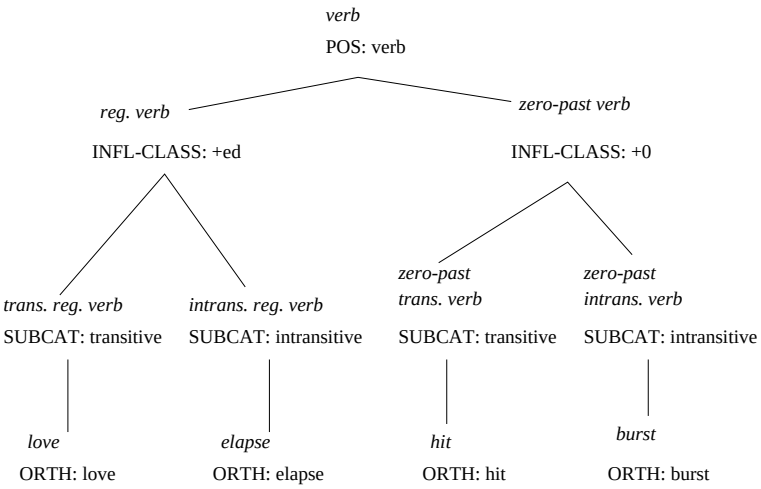


Figure 2.24: Monotonic single inheritance: subcategorisation information is duplicated

Multiple inheritance, however, permits an efficient representation of these facts: a given verb can inherit subcategorisation properties along one path and inflectional properties along another. Independent properties such as subcategorisation and inflection are called **dimensions** and are sometimes shown as rectangular boxes in an inheritance hierarchy. Figure 2.25 shows a multiple inheritance hierarchy representing the same information as the two single inheritance

hierarchies in Figures 2.23 and 2.24.

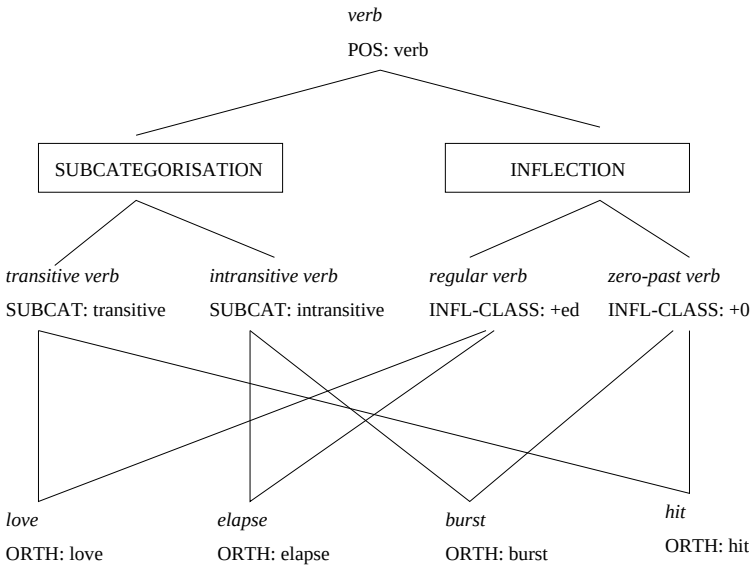


Figure 2.25: Monotonic multiple inheritance

The different subtypes within a dimension are said to **partition** the dimension, since they stand in an exclusive disjunction relation to each other and exhaust the dimension. In the hierarchy in Figure 2.25, the types *transitive verb* and *intransitive verb* partition the dimension SUBCATEGORISATION.

Monotonic inheritance means that a node inherits *all* properties (i.e. all attribute-value pairs) specified for its supernodes. The property set of a node is included in the property set of each of its descendants. **Non-monotonic** or **default inheritance** means that not all properties have to be inherited from a supernode. For example, if an attribute-value pair x that is specified for a supernode is not compatible with an attribute-value pair x' specified for the current node, x' will take precedence and block the inheritance of x. Default (or non-default) information can be explicitly marked in feature structures. Often, however, non-monotonic inheritance hierarchies do not explicitly mark defaults or non-defaults. In this case default information has to be identified in some other way. Usually this is done by adopting the principle that the less specific piece of information is the default, i.e. specific information takes precedence over general information and a property associated with a parent node will be overridden by a conflicting property associated with its descendant (*Specificity Principle*). This permits one to encode exceptional properties of a child node relative to the generalisations encoded on its

parents. A mechanism that deals with exceptionality is useful because the lexicon generally exhibits some degree of idiosyncrasy and it is difficult to find lexical generalisations that hold without exception.[11] For example, most English verbs form the past tense by adding the suffix *−ed* to their stem and it makes sense to state this principle as a general rule. However, there are several exceptions to this rule. Therefore the *−ed* inflection is best treated as a default (see e.g. Daelemans *et al.* 1992). Hence, in Figure 2.26 regular past tense inflection is specified as a default at the type *verb*. This default is overridden by the type *zero-past verb*. The implicit overriding of information associated with an ancestor node is sometimes also referred to as *blocking* (see Briscoe *et al.* 1995 for a discussion of blocking).

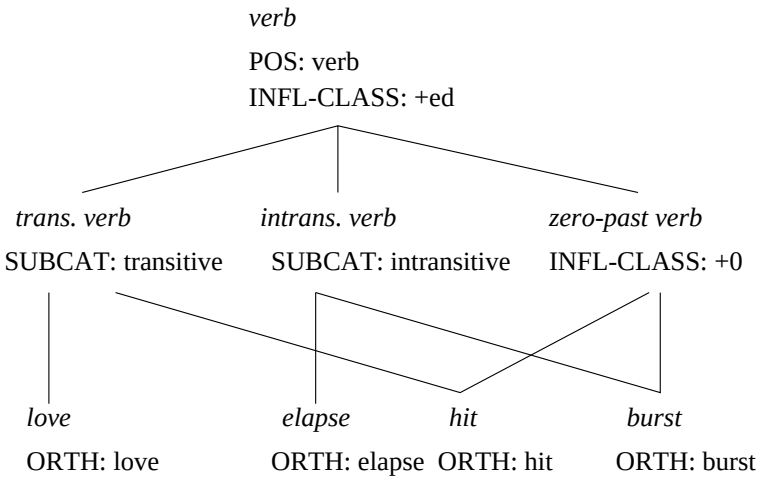


Figure 2.26: Non-monotonic inheritance

Non-monotonic inheritance can often be traded off against multiple inheritance in that using a default sometimes avoids the need for multiple inheritance and *vice versa*. However, many inheritance hierarchies are both non-monotonic and use multiple inheritance as in Figure 2.26. In general, defaults and multiple inheritance fulfil different functions: multiple inheritance is used to express independent properties while non-monotonic inheritance is used to encode exceptions to generalisations.

Non-monotonic inheritance can potentially lead to conflicts (i.e. unification failure). If the conflict occurs between nodes that are in a subsumption relation, the conflict is usually resolved by letting the more specific node take precedence, as mentioned above. However, combining non-monotonic inheritance with multiple inheritance potentially leads to another

[11] Bouma (1992) gives a brief overview of different areas in linguistics where defaults are useful.

source of conflict. This conflict appears when a node inherits incompatible information from two of its parent nodes. This is known as the *Nixon diamond* (Figure 2.27). Conflicts like these can only be resolved by assigning priority to one parent node (or to one of the conflicting properties) or by allowing disjunctive or underspecified inferences (e.g. “Nixon was either a pacifist or he was not a pacifist” or “It is not known whether Nixon was a pacifist”). An inheritance hierarchy which avoids this kind of conflict altogether by demanding that no property can be inherited from more than one parent node is called **orthogonal**.

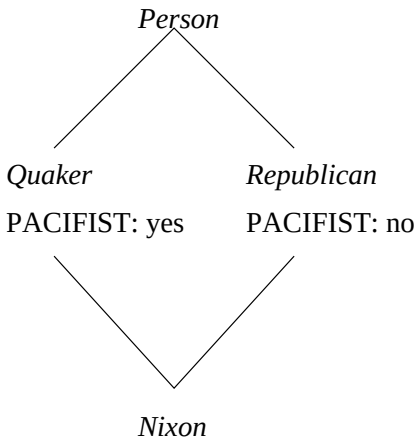


Figure 2.27: Nixon Diamond

In a monotonic inheritance hierarchy the unification between two feature structures is fairly straightforward. It will return the most general feature structure which is subsumed by the two unified feature structures. If no such feature structure exists unification fails.

Unification as defined in definition 2.7 above is inherently monotonic. To be able to unify feature structures non-monotonically, one has to extend the definition. Several definitions of non-monotonic unification have been proposed. Thus, Bouma (1992) defines an operation called *default unification*, Carpenter (1993) describes two versions of a default unification, one which returns a single feature structure (*skeptical default unification*) and one which returns a set of feature structures (*credulous default unification*), Young and Rounds (1993) and Lascarides *et al.* (1996) propose default unifications which require the default values to be marked explicitly, and Lascarides and Copestake (1999) present an extension of their original default unification called *YADU* (*Yet Another Default Unification*).

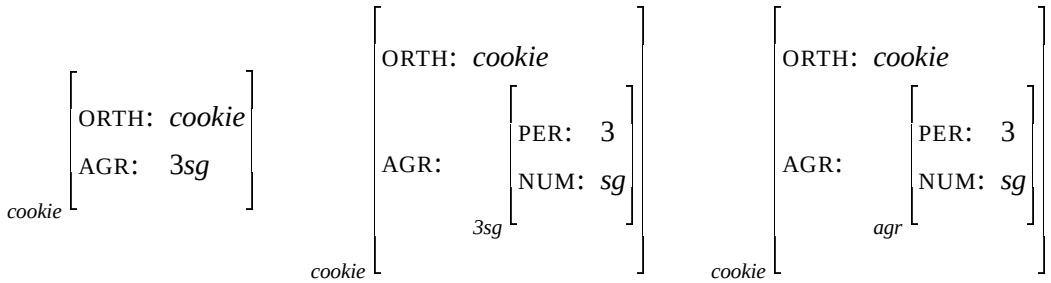


Figure 2.28: Three ways of expressing “3rd person singular”

2.3 A Note on Types vs. Attribute-Value Pairs

In type hierarchies, the same information can often be expressed in several different ways, depending on whether one uses atomic types or complex types with appropriate attribute-value pairs. For example, the information that *cookie* is a third person singular form can be expressed by defining an atomic type *3sg* (Figure 2.28, left), by defining a complex type *3sg* whose appropriate attribute-value pairs are PER:*3* and NUM:*sg* (Figure 2.28, middle), or by not introducing a special type for *3sg* and relying only on attribute-value pairs (Figure 2.28, right).

The suitability of each of these variants depends to some extent on how the rest of the grammar has been implemented (see Copestake 2002, p. 134ff). Thus, complex types with attribute-value pairs (variants 2 and 3, above) are useful if number and person information has to be accessed independently. For example, number agreement between verbs and their subjects can be modelled by specifying a reentrancy between the number of a verb and the number of its subject. But this would not work without a NUM attribute. Similarly the use of a type (complex or simple) *3sg* might be beneficial if one does not normally have to distinguish between all combinations of attribute-value pairs. For example, in English it is important to distinguish between forms that are 3rd person singular and forms that are not 3rd person singular but for most verbs it is not necessary to distinguish between, say, 1st and 2nd person singular forms. Introducing a *3sg* type allows one to define a *non3sg* type which can function as a “catch-all” for other forms in those cases where the other forms need not be distinguished (see the type hierarchy for *agr* in Figure 2.29).

Sometimes types are also introduced to function as antecedents for lexical rules, i.e. one might define a (simplistic) lexical rule which takes a *non3sg* verb and transforms it into a *3sg* verb by adding an “s”. However, Meurers (2001) argues that this is not a good reason for

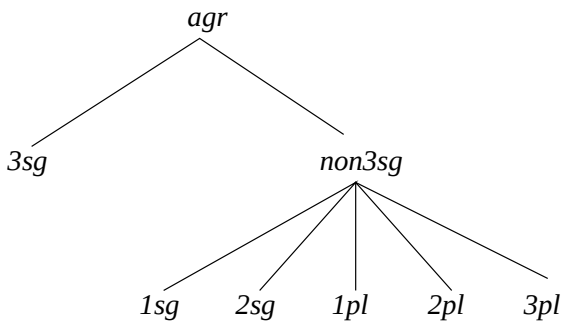


Figure 2.29: Type System, 3sg vs. non3sg

the introduction of types because lexical rules can just as well take bundles of attribute-value pairs (or complex types) as antecedents and defining a new type for every generalisation in a grammar complicates the type hierarchy and often leads to the duplication of information. Hence, while there are some guidelines for the usage of types *vs.* attribute-value pairs some design decisions remain fairly subjective.

2.4 Hierarchical Lexicons in Linguistic Theories

According to Daelemans *et al.* (1992) the use of inheritance hierarchies in linguistics and natural language processing can be traced back to three different traditions: semantic nets in artificial intelligence (Quillian 1968, Touretzky 1986), data abstraction in programming languages, which has led to object-orientation and the use of type hierarchies (Ait-Kaci 1984), and the notion of "markedness", which was introduced in the Prague School and re-used in generative phonology (Chomsky and Halle, 1968).

The use of inheritance hierarchies has grown with the spread of lexicalist linguistic theories, such as *Head-Driven Phrase Structure Grammar* (*HPSG*) (Pollard and Sag, 1987, 1994), *Categorial Grammar* (*CG*) (Wood, 1993), *Lexicalised Tree Adjoining Grammar* (*LTAG*) (Joshi and Shabes, 1991), or *Word Grammar* (*WG*) (Hudson, 1984), in which the syntactic rule system is radically reduced in favour of a complex lexicon. The lexicon in these theories commonly contains information about several linguistic aspects, for example orthography, phonology, syntax and semantics. Because the lexicons of such grammar theories can grow very big they need to be represented efficiently. As the lexicons used by lexicalist grammar theories contain many regularities as well as sub-regularities and exceptions (see the examples in Section 1.1), representing lexical entries as simple lists results in a lot of redundancy and in a significant loss of

generalisation. Inheritance hierarchies are therefore the preferred choice of representation for many lexicalist approaches. As mentioned in the introduction lexical inheritance hierarchies also have the advantage of being able to generalise beyond the lexicon.

HPSG has favoured the use of lexical inheritance hierarchies right from the start. Even Pollard and Sag (1987) suggest representing lexical information in an inheritance hierarchy. And the most comprehensive account of a hierarchical lexicon to date has been given by Flickinger (1987) within the HPSG framework.

Inheritance hierarchies in HPSG are usually multiple inheritance type hierarchies, with some authors preferring monotonic and some non-monotonic hierarchies. Other work within the HPSG paradigm that makes crucial use of inheritance networks includes Flickinger and Nerbonne (1992), Krieger and Nerbonne (1993), Sag and Wasow (1999), Riehemann (1998), Malouf (1998), Lüngen (2002), Koenig (1999) and Koenig and Jurafsky (1994).

In Categorial Grammar, inheritance hierarchies have been used by Villavicencio (2002) and Baldridge (2002). For Lexicalised Tree Adjoining Grammar, Vijay-Shanker and Schabes (1992) and Evans *et al.* (1994, 1995) proposed the use of inheritance hierarchies. And Word Grammar relies crucially on default inheritance hierarchies (Hudson 1984, Fraser and Hudson 1992).

The interest in lexical inheritance hierarchies has lead to the development of several tools which facilitate the implementation of hierarchical lexicons. The first formalism which supported lexical inheritance hierarchies of some sort was PATR-II (Shieber, 1986). PATR-II permitted the use of *templates*, which can be employed to implement *prototypes* and are thus similar to superclasses in an inheritance hierarchy. Nowadays the grammar development environments used most for HPSG-style grammars and lexicons are probably *ALE* (*Attribute-Logic Engine*),[12] which was developed by Bob Carpenter and Gerald Penn and the *LKB* (*Linguistic Knowledge Building*, Copestake 2002).[13] The LKB allows non-monotonicity while ALE is monotonic.[14]

Another popular representation language is *DATR*,[15] which was developed by Gerald Gazdar and Roger Evans (1989a, 1989b) with the aim of providing a minimalist formalism that is still expressive enough to represent lexical information at a variety of levels of language description (Evans and Gazdar, 1996) and to support default inheritance. Whereas ALE and

[12]The ALE homepage is: `http://www.cs.toronto.edu/~gpenn/ale.html` (7.02.03)

[13]The LKB homepage is: `http://www-csli.stanford.edu/~aac/lkb.html` (7.02.03)

[14]There are, however, extensions of ALE which allow default unification (e.g. Grover *et al.* 1994).

[15]The name "DATR" is a not an acronym but a reference to the PATR formalism (see e.g. Evans and Gazdar, 1996, fn. 3).

the LKB have been geared towards unification-based grammars (in particular HPSG), DATR was not developed for any particular grammar formalism. It is supposed to be an independent lexical representation language. Thus, while HPSG-style feature structures can be (fairly) straightforwardly represented in ALE and the LKB, encoding them in DATR is not trivial (see, for example, Krieger and Nerbonne 1993, Daelemans and van der Linden 1991). DATR has been widely used, often to represent morphological or phonological information (e.g. Reinhard and Gibbon 1991, Gibbon 1992), but it has also been employed to represent syntactic and semantic information (e.g. Kilgarriff 1993). It has also been used in fairly big NLP systems, e.g. Cahill (1993), Cahill (1994) and Andry *et al.* (1992). A short overview of DATR will be given in the next section.

Other grammar/lexicon development environments include PAGE[16] and ELU (Russell *et al.* 1992) for HPSG-style grammars and LexGram for CG (König 1999).

A different route has been taken by Daelemans and Smedt (1994), who argue that instead of developing special purpose formalisms for the encoding of lexical inheritance hierarchies one should re-use object oriented (programming) languages, which could possibly be extended into a hybrid between an object-oriented and a symbolic system. They claim that the creation of special purpose formalisms for the representation of lexical inheritance hierarchies is unnecessary and that the use of object-oriented languages potentially facilitates a closer integration between linguistic and other cognitive information.

2.5 Automatic Construction of Lexical Inheritance Hierarchies

So far there has been little work on machine learning of lexical inheritance hierarchies. The first and to date also the most comprehensive account has been given by Barg (1996a, see also Barg 1994) who investigated in her PhD thesis how DATR hierarchies can be acquired automatically.

To understand Barg's algorithm it is necessary to give a brief overview of the DATR formalism.[17] DATR theories contain a set of nodes with associated path-value pairs. Path-value pairs usually take the form of so-called *definitional sentences*.[18] Figure 2.30 shows a trivial

[16] `http://www.dfki.de/lt/systems/page/` (18.7.03).

[17] The overview does not describe all DATR constructs.

[18] The second kind of sentence in DATR is called *extensional sentence*. Extensional sentences consist of a *query* to a DATR theory and its value. A query is a node path pair, like `Hobble: <mor root>`. The associated value given the DATR theory in Figure 2.30 is `hobble`. The corresponding extensional sentence is `Hobble: <mor root> = hobble`. Note, that whereas definitional sentences contain two equals signs, extensional sentences contain only one.

DATR theory, without inheritance or defaults.

Inheritance in DATR is associated with paths rather than with nodes, i.e. it is paths that inherit not nodes. Inheritance is expressed by so-called *inheritance descriptors*, which occur at the right hand side of a sentence and specify where the left hand side of the sentence inherits its value from. Commonly, inheritance descriptors consist of a node-path pair. Thus, to express that the value of the path < `syn cat` > of the entry `Hobble` is inherited from the path < `syn cat` > of the node `Verb` one can write:

```
Hobble:
     <syn cat> == Verb: <syn cat>
```

If the path on the right hand side is identical to the path on the left hand side it can be left out, i.e. the statement above can be also be represented as:

```
Hobble:
     <syn cat> == Verb
```

Apart from inter-node inheritance DATR also allows intra-node inheritance, i.e. the value of a path can be inherited from another path of the same node:[19]

```
Hobble:
     <mor root>        == hobble
     <mor pres sg 1st> == Hobble: <mor root>
```

Again the node in the inheritance descriptor can be left out if it is identical to the current node.

All of the inheritance descriptors above express so-called *local inheritance*. There is also a second set of inheritance descriptors, which express *global inheritance*. In global inheritance descriptors, path references are quoted. Roughly speaking a quoted path is evaluated at the node where the evaluation of a DATR query was initiated. For example, given the theory below, the query `Hobble:<mor past>` is referred to the node `Verb`, where the sentence `<mor past> == ''< mor root>''+ed` evaluates to `hobble+ed` because the query was initiated at the node `Hobble`.

```
Verb:
     <mor past>   == ''<mor root>'' +ed
```

[19] Intra-node inheritance is related to reentrancies in feature structure grammars but not formally equivalent.

```
Hobble:
    <syn cat>              == verb
    <mor root>             == hobble
    <mor pres sg 1st>      == hobble
    <mor pres sg 2nd>      == hobble
    <mor pres sg 3rd>      == hobble+s
    <mor pres pl 1st>      == hobble
    <mor pres pl 2nd>      == hobble
    <mor pres pl 3rd>      == hobble
    <mor pres participle>  == hobble+ing
    <mor past sg 1st>      == hobble+ed
    <mor past sg 2nd>      == hobble+ed
    <mor past sg 3rd>      == hobble+ed
    <mor past pl 1st>      == hobble+ed
    <mor past pl 2nd>      == hobble+ed
    <mor past pl 3rd>      == hobble+ed
    <mor past participle>  == hobble+ed
    <mor pass participle>  == hobble+ed.

Come:
    <syn cat>              == verb
    <mor root>             == come
    <mor pres sg 1st>      == come
    <mor pres sg 2nd>      == come
    <mor pres sg 3rd>      == come+s
    <mor pres pl 1st>      == come
    <mor pres pl 2nd>      == come
    <mor pres pl 3rd>      == come
    <mor pres participle>  == come+ing
    <mor past sg 1st>      == came
    <mor past sg 2nd>      == came
    <mor past sg 3rd>      == came
    <mor past pl 1st>      == came
    <mor past pl 2nd>      == came
    <mor past pl 3rd>      == came
    <mor past participle>  == came
    <mor pass participle>  == came.
```

Figure 2.30: A DATR theory without inheritance and defaults

```
Hobble:
     <mor root>   == hobble
     <mor past>   == Verb
```

Defaults are expressed by means of prefix-matching, i.e. if a path does not have a value assigned to it, it is assigned the value of the longest matching prefix. Given the theory below, the query `Hobble:<mor pres sg 1st>` evaluates to `hobble` because the longest matching subpath is `<mor pres>` (which evaluates to `hobble`), whereas `Hobble:<mor pres sg 3rd>` evaluates to `hobble+s`.[20]

```
Hobble:
     <mor root>        == hobble
     <mor pres>        == <mor root>
     <mor pres sg 3rd> == <mor root> +s
```

Defaults also allow one to express that all paths inherit by default from a given node. Thus the node description below states that `Hobble` inherits all paths from the node `Verb`, with the exception of the path `<mor root>`, which is explicitly assigned a value.[21]

```
Hobble:
     < >               == Verb
     <mor root>        == hobble
```

Using inheritance and defaults, the rather long description of the verbs *hobble* and *come* in Figure 2.30 can be represented concisely (Figure 2.31).

Barg's algorithm transforms unstructured DATR theories, i.e. theories which do not contain any inheritance or default information, into structured DATR theories,[22] which do contain such information while still being consistent and complete with respect to the original theory.[23]

[20] This means that the treatment of defaults in DATR differs substantially from feature structure grammars like HPSG (cf. Daelemans and van der Linden 1991, p. 63). DATR's default system also means that, unlike standard feature logic, DATR does not make a *closed world assumption*, i.e. just because something is not explicitly expressed in a DATR theory does not mean that it cannot be evaluated. For example, the default mechanism that allows `Hobble:<mor pres sg 1st>` to evaluate also allows a nonsense query like `Hobble:<mor pres past syn cat>` to evaluate (to `hobble`).

[21] Note, only one node can be stated as a default parent in this way. Inheritance from all other parents is necessarily restricted to individual paths. This is sometimes called *mixin inheritance* (Daelemans and van der Linden 1991). Mixin inheritance ensures orthogonality.

[22] The set of theories learnable by Barg's algorithm is a proper subset of the set of valid DATR theories because the algorithm does not make use of all DATR features, like variables, compiler declarations and evaluable paths.

[23] *Consistency* and *completeness* will be discussed in detail in relation to the learning task presented here in

```
Verb:
    <syn cat>             == verb
    <mor pres>            == ‘‘<mor root>’’
    <mor pres sg 3rd>     == ‘‘<mor root>’’ +s
    <mor pres participle> == ‘‘<mor root> +ing
    <mor past>            == ‘‘<mor roor>’’ +ed
    <mor pass participle> == ‘‘<mor past>’’.

Hobble:
    < >                   == Verb
    <mor root>            == hobble.

Come:
    < >                   == Verb
    <mor root>            == come
    <mor past>            == came
    <mor past participle> == <mor root>.
```

Figure 2.31: A DATR theory with local and global inheritance and defaults

To achieve this, she proposes two sets of learning operators: transformation rules for introducing inheritance descriptors and default rules for introducing defaults. The application of each learning operator transforms a DATR theory into a new theory. Since the application of rules is not deterministic, i.e. it is usually the case that several rules can be applied to a given theory or that the same rule can be applied in several places, Barg also proposes a set of heuristics for selecting a sequence of rule applications.

Transformation rules are applied first. Each rule takes a sentence s_i as its input and produces the transformed sentence $s_{i'}$ as its output, subject to a set of constraints which specify under which conditions the rule can be applied. These constraints ensure that the new DATR theory (H_{n+1}) is complete and consistent with respect to the previous theory (H_n) and thus ultimately also with respect to the original theory (H_0).

For example, the rule that introduces a local, intra-node inheritance descriptor can be applied if the current DATR theory contains a sentence $s_i : (n, p_x, v)$, where n is the node at which the sentence occurs, p_x is the path (i.e. the left hand side of the sentence) and v is the value associated with the path (i.e. the right hand side of the sentence), under the condition that v is atomic and that the theory contains another sentence $s_j : (n, p_y, v)$, i.e. another sentence associated with node n whose value is identical to v. The output of the rule is a sentence $s_{i'} : (n, p_x, p_y)$. Thus because the paths <mor root> and <mor pres sg 1st> lead to the same value in $H1$ below, the theory can be transformed to $H2$. A similar rule introduces global intra-node inheritance.

H1

```
Hobble:
    <syn cat>         == verb
    <mor root>        == hobble
    <mor pres sg 1st> == hobble
```

H2

```
Hobble:
    <syn cat>         == verb
    <mor root>        == hobble
    <mor pres sg 1st> == <mor root>
```

Chapter 3. For Barg's learning problem, an inferred DATR theory is consistent with the original theory if queries which were contained in the original theory evaluate to the same value in the inferred theory. An inferred DATR theory is complete if *all* queries which were contained in the original theory do evaluate to something.

To derive a hierarchical structure it is also necessary to have rules which introduce new nodes. These rules take an empty sentence as input and generate an output sentence $s : (n, p, v)$, where v has to be an atomic value, n has to be a new node not yet contained in the theory and there has to be a sentence $s_i : (n_j, p, v)$ (i.e. a sentence that contains the path-value pair of the new node) in the theory. For example, the theory $H2$ could be transformed into:
H3

```
Node 1:
    <syn cat>          == verb

Hobble:
    <syn cat>          == verb
    <mor root>         == hobble
    <mor pres sg 1st> == <mor root>
```

To inherit from the new node a rule is introduced that transforms a sentence $s : (n, p, v)$ into $s' : (n, p, v')$ where v' is another node in the theory (here `Node 1`) under the condition that there is a sentence $s_i : (v', p, v)$, i.e. the node v' contains a sentence whose path is identical to p and whose value is identical to v. This yields the new theory:
H4

```
Node 1:
    <syn cat>          == verb

Hobble:
    <syn cat>          == Node 1
    <mor root>         == hobble
    <mor pres sg 1st> == <mor root>
```

Further rules allow the addition of further sentences to a newly created node and the introduction of global inter-node inheritance. There are also rules which allow the value of a path to be inherited from a node-path pair.

Default rules are only applied once all transformation rules have been applied[24] to avoid problematic interactions. Default rules essentially shorten the right hand side of a path. Thus

[24]The permissible sequences of transformation rules are potentially infinite, however the heuristics discussed below introduce further constraints that make the search finite.

applying them before all transformation rules have been applied might lead to transformation rules no longer being applicable. Mixing transformation and default rules could also lead to theories which are not complete and consistent with respect to the original theory.

Defaults are introduced by ordering the sentences of a node and looking at each of them in turn, starting with the longest. At each step, a sentence can be modified by shortening it by one element (i.e. by cancelling the last attribute of the path on the left hand side) or by removing the sentence altogether. The different default rules state the constraints under which a sentence can be shortened or removed. For example, one default rules states that a sentence $s : (n, p, v)$ can be shortened by one element to the sentence $s' : (n, p', v)$ if the sentence s' does not yet exist in the theory and if v is either a node or an atomic value. A sentence $s : (n, p, v)$ can be removed if a sentence $s' : (n, p', v)$ which fulfils these requirements already exists.

For example, in the theory $H'1$ below, the left hand side of the first sentence can be shortened to <mor pres sg>. The next step is to check whether the second sentence can be shortened by one element, however since the shortened sentence <mor pres sg> == come has just been introduced into the theory, the second sentence can be removed altogether. The third one, however can neither be shortened nor removed because there is already a sentence with the left hand side <mor pres sg> but a different value (come *vs.* come+s), namely sentence one, which has just been shortened in this way. Sentence four can be shortened by one element. Thus, after the application of three default rules the theory is transformed into $H'4$:[25]

H'1

```
Come:
    <mor pres sg 1st> == come
    <mor pres sg 2nd> == come
    <mor pres sg 3rd> == come+s
    <syn cat>         == verb
```

H'4

```
Come:
    s1: <mor pres sg>     == come
    s3: <mor pres sg 3rd> == come+s
```

[25]Note, that the outcome of applying the default rules depends on the order of the sentences. In the present case the first three sentences could have been considered in any order because they are all of equal length, however if the sentence <mor pres sg 3rd>==come+s had been considered first, it would have been possible to shorten it, making the third person singular ending the default. This result is clearly not desirable and prevented by the heuristics, which are discussed below.

```
s4: <syn>            == verb
```

Once all sentences of a node have been considered by the default mechanism, it is permissible to re-consider those sentences which had been changed the last time. Sentences which could not be changed (like sentence three) cannot be re-considered. Before the default mechanism is applied again the sentences have to be re-ordered according to their new lengths.

There are further default rules, like those dealing with global inheritance descriptors etc. These are formulated in a similar way but have more constraints associated with them to ensure that their application results in theories that are complete and consistent.

As was mentioned before, there are usually several ways in which a theory H_n can be transformed or expanded into theory H_{n+1} by applying a transformation rule. Consequently there are several permissible theories H_{n+1} that can be generated at any given point. The heuristics for guiding the search through the space of valid theories H_{n+1} consist of two parts. First, the application of transformation rules is further constrained, such that a rule cannot be applied in all cases where it would lead to a complete and consistent hierarchy but only in cases where it "makes sense" to apply the rule. For example, it does not make sense to create new nodes randomly. A new node only makes sense if it generalises, i.e. it must contain a sentence which is shared by at least two nodes at a lower level. Barg restricts this even further by calculating the similarity of a pair of nodes and requiring that a node that generalises over two nodes `N1` and `N2` is only created if `N1` is `N2's` most similar node or the other way round,[26] where similarity is calculated by counting the number of sentences that they have in common.[27] Restricting the creation of new nodes in this way also makes the search through sequences of rule applications finite.

The application of default rules is not always deterministic either and depends on how sentences of equal length are ordered. In the example above, considering sentence three before either sentence one or sentence two would have resulted in making the 3rd singular form the default. This is clearly less desirable and is avoided by a constraint which requires sentences of equal length and identical prefixes (in this case `<mor pres sg>`) to be ordered in such a way that those whose right hand side occurs more often come first.

The second part of the heuristics involves ranking all theories H_{n+1} and only considering the n-best theories for further expansion. This involves the formulation of a set of evaluation criteria. Barg distinguishes two sets of criteria: *search criteria* which guide the application of transformation rules and *selection criteria* which are used to choose the best theory (or theories

[26] The two nodes `N1` and `N2` have to be at the same level of the hierarchy.

[27] Note, that under this definition "similarity" is not symmetrical.

if there are several equally good ones) at the end of the inference process (i.e. after the default mechanism has been applied).

Barg argues that both sets are domain-dependent. For a linguistic domain, she suggests criteria related to the size, simplicity or homogeneity of a DATR theory or certain parts of a DATR theory (smaller hierarchies are assumed to be simpler and thus better). Since these criteria are often fairly abstract, Barg defines a set of "indicators" for each criterion, e.g. the size of a theory is measured in terms of the number of objects (i.e. nodes), the number of attribute-value pairs and the number of levels in the hierarchy.

Not all criteria are equally useful. For instance, the size of a theory does not say much about its quality. A small theory may not capture many generalisations but a big theory does not necessarily capture many, good generalisations either; it may merely indicate a bad organisation of the hierarchy. Likewise not all criteria are equally suitable as search and as selection criteria. For example, the size of a hierarchy is not a particularly good search criterion because all theories grow bigger (with respect to the number of sentence) during the application of transformation rules.

Barg demonstrates the performance of her system on two learning tasks: constructing hierarchies for German substantive inflexion and for verb classes. To evaluate her results she compares them manually to existing hierarchies for the same domain. She concludes that while the automatically constructed hierarchies tend to differ structurally from the manually built hierarchies they nonetheless capture the most important linguistic generalisations. However, it has to be said that she only tests her program on fairly small lexicons (around 15 entries and 10 attribute-value pairs per entry) and for fairly small domains. She does not try to infer a hierarchy for a big lexicon covering all parts-of-speech. In addition, she partly tunes the selection of her evaluation criteria to the desired output hierarchy, even though she achieves some degree of consistency, e.g. the best search criterion for all tasks seems to be the number of different inheritance descriptors. Also, it is difficult to re-apply Barg's learning system to HPSG-style inheritance hierarchies because DATR hierarchies are formally quite different and Barg's system is (naturally) fairly DATR specific.

Petersen (2001) takes a set-theoretic approach, using ideas from *Formal Concept Analysis* (Ganter and Wille, 1999) to construct inheritance hierarchies automatically. Formal Concept Analysis is a field of applied mathematics that formalises the notion of a *concept* and provides tools for mathematically analysing conceptual data. The term *concept* as used in this branch of mathematics will be defined formally in Chapter 4. Informally, a concept is a set of properties

	POS	SUBCAT	PAST
love	verb	trans	+ed
elapse	verb	intr	+ed
hit	verb	trans	+0
burst	verb	intr	+0

Table 2.1: Some lexical data

paired with a set of objects sharing these properties. The set of properties is the *intension* of the concept, the set of objects is the *extension*. In a lexical domain, the properties are attribute-value pairs and the objects are lexical entries. For example, given the lexical data in Table 2.1, a possible concept is ({POS:*verb*, SUBCAT:*intr*}, {elapse, burst}). Another concept is ({POS:*verb*}, {love, elapse, hit, burst}). It is possible to construct a partial order over the concepts contained in the data by looking at intension or extension sets that include each other. The result is a *Concept/Galois lattice* (Figure 2.32). This lattice contains a lot of redundancy because several attribute-value pairs and object names occur repeatedly. However, it can easily be turned into a hierarchy that is minimal with respect to both attribute-value pairs and objects by only stating attribute-value pairs at the highest node at which they occur and only stating objects at their lowest node (Figure 2.33). The result is a hierarchy which contains every attribute-value pair exactly once, i.e. a hierarchy which is redundancy free with respect to attribute-value pairs.

Petersen's approach bares some similarity with the approach that I propose in that she also suggests constructing an inheritance hierarchy by first building a Galois lattice and then pruning it. However, I will argue in the next chapter that a "redundancy-free" hierarchy is not the best solution, as manually built hierarchies are rarely redundancy-free because a redundancy-free option is often not the most linguistically plausible solution.

The hierarchies derived by Petersen are monotonic. Petersen argues that non-monotonic hierarchies are best constructed semi-automatically but she makes some interesting suggestions how ideas from Formal Concept Analysis could be exploited to suggest possible default sets to the user. She argues that it is possible to identify a so-called *acceptable set of default information*. This is a set of attribute-value pairs that fulfils two conditions. First, its extension is non-empty, i.e. there exists an object which contains all of the attribute-value pairs in its intension. Second, the set of attribute-value pairs is complete with respect to its implications, i.e. if an attribute-value pair *x* that is contained in a default set implies an attribute-value pair *y*

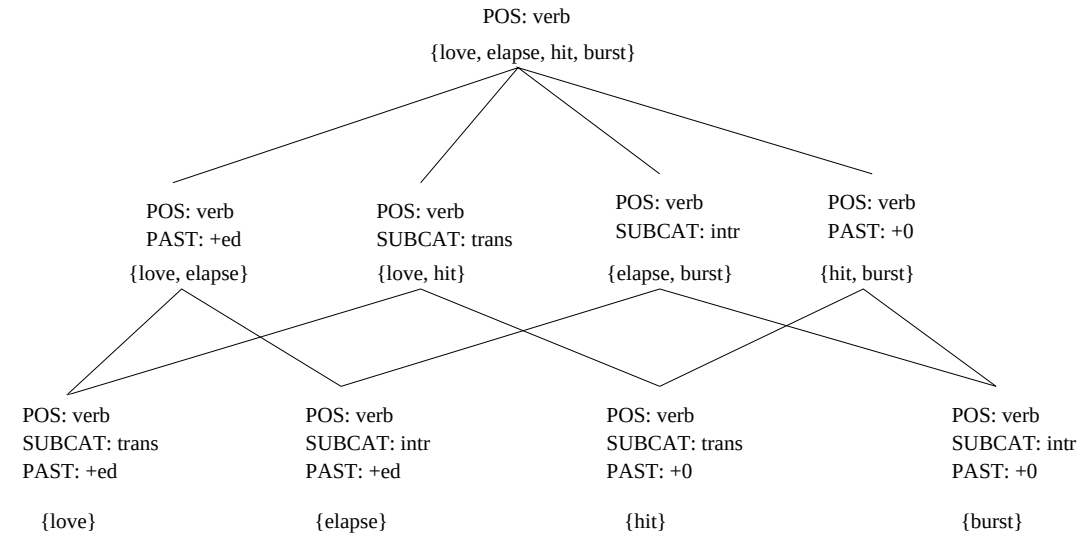


Figure 2.32: Concept lattice

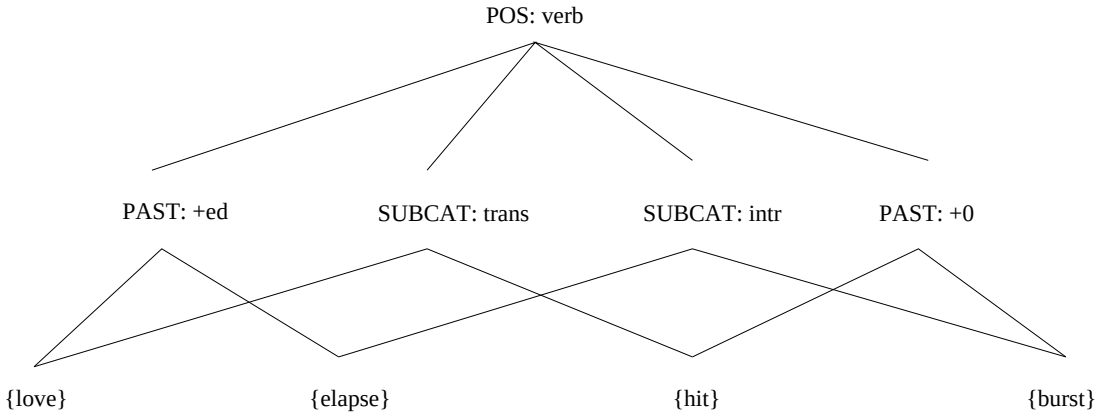


Figure 2.33: Reduced concept lattice

then *y* also has to be included in the set. This approach can deal with regularities (i.e. top-level defaults) and exceptions. To deal with sub-regularities, one has to look at partial implications instead. Petersen suggests that the user might be able to choose among different default sets on the basis of the size of their extensions and intensions.

Cahill (1998) is concerned with the semi-automatic construction of multi-lingual (Dutch, English, German) hierarchies. While she uses DATR as a representation language her approach is fairly DATR-independent. The input to her system are three lexicons, one for each language.

The output is a set of four connected hierarchies, one for each language and one additional hierarchy that expresses (defeasible) generalisations that hold across the languages. The general structure of the hierarchies (i.e. the partial order of nodes) is mostly hard-wired. For example, the general hierarchy consists of a node *word* from which nodes that correspond to the other parts-of-speech inherit. Somewhere further down the hierarchy are nodes that correspond to individual lexemes. The language-specific hierarchies have a similar structure, however each node in a language specific hierarchy not only inherits from its parents in that hierarchy but also from the corresponding node in the general hierarchy. For example, a node *fish* in the English hierarchy inherits from the node *noun* in the English hierarchy but also from the node *fish* in the general hierarchy, where the general type *fish* expresses properties of the lexeme that are shared by all three languages, for example its semantics. Likewise the node *noun* in the English hierarchy also inherits part of its properties from the *noun* node in the general hierarchy.

Cahill's algorithm infers morphological and phonological generalisations over the three lexicons. These generalisations are extracted automatically by applying a pattern-matching algorithm to the orthographic and phonological forms of lexical entries. Properties that are shared by at least two of the three languages are treated as defaults and specified in the general hierarchy.

Finally, in previous work (Sporleder 1999, see also Lüngen and Sporleder 1999), I proposed a modified decision tree algorithm to derive monotonic and non-monotonic single inheritance hierarchies. The algorithm successively splits the whole set into suitable subsets. Similar to decision trees a set is split into subsets based on the values of one attribute. However unlike a decision tree algorithm, the algorithm selects attributes that *minimise* information gain. Information gain has to be minimised because it is the purpose of an inheritance hierarchy to capture generalisations. A construction algorithm for inheritance hierarchies thus has to find suitable generalisations whereas a decision tree algorithm has to find discriminating properties. I also proposed several extensions to the basic algorithm. One extension permitted the construction of non-monotonic hierarchies by allowing a certain percentage of lexical entries to be "misclassified", i.e. placed under a node with which they did not share all attribute-value pairs. However, defaults and exceptions were treated in a purely numerical fashion, i.e. an exception was allowed if it did not result in too many attribute-value pairs being overridden, there were no *linguistic* constraint that guided the default mechanism. The main drawback of this method is that it cannot easily be extended to multiple inheritance hierarchies. Constructing subclasses on the basis of differences in *one* attribute also turned out to be problematic as sister classes are

often distinguished from each other on the basis of a *conjunction* of attributes or on the basis of presence or absence of certain attribute-value pairs.

There has also been work on how new lexical entries can be inserted into an existing hierarchy. For example, Light (1994) presents an algorithm for inserting new lexical entries into an existing non-monotonic multiple inheritance hierarchy. The crucial problem of this task is to insert a lexical entry in the "right" place of the hierarchy. For a monotonic single inheritance hierarchy the insertion problem would be trivial. One could simply follow a path down the hierarchy and insert a lexical item immediately below the most specific non-conflicting node. The presence of multiple inheritance makes this problem more complex. Since there may now be several paths from one node to another the worst case search time is no longer linear. If the hierarchy is also non-monotonic, finding a good insertion place is even more difficult because a conflict between the attribute-value pair set of the new entry and the attribute-value pair set of a potential parent no longer rules out the potential parent.

The first step in Light's algorithm is to compile out the hierarchy, such that every node is viewed as the set of attribute-value pairs that can be inherited from it. Finding the best insertion place for a new entry then amounts to finding the best set of parent nodes. Light defines an optimal insertion as one which minimises the following sum:

```
  # of superclasses
+ # of attribute-value pairs which cannot be inherited from these superclasses
+ # of attribute-value pairs that have to be listed to block incorrect inheritance
    from superclasses
```

Evaluating all possible subsets of nodes with respect to this formula is NP-complete and thus infeasible for all but the most trivial hierarchies. Light gives a polynomial time approximation algorithm that employs greedy search. During each iteration, a set of attribute-value pairs (i.e. a node in the hierarchy) is chosen such that the sum of the attribute-value pairs covered by it minus the attribute-value pairs clashing with the new entry's attribute-value pairs is maximised. The algorithm terminates when it can no longer find a set of attribute-value pairs which improves the result, i.e. when for all sets of attribute-value pairs the number of incorrectly inherited attribute-value pairs is higher or equal to the number of correctly inherited attribute-value pairs.

Since the algorithm is greedy it does not always produce the best result; it may get stuck in a local optimum. Another problem is that the algorithm may produce redundant links because there is nothing to stop a node from inheriting from two nodes which in turn inherit from each other. This is due to the fact that the algorithm only deals with sets of attribute-value pairs

without having access to the original inheritance relations within the hierarchy. Light suggests tackling this problem by weighting the attribute-value pairs with respect to the position of the attribute-value pair's original node (i.e. the node where the attribute-value pair was introduced) in the hierarchy. Attribute-value pairs with lower weights would be preferred over attribute-value pairs with higher weights biasing the program to inherit a attribute-value pair from the lowest possible node. Another suggestion would be to use post-processing to remove redundant links.

The algorithm is not suitable for learning hierarchies from scratch because it does not introduce new non-terminal nodes.[28] It merely chooses the ancestors of a new lexical entry from the set of existing nodes. Thus it is only useful for inserting a limited number of new lexical entries into an existing hierarchy.

Villavicencio (2002) describes how categorial grammars can be learned from a corpus. She uses an inheritance hierarchy as the backbone for her learning system and starts with an initial hierarchy that represents an underspecified universal grammar. Acquiring a grammar for a given language then involves extending the original hierarchy incrementally by adding new lexical entries that have been acquired from the corpus to the hierarchy. Villavicencio uses Light's (1994) algorithm to find the best insertion place. While Light's algorithm cannot explicitly create non-terminal nodes, it is possible to insert a new lexical entry e_1 as a subtype of a previously inserted entry e_0. As a consequence e_0 will become a non-terminal node. However, this only works if (i) $e_0 \sqsubset e_1$ and (ii) e_0 is inserted before e_1. If the entries correspond to lexemes the first condition is usually violated because different lexemes are normally incompatible (having different values for orthography or different id numbers etc.). However, Villavicencio's entries are grammatical parameters or categories. For example, she might want to insert the category *transitive verb*, represented as S|NP|NP in Categorial Grammar (i.e. a transitive verb is something that combines with two noun phrases (NP) to form a sentence (S)). Once the transitive verb category has been inserted, a ditransitive verb category (represented as S|NP|NP|NP) can be added as a subtype of the transitive category thereby making the latter a non-terminal node. However, this only works because Villavicencio's algorithm guarantees that the transitive category is always added *before* the ditransitive category. If the algorithm comes across a sentence containing a ditransitive verb before it has acquired the transitive category, the sentence is ignored (Villavicencio 2002, p. 175f).

[28] However, a terminal node may later become a non-terminal if a new entry is added which is subsumed by it. This is exploited by Villavicencio (2002), as discussed below. In general, however, it is not the case that entries subsume each other.

Basili *et al.* (1997) derive a hierarchical subcategorisation lexicon for verbs from a corpus. Like Petersen (2001) and the system proposed here they generate their hierarchies by pruning Galois lattices. As a first step they automatically extract the set of subcategorisation frames for a given verb from a text corpus. Subcategorisation frames can be partially ordered. For example, the intransitive frame *[subject, verb]* (as in *Peter ate*) subsumes the transitive frame *[subject, verb, direct object]* (as in *Peter ate the cake*). The second step of Basili *et al.*'s algorithm is to generate the partial order of the subcategorisation frames for a given verb. This partial order is effectively a Galois lattice. Every node (C) in the lattice consists of two sets: its *intension* (F), i.e. the syntactic arguments of the frame (such as *subject, direct object*), and its *extension* (S), i.e. the verb occurrences (i.e. verb tokens) in the corpus with which it is associated. For instance, given the two sentence mini-corpus:

(s_1): *Peter ate.*
(s_2): *Peter ate the cake*

the node corresponding to the intransitive verb frame for *eat* is:

({ *subject* }, { eat_{s_1}, eat_{s_2} })

Since the automatically extracted set of subcategorisation frames is likely to contain a certain amount of noise a third step involves the pruning of the lattice in such a way that erroneous frames are removed. To decide whether a node should be pruned, two selection criteria are defined: *selectivity* measures how *relevant* a frame is for the elements of its extension, *linguistic preference* is a probabilistic measure of the plausibility of a frame. *Selectivity* (s) basically measures what proportion of syntactic contexts which are covered by a subcategorization frame C are also covered by a descendant of C and is formally defined as:

$$
\begin{aligned}
s(C) &= \textstyle\sum_{t \in S} \frac{w(t,C)}{n_L}, \text{ where} \\
w(t,C) &= \textstyle\prod_{C' \in Descendants(C)} 1 - s(C')\delta(t,C') \\
\delta(t,C) &= \begin{cases} 1 & t \in S \\ 0 & \text{otherwise} \end{cases} \\
n_L &= \text{the number of verb occurrences on which the lattice has been built}
\end{aligned}
$$

To calculate *linguistic preference* (l), each grammatical relation p (such as *subject* or *direct object*) is assigned a weight, $w(p)$, that reflects the likelihood of the verb occurring with p,[29] where:

[29] Basili *et al.* do not go into the details of how $w(p)$ is estimated.

$\sum_p w(p) = 1$

Let *ARG* be the set of grammatical relations occurring in the lattice. The linguistic preference of a node is then defined as:

$$l(C) = \prod_{p \in ARG} w(p,C), \text{ where}$$

$$w(p,C) = \begin{cases} w(p) & p \in F \\ 1 - w(p) & \text{otherwise} \end{cases}$$

Selectivity and *linguistic preference* are combined into an overall *preference* function:

$$pref(C) = s(C)l(C)$$

A node C is retained in the lattice if $pref(C) > \sigma$ where σ is an experimentally set threshold.

Basili *et al.* evaluate the subcategorisation lexicon extracted by their algorithm against a set of existing resources (e.g. manually built subcategorisation lexicons). They do not evaluate the hierarchy as such.

It has to be noted that Basili *et al.* build their hierarchy from a corpus rather than from an input lexicon. That is, their task is fundamentally different from the task discussed here. Hence, their selection criteria cannot be transferred to the task of building hierarchies for flat lexicons without the use of a corpus.

There has also been work on the automatic construction of semantic taxonomies from machine-readable dictionaries (Copestake 1990, Copestake 1992). In this research the entries in a dictionary are used to extract hyponymy relations. For example, the entry below[30] suggests that *autobiography* is a hyponym of *book* (i.e. *book* subsumes *autobiography*). This information can then be used to build an inheritance hierarchy in which the semantic type *book* subsumes the semantic type *autobiography*.

autobiography: a book written by oneself about one's own life.

While this research also falls under the heading of hierarchy construction, the hierarchies are not really lexical inheritance hierarchies but bear more resemblance to semantic nets, i.e. the task is more about inducing world knowledge than structuring lexical knowledge. The approach differs crucially from the work presented here in that the *classes* of the resultant hierarchy are not constructed automatically. They are already implicit in lexeme definitions in the dictionary.

Some work on automatic hierarchy construction has also been undertaken within a general machine learning context. For example, clustering techniques can be used to create a hierarchy

[30] The example has been taken from Copestake (1990).

of classes (e.g Michalski and Stepp, 1983). However, most hierarchical clustering algorithms create tree-structures and are thus not suitable to build multiple inheritance hierarchies. There are, however, clustering techniques where an object can simultaneously belong to several clusters or where probabilities for an object's membership in different clusters are calculated. This technique is called *clumping* (Fisher and Langley 1985). It is possible that clumping techniques could be extended to derive multiple inheritance hierarchies.

2.6 Summary

This chapter formally introduced lexical inheritance hierarchies and defined the relevant concepts. It also gave an overview of the use of lexical inheritance hierarchies and of previous approaches to construct them automatically. The next chapter will discuss the task in more detail and sketch the approach taken here.

Chapter 3

Learning Lexical Inheritance Hierarchies

This chapter outlines the learning task and the approach suggested here. Section 3.1 defines the task in more detail and constrains it in various ways. Section 3.2 outlines some of the difficulties that are inherent in the task. Section 3.3 illustrates the task with an example. Section 3.4 discusses previous approaches and outlines some problems with them. Finally, Section 3.5 outlines the approach taken in this thesis.

3.1 The Task

Automatic hierarchy construction can be defined as follows: given a "flat" lexicon L, find an inheritance hierarchy H such that:

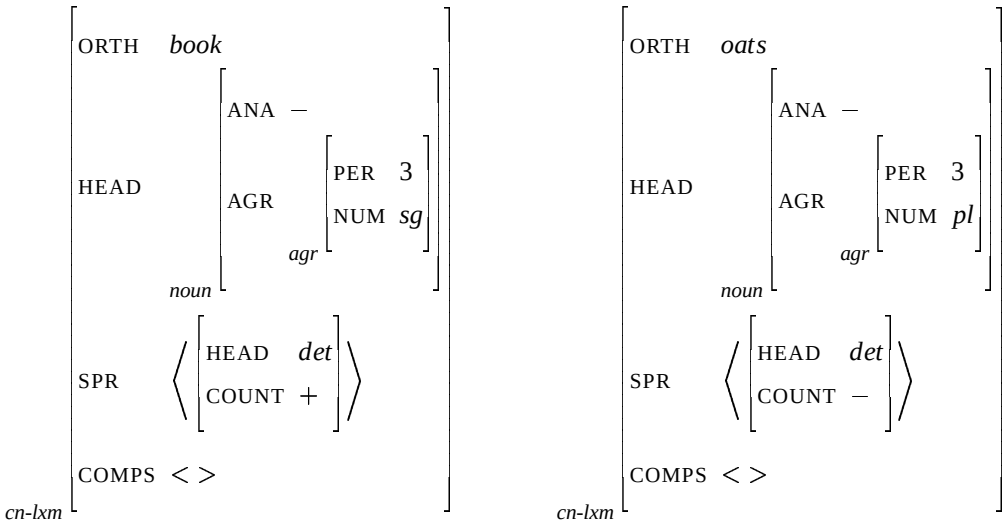
- H is sound with respect to L
- and the data in L is well structured by H

A hierarchy H is *sound* with respect to the input lexicon L if the input lexicon L can be retrieved by compiling out the hierarchy H via inheritance, i.e. if the hierarchy construction does not add, omit or change information.

The following sections discuss the task in more detail but before that I am going to constrain the task in three ways: First, I will assume that both the input lexicon and the output hierarchy are untyped. Second, the input lexicon should not contain any nested feature structures; complex feature structures need to be converted into sets of atomic feature structures. Both

constraints are in line with previous approaches to automatic lexical inheritance hierarchy construction (see Petersen 2001, Barg 1996a, Light 1994). Furthermore, the approach suggested here is restricted to monotonic hierarchies. This is a consequence of the availability of training data. The following paragraphs discuss these constraints in more detail.

That the input lexicon will usually be untyped follows from the task itself since the types are what needs to be inferred. The existence of types in the lexicon implies a particular hierarchy and in the unlikely event that one has the type information in the lexicon but not the hierarchy, it will usually be possible to simply "read off" the hierarchy implied by the types. For example, from the fully specified typed lexical entry for *book* in Figure 3.1(a) it is relatively easy to infer that there should be a type/node *cn-lxm* (common noun lexeme) from which *book* inherits. If there are other lexical entries of type *cn-lxm* like *oats* (Figure 3.1(b)) it is also straightforward to determine how the node *cn-lxm* should look: all properties shared by all entries of that type should be part of the underlying intension of the node for *cn-lxm*. Hence, from the two entries in Figure 3.1 one can infer that the common noun portion of the hierarchy should look as in Figure 3.2.



(a) book (b) oats

Figure 3.1: Typed lexical entries

Usually hierarchy creation and lexicon creation go hand in hand. Typically, the normal development cycle for grammars which make use of a hierarchical lexicon starts with a relatively small set of sentences that have to be processed or linguistic phenomena that need to be

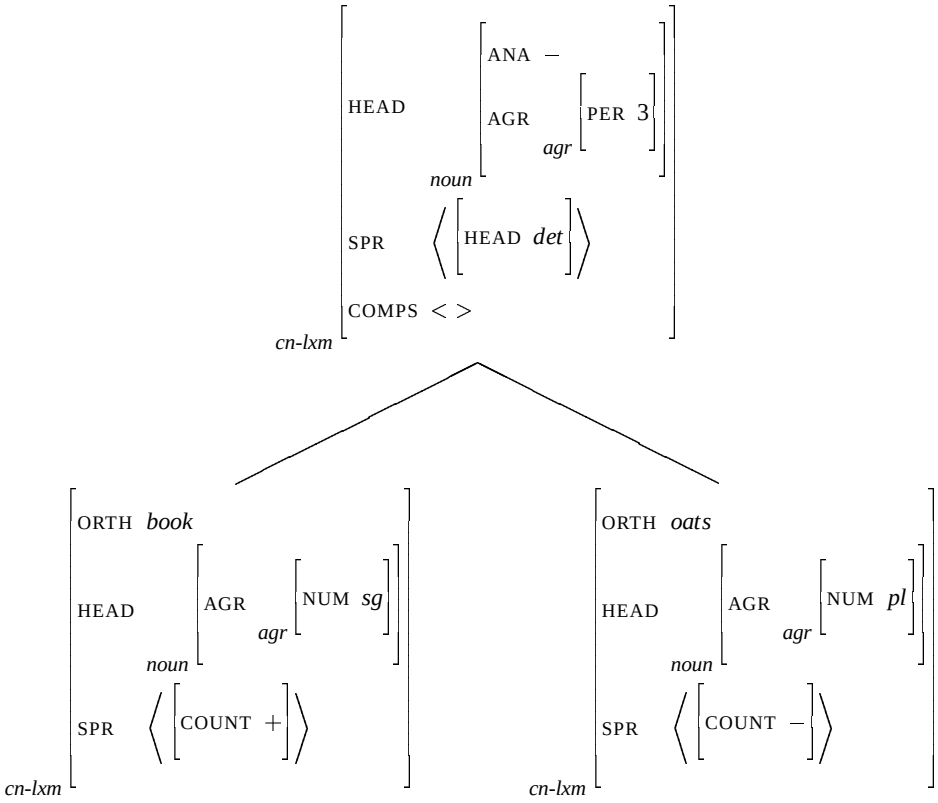


Figure 3.2: Inferred hierarchy

dealt with. Then a small lexicon and a set of rules is implemented in a grammar development environment like the LKB (Copestake, 2002) or ALE (Carpenter and Penn, 2001). This is then tested on a set of example sentences and stepwise refined and extended to additional linguistic phenomena. Hierarchy refinement and lexicon refinement —such as the addition of new attribute-value pairs— are interleaved. Automatic hierarchy construction will be of limited use under these circumstances even though it is possible that a tool which makes it easier to extend an existing hierarchy and ensures that the new hierarchy will be consistent and compatible with the rest of the grammar might be useful.

However, there are two scenarios where it may be necessary to construct a hierarchy after the lexicon has been created. It is in these cases that a system for automatic hierarchy construction is most useful. The first scenario is one where a manually created lexicon was not originally intended to be represented as an inheritance hierarchy. This scenario could arise if

one wanted to include an existing flat lexicon, such as the CELEX lexicon,[1] in an application which requires a hierarchical lexicon, such as the LKB. In this case a non-trivial lexicon exists and creating a hierarchy for it manually would be impractical. The second case arises if the lexical data has been extracted (semi-)automatically from a corpus (see the discussion on p. 7f). In both cases the input lexicon will be untyped. Hence, a system that builds hierarchies automatically has to work on an untyped input lexicon. The output hierarchy could be typed (i.e. be a *type hierarchy*) or untyped. However, there are good reasons to aim for an untyped hierarchy. An untyped hierarchy contains a set of untyped feature structures which are partially ordered. A type hierarchy contains a set of partially ordered *typed* feature structures. Hence, a valid type hierarchy has to specify which attributes are appropriate for which types and the appropriateness conditions have to be fulfilled (see Section 2.1). For every attribute there has to be a most general type for which it is appropriate (*Minimal Introduction*) and all attributes that are appropriate for a type σ also have to be appropriate for a type τ if $\sigma \sqsubseteq \tau$ (*Upward Closure/Right Monotonicity*). Furthermore, every attribute of (the underlying intension of) a feature structure F has to be appropriate for F and take an appropriate value and F's underlying intension has to contain all attributes that are appropriate for it (*Total Well-Typing*).

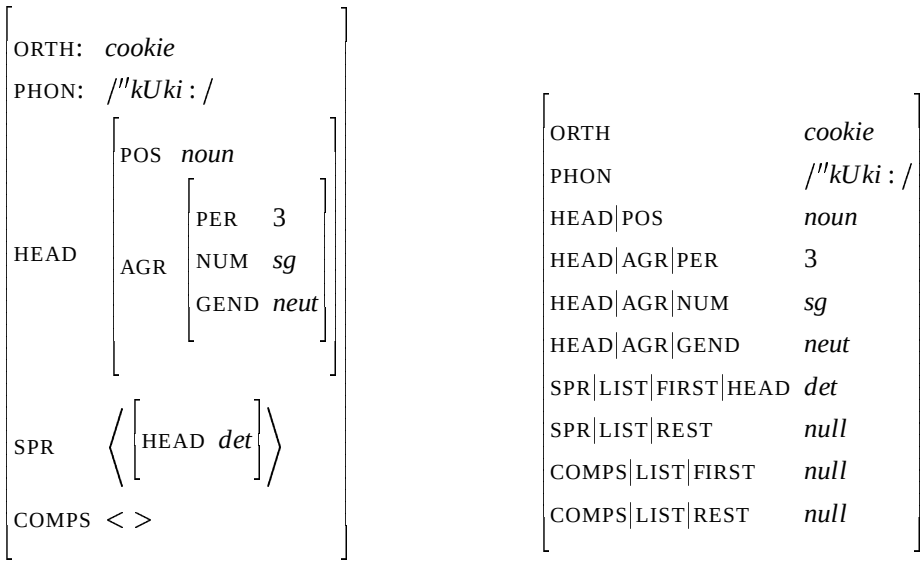
It should be possible to automatically convert an untyped inheritance hierarchy into a valid type hierarchy, even though —to my knowledge— no such algorithm exists at the present time. A conversion algorithm could start by introducing a basic type system and then iteratively refine it until the appropriateness conditions are consistent and the hierarchy is totally well-typed.[2] In any case it seems that inferring good super-nodes for an untyped lexicon is more difficult than typing an untyped hierarchy. Previous approaches to automatic hierarchy construction have been restricted to inferring untyped hierarchies. Barg (1996a) only deals with untyped hierarchies because her work has been done within the DATR formalism and DATR does not support type hierarchies in the way the LKB or ALE do. Petersen (2001), too, infers untyped hierarchies.

Requiring that the lexical entries in the input lexicon are "flat" feature structures, i.e. sets of atomic feature structures, makes it more straightforward to talk about intersections between

[1] See `http://www.kun.nl/celex/` (3.7.03).

[2] This may sound prohibitively computationally complex but the LKB grammar environment system (Copestake, 2002) already implements several functions to make a non-well typed hierarchy well-typed, such as type inference (for making typed feature structures well-typed, ensuring minimal introduction etc.) and automatic calculation and insertion of greatest lower bounds to ensure boundedness. These functions have been implemented to make it easier for the user to develop a hierarchy without having to worry about well-typedness all the time, i.e. the user does not have to specify a well-typed hierarchy but a hierarchy that can automatically (and deterministically) be made well-typed. However, the LKB still requires the user to specify the complete type-system.

entries. And the approach suggested here defines lexical generalisations as intersections between entries. It has to be stressed, however, that re-representing nested feature structures as sets of atomic feature structures does not change their information content; it is purely a representational issue. In an untyped input lexicon it is relatively straightforward to convert complex feature structures to sets of atomic feature structures. Converting a feature structure in this way means that the path prefix of each attribute will be made explicit. For example, the entry in Figure 3.3(a) will become the entry in Figure 3.3(b).



(a) Complex FS

(b) Set of atomic FSs

Figure 3.3: Flattening Feature Structures

As can be seen, lists are represented by the attributes FIRST and REST. This is the way they are usually implemented in grammar development environments such as the LKB or ALE. At this stage reentrancies also have to be re-represented. The two pieces of information potentially contained in a reentrancy, i.e. the value of an attribute and the fact that it is reentrant, are teased apart by introducing a new attribute REENTR to encode path identity (Figure 3.4). Once a feature structure has been converted it makes more sense to talk about *path-value pairs* rather than *attribute-value pairs*. Assuming flat feature structures is again in line with previous research. Feature-structure flattening is discussed in more detail in Section 6.2.2.

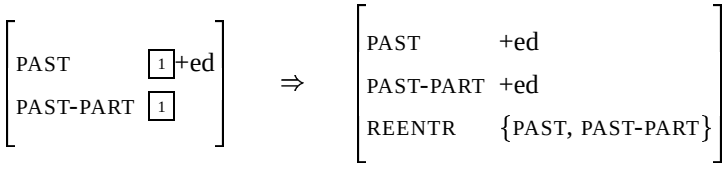


Figure 3.4: Re-representing reentrancies

The third constraint is that the output is restricted to a monotonic hierarchy. In this respect the system proposed here deviates from both Barg and Light. Petersen, however, keeps her system essentially monotonic too, but makes some interesting suggestions for a non-monotonic extension. Inferring non-monotonic hierarchies automatically is very difficult because it requires a good deal of background knowledge to decide where a default would be useful and which value should be the default. A naive approach would be to assume that it is always the most frequent value of an attribute that should be the default. However this does not work in all cases. First, it is possible that the relative frequencies in the lexicon do not correspond to the relative frequencies in the language overall. Second, one could also argue that the productive pattern should be the default. However, sometimes more than one pattern is productive, which one is chosen often depends on the phonological context (e.g. the English verbalisation suffixes *-ify* and *-ize*, or the German diminutives *-chen* and *-lein*). Furthermore, while the most frequent pattern is usually the productive pattern there seem to be examples where this is not the case. For example, Clahsen (1999) argues that the productive suffix for plural formation in German is *-s*. This is the suffix applied to product names (*Golfs*), nominalised conjunctions (*die Wenns und Abers* 'the Ifs and Buts') etc. However, Clahsen found that only 7% of the plural forms (types not tokens) in the CELEX lexicon for German were formed by adding *-s*. A second example of a minority default that is discussed in the literature is the case of Arabic plurals (see McCarthy and Prince (1990)).[3]

Defaults also bring with them the possibility of an unresolvable conflict in the inheritance hierarchy. It is usually assumed that a more specific node takes precedence over a less specific node. This makes it easy to resolve conflicts between nodes that are in a subsumption relation. However, it is also possible that a conflict occurs between nodes which do not subsume each other. For example, the parents of a node may contain conflicting information. Thus whenever a default is introduced into the hierarchy one has to take precautions that this does not lead to conflicts or that there is a way of dealing with conflicts, such as default unification (e.g. Bouma

[3]However, Boudelaa and Gaskell (2000) dispute the claim that Arabic plurals are a case of minority default.

1992, Carpenter 1993, Lascarides *et al.* 1996).

A particular problem for the approach proposed here arises from the fact that most hierarchies implemented in formalisms such as the LKB or ALE are monotonic. ALE does not support non-monotonic inheritance. The LKB allowed defaults right from the beginning but despite this the vast majority of LKB grammars are monotonic. Consequently, it is difficult to find enough suitable non-monotonic training and test hierarchies for a supervised machine learning approach.[4]

3.2 Why the Task Is Difficult

It was said above that a good hierarchy has to be sound and structure the input data well. The first of these goals is relatively easy to achieve, what is difficult is to ensure that the data in the lexicon is well structured by the hierarchy. Three circumstances make the task particularly hard: First, there is no unique best solution. What is considered the best solution often depends on personal taste and the application for which the hierarchy is used. This makes it impossible to build a system which can automatically generate *the best* hierarchy. The most one can hope for is to automatically find a reasonably good hierarchy. Indeed an automatically derived hierarchy may differ from a manually built one for the same the lexicon and still be equally linguistically plausible. Given that one cannot discriminate completely among the plausible hierarchies, a semi-automatic system for constructing the hierarchy will often be the best solution.

A second and related reason why it is difficult to achieve very good results is that background knowledge, i.e. knowledge that is extraneous to the lexicon itself, is often needed to assess whether a hierarchy structures the input data well. It would perhaps be possible for a linguistic expert to feed in such knowledge but this has several disadvantages. The main disadvantage is that it is impractical and subject to human error. But it also makes the system less generally applicable. For example, an algorithm tuned towards distinguishing parts-of-speech would under-perform if applied to a lexicon which, for some reason, only contained verbs. Furthermore, relying on background knowledge hampers a program's ability to be data-oriented and detect unexpected interrelations in the lexicon. Therefore, it was decided not to feed in extraneous knowledge. In this respect the approach taken here is in tune with previous approaches, such as Barg (1996a), Petersen (2001) and Light (1994), neither of which

[4] In particular, there is no non-monotonic hierarchy that is big enough to serve as a training hierarchy, as the biggest LKB hierarchy, LinGO ERG (English Resource Grammar), is monotonic (see Section 6.1).

feed in background knowledge. Such background knowledge can be factored in however, if a semi-automatic system is used (in which case the background knowledge is provided by the user).

The third reason that makes the task difficult is that the search space is very big indeed. For a given input lexicon the number of sound monotonic hierarchies is exponential (with respect to the number of path-value pairs) while the number of sound non-monotonic hierarchies is infinite. The reason why the search space is infinite for non-monotonic hierarchies is that path-value pairs can be overridden. This implies that path-value pairs can also be *re-set* to their original value further down the inheritance chain. Theoretically a path-value pair can be overridden and re-set infinitely often which means that there are infinitely many (sound) non-monotonic inheritance chains.

It is theoretically possible to disallow path-value pair re-setting, i.e. by adding a constraint that the same path-value pair can occur only once in an inheritance path. This would make the search space finite for non-monotonic hierarchies. However, there are a few cases where path-value pair re-setting is actually quite useful. For example, suppose a lexicon contains a boolean valued attribute QUESTION-INV which specifies whether or not a verb will be inverted in a question. Main verbs generally cannot be inverted in questions and are therefore specified as QUESTION-INV:− (Figure 3.5). Auxiliaries, which are a subclass of verbs, are inverted in question and thus override the path-value pair QUESTION-INV:− with QUESTION-INV:+. Modals, which are a subclass of auxiliaries, are generally also inverted in questions. However, for the modal *ought*, the inverted use in questions is somewhat rare (*ought* is usually replaced by *should* in questions). So it may be desirable to specify *ought* as QUESTION-INV:−. This effectively sets the value of QUESTION-INV back to its original value (as specified at the verb node). Sub-regularities like this suggest that re-setting of path-value pairs should in principle be allowed in non-monotonic inheritance hierarchies. Of course there will be a bound to the depth of value re-setting that is linguistically useful or necessary. Hence, it would be possible to restrict the search space for practical purposes.

The search space for monotonic hierarchies is finite but unless it is further constrained it contains 2^{2^l} hierarchies in the worst case for an input lexicon that contains l different path-value pairs. This can be estimated as follows: A node in a hierarchy generalises over its terminal descendants by listing all the properties that its terminal descendants have in common. That is, a node is effectively an intersection of the (underlying) intensions of its terminal descendants. In the worst case each subset of the powerset —except the empty set— of lexical entries has a

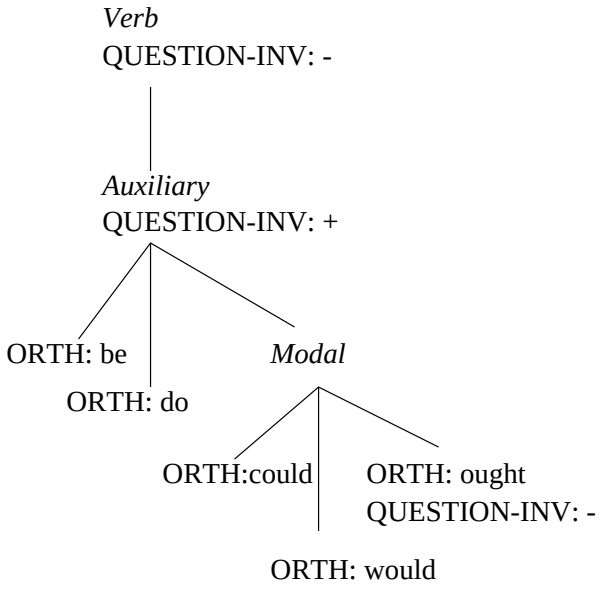
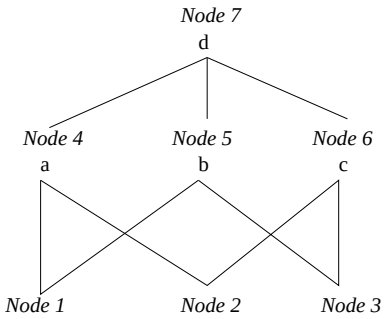


Figure 3.5: Re-setting the value for QUESTION-INV

unique non-empty intersection, giving rise to $2^n - 1$ potential non-terminal nodes for a lexicon with n entries. This is demonstrated by the data in Figure 3.6, where three entries in a lexicon (Figure 3.6(a)) give rise to a maximal hierarchy with 8 nodes (Figure 3.6(b)). As will also become clear from the data, this worst case scenario can only occur if every lexical entry contains all but one path-value pair and if the path-value pair that is missing is different for all lexical entries.

entries	pvps		
1	a	b	d
2	a	c	d
3	b	c	d

(a) lexicon



(b) maximal hierarchy

Figure 3.6: Worst Case Scenario

Unfortunately, this is not the whole story because manually built hierarchies commonly

contain nodes whose intension is not an intersection of their descendants but merely a subset of the (underlying) intension of a descendant. In other words, manually built hierarchies often contain nodes that generalise over one entry and a set of linguistic entities not contained in the lexicon. For example, a lexicon may only contain one intransitive verb but still contain a non-terminal node that generalises over all intransitive verbs (Figure 3.7, *Node* 2).

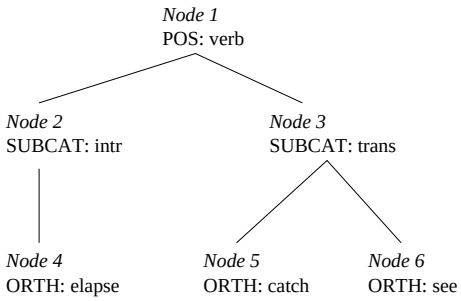
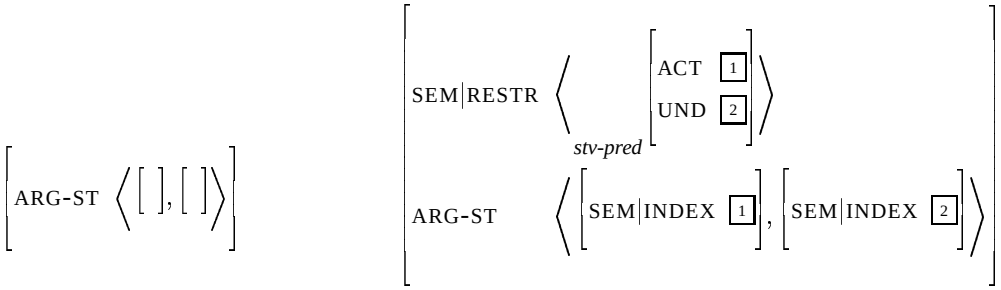


Figure 3.7: Hierarchy with a generalisation over less than two entries

A real example occurs in the Sag and Wasow (1999) lexicon. This lexicon separates types that deal with semantics from types that deal with syntax. For example, there is a type *stv-lxm*, which deals with the syntax of strict transitive verbs (Figure 3.8(a)), i.e. the fact that strict transitive verbs have two complements, and a type *stv-sem* which deals with the semantics (Figure 3.8(b)), i.e. the fact that the subject of the verb is identified with the actor of the action the verb describes and the object is the undergoer of this action. In the manually built hierarchy, the syntactic type inherits from the semantic type. In some cases two syntactic types inherit from the same semantic type, e.g. *adj-lxm* and *adv-lxm* both inherit from *modifier-sem*. However, in some cases the semantic type only has one immediate descendant. For example, the only type inheriting from *stv-sem* is *stv-lxm*. This case is slightly different from the one above (i.e. a lexicon only containing one intransitive verb), in that the division between syntactic and semantic properties does not have anything to do with generalising beyond the entries in the lexicon but is rather motivated by editorial considerations (the creators of the hierarchy claim the separation improves readability). However, it leads to the same problems because the intension of *stv-sem* is not an intersection of the intensions of its descendants (since it has only immediate descendant) but rather a subset of the (underlying) intension of its descendant.

If the (underlying) intensions of nodes can simply be subsets of the intensions of one of their descendants, the number of potential nodes in a hierarchy is bounded by the number of path-value pairs (l) in the lexicon rather than by the number of nodes, i.e. the number of



(a) *stv-lxm*

(b) *stv-sem*

Figure 3.8: Overt intensions of *stv-lxm* and *stv-sem*

potential nodes is 2^l. This is bad news because the number of path-value pairs in a lexicon is typically much higher than the number of nodes unless the lexicon is very big indeed. For example, the lexicons used in the experiments in Chapter 7 contained between 3 and 10 times as many path-value pairs as lexical entries.

Going from the number of potential nodes in a hierarchy to the number of sound hierarchies adds another layer of complexity. A hierarchy commonly only contains a subset of the potential nodes and in the worst case every subset in the powerset of potential nodes gives rise to a sound hierarchy. This means that the number of sound monotonic hierarchies for an input lexicon with l different path-value pairs is 2^{2^l} in the worst case.

In practice, this worst case scenario will of course never arise. First, the fact that path-value pairs are not atoms but complex entities means that some combinations are impossible. For example, a monotonic hierarchy cannot contain a node whose underlying intension contains the same attribute twice but with different values. Hence the worst case scenario could only arise if every attribute in the lexicon had only one value, which will not normally be the case. Second, not all combinations of nodes will give rise to a valid hierarchy because every hierarchy is required to contain at least a unique root and nodes corresponding to the lexical entries.

All previous approaches have been restricted to hierarchies which only contain nodes that do generalise over at least two lexical entries. In Barg's (1996a) system this is a consequence of the set of heuristics which she defines to restrict the search space (see page 46). Part of the heuristics constrains the application of transformation rules and the rule which creates new nodes is restricted in such a way that a new node can only be created if at least two other nodes will inherit from it. For Petersen (2001) the restriction follows from her use of concept

lattices as the search space for the algorithm. Since in a concept lattice every node has at least two immediate descendants she cannot derive any hierarchies that contain nodes which do not generalise over the lexicon.

Indeed there are very good reasons for restricting the search space in this way. First, it is impossible to determine from the lexicon alone whether a generalisation that is not contained in it is a good generalisation (see Section 4.2). This again requires some form of linguistic knowledge that is extraneous to the lexicon. Like previous approaches, the approach taken in this thesis is restricted to generalisation over at least two entries. If one restricts the search space in this way the overall worst case complexity is bound by the number of entries, i.e. $O(2^{2^n})$ for a lexicon with *n* entries. An exhaustive search of all sound monotonic hierarchies is, of course, still intractable and a set of heuristics has to be employed. This bears the risk of getting stuck in a local maximum, making it even more difficult to achieve good results.

3.3 An Example

This section discusses lexicon and hierarchy development using derivational morphology in German as an example (cf. Lüngen (2002)). The aim is to outline some of the principles governing manual development and to make explicit some criteria which could be used to distinguish good hierarchies from less good ones.

For German it has been suggested that there are three prefix classes (Lüngen 2002, Steinbrecher 1995): non-native prefixes (*prenn*), class I native prefixes (*pre1*), and class II native prefixes (*pre2*). Distinguishing non-native from native prefixes is motivated by the fact that they behave differently. Non-native prefixes do not usually combine with native lexical roots. For example, the Latin prefix *re-* cannot be attached to the German verb *bauen* ("build", **re-bauen* - "rebuild") but it can be a attached to verbs which have a non-native origin, such as *generieren* ("generate", *regenerieren* - "regenerate").

The distinction between *pre1* and *pre2* prefixes draws on a distinction that has long been proposed for English (cf. Siegel 1979). Class I prefixes in English are, for example, *in-* and *con-*, class II prefixes are *non-* and *un-*. Class I prefixes can cause stress shift (cf. *finite* $\rightarrow$ *infinite*), class II prefixes do not (cf. *nonfinite*). Furthermore, class I prefixes may undergo automatic phonological processes, such as assimilation (*illegal* vs. **inlegal*), class II prefixes do not (*unlawful* vs. **ullawful*). Finally, if class I and class II prefixes are combined in a word, class II prefixes have to precede class I prefixes, e.g.:

(3.1) a. * ir + re + fill + able
* class I + class II + root + suffix

b. semi + in + ept
class II + class I + root

The behaviour of *pre1* and *pre2* prefixes in German differs slightly from the English case. Class I prefixes (*pre1*) do not attract primary lexical stress, e.g. /Entv''Ertn/ (*ent+werten*, "invalidate"), whereas class II prefixes (*pre2*) can attract primary lexical stress, as in /''ErtsfaInt/ (*Erz+feind*, "arch-enemy").[5] Furthermore, *pre1* prefixes cannot be attached to a stem that already contains a *pre2* prefix while *pre2* prefixes can be attached to a stem that already contains a *pre1* prefix:

(3.2) a. * ent + miß + deut + en ("de-misinterpret")
* pre1 + pre2 + root + suffix

b. un + er + klär + lich ("inexplicable")
pre2 + pre1 + root + suffix

The prefix classification proposed by Steinbrecher (1995) and Lüngen (2002) is shown in Table 3.1. The attribute NAT encodes whether or not a prefix is native, STR encodes whether it can carry lexical stress,[6] and ADJ ("adjacency") specifies whether the prefix can attach to a complex stem (ADJ:-) or not (ADJ:+). Note, that only two of the three attributes are needed to distinguish the three classes. For instance, the value of ADJ is predictable from the values of the other two attributes:

- (NAT:+ $\wedge$ STR:+) $\rightarrow$ ADJ:-
- (NAT:- $\vee$ (NAT:+ $\wedge$ STR:-)) $\rightarrow$ ADJ:+

Consequently, it would be possible (even though not necessarily desirable) to drop ADJ from the lexicon, provided that the grammar contains a morphological rule which predicts the appropriate attachment behaviour from the values for NAT and STR.

Suppose that ADJ is indeed dropped from the lexicon and that the untyped, flat lexicon in Figure 3.9 is the input to a system which builds lexical inheritance hierarchies automatically. Figure 3.10 shows some of the sound hierarchies for the prefix lexicon in Figure 3.9. Given a set

[5] Again, SAMPA has been used for the transcriptions (http://www.phon.ucl.ac.uk/home/sampa/home.htm). Double quotation marks ('') indicate primary lexical stress.

[6] Note, that the specification STR:+ only means that the prefix can *potentially* carry stress. It is used by a stress assignment component to assign a stress pattern to a morphologically complex word (Lüngen 2002).

class	NAT	STR	ADJ	examples
pre1	+	-	+	*ent-, ver-, er-*
pre2	+	+	-	*un-, erz-, miß-*
prenn	-	+	+	*hyper-, in-, re-*

Table 3.1: Prefix classes after Steinbrecher (1995) and Lüngen (2002)

of sound hierarchies the task is to select a hierarchy that structures the data well. Unfortunately, as mentioned above, there is not a single best solution to this task. In this respect automatic inheritance hierarchy construction is like tasks such as natural language generation or text summarisation where there are usually several acceptable solutions and different individuals may prefer different solutions. However, choosing a good hierarchy is not entirely subjective. In the set of sound monotonic hierarchies for a lexicon there will be many that most human experts will consider relatively implausible. Hierarchy 2 in Figure 3.10, for example, represents the three prefix classes most clearly: for each prefix class there is a node in the hierarchy such that (i) its extension is identical to the set of members of that prefix class (e.g. *ent-* and *ver-* for *pre1*) and (ii) its (underlying) intension is identical to the set of attribute-value pairs shared by the members of the class (e.g. NAT:+ and STR:+ for *pre1*). Hierarchy 3 and 5 also represent the three prefix classes but, in addition, they capture further similarities between the classes: Hierarchy 3 represents the fact that members of the classes *pre1* and *pre2* share the property of being native, Hierarchy 5 represents the fact that members of *pre2* and *prenn* share the property of potentially carrying lexical stress. Which of the three hierarchies is considered to be the "best" is probably a matter of personal choice as all of them are relatively plausible.

	NAT	STR	as in:	
ent-	+	-	*ent+laden*	(*discharge*)
ver-	+	-	*ver+meiden*	(*avoid*)
un-	+	+	*un+persönlich*	(*impersonal*)
erz-	+	+	*erz+feind*	(*arch-enemy*)
hyper-	-	+	*hyper+aktiv*	(*hyper-active*)
in-	-	+	*in+aktiv*	(*in-active*)

Figure 3.9: Input Lexicon

Intuitively, the other hierarchies are less plausible as they do not capture important facts

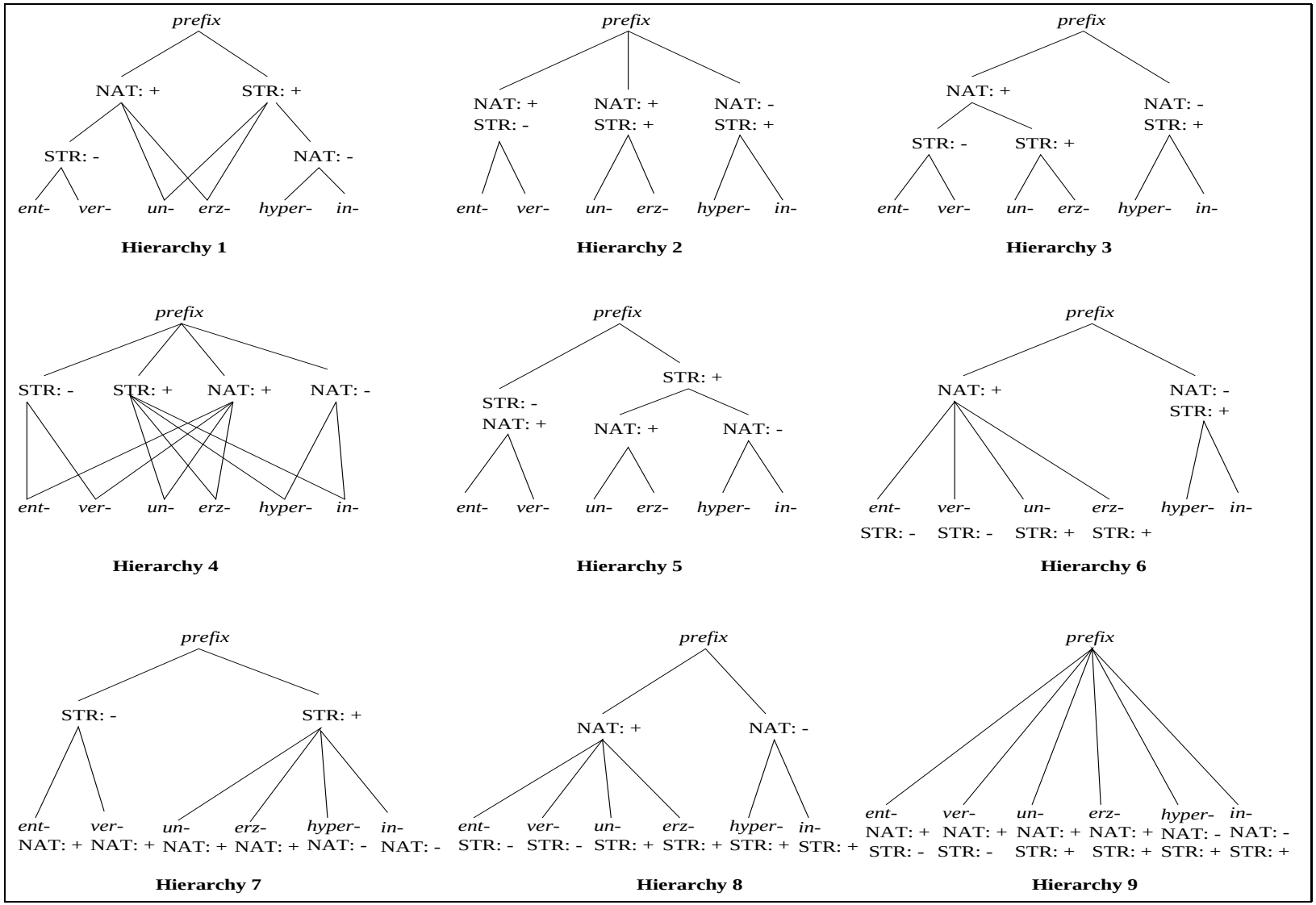


Figure 3.10: Different Hierarchies for the Data in Figure 3.9

about German prefixes as well as hierarchies 2, 3, and 5. In particular, they do not represent the three prefix classes very well. Hierarchy 1 represents only two of the three prefix classes, namely *pre1* and *prenn*. *Pre2* is not directly represented. Hierarchy 6 only represents one prefix class, namely *prenn*, as it contains a node (labelled with NAT:+ and STR:-) whose extension is identical to the set of non-native prefixes and whose intension is identical to the set of attribute-value pairs shared by non-native prefixes. Hierarchies 7 and 8, on the other hand, only partially represent one prefix class. For example, Hierarchy 7 contains a node whose extension is identical to the set of *pre1* prefixes but whose intension only partly represents this class, i.e. the hierarchy fails to capture the fact that all *pre1* prefixes are native. In other words, it fails to capture an implication between two attribute-value pairs. The same is true for Hierarchy 8 and the class *prenn*.

The hierarchies 9 and 4 are special cases. Hierarchy 9 is essentially a flat lexicon with a root node added to it. Consequently, one would not expect it to be linguistically interesting in any way. However, Hierarchy 4 is not flat, it contains four intermediate nodes, but it is not intuitively appealing either. This is despite the fact that Hierarchy 4 partially represents two of the prefix classes (*pre1* and *prenn*) as there is a node (labelled STR:-) whose extension is identical to the members of *pre1* and there is a node (labelled NAT:-) whose extension is identical to the members of *prenn*. However, it is again the case that the intensions of these two nodes do not represent the two classes very well. For example, it is not clear from looking at the hierarchy that all *pre1* prefixes are native and all *prenn* prefixes are STR:+. Like hierarchies 7 and 8, Hierarchy 4 fails to represent implications that hold between attribute-value pairs. One could argue that Hierarchy 4 is not any better than a flat lexicon. In a flat lexicon it is easy to go from extension to intension. That is, for a given entry it is relatively easy to read off which attribute-value pairs it contains but it is relatively difficult to go in the other direction, i.e. to determine for a given attribute-value pair which entries share it. In Hierarchy 4, on the other hand, it is easy to go from intension to extension, i.e. for a given attribute-value pair one can just read off which entries share it, but it is quite difficult to start with an entry and determine which attribute-value pairs it contains. Thus a flat lexicon and a hierarchy like Hierarchy 4 really are two sides of the same coin: one starts with a lexical entry and pairs it with a set of attribute-value pairs, the other starts with an attribute-value pair and pairs it with a set of entries. Crucially, neither of them identifies *classes* of entries which exhibit similar behaviour, so in a way neither of them captures any interesting generalisations about the lexicon.

This discussion showed that one can reason about the plausibility of a set of sound hierar-

chies if one knows something about the linguistic domain of the lexicon. That is, if one knows that there are good linguistic reasons to postulate three prefix classes for German, it is relatively easy to argue that hierarchies which in some way represent all or some of these classes are better than those which do not. However, a system which builds hierarchies automatically does not know which linguistic classes have been proposed for the domain; this is what it is supposed to determine itself. Hence, the question is: What makes a linguistically plausible class? It is impossible to answer this question exhaustively as things like linguistic intuition play into it and there is also a certain degree of subjectivity. However, I will try to identify at least some properties that justify the existence of a class. One thing that has already been touched upon is the existence of implications. The data contains two implications: NAT:- implies STR:+ and – because both attributes are boolean – STR:- also implies NAT:+ (by *modus tollens*). The existence of an implication often lends more plausibility to a class. Implications cannot only hold between two attribute-value pairs but also, for instance, between an attribute-value pair and an attribute. This underlies the idea of appropriateness constraints. For example, if a word is specified as POS:*noun*, this usually implies that the word also contains an attribute CASE. Hence, one could argue that the class of nouns is considered linguistically important because noun-hood also implies several other properties, such as case-marking and subcategorisation for a determiner.[7] Another –albeit probably rarer– form of implication is between an attribute-value pair and a set of values for another attribute. That is, an attribute-value pair sometimes implies that the value for another attribute has to be drawn from a given subset of all permissible values for that attribute. For example, for a highly inflected language one could introduce an attribute INFL which is appropriate for all inflectable lexemes and where each value encodes an inflection class but not all inflection classes are valid for all parts-of-speech. In this case the attribute-value pair POS:*noun* implies that the value for INFL is drawn from a certain subset, namely the subset of noun inflections.

Sometimes linguistic implications exist but are not expressed in the lexicon. For example, it was mentioned above that the values of the two attributes NAT and STR imply the value of the attribute ADJ. This complex interaction between NAT, STR, and ADJ is one reason why it makes sense to have three prefix classes. For example, the class *pre2* is justified by the fact that it is the only prefix class whose members can attach to a complex stem (ADJ:-). However, the attribute ADJ is not expressed in the input lexicon, hence the implication associated with the class *pre2* (i.e. (NAT:+ $\wedge$ STR:+) $\rightarrow$ ADJ:-) is not expressed in the lexicon. This is therefore a case where

[7] For bare plurals, the determiner is optional, of course.

background knowledge is important. It is difficult to estimate how frequent cases are where a linguistic implication is not expressed in the lexicon because the value of the affected attribute can be predicted from other attribute-value pairs but it is clear that those cases can be difficult to handle for an automatic system.

However, even if this implication is absent there are other reasons to believe that one should actually have three prefix classes, as in Hierarchy 2, and not two, as in Hierarchy 1: the former is somehow better balanced than the latter. Hierarchy 2 splits the set of prefixes into three classes and membership for each class is determined by the values of the two attributes NAT and STR. Hierarchy 1, on the other hand, splits the set of prefixes into two classes plus a set of prefixes which share some properties with one class and some with the other but which do not form a homogeneous class themselves. That is, Hierarchy 1 does not make explicit the fact that *un-* and *erz-* actually share a set of properties (namely being native and potentially stressed), even though this fact can of course be teased out by following the inheritance links and comparing the properties that these two prefixes inherit. In a way, Hierarchy 1 treats the two prefixes *un-* and *erz-* as exceptions or subregularities, i.e. *un-* is like a *pre1* prefix but not quite because it can be stressed and it is like a *prenn* prefix but not quite because it is native. In this respect, the treatment of *un-* and *erz-* in Hierarchy 1 is entirely analogous to Riehemann's (1998) treatment of the German adjective *fruchtbar* as discussed in Chapter 1 (a simplified version of Riehemann's hierarchy is repeated in Figure 3.11 below). Remember, that *fruchtbar* differs from most other adjectives ending in *-bar* because it does not carry a notion of possibility. Her solution was not to specify the property of indicating possibility at the *bar-adj* node itself but rather at a descendant (*poss-bar-adj*).[8] *Fruchtbar* was then inserted as a sub-regularity under *bar-adj* while all other *bar*-adjectives inherited from *poss-bar-adj*. Analogously, Hierarchy 1 treats *un-* and *erz-* as sub-regularities of *pre1* and *prenn* without expressing the similarities between them. For a lexicon with only six entries this imbalance may not be very obvious but if each prefix class had 20 members, it would be very obvious as there would be two prefix classes *pre1* and *prenn*, each with 20 members, and 20 sub-regularities, which overlap with both of these classes. Of course, proportion is important here. For example, if there were 20 *pre1* prefixes and 20 *prenn* prefixes but only two *pre2* prefixes one could argue that these could indeed be treated as sub-regularities.

But it is not only the failure of capturing important generalisations that renders a hierarchy implausible. Sometimes a hierarchy can also capture the *wrong* generalisations. Some

[8] Actually, *poss-bar-adj* does not specify this property itself but inherits if from another node *possibility*.

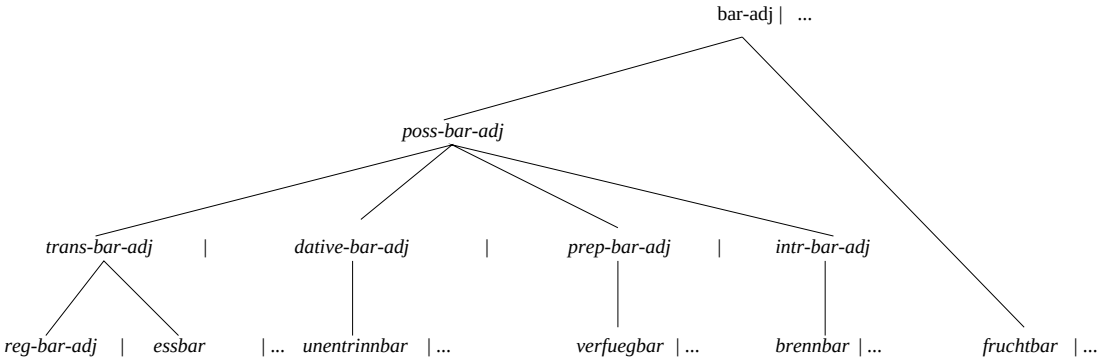


Figure 3.11: Fragment from Riehemann's (1998) hierarchy (adapted from page 5)

generalisations may be spurious and due to data sparseness. For example, it could be that all non-native lexemes in a lexicon happen to be verbs but one would not want the hierarchy to express the generalisation that all non-native lexemes are verbs because this is not generally the case. Accidental properties of the data are usually very tricky for linguistically knowledge-poor learning algorithms since these cannot go beyond the data and therefore have no way of distinguishing between spurious and true interdependencies.

On the other hand, some bad generalisations can be ruled out on formal grounds. For example, in the grammar supplied with Sag and Wasow (1999)[9] there are several lexical entries that are underspecified for number, i.e. they contain the path-value pair HEAD|AGR|NUM:*num*. Assume for the argument that the lexicon is untyped, i.e. *num* is a catch-all value for NUM —covering all entries which are for some reason underspecified for number— rather than a supertype for *sg* and *pl*. Entries which are underspecified in this way are: the determiner *the* (which can be plural as well as singular) and all common noun lexemes (which obtain their number specification by undergoing a lexical rule). Not underspecified are: determiners other than *the*[10] and proper noun lexemes (which are directly specified for number in the lexicon and not changed by lexical rule). Consequently, it would be possible to introduce a class which generalises over the determiner *the* and all common nouns. However this class would not be particularly felicitous as this generalisation is not linguistically meaningful or plausible. It could be ruled out on formal grounds by requiring that each class should have at least two path-value pairs in its underlying intension. This class would only have one as common nouns

[9]The grammar can be downloaded from `http://lingo.stanford.edu/teaching-grammars.html` (4.3.02).

[10]There are other determiners in English that can be both singular and plural, for example *some*. However, *the* is the only determiner underspecified in this way in the Sag & Wasow lexicon.

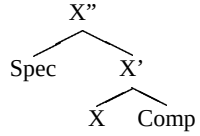
and *the* have nothing else in common.[11] It is also possible to reason about the importance of the path-value pair HEAD|AGR|NUM:*num*. For example, it is not in an interdependence relation with any other path or path-value pair. This may suggest that it is not very important and a generalisation solely based on underspecification for number should therefore be ruled out. In this respect underspecification for number differs from a path-value pair encoding part-of-speech, such as HEAD|POS:*noun*. The latter is in an interdependence relation with the path HEAD|CASE, i.e. all entries that are nouns also contain an attribute for case. Interdependencies of this kind are often important clues indicating the importance of a path-value pair and this in turn can have an influence on how good a generalisation based on this path-value pair is.

Another example from the same lexicon deals with entries that have an empty specifier.[12] These are: proper nouns (like *Peter*), determiners (like *the* and *few*), and so-called argument-marking prepositions (e.g. some uses of *to* and *in*). It would be possible to include a class in the hierarchy which generalises over proper noun phrases, argument-marking prepositions and determiners. However, such a class may not be considered linguistically plausible, even though it is —arguably— not quite as bad as a class which generalises over entries that are underspecified for number. The class could be ruled out for reasons similar to those mentioned above (e.g. determiners, proper nouns and argument-marking prepositions have nothing in common apart from an empty specifier).

Thus, while there is a continuum from plausible hierarchies to implausible hierarchies for any given lexicon, and while people may not agree entirely which hierarchy is the best, it is nonetheless possible to identify a set of hierarchies which would be judged to be relatively implausible by most people, e.g. hierarchies 4 and 9 in Figure 3.10. Likewise it is also possible to identify a set of hierarchies which would generally be regarded as relatively plausible, e.g. hierarchies 2, 3 and 5 in Figure 3.10. In between these two extremes there is probably a transition area of hierarchies which are marginally plausible, e.g. hierarchies 2, 6, 7 and 8 in

[11] How many properties different sets of entries have in common depends to some extent on the particular linguistic analysis that is used. In the grammar supplied with Sag and Wasow (1999) common nouns and *the* have nothing in common apart from underspecification for number.

[12] Formally the notion of a specifier was introduced by Jackendoff (1977) within the framework of X-bar syntax. X-bar syntax assumes that the syntactic arguments of a head X are ordered hierarchically in the following way:



That is, a head first combines with its complement(s) (*Comp*) to form an intermediate unit X' and then with its specifier (*Spec*) to form the full constituent X''. For example, a verb first combines with its complements (i.e. direct object, indirect object etc.) to form a V’ and then with its specifier (the subject NP) to form a V”.

Figure 3.10. The borders between the three areas will, of course, be rather fuzzy. I will assume that manually built hierarchies fall by definition in the plausible area as they presumably were considered plausible by at least one person (i.e. the linguistic expert who developed them). Furthermore manually built hierarchies are often carefully constructed and developed over several weeks, months and sometimes years.

A successful learning algorithm is one that derives hierarchies which fall into the "relatively plausible" set. Given the subjectivity of the task and the fact that there are usually several equally good hierarchies it is possible that the best approach is a semi-automatic system. For example, the system could reduce the search space if it could reliably identify many of the implausible solutions and a linguist could then assess the remaining hierarchies —possibly with the help of a suitable data exploration tool— and decide on which hierarchy would be best suited for his or her purposes. Different scenarios of this kind are explored in Chapter 7.

The fact that there are usually several equally good hierarchies for a lexicon also makes it difficult to evaluate the derived hierarchy against a gold standard. If a gold standard is used this will inevitably usually lead to a rather conservative evaluation method which penalises plausible hierarchies that are different from the gold standard (see Section 6.5).

Selecting a good hierarchy or a set of good hierarchies requires a way to assess different hierarchies according to a "goodness criterion" (or a set of goodness criteria). The next section discusses different approaches to these criteria.

3.4 Goodness Criteria

When constructing a hierarchy automatically it is important to have some criteria for deciding under which circumstances a hierarchy structures the input data well. While there has been a lot of work on hierarchical lexicons, the discussion focuses mainly on lexical design in general, e.g. whether a certain attribute is useful for the analysis of a particular linguistic phenomenon. Not much has been written about *hierarchy design* and what constitutes a "good" hierarchy. However, it is generally assumed that an inheritance hierarchy should (i) reduce redundancy and (ii) capture (linguistically plausible) generalisations (e.g. Pollard and Sag 1987, p. 192; Flickinger 1987, p. 5-7; Flickinger and Nerbonne 1992, p. 270).

The previous section provided an informal discussion about linguistic plausibility. It was argued that it is generally possible to distinguish relatively plausible hierarchies from fairly implausible ones, although there may be hierarchies which fall between these two extremes and even though linguists may not agree on the *most* plausible hierarchy for a lexicon. However,

while the previous section indicated some criteria for distinguishing plausible from implausible classes, it is generally difficult to specify formal criteria which can be used to assess the plausibility of a hierarchy: whether a hierarchy is considered plausible or not is to some extent a matter of linguistic intuition. This poses a major problem for automatic hierarchy construction as an automated system requires a well-defined set of formal criteria. Consequently, previous approaches have relied on the first of the criteria mentioned above, i.e. the reduction of redundancy, as a measure of hierarchy quality. The underlying –although not usually explicitly expressed– idea is that, since a good hierarchy is supposed to both reduce redundancy and be linguistically plausible, reducing redundancy will automatically lead to plausible hierarchies.

The remainder of this section summarises the redundancy criteria which have been used in this way. It is argued that a focus on minimal redundancy may not be the best way forward since the notion of redundancy is not well defined in the context of inheritance hierarchies and since it is not clear that using an *ad hoc* definition of redundancy does indeed lead to plausible hierarchies.

Barg (1996a) defines 24 so-called "search" and "selection criteria". The former are applied to the set of hierarchies that can be generated at the next step to choose the *n* best for further expansion. The latter are only applied at the final step to choose *the* best hierarchy among the *n* best. The criteria deal with aspects of size and simplicity, such as the number of nodes, the average number of path-value pairs per node or the number of inheritance links.

Petersen (2001), on the other hand, uses only one criterion. She aims at hierarchies which are minimal with respect to the number of path-value pairs, i.e. hierarchies where every path-value pair occurs exactly once.

Light (1994) does not deal with constructing hierarchies from scratch but rather with inserting new entries into an existing hierarchy. However, he also faces the problem of assessing different possible hierarchies (corresponding to different insertion places) and selecting the best. And he, too, tries to minimise the resulting hierarchy by inserting a new entry *n* in such a way that the following sum is minimised:[13]

number of parents + number of path-value pairs in *n*'s overt intension

Aiming for a hierarchy which is small and simple is in accordance with the belief that an inheritance hierarchy should reduce redundancy. It is also in accordance with the logical principle of *Parsimony* (also called "Occam's Razor")[14] which is often used in machine learning.

[13] Villavicencio (2002) uses the same principle to build her hierarchies.

[14] The principle gets its name from the medieval philosopher William of Occam to whom it has been attributed.

The principle states that one should choose the simplest theory that fits the facts. The idea is that choosing the simplest theory avoids unnecessary complication, which does not increase the performance of an algorithm. For example, if there are two different decision trees for a given classification task, both with similar performance but one containing 5 nodes and the other containing 500 nodes, one should choose the former.

However, applying this principle of simplicity directly to inheritance hierarchies is problematic. The first problem is that "minimality" and "simplicity" are not very well defined in the context of inheritance hierarchies. While it is relatively plausible to define the size of a decision tree in terms of the number of nodes it contains, this is not so easy for inheritance hierarchies. The hierarchy with the smallest number of nodes is a two-level hierarchy consisting of the lexical entries and a root node (e.g. Hierarchy 9 in Figure 3.10, repeated below as Figure 3.12). However, this hierarchy is not minimal with respect to the number of path-value pairs as all path-value pairs occur repeatedly.

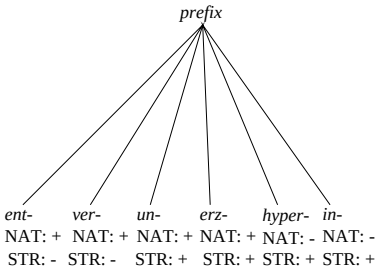


Figure 3.12: Minimal number of nodes

The other obvious measure which could be used is the number of path-value pairs. Figure 3.13 shows two hierarchies which are minimal in this respect. For every lexicon it is possible to find a hierarchy in which every path-value pair occurs exactly once, provided that multiple inheritance is used. One solution is to just create a node for every path-value pair in the lexicon and then have each entry inherit each path-value pair from a different node. The resulting hierarchy will consist of three levels: the root node, an intermediate level with the path-value pair nodes, and the level which contains the lexical entries (e.g. Hierarchy 4 in Figure 3.10, repeated below as Figure 3.13 (right)).

The hierarchy on the left in Figure 3.12 shows the kind of hierarchy returned by Petersen's (2001) algorithm. It differs from the hierarchy on the right in that it captures path-value pair co-occurrence relations of the form:

[x:a] $\rightarrow$ [y:b]

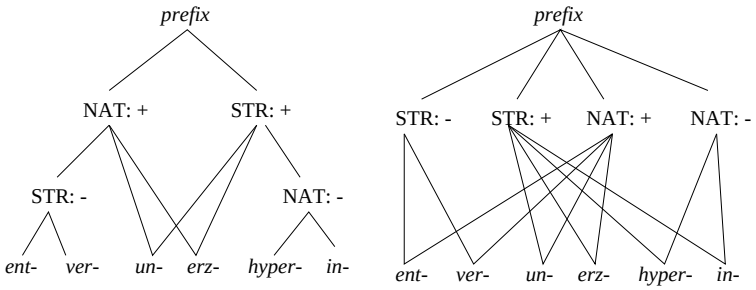


Figure 3.13: Minimal number of path-value pairs

For example, the path-value pair NAT:- implies the presence of the path-value pair STR:+, i.e. all entries that contain the former also contain the latter. In the hierarchy this is represented by the NAT:- node inheriting from the STR:+ node.

While the two hierarchies in Figure 3.13 are minimal with respect to the number of path-value pairs they are not minimal with respect to the number of nodes. The hierarchy in Figure 3.12 contains 7 nodes whereas the hierarchies in Figure 3.13 both contain 11 nodes. This illustrates that there is a trade-off between minimising the number of nodes and minimising the number of path-value pairs. A hierarchy which is minimal with respect to the number of path-value pairs is not normally minimal with respect to the number of nodes as a repetition of path-value pairs often can only be avoided by the introduction of new nodes. Likewise a hierarchy that is minimal with respect to the number of nodes is not normally minimal with respect to the number of path-value pairs. This trade-off arises from a tension between inheriting a path-value pair and specifying it directly at the node as part of its overt intension. This means that there is no obvious definition of "minimality" in the context of lexical inheritance hierarchies. This observation is also supported by the fact that Barg, Petersen, and Light all use different definitions of minimality.

A second problem with minimality-based approaches is that they tend to define their formal criteria in a rather *ad hoc* way (i.e. not empirically determined). They start with a definition of minimal redundancy and then try to prove that this leads to good hierarchies. However, there is no guarantee that an *ad hoc* definition of minimality automatically leads to a linguistically plausible hierarchy. For example, the hierarchies in Figures 3.12 and 3.13 (right) are clearly not linguistically insightful. Neither captures any interesting generalisations (see the discussion on page 72). The hierarchy in Figure 3.13 (left) fares a bit better but the manually constructed hierarchy for the data (Hierarchy 2 in Figure 3.10, repeated below as Figure 3.14) turns out

to be neither minimal with respect to the number of nodes nor with respect to the number of path-value pairs.

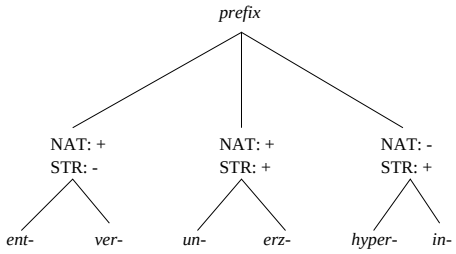


Figure 3.14: Manually built hierarchy

It is possible that a strategy which aims at deriving hierarchies that lie between the two extremes fares best. This is the strategy followed by Light's algorithm and to some extent also by Barg. A simple combined minimality measure would be the sum of the number of nodes and the number of path-value pairs. However, this does not always work either. For example, if one calculates this sum for the hierarchies in Figure 3.10 and then selects the hierarchies which minimise this sum, one would again pick out hierarchies 1 and 4, both of which contain 11 nodes and 4 path-value pairs. Hierarchies 1 and 4 happen to be the two hierarchies which are also be minimal with respect to the number of path-value pairs and at least Hierarchy 4 (shown in Figure 3.13 on the right) is not particularly insightful. The fact that the combined minimality measure picks out the hierarchies which are minimal with respect to the number of path-value pairs is of course to some extent a consequence of fact that there are only two attributes in the prefix toy example, hence minimising the number of path-value pairs does not lead to a big increase in the number of nodes. In a 'real' lexicon the combined measure and the path-value pair minimality measure will not normally lead to the same solution and the combined measure will pick out hierarchies which fall between the two extremes 'minimal number of nodes' and 'minimal number of path-value pairs'. However, while a combined measure like this is probably not a bad idea it has never been investigated whether it does indeed lead to plausible hierarchies or whether there are better combined measures. This reveals a fundamental problem with minimality-based approaches: they start with some notion of minimality and then try to prove that this leads to plausible hierarchies. However, it makes much more sense to go in the other direction by starting with a manually built hierarchy and use it to determine the optimal way to combine a set of formal criteria into one overall criterion of hierarchy quality. This is the basic idea behind the supervised machine learning approach suggested in the next section.

There is a third reason why a minimality-based approach may not be the best one: minimality-based approaches tend to focus on fairly global and superficial criteria, such as the overall number of inheritance links or the average number of path-value pairs per node, while excluding more fine-grained criteria, such as interdependencies between path-value pairs. They also tend to keep the set of criteria small. This is understandable since the set of formal criteria has to be combined by hand into one overall criterion of hierarchy quality. This makes it difficult to accommodate many criteria. However, it is possible that one needs fine-grained criteria to develop a reasonably good model of hierarchy quality and this in turn means that one needs an *automatic* way of combining criteria.

3.5 Learning Architecture

The approach proposed here uses a supervised machine learning technique (Maximum Entropy Modelling) to combine different goodness criteria. This allows a very fine-grained modelling of context. It also means that linguistic knowledge which has gone into the development of manually built hierarchies is re-used to (potentially) discriminate among sound hierarchies such as those in Figure 3.10. On an intuitive level this means that the hierarchy is 'mined' for the linguistically informed decisions that the linguist made to create it.

3.5.1 Search Space

It was mentioned above that there is usually more than one sound inheritance hierarchy for a given input lexicon. One can view the task of automatically building a lexical inheritance hierarchy for an input lexicon as a search through the set of sound hierarchies for that lexicon. One possible solution would be to generate all sound hierarchies, assess each of them and then choose the best one. However it is also possible to view the problem as a search over the set of potential nodes. The task is then reduced to finding the best combination of intermediate (i.e. non-root, non-terminal) nodes. The root node is determined by requiring that the hierarchy is sound (i.e. there has to be a unique root). And the terminal nodes will correspond to the lexical entries. A valid intermediate node is defined as a node which generalises over a set of lexical entries, i.e. the intension of a valid intermediate node is an non-empty intersection of the intensions of at least two lexical entries.[15]

[15]This is a somewhat simplified interpretation of what a "generalisation" is (see Section 4.2). But this simplification is necessary to make the task computationally feasible.

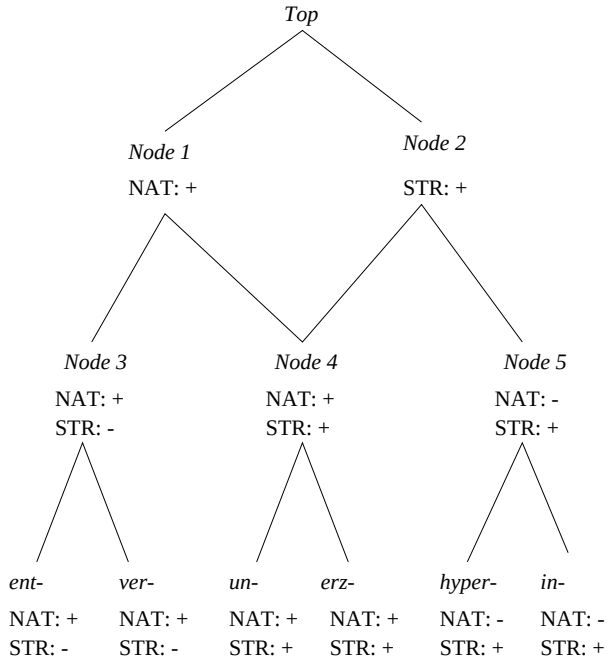


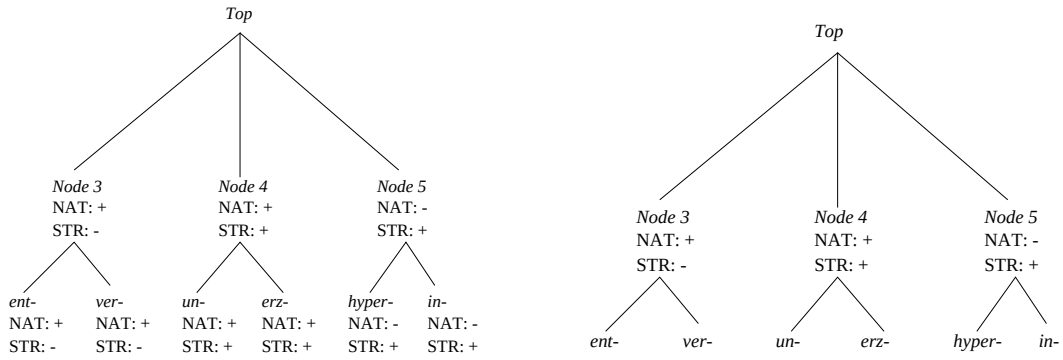
Figure 3.15: Galois (semi-) lattice for the lexicon in Figure 3.9

The set of valid nodes can be partially ordered. Figure 3.15 shows a partial order of all possible nodes for the prefix lexicon (Figure 3.9). Node 3 is an intersection of *ent-* and *ver-*, Node 4 is an intersection of *un-* and *erz-*, Node 1 is an intersection of Node 3 and Node 4 and so on. Formally, the structure in Figure 3.15 is a *Galois* (*semi-*) *lattice* over the prefix data.[16] Galois lattices are discussed in detail in Chapter 4.

Most sound, monotonic hierarchies for an input lexicon can be derived from its Galois latticeby pruning unwanted generalisations and omitting path-value pairs that can be inherited. For example, pruning Node 1 and Node 2 in the above semi-lattice results in the hierarchy in Figure 3.16(a). Omitting path-value pairs which can be inherited from an ancestor results in the hierarchy in Figure 3.16(b), which corresponds to Hierarchy 2 in Figure 3.10. Likewise, Hierarchy 1 in Figure 3.10 can be derived by pruning Node 4 and omitting path-value pairs that can be inherited (Figure 3.17).

Using Galois lattices has a couple of important advantages. First, it guarantees a (nearly) complete search space for monotonic hierarchies. The only hierarchies that are not in the search

[16]The structure is a semi-lattice rather than a lattice because there is no unique greatest lower bound for every compatible pair of nodes. In a Galois lattice, a bottom node would be added to Figure 3.15, ensuring the lattice property. However, the bottom node is irrelevant for the learning algorithm.



(a) Pruning nodes (b) Omitting inheritable AVPs

Figure 3.16: Deriving an inheritance hierarchy from a Galois lattice

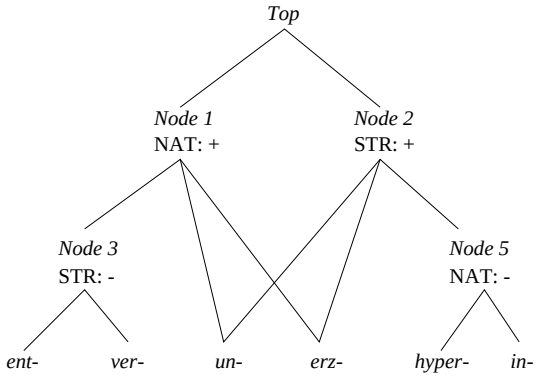


Figure 3.17: Another Derived Hierarchy

space are those that generalise beyond the lexicon and a few hierarchies that differ structurally from Galois lattices (e.g. Hierarchy 4 in Figure 3.10, repeated as Figure 3.14 (right), see Section 4.2 for a detailed discussion). Second, it is relatively easy to guarantee the soundness of the derived hierarchies. Since the lexical entries of the input lexicon occur as terminal nodes in the Galois semi-lattice and each non-terminal node is by definition consistent with its descendants (because it is an intersection of its descendants), soundness is guaranteed provided that (i) no terminal nodes are pruned and (ii) path-value pairs are only omitted if they can be inherited. To ensure that the derived hierarchy is rooted the root node also has to be retained.

3.5.2 Search Strategy

Galois lattices are an elegant way of defining the search space as they avoid the need to enumerate the sound hierarchies for a lexicon. They also reduce the overall complexity of the search space to some extent by restricting the set of intermediate nodes to nodes that generalise over at least two entries. However, it is impossible to search over all node *combinations* in the Galois lattice. One reason for this is that a lexicon with n entries still gives rise to 2^{2^n} sound hierarchies in the worst case. For a very small lexicon searching all combinations, evaluating them and then choosing the one that scores best may still be feasible but for bigger lexicons the search space can easily hold several hundred thousand sound hierarchies. Given that each of these hierarchies may in turn contain hundreds of nodes, a naive search strategy is doomed to failure.

For this reason, previous approaches have employed greedy search strategies. Barg (1996a), for example, restricts the application of her transformation rules and uses an n-best beam search through the space of possible hierarchies. Similarly, Light's (1994) algorithm for inserting new entries into an existing hierarchy is greedy in that it does not evaluate all combinations of possible parents for a new entry but iteratively adds the best parent to the set of parents.

Since searching over all node combinations in the Galois lattice is not feasible one can reduce the search space by making the additional simplifying assumption that the quality of a node depends only on its immediate context. It is then possible to use a greedy search over individual nodes rather than over combinations of nodes. This reduces the worst case complexity for a lexicon with n entries from $O(2^{2^n})$ to $O(2^n)$, i.e. the size of the Galois lattice. While this is still intractable in theory, it is feasible in practice in most cases because the search space will usually be much smaller than the worst case scenario (see Section 4.1.2).

In probabilistic terms, restricting the context means that instead of calculating the condi-

tional probability of a node n_k given all other nodes in a hierarchy with k nodes, one can make a *Markov assumption* and only calculate the probability of a node given l other nodes ($l < k$), where the l other nodes could be the immediate neighbours of n_k, i.e.

$$P(n_k|n_{k-1},\ldots,n_1) \approx P(n_k|n_{k-1},\ldots,n_{k-l})$$

This restriction of context to a fairly local area is quite common in statistical natural language processing. For example, part-of-speech tagging is often done by employing so-called *n-gram models* where the context is restricted to a few words to the left and a few words to the right of the target word. There are two reasons for restricting context. One is data sparseness, e.g. it is usually impossible to get reliable probability estimates for long sequences of words as it is unlikely that these sequences have been seen often enough in the training set. A supervised machine learning system for inheritance hierarchy generation would run into similar problems as there are not enough training hierarchies to reliably estimate the probability of each combination of nodes. However, the second reason for restricting the context is that it also reduces the computational complexity. Furthermore it is often not necessary to take the complete context into account as for many phenomena in natural language processing it is enough to know about a fairly local context. For example, in part-of-speech tagging is not normally necessary to know the tags of all previous words in the sentence to predict the current tag.

Different search directions would be possible. The obvious choices are top-down or bottom-up but theoretically one could also use more complicated heuristics which combine a top-down and/or bottom-up strategy with some kind of horizontal search. It is likely that humans use a complex search strategy when building lexical inheritance hierarchies, i.e. they probably have a more holistic view of the hierarchy which allows them to jump from one part of the hierarchy to another without traversing it along subsumption lines. But even for humans it is likely that a vertical search direction, i.e. top-down or bottom-up, is prevalent. Intuitively, hierarchies are built by either starting with a general concept and then splitting it up into more specific concepts or by starting with a set of specific concepts and gradually working up to a general concept. However, it is difficult to formalise the heuristics used by humans. In addition, trying to emulate human behaviour may not be practical for a computer system. One reason for this is that, while humans can rely to some extent on a graphical representation of the hierarchy, i.e. they can navigate along the two dimensions up/down and left/right, a computer program cannot do this. It has to rely on strategies, such as traversing the hierarchy along inheritance links or keeping track of each node's level and then searching level by level. This makes a more holistic search relatively slow. Since it is likely that humans do rely to a large extent, albeit not

exclusively, on vertical search it makes sense to use a vertical search for the lattice pruning step as well. Humans may combine top-down and bottom-up or may have individual preferences for one or the other. However, for efficiency reasons, I decided to stick with one search direction in the system suggested here. Top-down seems a better choice than bottom-up because that way it is clear which path-value pairs can be inherited by a given node when that node is considered for pruning. This means that one can do node pruning and path-value pair pruning at the same time. In addition, the proportion of path-value pairs that can be inherited by a node may have consequences for the likelihood of pruning this node. Thus, one could argue that a node which is "empty" —i.e. can inherit all its path-value pairs from its ancestors— should always be pruned. For example, if Node 1 and Node 2 are retained in the Galois semi-lattice in Figure 3.15 then Node 4 will be empty as it can inherit all its path-value pairs. There is little reason for retaining an empty node unless one wants to ensure that the derived hierarchy is bounded complete and the node is necessary to ensure this property.[17] However, it is relatively easy, if computationally expensive, to add greatest lower bounds to a hierarchy which is not bounded complete, therefore boundedness is not enforced in the current system.

3.5.3 Classification

Deciding whether a node should be pruned or retained is essentially a classification problem, i.e. it has to be decided whether the node belongs to the class *prune* or to the class *retain*. One could base a classification on simple minimal redundancy criteria like the ones discussed in Section 3.4. However, as has been outlined above, there are a couple of problems with this. First, criteria should not be combined in an *ad hoc* fashion but an empirical approach should be taken to find a good way of combining criteria. Second, it may be that a more fine-grained set of criteria is needed. One that contains local criteria and criteria that pay more attention to the actual data set,[18] e.g. by taking interdependencies between path-value pairs into account.

Both problems can be addressed by using maximum entropy models for the classification. Maximum entropy modelling is discussed in detail in Chapter 5. Basically, a maximum entropy model assigns probabilities to the class membership of an entity. For example, the probability

[17] Of course whether or not a node is empty depends on its parents, i.e. on pruning decisions that have been made further up in the hierarchy. These decisions may have been wrong. Thus if manual post-processing is done on the derived hierarchy and it is not absolutely certain whether a node's parents were retained correctly one may nonetheless want to keep an empty node because the fact that the node can inherit all its path-value pairs may arise from wrongly retaining some of its parents (see Chapter 7).

[18] However, because the training and the test hierarchy will usually differ in their path-value pair sets the criteria cannot refer to actual path-value pairs. Otherwise it would not be possible to generalise from the training data to the test data. The criteria can, however, refer to inter-relations between path-value pairs (see page 113ff).

that a node will be classified as belonging to the class *prune* may be 0.6. A node will be pruned if its pruning probability is above a user-set threshold. The probability of a class depends on the *context* in which a node occurs, modelled by so-called maximum entropy *features*,[19] and the assumed importance of each feature. The importance of a feature is modelled by weights and these weights are set by a parameter estimation algorithm in a supervised machine learning step using pre-classified training data. For the task of automatic inheritance hierarchy construction, training data in the form of pre-classified Galois lattice nodes is required. This is obtained from a manually built hierarchy or a set of these. The hierarchy is first compiled out to retrieve the lexicon. Then a Galois lattice is built for the lexicon. To determine which nodes in the Galois lattice belong to the class *prune* and which belong to the class *retain*, the lattice is matched to the original, manually built hierarchy. Nodes which occur in the hierarchy are classified as *retain* those that do not as *prune*. This is illustrated by Figure 3.18.

Maximum entropy models can easily combine several hundreds or thousands of features, thereby addressing the need for a fine-grained set of criteria. The supervised learning step addresses the need for a non *ad hoc* combination of criteria because the way in which criteria are combined, i.e. their weights, is determined empirically by a parameter estimation algorithm using an existing, manually built, hierarchy. Criteria which are not useful will be assigned a low weight. Hence, it does not do much harm (apart from potentially slowing down the training) to include a criterion even if its benefit is somewhat doubtful.[20] Using a supervised machine learning approach avoids the need to formulate hand-crafted rules about which criteria may be useful and how they should be combined. In effect the system is *re-using* knowledge about linguistic plausibility that has gone into the development of the training hierarchy. Since large inheritance hierarchies are often developed over several years by linguistic experts they are a valuable resource. Supervised machine learning techniques allow one to exploit this resource. Another nice property of maximum entropy modelling is that features do not have to be independent. This is potentially useful since it is likely that there are dependencies between different criteria, as is the case for the criteria discussed in Section 3.4.

[19] Strictly speaking, a maximum entropy feature is a pairing of (partial) context and class. The formal definition will be given in Chapter 5.

[20] Unsuitable criteria also increase the risk of overfitting, so precautions have to be taken to avoid overfitting (see page 184).

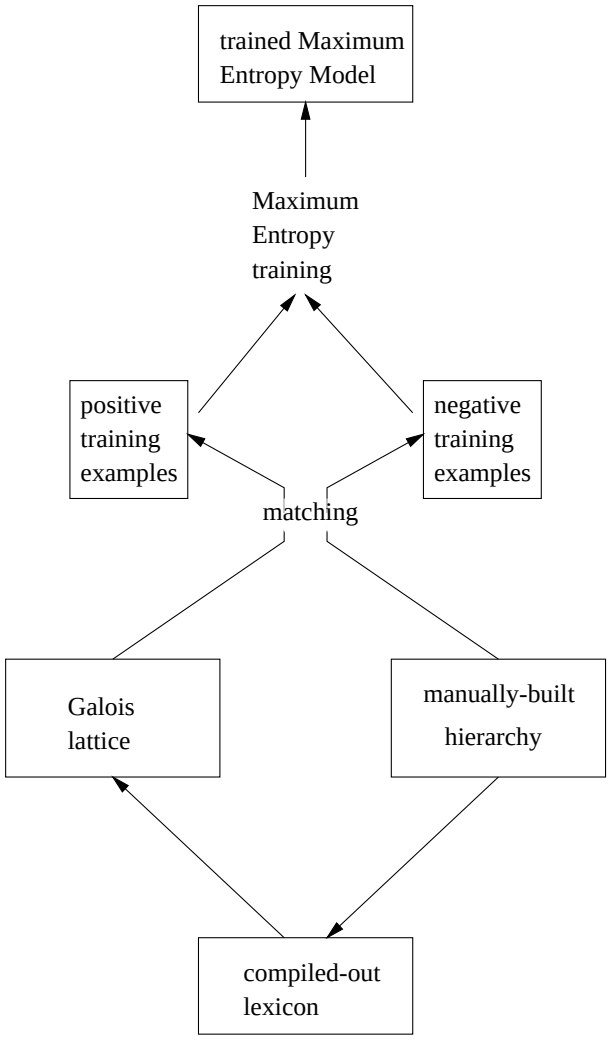


Figure 3.18: Overview of learning architecture

3.6 Summary

This chapter described the learning task and why it is difficult. It also discussed some weaknesses of previous approaches and gave an overview of the learning system proposed in this thesis.

The task is to build an untyped lexical inheritance hierarchy from an untyped input lexicon. The automatically generated hierarchy has to be sound with respect to the input lexicon and the data should be structured reasonably well by it, i.e. the hierarchy should be linguistically plausible.

A major difficulty of the task lies in the fact that it is virtually impossible to define linguistic plausibility in a way that is precise enough to be used directly in a hierarchy construction system. Furthermore linguistic plausibility is to some extent subjective; what is regarded as a good hierarchy is often a matter of taste and is sometimes also influenced by the application for which the hierarchy is used. Therefore the best a system can do automatically is to build a hierarchy which is reasonably plausible. In circumstances where this is not good enough a semi-automatic system is the only way forward.

Given that it is impossible to define linguistic plausibility precisely, previous approaches have relied on formal properties to assess the quality of a given hierarchy. The general focus has been on goodness criteria which measure the "compactness" of a hierarchy, with more compact hierarchies assumed to be better than less compact hierarchies. I argued that a simple definition of compactness, using either the number of nodes or the number of path-value pairs as the criterion, does not work because hierarchies which are minimal with respect to either of these criteria are usually not very good at capturing generalisations and structuring the data well. Furthermore there is a trade-off between these two criteria due to a tension between inheriting a property and listing the property directly at a node. If a hierarchy is compact with respect to one of the criteria it will not be compact with respect to the other. And there is no reason to prefer one kind of compactness over the other.

Manually built hierarchies often occupy the middle ground between these two extremes. Therefore it seems better to combine different criteria. This has been done in most previous approaches. However, the selection of criteria and their combination has usually been fairly *ad hoc*. The system proposed here aims to overcome this problem by making use of a supervised machine learning technique (i.e. maximum entropy modelling). Criteria are selected and automatically combined by analysing a manually built hierarchy. This allows the re-use of information and expertise that has gone into the development of these hierarchies.

Because criteria are combined automatically many more of them can be used than in previous systems where the combination was done manually. This allows for a more fine-grained model, which may fare better when it comes to emulating linguistic plausibility on the basis of formal criteria. It also allows one to take into account interdependencies in the data, such as dependencies between different paths or path-value pairs. It was argued that these interdependencies are often useful clues as they can often indicate the importance of a path-value pair and thereby help to assess the quality of the generalisations captured by a hierarchy.

It was outlined how maximum entropy learning can be combined with Galois lattices and a

simple top-down search strategy to automatically generate lexical inheritance hierarchies. The use of Galois lattices combined with the search strategy guarantees that each derived hierarchy will be sound. If one makes the assumptions that the quality of a hierarchy depends on the quality of its nodes and that the quality of a node can be determined by taking a fairly local node context into account, the worst case complexity of the search can be reduced from $O(2^{2^n})$ to $O(2^n)$ (where n is the number of entries in the input lexicon).

The next chapter discusses Galois lattices in more detail while Chapter 5 discusses the maximum entropy modelling.

Chapter 4

Galois Lattices

Galois lattices (also called *concept lattices*) are a popular tool for manual data exploration. Their use as a data analysis tool is linked to a research area in applied mathematics called *Formal Concept Analysis* (Ganter and Wille, 1999).

In recent years, however, Galois lattices have been used increasingly in automated tasks. Aside from Petersen (2001), who uses them to generate inheritance hierarchies, and Basili *et al.* (1997), who employ them to construct hierarchical subcategorisation lexicons, Galois lattices have been used in various other machine learning and data mining applications, such as rule learning (e.g. Sahami 1995), information retrieval (e.g. Carpineto and Romano 1996, Godin *et al.* 1995b, van der Merwe and Kourie 2001), knowledge acquisition (e.g. Stumme 1996, Yang and Oh 1993, Girard and Ralambondrainy 1997), hypernymy modelling (Priss 1998) and building and maintaining class hierarchies in object-oriented programming languages (Godin *et al.* 1998).

This chapter formally introduces Galois lattices. Section 4.1 discusses Galois lattices in general. Section 4.2 discusses how Galois lattice can be applied to the task of automatic hierarchy construction, and in particular which simplifying assumptions have to be made if hierarchies are derived from Galois lattices.

4.1 Background

Each class in an inheritance hierarchy expresses that a set of path-value pairs (the underlying intension of the class) is shared by a set of entries (the extension of the class). Hence, each class has a set-theoretic interpretation in that it relates pairs of sets to each other. The same idea

underlies Galois lattices. Galois lattices are a partial order over *concepts*, where each concept is a pair of a set of objects and a set of properties shared by these objects. The next section defines Galois lattices formally. Section 4.1.2 debates some complexity issues.

4.1.1 Formal Definitions

Formally, a Galois lattice is a partially ordered set of so-called **formal concepts**. The definition of a formal concept relies crucially on the definition of **formal context** (cf. Ganter and Wille 1998, p. 591f, Ganter and Wille 1999, p. 17f):

Definition 4.1 (Formal Context) *A triple* (G,M,I) *is called* formal context *if* G *and* M *are sets and* $I \subseteq G \times M$ *is a binary relation between* G *and* M. *The elements of* G *are called* objects*, those of* M descriptors*, and* I *is called the* incidence *of the context* (G,M,I). *In order to express that an object* g *is in a relation* I *with a descriptor* m*, we write* gIm *or* $(g,m) \in I$ *and read it as "the object* g has *the descriptor* m*".*

Definition 4.2 (Formal Concept) *A pair* (A,B) *is a* formal concept *of the context* (G,M,I) *if and only if* $A \subseteq G$*,* $B \subseteq M$*,* $B = A'$*, and* $A = B'$*, where:*

$$A' := \{m \in M | (g,m) \in I \text{ for all } g \in A\}$$
$$B' := \{g \in G | (g,m) \in I \text{ for all } m \in B\}$$

A is called the extension *(or* extent*) and B the* intension *(or* intent*) of the concept (A,B).* $\mathfrak{B}(G,M,I)$ *denotes the set of all concepts of the context* (G,M,I).

Transferred to lexicons, the objects of a formal context are the linguistic entities which are described by lexical entries, i.e. the domain of the lexicon (see Section 2.1 for a definition of *domain*). The descriptors of a formal context are the path-value pairs of the lexicon. Thus, the formal context (G,M,I) is the input lexicon where G is the domain of the lexicon and M is the set of path-value pairs that occur in the lexicon. A formal concept (A,B) is then a set of lexical entries A paired with a set of path-value pairs B, where B contains all path-value pairs shared by the entries in A.

The descriptors of a formal context are usually taken to be atomic (**one-valued context**) whereas path-value pairs are not atomic in that they consist of a path and an associated value. Contexts which make use of path-value pair descriptors are called **many-valued contexts**. Table 4.1 shows the prefix lexicon (introduced on page 70, Figure 3.9) as a many-value con-

	ent-	ver-	un-	erz-	hyper-	in-
NAT	+	+	+	+	-	-
STR	-	-	+	+	+	+
ID	1	2	3	4	5	6

Table 4.1: Prefix lexicon as many-valued context

	ent-	ver-	un-	erz-	hyper-	in-
NAT: +	X	X	X	X		
NAT: -					X	X
STR: +			X	X	X	X
STR: -	X	X				
ID: 1	X					
ID: 2		X				
ID: 3			X			
ID: 4				X		
ID: 5					X	
ID: 6						X

Table 4.2: Prefix lexicon as one-valued context

text. An id-number has been added to distinguish between different entries.[1] Formal Concept Analysis deals with many-valued contexts by transforming them into one-valued contexts, i.e. path-value pairs are treated as if they were atomic (Ganter and Wille 1999, p. 36). Table 4.2 shows how the prefix lexicon can be represented as a one-valued context. An "X" in the table indicates the presence of the path-value pair in the lexical entry for the prefix. Such a table is sometimes called a **context table**.

There are 13 formal concepts in the data in Table 4.2:

- $A = \{ent-, ver-, un-, erz-, hyper-, in-\}$, A'=$\{\ \emptyset\ \}$
- $A = \{ent-, ver-, un-, erz-\}$, A'=$\{$NAT:+$\}$
- $A = \{un-, erz-, hyper-, in-\}$, A'=$\{$STR:+$\}$

[1] For the construction of a Galois lattice it is important that every entry is unique in at least one path-value pair as this ensures that the lattice contains an individual node for each lexical entry. In the implementation this is ensured by adding a unique id to every entry.

- $A = \{ent-, ver-\}$, A'={NAT:+, STR:-}
- $A = \{un-, erz-\}$, A'={NAT:+,STR:+}
- $A = \{hyper-, in-\}$, A'={STR:+,NAT:-}
- $A = \{ent-\}$, A'={STR:-, NAT:+, ID:1}
- $A = \{ver-\}$, A'={STR:-, NAT:+, ID:2}
- $A = \{un-\}$, A'={STR:+, NAT:+, ID:3}
- $A = \{erz-\}$, A'={STR:+, NAT:+, ID:4}
- $A = \{hyper-\}$, A'={STR:+, NAT:-, ID:5}
- $A = \{in-\}$, A'={STR:-, NAT:-, ID:6}
- $A = \{\emptyset\}$, A'={NAT:+, NAT:-, STR:+, STR:-}

Note, that the definition of *formal concept* permits A or A' to be empty.

Formal concepts can be partially ordered and this gives rise to the definition of a **Galois lattice** (or *Concept lattice*), cf. (Ganter and Wille, 1999, p. 19f):

Definition 4.3 (Subconcept, Superconcept, Galois lattice) *If (A_1, B_1) and (A_2, B_2) are concepts of a context, (A_1, B_1) is called a* subconcept *of (A_2, B_2), provided that $A_1 \subseteq A_2$ (which is equivalent to $B_2 \subseteq B_1$). In this case, (A_2, B_2) is a* superconcept *of (A_1, B_1), and we write $(A_1, B_1) \leq (A_2, B_2)$. The relation $\leq$ is called the* hierarchical order *of the concepts. The set of all concepts of (G, M, I) ordered in this way is called the* Galois *(or* Concept*)* lattice *of the context (G, M, I).*

The Galois lattice for the data in Table 4.2 is shown in Figure 4.1. The intension of each node is shown in the upper part of each box, the extension is shown in bold face in the lower part. The bottom node of a Galois lattice is not important for the task of deriving lexical inheritance hierarchies since it is guaranteed to have an empty extension, i.e. no lexical entry contains all path-value pairs, and is therefore always pruned (or rather not generated in the first place). I will usually omit it when drawing Galois lattices, in which case the Galois lattice becomes a Galois semi-lattice. However, I will still refer to what is formally a semi-lattice as a "Galois lattice". Likewise, the expression "terminal nodes of the Galois lattice" will refer to

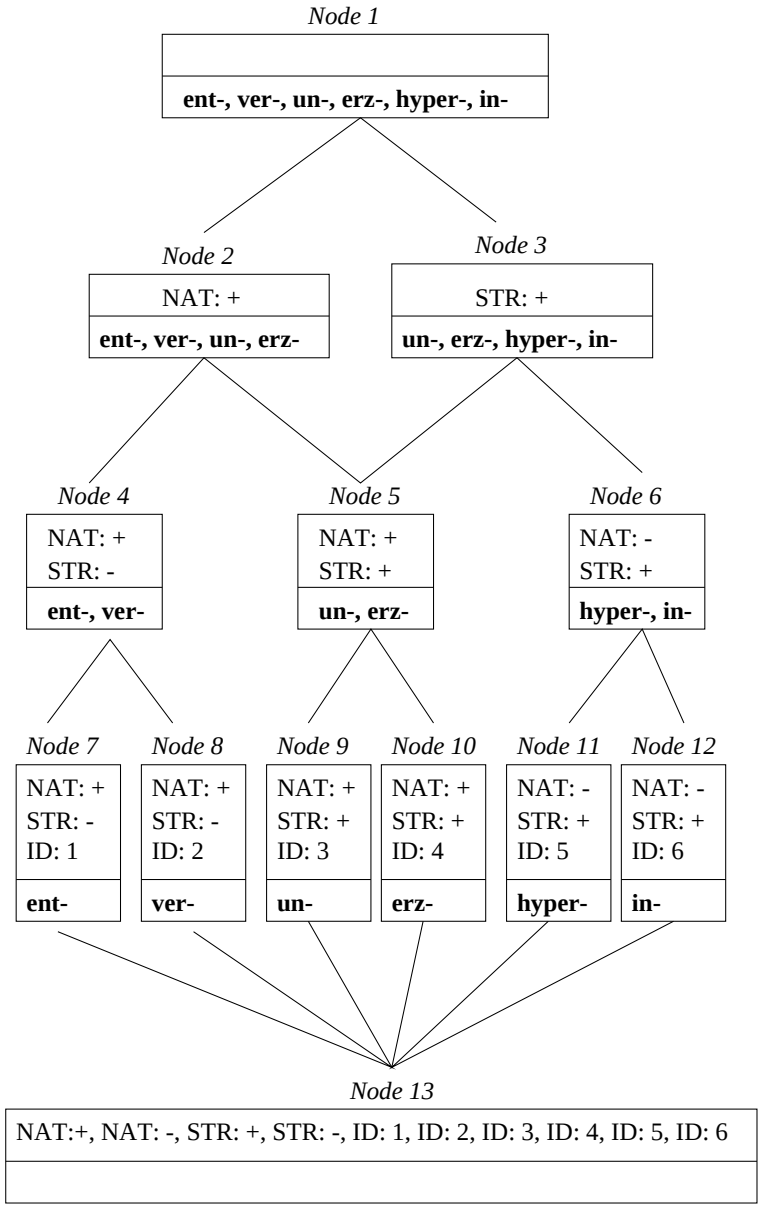


Figure 4.1: Galois lattice

the terminal nodes of the semi-lattice, i.e. the nodes corresponding to the lexical entries (nodes 7 to 12 in Figure 4.1).

From the Galois lattice two further structures can be derived (see Godin *et al.* 1998). A so-called *inheritance concept lattice* can be derived by omitting a descriptor from the (overt)

intension[2] of node *n* if the descriptor also occurs at an ancestor of *n*. Likewise, an object is omitted from the extension of a node if it also occurs in the extension of any of its descendants. In a Galois lattice derived from a lexicon this will usually mean that objects will only occur in the extension sets of the nodes corresponding to the lexical entries. Figure 4.2 shows the inheritance concept lattice that is derived from the Galois lattice in Figure 4.1.

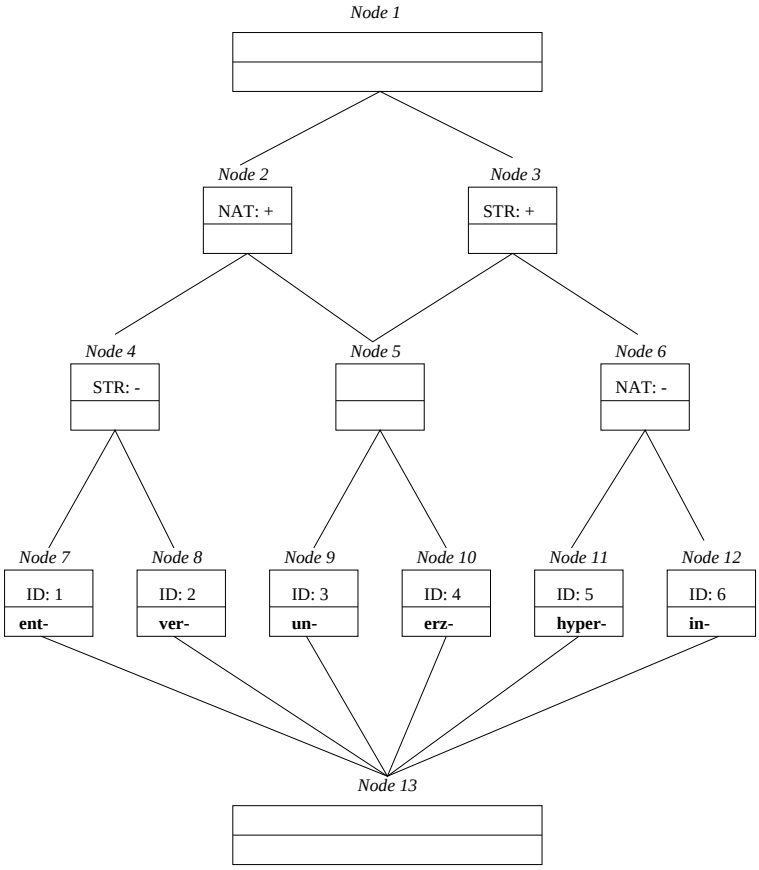


Figure 4.2: Inheritance Concept Lattice

Omitting descriptors and objects can result in nodes with empty overt intensions or empty (overt) extensions. If both the intension and the extension of a node are empty it can be pruned from the lattice. The resulting structure is called *pruned inheritance concept hierarchy*. The lexical inheritance hierarchies automatically derived by Petersen (2001) are effectively pruned inheritance concept lattices. Figure 4.3 shows the pruned inheritance concept hierarchy that can be derived from the lattice in Figure 4.2 (by pruning nodes 1, 5, and 13). Note, that pruning

[2]See Section 2.1 for a definition of the term *overt intension*.

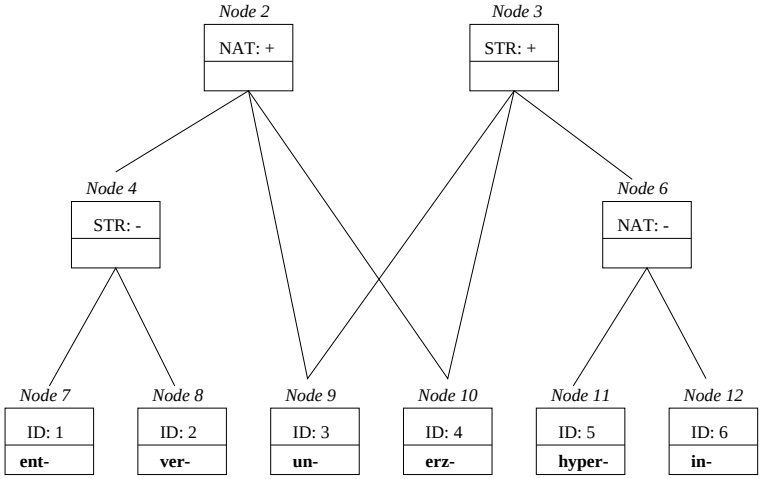


Figure 4.3: Pruned Inheritance Concept Hierarchy

may result in a structure that is no longer a lattice.

In many domains object descriptors are graph-structured, i.e. it is possible to define a partial order over the set of descriptors. This is for example the case if a Galois lattice is built for a typed lexicon because types (and by extension path-value pairs whose values are types) can be partially ordered. To deal with these domains an extension to the above definition of Galois lattice has been proposed which takes ordering relations between descriptors into account (Carpineto and Romano 1996, Godin *et al.* 1998):

Definition 4.4 (Galois lattice, data with partially ordered descriptors) *If* (A_1, B_1) *and* (A_2, B_2) *are concepts of a context,* $(A_1, B_1) \leq (A_2, B_2)$ *if and only if*

$$\forall d_2 \in B_2, \exists d_1 \in B_1, d_1 \leq_{D^*} d_2$$

where $\leq_{D^*}$ *is the partial order relation over the set of descriptors.*

However, as discussed in Chapter 3, I will assume that the input lexicon is untyped.

4.1.2 Complexity Issues

Galois lattices have several nice properties. Their mathematical foundations are fairly well researched and they provide are a very natural way of structuring a domain. Consequently they have been used in several machine learning tasks. However, Galois lattices have the serious drawback that their space and construction time requirements are worst case exponential to

the number of objects in the context, i.e. the worst case complexity is $2^{|G|}$. But this worst case scenario only arises if (i) all objects contain all but one descriptor and (ii) if each object lacks a different descriptor (Kourie and Oosthuizen, 1998). This is hardly ever the case and in practice, Galois lattices are usually much smaller than this, though it is difficult to determine the size of a lattice for a given context without actually building it. Lindig (2000) investigated which context parameters best determine the size of a Galois lattice. He found that $|I|$, i.e. the number of X's in the context table, is the best predictor. For lexicons, $|I|$ can be computed by summing the number of path-value pairs per entry over all entries. Lindig found that for sparse contexts, i.e. contexts in which $|I| \ll (|G| \times |M|)$, the Galois lattice tends to grow quadratically with respect to $|I|$ rather than exponentially with respect to $|G|$.

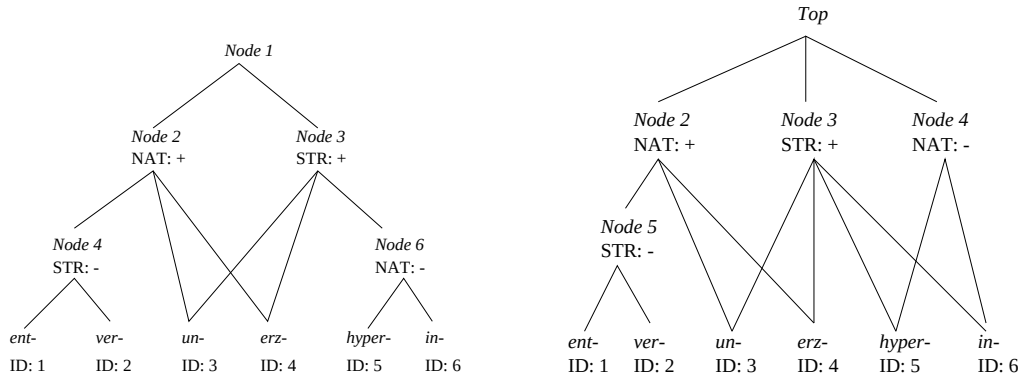
Lexicons tend to have relatively large context tables. In particular the number of descriptors tends to be large since every path-value pair corresponds to one descriptor. Hence, while the attribute set of a lexicon may be fairly small there will be many more attribute paths. Moving from a many-valued context to a one-valued context again adds an order of magnitude to the number of descriptors. For example, the training lexicon used in the experiments in Chapter 7 (i.e. LinGO ERG) contains 21,064 descriptors. However this also means that lexicons are very sparse contexts. For example, the transformation of a many-valued context to a one-valued context not only increases the number of columns in the context table, i.e. one column per path-value pair rather than per path, but it also increases sparseness since a lexical entry cannot contain the same path twice with two conflicting values, e.g. a prefix cannot be both STR:+ and STR:-; at least one of the cells STR:+ or STR:- in the context table has to remain empty. If every path has two values, half of the context table will remain empty. However most paths have more than two values which increases the sparseness even more. Aside from incompatible *path-value pair* combinations there are also incompatible *path* combinations. This follows from the existence of appropriateness conditions for attributes, i.e. some attributes are appropriate for some entries but not for others and by extension some paths are appropriate for some entries but not for others. For example, verbs in English are not marked for case and lexical entries of verbs will thus not contain an attribute CASE while nouns are not marked for tense and their entries will not contain an attribute TENSE. Consequently, the context tables for lexicons are very sparse indeed. Assuming that Lindig's findings are correct, this means that the corresponding Galois lattice will be quadratic with respect to the context size rather than exponential. However, in practice there can still be problems with very big lexicons (see Section 6.3).

There has also been a lot of work on the development of efficient algorithms for the construction of Galois lattices and several authors claim to have developed algorithms that are approximately polynomial (and usually quadratic) with respect to the concepts in the lattice (e.g. Lindig 2000, Njiwoua and Nguifo 1997, Valtchev *et al.* 2000, Kourie and Oosthuizen 1998). There are also incremental algorithms (Godin *et al.*, 1995a).

4.2 Galois Lattices vs. Inheritance Hierarchies

This section compares Galois lattices and inheritance hierarchies and discusses some of the mismatches that occur. Usually, a manually built hierarchy can be derived from the corresponding Galois lattice by pruning unwanted nodes and path-value pairs but sometimes this is not possible, usually because the hierarchy contains one or more nodes that do not have a counterpart in the Galois lattice. The counterpart of a manual node is identified by looking at the node's extension and at its *underlying* intension. A node *n* in a manually built hierarchy does have a counterpart in the lattice if the lattice contains a node n' whose extension is identical to *n*'s extension and whose intension is identical to *n*'s underlying intension. If every node in a manually built hierarchy has a counterpart in the lattice it is possible to derive the hierarchy from the lattice in the suggested way. If the manually built hierarchy contains a node *n* which does not have a counterpart in the lattice it may still be possible to derive a hierarchy which contains a node n' which is superficially similar, i.e. has the same extension and *overt* intension, but from the fact that n' has a different *underlying* intension it follows that its position in the hierarchy must be different; hence the two hierarchies cannot be the same. For example, Figure 4.4 shows two inheritance hierarchies for the prefix lexicon (see Figure 3.9, page 70). The one on the left has been derived from the Galois lattice in Figure 4.1 (page 97), the one on the right has not. Despite this, for every intermediate node in Hierarchy (b) there is a node in Hierarchy (a) which has the same extension and the same *overt* intension. However, *Node 4* in Hierarchy (b) differs from *Node 6* in Hierarchy (a) in its underlying intension. The underlying intension of *Node 6* in (a) is {STR:+, NAT:-} whereas the underlying intension of *Node 4* in (b) is {NAT:-}. While the Galois lattice contains a node with extension {*hyp*-, *in*-} and intension {STR:+,NAT:-} (namely *Node 6*), it does not contain a node with that extension and intension {NAT:-}. This is why the hierarchy on the left is derivable from the lattice but the hierarchy on the right is not.

If one wants to guarantee that a hierarchy is derivable from a lattice, one has to guarantee that each node in the hierarchy has a counterpart in the lattice. Furthermore, whether a node



(a) derived from GL

(b) not derived from GL

Figure 4.4: Two hierarchies

has a counterpart has to be determined on the basis of its extension and underlying intension. In the remainder of this section I will explain why it is not always the case that all nodes in a manual hierarchy have a counterpart in the lattice, i.e. a Galois lattice only *approximates* the search space for the problem, but I will also argue that this approximation is a relatively good one.

By definition each non-terminal node in a manually built hierarchy is a generalisation over the lexicon. Each non-terminal in a Galois lattice also is a generalisations in the sense that its intension is an intersection of the intensions of its descendants. However, some nodes in some manually built hierarchies do not have a counterpart in the corresponding lattice because they are not generalisations in the strict sense (i.e. "intersection between descendants"). In the context of inheritance hierarchies, the term "generalisation" can be defined in at least three ways:

1. **generalisation as intersection:** the generalisation over a set of n entities each of which described by a set s_i of properties is the set s_j, such that $s_j = \bigcap_{i=1}^{n} s_i$
2. **generalisation as subset:** the generalisation of an entity described by a set of properties s_i is a set s_j, such that $s_j \subset s_i$
3. **generalisation as subset of intersection:** the generalisation over a set of n entities each of which described by a set s_i of properties is a set s_j, such that $s_j \subset s_k$ where $s_k = \bigcap_{i=1}^{n} s_i$

Figure 4.5 shows examples of generalisations according to this definition. The hierarchy

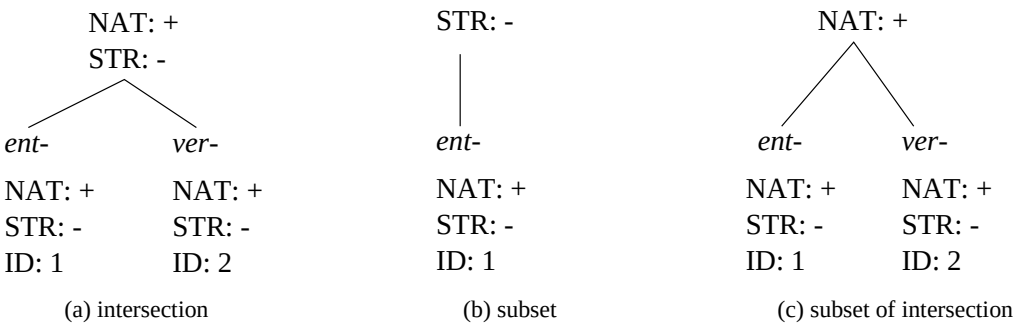


Figure 4.5: Examples of generalisations

fragment on the left contains a generalisation that is obtained by intersecting the entries for the prefixes *ent-* and *ver-*, the fragment in the middle shows a generalisation that is a subset of the set of properties of *ent-*, and the fragment on the right shows a generalisation that is obtained by taken a subset of the intersection between the two entries. In a Galois lattice all non-terminal nodes are generalisations of the first kind. In (monotonic) lexical inheritance hierarchies most generalisation are of the first kind, too, i.e. the underlying intension of most non-terminal nodes are intersections of the underlying intensions of their terminal descendants.

However, some lexical inheritance hierarchies contain generalisations of the second kind. This was already touched upon in Section 3.2 (page 65). Because a lexicon is never complete, it happens sometimes that the developer of a lexical inheritance hierarchy wants to express the existence of a linguistically meaningful class despite the fact that only one member of this class is contained in the lexicon. The way to do this is by having a node which lists the properties of the class and has only one immediate descendant, namely the one member of the class which is actually contained in the lexicon. For example, the type hierarchy supplied with Sag and Wasow (1999) contains a type *comp2-lxm* which covers complementizers that subcategorise for verb phrases, such as *that* and *whether* in example (4.1).

(4.1) a. Peter thought that Tom stole the money.

b. Peter wondered whether Tom stole the money.

However, it is not possible to generalise to the class of *comp2-lxm* complementizers by intersecting a set of lexical entries, as the lexicon only contains one entry belonging to that class of complementizers, namely *that*. A linguist constructing a lexical inheritance hierarchy by hand can draw on his or her background knowledge when inserting a class *comp2-lxm*, because a

linguist will know that there are other complementizers that behave like *that* and that may be added to the lexicon later. For a computer program which has only the lexicon itself to draw on these kinds of generalisations are very difficult to deal with. It is of course possible to *generate* generalisations of this kind, e.g. if for each lexical entry one took the powerset of the set of its attribute-value pairs, one would have the complete set of a generalisations possible under this definition. However, as has been outline in Section 3.2 this is intractable. Furthermore, these kinds of generalisation are virtually impossible to assess, i.e. there is no way of knowing whether such a generalisation is linguistically meaningful as its meaningfulness depends on the existence of lexemes which are not (yet) contained in the lexicon.

The existence of these kinds of generalisations in manually built hierarchies and their non-existence in Galois lattices means that some nodes in a manually built hierarchy cannot be derived from the corresponding Galois lattice. This problem seems to be quite widespread, at least for small and medium size lexicons. For example, in the manually built hierarchies used for the experiments described in Chapter 7 around a third of all non-terminal nodes expressed this kind of generalisation and were therefore not contained in the Galois lattice . While this is a problem, it is a problem for all knowledge-poor approaches to automatic hierarchy construction. It can only be addressed by supplying background knowledge of some form or by doing manual post-processing.

Galois lattice also do not contain generalisations of the third kind, i.e. generalisations which are a subset of an intersection between entries. The reason for this is that Galois lattices only contain *complete* intersections, i.e. if a set E of lexical entries shares a set S of properties, there will be a node in the Galois lattice which has S as its intension and E as its extension, but there will not be a node which has E as its extension and S' ($S' \subset S$) as its intension. In this sense a Galois lattice captures all implications between attribute-value pairs in the lexicon. That is, if a path-value pair A implies a path-value pair B then all Galois lattice nodes that contain A in their intension also contain B. For example, in the prefix lexicon NAT:- implies STR:+. In the Galois lattice for the lexicon (repeated below as Figure 4.6), every node that contains NAT:- (nodes 6, 11, 12, and 13) also contains STR:+. For hierarchies derived from a Galois lattice, like the hierarchies in Figure 4.7, this means that there is either at least one node which contains both A and B in its *overt* intension (e.g. *Node 6* in the hierarchy on the left) or that there is a node which has B in its overt intension (e.g. *Node 6* in the hierarchy on the right) and which inherits from a node which has A in its overt intension (e.g. *Node 3*).

However, lexical inheritance hierarchies do not have to explicitly express all implications.

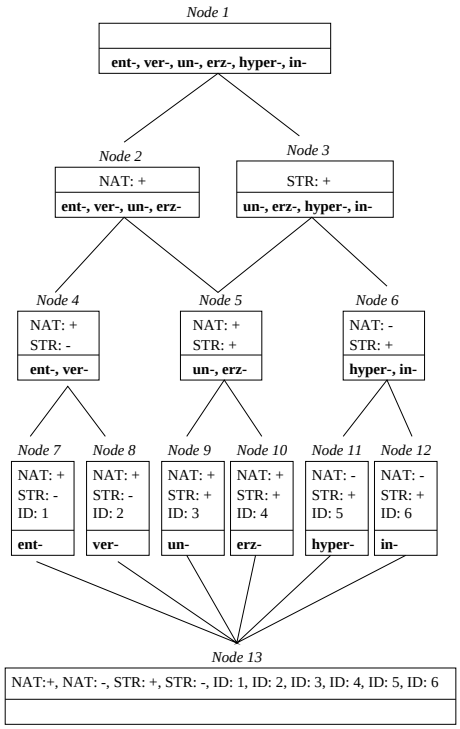


Figure 4.6: Galois lattice

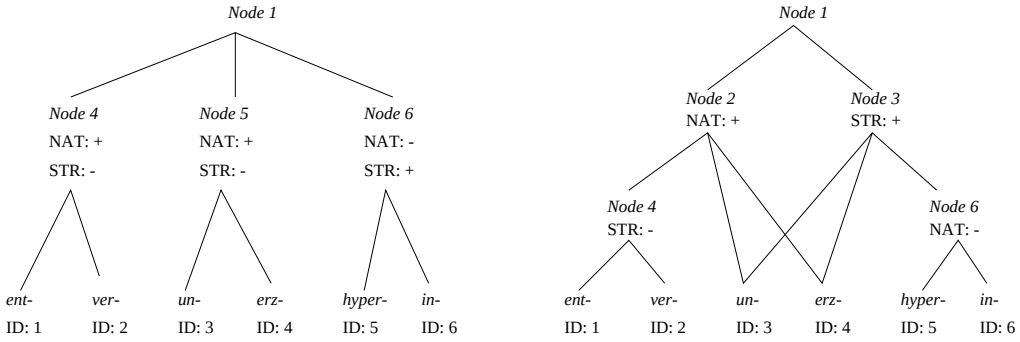


Figure 4.7: Hierarchies derived from the lattice in Figure 4.6

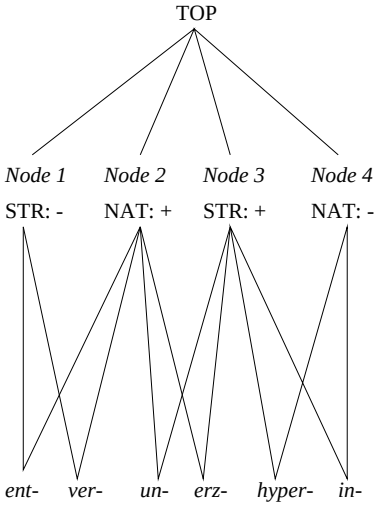


Figure 4.8: A hierarchy which does not capture implications

The hierarchy in Figure 4.8 is a perfectly sound inheritance hierarchy for the prefix lexicon, albeit not particularly meaningful, but it does not express the generalisation between NAT:- and STR:+, i.e. STR:+ is not expressed at the same node as NAT:- nor is it expressed at an ancestor. The two attribute-value pairs are completely independent in the hierarchy. Strictly speaking, none of the nodes in the hierarchy is contained in the lattice. For example, *Node 4* in the hierarchy does have the extension {*hyper-*, *in-*} and the underlying intension {NAT:-}. No such node exists in the lattice. The closest node in the lattice is *Node 6* which has the same extension but an additional attribute in its intension (namely STR:+). Consequently, it is not possible to derive the hierarchy in Figure 4.8 from the lattice in Figure 4.6. The closest one can get is the hierarchy on the right in Figure 4.7.

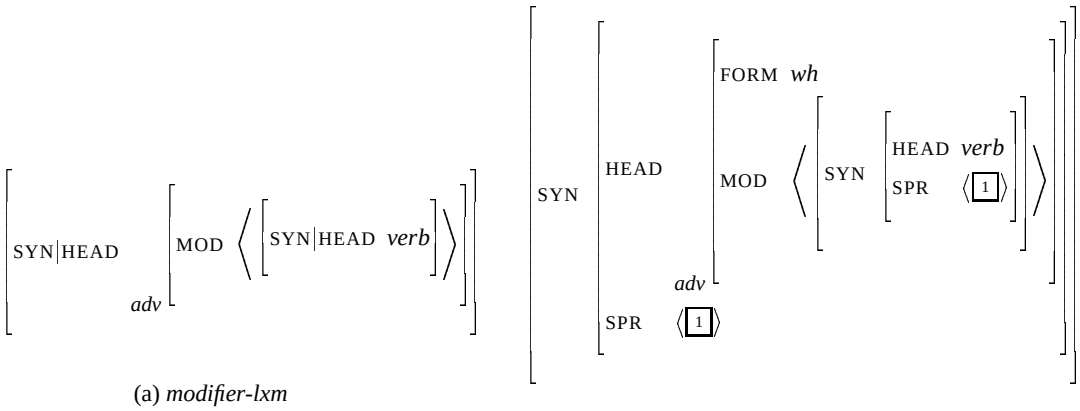
However, it is highly unlikely that a linguist would want to ignore all implications between path-value pairs and represent a lexicon as a bipartite graph since a bipartite representation arguably does not capture any generalisations (see the discussion in Section 3.3, page 72f) and capturing generalisations is a major reason why one would want to represent lexical data as an inheritance hierarchy in the first place. It is however possible that a manually built hierarchy does not represent *all* implications between path-value pairs, either because the linguist building the hierarchy thought that an implication was spurious and deliberately avoided it or because he or she simply was not aware of the implication. One example of this is the separation between syntactic and semantic properties in the Sag & Wasow lexicon (Sag and Wasow 1999). This hierarchy separates types that deal with semantic argument structure from types

that deal with syntactic argument structure (cf. Section 3.2 page 66). Apparently, this happened for reasons of readability. However, a given semantic argument structure often implies a particular syntactic argument structure. For example, the fact that the actor is the first element on the ARG-ST list and the undergoer is the second argument implies that there are indeed two arguments. Splitting semantic from syntactic information means that this implication cannot be captured.

Again it would be possible to define the search space in such a way that generalisations of this kind (i.e. those that are subsets of intersections) are contained in it. One reason for not doing this is the same as for the second type of generalisation: doing this would massively increase the search space and is therefore not feasible. In addition, it seems that this kind of generalisation is not very common in manually built hierarchies; hierarchies usually *do* capture implications and hierarchies which do not, like the one in Figure 4.8, are usually not very meaningful.

One other case of mismatch between lattice and hierarchy came up in the experiments: one manually built type hierarchy (the Spanish lexicon (Quirino Simões, 2001), see Section 6.1) failed to capture a subsumption relation between two types. The two types, *modifier-lxm* and *wh-adv-lxm* are shown in Figure 4.9. As can be seen, *modifier-lxm* subsumes *wh-adv-lxm* but in the Spanish hierarchy there is no path connecting the two types. This means that the objects of *wh-adv-lxm*'s extension are not part of *modifier-lxm*'s extension in the type hierarchy but they are in the lattice because the lattice does contain a path between the two nodes (since preservation of subsumption is guaranteed by the Galois lattice construction algorithm). However, this kind of mismatch again seems to be rare and only occurred once in the experiments and the non-observance of subsumption may well have been accidental in this case.

To sum up, while it is not strictly true that all monotonic inheritance hierarchies can be derived by generating the Galois lattice for the underlying lexicon and then pruning it, this is not a big limitation in practice. Mismatches that arise from the hierarchy's failure to capture implications seem to be quite rare. Mismatches that arise because a hierarchy contains generalisations that are not intersections of lexical entries are more widespread. However, capturing these generalisations is tricky if not impossible unless background knowledge is taken into account. Thus this is a limitation of data-driven techniques in general not of a Galois lattice approach. Adopting the simplifying assumption that monotonic inheritance hierarchies can be derived from Galois lattices therefore seems to be justified in practice.



(a) *modifier-lxm*

(b) *wh-adv-lxm*

Figure 4.9: Two types from the Spanish lexicon

4.3 Summary

This chapter gave an overview of Galois lattices and how they can be used to derive lexical inheritance hierarchies.

A Galois lattice over a lexicon is basically a partial order of all the generalisations that can be made over the lexical data. Hence, an inheritance hierarchy can be derived from a Galois lattice by pruning away unwanted generalisations. However, there are some cases where a manually built hierarchy or a part of such a hierarchy are not contained in the corresponding lattice. The most common of these is where a hierarchy expresses a generalisation that is not contained in the lexicon. These generalisations require background knowledge and therefore pose problems for any data-driven machine learning technique and are not a limitation of Galois lattice based approaches. Apart from this Galois lattices offer a (nearly) complete search space and together with a suitable top-down pruning strategy guarantee that the derived hierarchy is sound. While the worst case complexity of a Galois lattice is exponential to the number of lexical entries, there are indications that the complexity of a Galois lattice which is build over a lexicon will usually be much smaller than this due to the fact that lexicons typically have very sparse context tables.

The next chapter introduces maximum entropy models and discusses how they are used in the system proposed here.

Chapter 5

Maximum Entropy Modelling

The maximum entropy framework has been successfully applied to a range of natural language processing tasks. Ratnaparkhi (1998) discusses its use in sentence boundary detection, part-of-speech tagging, parsing and prepositional phrase attachment detection. Maximum entropy modelling has also been applied to machine translation (Berger *et al.* 1996), stochastic attribute-value grammars (Abney, 1997), chunking (Koeling 2000), text classification (Nigam *et al.*, 1999), named entity recognition (Mikheev *et al.*, 1999; Borthwick *et al.*, 1998), sentence extraction (Osborne, 2002) and dialogue act recognition (Poesio and Mikheev, 1998).

This chapter gives an overview of the maximum entropy framework and discusses how it can be applied to the task of pruning a Galois lattice into an inheritance hierarchy.

5.1 Background

Many machine learning tasks are essentially classification problems, i.e. the aim is to find a classifier which correctly predicts the class of an object given its context. In the case of pruning a Galois lattice the set of objects is the set of intermediate (i.e. non-terminal, non-root) nodes in the lattice. There are two classes: *prune* and *retain*. Statistical approaches assign each object a probability for each class, where the probabilities are determined by the context in which an object occurs. Once probabilities have been assigned, building a classifier is fairly straightforward. One possibility is to simply assign the class which has the highest probability. The following section gives an overview of some crucial ideas in probability theory and introduces maximum entropy modelling.

5.1.1 Formal Definitions

Probability theory deals with modelling how likely it is that a particular *event* will happen, where an event is taken to be the *outcome* of a *trial*. The set Ω of basic outcomes of a trial is called the *sample space*. For example, tossing a coin would be a trial with two possible outcomes: heads or tails (i.e. $\Omega = \{heads, tails\}$). Another trial would be tossing a coin three times and the set of outcomes would then be the set of all possible sequences of length three of heads (H) and tails (T), i.e. TTT, HTT, HHT etc. To be able to discuss trials in a more general form the term *random variable* is introduced. A random variable X ranges over a sample space of k mutually exclusive outcomes.[1] The outcome of a trial is denoted as x_i, where $1 \leq i \leq k$. It is then possible to define a probability distribution over X, which assigns probabilities to individual values of X. The probability of a particular outcome x_i is denoted as $P(X = x_i)$ or simply as $P(x_i)$, where:

$$\sum_{x_i \in \Omega} P(x_i) = 1$$

If a sequence of trials is conducted it is assumed that the observed outcomes are generated by a *stochastic process* which produces outcomes according to some underlying probability distribution.

In machine learning, one has to predict future outcomes on the basis of knowledge about past outcomes, usually without (much) knowledge of the stochastic process itself. Thus the aim is to find a *stochastic model* which fits previously observed data and will therefore hopefully also predict future data with a reasonable degree of accuracy. This task is called *statistical modelling* (Berger *et al.* 1996). Maximum entropy modelling is an example of statistical modelling, i.e. one has to find a maximum entropy solution which correctly models the probability distribution of a random variable X.

The probability of an output value x may be influenced by contextual information y ($y \in Y$). For example, the probability of pruning a node in a Galois lattice may depend on the size of its overt intension (i.e. on the number of path-value pairs that it cannot inherit from an ancestor) or on the number of ancestors or descendants. The maximum entropy model then needs to estimate the *conditional probability* that the outcome of a trial will be x given that the context is y, i.e. the model needs to estimate $P(x|y)$.

Statistical modelling consists of two subtasks: (i) deciding which elements of the context

[1] Strictly speaking, a *random variable* X is taken to be a function which maps the set of outcomes to the set of real numbers: $X : \Omega \to \mathbb{R}$. The idea is that it is mathematically more straightforward to talk about the probabilities of numeric values that are mapped to events rather than to talk about the (potentially complex) events themselves (deGroot and Schervish 2002, p. 97ff). However, this mapping is irrelevant for the discussion here.

are likely to influence the stochastic process (*feature selection*) and (ii) selecting the model which fits the data best by estimating the parameters (*model selection*).[2] Feature selection is discussed in Section 5.2 while the remainder of this Section deals with the maximum entropy framework and model selection.

Maximum entropy modelling is a *supervised learning* method. This means that the learning method makes use of a collection of *training data*, consisting of objects which have been classified beforehand, to select a model. For Galois lattice pruning the training data are obtained from a manually built hierarchy as discussed in Section 3.5.3. There will usually be many (possibly infinitely many) models that fit the data but the *maximum entropy principle* specifies that a model should be chosen which is consistent with the facts in the training data but does not make any further assumptions, i.e. a model which maximises the *entropy* of a random variable. Entropy is a measure of uncertainty and the entropy of a random variable X is defined as follows:[3]

$$H(X) = -\sum_{x \in X} P(x) log_2 P(x)$$

Selecting a model which maximises the entropy of a random variable captures the intuition that a model should model the facts but be otherwise unbiased.

Following Ratnaparkhi (1998), I will assume that contextual evidence is represented by **contextual predicates** and **features**. Contextual predicates (*cp*s) map from the set of possible contexts, denoted by $\mathcal{B}$, to *true* if the object occurs in that particular context or *false* if it does not:

$$cp : \mathcal{B} \rightarrow \{\text{true, false}\}$$

Features map from the set of classes, denoted by $\mathcal{A}$, and contexts to the boolean values 0 or 1:

$$f : \mathcal{A} \times \mathcal{B} \rightarrow \{0, 1\}$$

The general form of a feature is:

$$f_{cp,a'}(a,b) = \begin{cases} 1 & \text{if } a\text{=}a\text{' and } cp(b)\text{=}true \\ 0 & \text{otherwise} \end{cases}$$

[2] The terms have been taken from Berger *et al.* (1996). Instead of *model selection* the term *parameter estimation* may be more appropriate.

[3] For maximum entropy modelling it is usually the conditional entropy of X given the context Y that is maximised. The conditional entropy is defined as:
$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) log_2 p(x|y)$

Thus a feature returns 1 if a class label co-occurs with a particular piece of context and 0 otherwise. If a feature returns 1 it is said to be *active*.

Features are weighted according to their relative importance as estimated from the training data and then combined in an exponential model:

$$p(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

where k is the number of features and $Z(b)$ is the normalisation factor which ensures that $\sum_a p(a|b) = 1$. $Z(b)$ is defined as:

$$Z(b) = \sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

The parameter α_j is the weight of the feature $f_j(a,b)$. Different maximum entropy models differ in their set of weights so finding the best model amounts to finding the best set of weights.

The weight of each feature can be estimated from the training data by a parameter estimation algorithm such as *Generalized Iterative Scaling* (Darroch and Ratcliff, 1972). Once the model has been trained, i.e. the set of weights has been estimated, it can be applied to new data. The probability of an item in the new data set belonging to a class *a* given the context *b* (i.e. p(a|b)) can then be determined by taking the normalised product of the active features for the pair (a,b).

It is important to note, however, that a weight on its own is not interpretable, i.e. if a feature's absolute weight is high this does not necessarily mean that the feature is in itself important to predict the class of a node. The weight may simply be high to counteract the effects of other features. That is, the feature weights have to be taken in combination it is not possible to draw conclusions from isolated weights.

5.1.2 Advantages of Maximum Entropy Modelling

The maximum entropy framework has several nice properties which account for its popularity in machine learning and natural language processing. For a start, the maximum entropy framework does not assume statistical independence between different contextual features. Because the features do not have to be independent, maximum entropy models are very flexible. Since they can also incorporate a large amount of features, maximum entropy models permit a very fine-grained modelling of context. These properties are quite important for the task of automatically building inheritance hierarchies. It was argued in Chapter 3 that a relatively fine-grained

model of node quality is needed. Furthermore, it was shown that redundancy criteria are often not independent (e.g. the number of path-value pairs is not independent from the number of nodes). Ratnaparkhi (1998) also demonstrated that maximum entropy models can achieve fairly good results for several NLP tasks with relatively knowledge-poor features. As will be seen in the next section this is beneficial since maximum entropy features that model node context need to be fairly abstract to ensure that the model can generalise over different lexicons. They also need to be sufficiently general to avoid sparse data problems. A practical advantage of maximum entropy modelling is the existence of several fairly efficient parameter estimation algorithms (cf. Malouf 2002) and the availability of software packages which implement the maximum entropy framework, allowing the user to focus on the implementation of suitable contextual predicates.[4]

5.2 Maximum Entropy Features for Galois Lattice Pruning

The main design decision in maximum entropy modelling lies in choosing the features or more precisely in choosing the contextual predicates within the features. Features are usually selected by hand but there have also been approaches to select them automatically for some learning tasks using pre-defined templates (see e.g. Della Pietra *et al.* 1997).

Once the contextual predicates have been chosen, the weights for the features can be determined automatically from the training data by a parameter estimation algorithm. Features which are not found to be particularly useful in predicting the class of a node will automatically be assigned a low weight. Provided that the training set does not contain too much noise or that a good smoothing technique is used, defining additional, unsuitable features therefore does not do much harm (apart from slowing down the training)[5] because their effects will be filtered out by the low weight. However, a failure to incorporate a feature which is useful will lower the performance of the model. Therefore contextual predicates for inheritance hierarchy learning have been chosen quite liberally. If it was felt that a particular piece of contextual information *might* be useful in determining whether or not a node should be pruned, it was included.

Since the probabilities derived from the training data should generalise across different lexicons, the context of a node has to be defined in terms of fairly abstract properties. In particular, it is not practical to define features which contain a direct reference to specific path-value or

[4] I used a maximum entropy package implemented by Jason Baldridge, Gann Bierner, Thomas Morton and others. This package is available from: `http://maxent.sourceforge.net/` (3.7.03).

[5] Maximum entropy models can, in general, be trained efficiently. The effect of defining a few features more or a few features less is therefore minute.

attribute-value pairs (such as HEAD|POS:*verb*) because the training and test lexicon will usually differ in their attribute-value pairs. Lexicons vary greatly in their number of attribute-value pairs. Of the two English lexicons used in the experiments in Chapter 7, one contained 43 different attributes while the other contained 119. And the smaller attribute-path set is not simply a subset of the larger one. In fact, only 19 of the 43 attributes in the smaller set also occur in the larger set. Consequently, if maximum entropy features referred directly to attribute-value pairs, one would have to map the attribute-value pairs in the test lexicon to the attribute-value pairs in the training lexicon (i.e. to those used by the maximum entropy model).

Sometimes this mapping is one-to-one and relatively trivial. For example, one lexicon may contain an attribute ORTH to encode the orthography of an entry while another lexicon may contain an attribute STEM that more or less serves the same purpose. However, even in this case it is tricky to do the mapping automatically because there is no fixed inventory of names for attributes that encode orthography.[6] For orthography it would be relatively easy to come up with an algorithm to detect which attribute in a lexicon encodes orthography. However, this would not be so easy with many other attributes. One problem which makes this task quite difficult is that the correspondence between attributes is usually not one-to-one but much more complex.

Take the count and mass noun distinction, for example. Count and mass nouns can be distinguished by allowing three values for the attribute NUM: *sg*, *pl*, and *mass*. A sentence like:

(5.1) *A furniture was broken.

is then ruled out by requiring that determiners and nouns agree with respect to their value for NUM. But it is also possible to use a binary attribute NUM which takes the values *sg* and *pl* and introduce an additional boolean attribute COUNT for determiners. The sentence in (5.1) could then be ruled out by requiring that mass nouns subcategorise for specifiers that are COUNT:- and count nouns subcategorise for specifiers that are COUNT:+.[7] Thus to model the count and mass distinction, the first analysis uses one attribute (NUM) with three values whereas the second uses two binary-valued attributes (NUM and COUNT). To map a lexicon that employs the first analysis into a lexicon that employs the second analysis one would have to convert an occurrence of an attribute-value pair into an attribute-value pair and a selectional restriction (Figure 5.1).[8]

Coming up with an algorithm that can automatically discover the relationship between these two analyses and then consistently map one into the other is a major task. It is also

[6] Mapping the values of the ORTH/STEM attributes to each other will, of course, usually be impossible as different

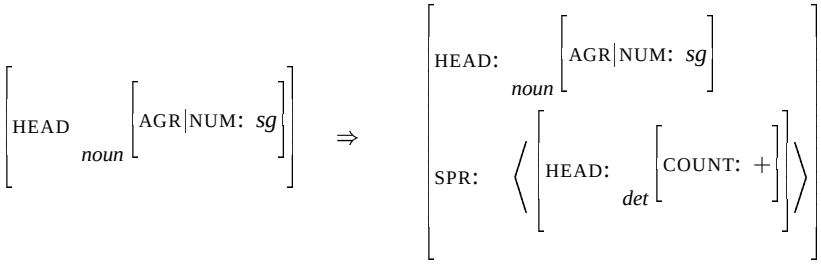


Figure 5.1: Mapping one analysis to another

questionable whether it is possible at all to do this automatically as relating the two analyses to each other requires a large amount of linguistic insight.

The mapping could of course be done manually. However, converting one attribute-value pair set consistently into another attribute-value pair set will usually be quite a bit of work, often amounting to a partial re-implementation of the original lexicon. Furthermore, changing the attribute-value pair set of a lexicon will sometimes involve changing the corresponding hierarchy, too. This is because the attribute set of a lexicon determines to some extent which generalisations one can express, and hence which nodes a hierarchy can contain. Difficulties can arise if a linguistic phenomenon that is modelled by one attribute in one lexicon is modelled by more than one attribute in the other lexicon as the hierarchy for the second lexicon may split up these two attributes and use them at different nodes. This cannot be done with the attribute set of the first lexicon.

For example, assume that the training lexicon models the difference between count and mass nouns with two attributes COUNT and NUM whereas the test lexicon uses a three-valued attribute NUM. To keep the feature structures simple I will assume that the training lexicon implements the attribute COUNT as a head feature on both nouns and determiners and that the grammar contains a constraint that requires a noun and its determiner to have identical COUNT values. The noun part of the training lexicon could then look as shown in Figure 5.2(a), the noun part of the test lexicon as in Figure 5.2(b).

To enable the maximum entropy model to make use of attribute-value pairs one could either map the attribute-value pair set of the training lexicon into that of the test lexicon or the other way round. Consider the first approach first. Changing the attribute-value pair set of the

lexicons are hardly ever identical with respect to their lexical entries.

[7]This example is taken from Sag and Wasow 1999, p. 93f.

[8]There is a problem with mapping nouns that are NUM:*mass* as the first analysis does not distinguish between singular and plural mass nouns. This will be discussed below.

ORTH	POS	COUNT	NUM
chair	noun	+	sg
scissors	noun	+	pl
furniture	noun	-	sg
oats	noun	-	pl

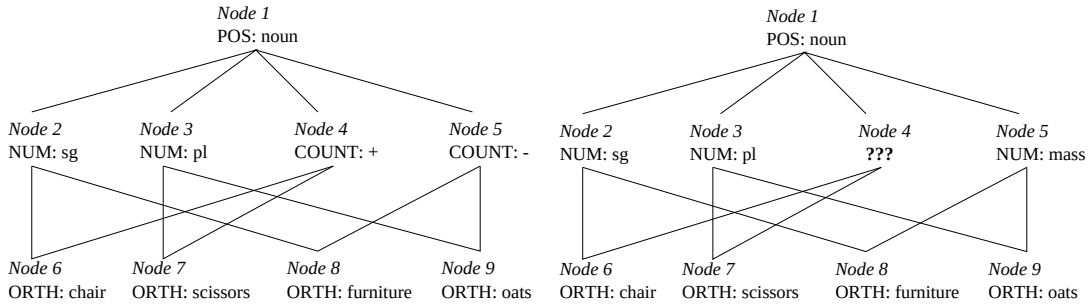
(a) training lexicon

ORTH	POS	NUM
chair	noun	sg
scissors	noun	pl
furniture	noun	mass
rice	noun	mass

(b) test lexicon

Figure 5.2: Different attribute-value pair sets

training lexicon would almost certainly require re-training the maximum entropy model as it may be difficult to change the attributes within the model. This means that the attributes have to be changed in the training hierarchy and the model then has to be re-trained with the new hierarchy. The original training hierarchy is shown in Figure 5.3(a). Converting the attribute-value pairs in the hierarchy works reasonably well for most nodes but fails for *Node 4* as the new attribute-value pair set does not permit a generalisation over count nouns (Figure 5.3(b)). The only solution would be to get rid of *Node 4* altogether.[9] However removing a node may affect the quality of the training hierarchy and altering a hierarchy in this way may not be a good idea.



(a) original

(b) converted

Figure 5.3: Converting the training hierarchy

[9]It is generally possible to remove an intermediate node automatically from a (monotonic) hierarchy without jeopardising the soundness of the hierarchy. Soundness is guaranteed if the overt intensions of the removed node's children are re-computed. Attribute-value pairs specified in the overt intension of the removed node have to be added to the overt intensions of the node's children.

Converting the test lexicon runs into similar problems. The first problem is that the test lexicon does not distinguish between singular and plural mass nouns. That mass nouns are either singular or plural becomes evident when they are combined with a verb:[10]

(5.2) a. The rice is in the bowl.

b. * The rice are in the bowl.

(5.3) a. The oats are in the bowl.

b. * The oats is in the bowl.

One could argue that the analysis employed by the test lexicon is flawed in this respect. However, since the test lexicon only contains singular mass nouns, a distinction between singular and plural mass nouns is not strictly necessary as long as it is ensured that mass nouns only combine with singular verbs. This is not a particularly elegant solution but it is possible and few lexicons are entirely elegant. This is also a case of a difference in linguistic coverage. It is very difficult to find two lexicons which are absolutely identical in the linguistic areas they cover. Even if two lexicons cover more or less the same areas there may be small differences, like one lexicon only containing singular mass nouns and another lexicon containing singular and plural mass nouns. These differences in coverage may result in slightly different analyses of the linguistic phenomena involved which will in turn make the mapping of the different attribute-value pair sets more difficult.

The fact that the test lexicon does not mark mass nouns for number means that it has to be decided on a case by case basis whether a mass noun is singular or plural (Table 5.1). This requires linguistic background knowledge and will usually involve some manual intervention even though it could be done automatically by extracting the number information of each mass noun in the lexicon from a suitable corpus.

Once the test lexicon has been converted, a Galois lattice can be built for it and pruned into an inheritance hierarchy. However, the automatically built hierarchy then has to be converted back into a hierarchy that uses the original attribute set. This may cause problems similar to those encountered in converting the training hierarchy, i.e. it is possible that the new hierarchy contains generalisations that cannot be made with the old attribute set. This would then again involve the removal of nodes from the derived hierarchy, which may affect its quality.

This shows that converting one attribute-value pair set into another is generally not feasible. An automatic conversion is usually ruled-out by the fact that there is no fixed inventory of

[10] Sag and Wasow 1999, p. 94.

ORTH	POS	COUNT	NUM
chair	noun	+	sg
scissors	noun	+	pl
furniture	noun	-	?
rice	noun	-	?

Table 5.1: Converting the test lexicon

attributes and even less so of alternative linguistic analyses. Detecting which attributes in one set would have to be mapped to which attributes in the other set requires linguistic insight and can usually only be done manually. Manual mapping is not only time-consuming it also runs into problems if the mapping is not one-to-one, i.e. if one attribute in one lexicon has to be converted into several attributes in the other lexicon. Since the generalisations one can make over a lexicon depend to some extent on the attribute set, changing the attribute set may change the set of hierarchies that can be built for the lexicon. This may mean that either the training hierarchy or the hierarchy that is automatically generated for the test lexicon has to be changed in a fundamental way because it is not compatible with the new attribute set. Changing a hierarchy to fit the attribute set, however, may have severe consequences for the quality of the hierarchy.

Given that two lexicons will usually differ substantially in their attribute-value pair sets and converting one set into the other is generally not an option, a hierarchy construction algorithm has to generalise over different attribute-value pairs. Consequently, contextual predicates should be kept fairly abstract and not refer to individual attribute-value pairs directly. Otherwise the maximum entropy model will not be able to generalise over different lexicons. Contextual predicates may however refer to properties of attribute-value or path-value pairs or relate path-value pairs to each other within a lexicon, e.g. by measuring the similarity between nodes with respect to their intensions. Using abstract features in the maximum entropy model presupposes that one postulates that linguists, too, base their decisions about whether to include a non-terminal node on abstract properties, like the proportion of attribute-value pairs that a node can or cannot inherit from its parent(s). Such abstract properties are independent of a particular linguistic analysis and particular attribute-value pair set. There are indeed situations where linguistic decisions are made on the basis of abstract —often quantitative— properties. For example, linguists often decide which value of an attribute should be the default by determining

which value occurs most frequently, even though this may be an oversimplification in some cases (see the earlier discussion about minority defaults, page 62).

The contextual predicates that have been implemented are made up of 69 node property *measures*. These are called *measures* rather than *contextual predicates* because the latter are usually assumed to be boolean while many of the measures are real-valued and have to be quantised to be used as contextual predicates. This is discussed in Section 5.2.3. As the inclusion of unimportant measures does not cause much harm but the exclusion of important ones does, measures have often been included because it was felt that they *might* be useful. The measures fall into two broad classes: *intra-node measures* model the properties of a node itself while *inter-node measures* model the relation between a node and its neighbours. However the borderline between these two sets of measures is not always entirely clear-cut.

Path-value pairs that encode a reentrancy between two or more attributes (i.e. those beginning with REENTR) are disregarded by most intra- and inter-node measures. They are only taken into account by measures that deal specifically with reentrancies. Remember, that reentrancies are re-represented in such a way that the two pieces of information, i.e. the value of the reentrancy and the fact that two or more attributes share a value, are split up (see Chapter 3, page 62). The attribute REENTR is used to encode the second piece of information, i.e.:

$$\begin{bmatrix} \\ F & \boxed{1} v1 \\ G & \boxed{1} \\ H & \boxed{2} v2 \\ I & \boxed{2} \\ \\ \end{bmatrix} \Rightarrow \begin{bmatrix} \\ F & v1 \\ G & v1 \\ H & v2 \\ I & v2 \\ \text{REENTR} & \{F,G\} \\ \text{REENTR} & \{H,I\} \\ \\ \end{bmatrix}$$

Thus, the attribute REENTR is different from other attributes in that it encodes a relation between attribute paths rather than equating a path with a value. For measures that do not deal with reentrancies it therefore makes sense to disregard this attribute. For the feature structure above this means that only the first four path-value pairs are taken into account, i.e.:

$$\begin{bmatrix} F & v1 \\ G & v1 \\ H & v2 \\ I & v2 \end{bmatrix}$$

The next two sections describe the measures in detail. Some measures are well-motivated while the motivation for others is a bit weaker. As was mentioned above, some measures were included just because they might help without having a clear-cut justification. Furthermore, it is sometimes difficult to give an exact motivation for a measure because —while it seems likely that the measure will have an effect on the pruning probability of a node— it is not clear in exactly what way. That is, one cannot say "if this measure has a high value then the pruning probability will increase" as it is the *interaction* between different measures that is important (see the discussion about maximum entropy weights on page 112). Therefore, the justifications that are given for the measures are only indications of the way in which they *could* be useful.

5.2.1 Intra-Node Measures

Intra-node measures deal mainly with the overt intension of a node *n*, i.e. with those path-value pairs that a node cannot inherit from any of its ancestors. In general, a node's overt intension is more interesting than its underlying intension since it contains the path-value pairs as they will be represented in the hierarchy. That is, the overt intension contains the generalisation which will be expressed by the hierarchy.

Obviously the overt intension depends on *n*'s parents. It could be calculated with respect to *n*'s positive parents or with respect to all of its parents in the Galois lattice but it is the former that is more interesting. Thus the number of new path-value pairs is calculated with respect to a node's *positive* parents: In the training step parents that are classified as *prune* are ignored when a node's overt intension is calculated. All path-value pairs that cannot be inherited from a parent that is classified as *retain* are part of a nodes overt intension regardless of whether they could be inherited from a parent that is classified as *prune*. For the testing step, pruning is done top-down, so when a node *n* is considered the algorithm has already decided which of its parents should be retained and the overt intension is calculated with respect to the retained parents only. Obviously, defining this measure in this way means that errors at one level of

the lattice may affect the pruning probability of nodes further down. One way to address this problem is by using a semi-automatic pruning method. This is further discussed in Chapter 7.

The following 15 intra-node measures are used in the maximum entropy model:

Number of New Path-Value Pairs (M 1):

• the size of a node's overt intension

The number of path-value pairs in a node's underlying intension indicates how many path-value pairs are shared by its descendants. Each of those path-value pairs is a generalisation over the descendants. However, some of these generalisations may have been captured higher up in the hierarchy and it is, arguably, more interesting to measure how many of the potential generalisations have not yet been captured. This can be done by looking at the size of a node's overt intension. Intuitively, a node that captures many generalisations that have not been captured before should be more likely to be retained than one that does not. Barg uses a related measure as one of her selection criteria, namely the average number of DATR sentences per node.

Proportion of New Path-Value Pairs (M 2):

• the size of a node's overt intension in proportion to its underlying intension

$$\text{M 2} = \frac{|\textit{overt intension}|}{|\textit{underlying intension}|}$$

Note that the denominator of this fraction is guaranteed to be non-zero as all intermediate nodes in the Galois lattice have —by definition— a non-empty intension.

This measure is a variation of the previous one. A low proportion of new path-value pairs means that most generalisations over the node's descendants have been captured before. This should reduce the likelihood of retaining the node.

Contains Reentrancy (M 3):

• whether the overt intension contains a reentrancy

Proportion of Reentrancies (M 4):

• what proportion of path-value pairs is reentrant

$$\text{M 4} = \begin{cases} 0 & \text{if } |\text{overt intension}| = 0 \\ \dfrac{|\textit{reentrant path-value pairs in overt intension}|}{|\textit{overt intension}|} & \text{otherwise} \end{cases}$$

The measure is implemented to return '0' if the overt intension is empty. This is to some extent an arbitrary decision. It would also have been possible to return '1' in this case. The reason for choosing the first alternative was that a situation where all new path-value pairs are reentrancies is intuitively more interesting (or more marked, given that reentrant path-value pairs are usually a minority) than a situation where none of the new path-value pairs are reentrant. Hence the first situation should not be conflated with a situation where the overt intension is empty.

Whether or not a node contains a reentrancy and what proportion of its new path-value pairs is reentrant may be an important cue as to whether or not a node should be retained, even though it is difficult to say whether a high proportion of reentrancies should increase or decrease the pruning probability of a node.

Proportion of Reentrancies with a Specified Value (M 5):

• what proportion of the node's reentrancies has a value specified

The intension of a node can contain a reentrancy without specifying a value for it. Reentrancies in terminal nodes nearly always specify a value. However, reentrancies in non-terminal nodes commonly do not. That is, the fact that two attributes are reentrant is often specified for a whole class of entries (i.e. at a high level in the hierarchy) whereas the actual value of the attribute often depends on a particular entry and is thus only specified at terminal level. Reentrancies without value occur particularly often in connection with subcategorisation constraints.

For example, the node *noun* may specify that a noun and its determiner agree in number by making the NUM value of the noun and the NUM value of its specifier reentrant. But the actual value of NUM depends on the specific noun inheriting from type *noun*, as in Figure 5.4.

The proportion of reentrancies with a specified value may be a clue for deciding whether or not to prune a node. For example, it may be that reentrancies with no value are more important for non-terminal nodes than reentrancies with a value because the former often indicate a generalisation corresponding to a particular subcategorisation pattern.

The measure M 5 calculates the proportion of reentrancies that do have a value specified and is defined as follows:

$$\text{M 5} = \begin{cases} 0 & \text{if } |\text{reentrancies in overt intension}| = 0 \\ \frac{|\textit{reentrancies in overt intension with value}|}{|\textit{reentrancies in overt intension}|} & \text{otherwise} \end{cases}$$

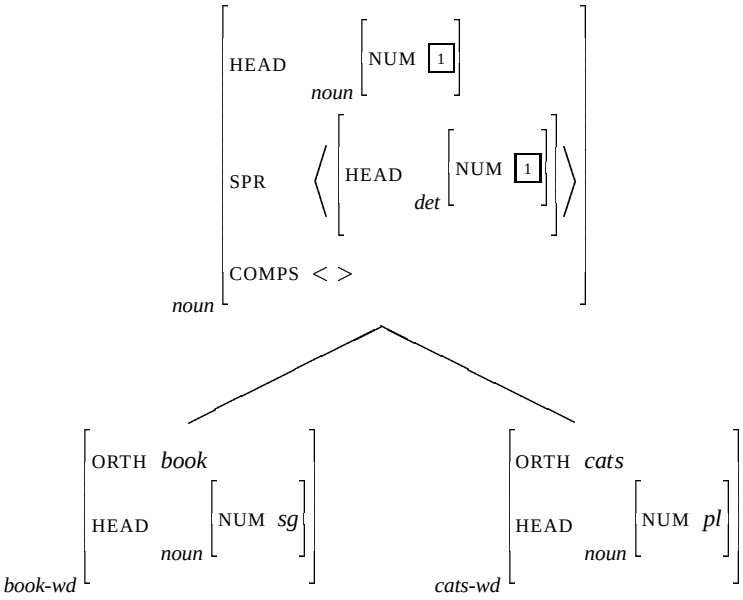


Figure 5.4: Reentrancy with no specified value

Path Prefix Coherence (M 6):

• how similar the beginnings of the path-value pairs in the node's intension are

This measure is calculated by first sorting the attribute-paths in such a way that those which share a common prefix of attributes are adjacent (with the longest path first). Then adjacent path-value pairs are compared and if they share a common prefix the identical attributes in that shared prefix are added up. The overall number of shared attributes in common prefixes is then divided by the possible maximum of shared attributes. Since one cannot equate the same value with two different paths in a node's intension, it is not possible for two attribute-value pairs within a node to have completely identical attribute-paths (i.e. if two path-value pairs have identical prefixes the last attributes in their path have to be different). Consequently, the possible maximum is the number of attributes minus one (i.e. the last one) summed over all path-value pairs except the first one, which does not contribute to the maximum.

An example is given below. The attributes contributing to the numerator of the fraction are given in bold face while the attributes contributing to the denominator are underlined. The path-prefix-coherence for the node is 0.6154.

$$\mathrm{M}\ 6\left(\begin{bmatrix} \mathrm{O}|\mathrm{P}|\mathrm{Q}|\mathrm{R}|\mathrm{X}|\mathrm{Y}|\mathrm{Z} & v1 \\ \underline{\mathbf{O}}|\underline{\mathbf{P}}|\underline{\mathbf{Q}}|\underline{\mathbf{R}}|\underline{\mathrm{S}}|\mathrm{T} & v2 \\ \underline{\mathrm{A}}|\underline{\mathrm{B}}|\mathrm{C} & v3 \\ \underline{\mathbf{A}}|\underline{\mathbf{B}}|\mathrm{D} & v4 \\ \underline{\mathrm{K}}|\underline{\mathrm{L}}|\mathrm{M} & v5 \\ \underline{\mathbf{K}}|\underline{\mathbf{L}}|\mathrm{N} & v6 \end{bmatrix}\right) = 8/13 \approx 0.6154$$

The rationale for this measure is that path-value pairs are often grouped into linguistic areas by their prefixes. For example, path-value pairs that deal with syntactic aspects contain the attribute SYN as their first element while path-value pairs that deal with semantics will be prefixed by SEM. Thus Path Prefix Coherence is essentially a measure of node homogeneity in that it measures whether a node mainly deals with a few linguistic areas or with many. Of course, the correlation between prefixes and linguistic area is not perfect. For example, path-value pairs are often grouped together with a common prefix because they have to be treated in a similar way rather than because they deal with similar areas. Take, for instance, head features, which deal with quite different linguistic areas such as agreement and part-of-speech, and are only bundled together because they need to be shared between a constituent and its head daughter.

The effectiveness of this measure depends on how fine-grained the attribute bundling is, i.e. how long the paths are on average in the lexicon: the longer the paths, the more effective the measure.

Manually constructed lexicons usually make more use of attribute bundling and nest feature structures more deeply than lexicons that were built automatically or semi-automatically from a corpus. But even in automatically extracted data it is possible to introduce some degree of attribute bundling, e.g. by prefixing attributes that deal with a particular linguistic area with an appropriate attribute (such as PHON for phonology, SYN for syntax etc.).

Path-Value Pair Suffix Coherence (M 7):

- how similar the endings of a node's path-value pairs are

This measure is defined in analogy to the previous measure. Two path-suffixes are only considered identical if the values are identical as well. However two path-value pairs sharing identical values without also sharing the final attribute do not count towards a node's path-value pair suffix coherence as value identity without attribute identity is likely to be accidental rather than

linguistically significant, e.g. if two boolean attributes both have the value *false*.

Since identical values contribute to the numerator they also contribute to the denominator. That is the maximum is the number of attributes in a path minus "1" (the first attribute) plus "1" (the value). This is demonstrated by the following example (again bold face indicates attributes which contribute to the numerator, attributes which contribute to the denominator are underlined):

$$\mathrm{M}\ 7\left(\begin{bmatrix} \mathrm{U}|\mathrm{V}|\mathrm{A}|\mathrm{B}|\mathrm{C} & v1 \\ \mathrm{X}|\underline{\mathrm{A}}|\underline{\mathrm{B}}|\underline{\mathrm{C}} & \underline{v2} \\ \mathrm{Y}|\underline{\mathbf{A}}|\underline{\mathbf{B}}|\underline{\mathbf{C}} & \underline{\mathbf{v2}} \\ \mathrm{Q}|\underline{\mathrm{R}}|\underline{\mathrm{S}}|\underline{\mathrm{T}} & \underline{v3} \\ \mathrm{V}|\underline{\mathrm{W}}|\underline{\mathrm{X}}|\underline{\mathbf{T}} & \underline{\mathbf{v3}} \\ \mathrm{H}|\underline{\mathrm{I}}|\underline{\mathrm{J}}|\underline{\mathrm{K}} & \underline{v6} \\ \mathrm{O}|\underline{\mathrm{S}}|\underline{\mathrm{J}}|\underline{\mathrm{L}} & \underline{v6} \end{bmatrix}\right) = 6/24 = 0.25$$

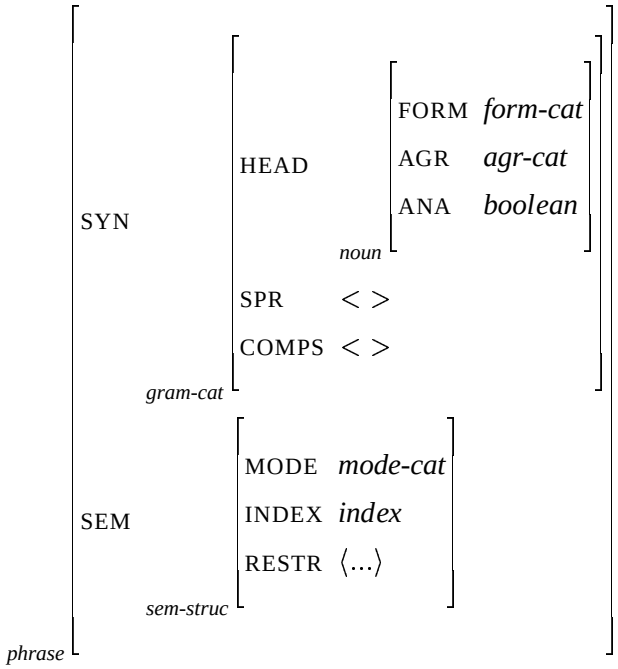
The presence of shared path-value pair suffixes in flat features structures indicates the repetition of embedded feature structures in the original nested feature structure. This is often caused by a reentrancy between two embedded feature structures and may indicate a subcategorisation constraint. For example, object-raising verbs, such as *expect* in:

(5.4) I expected Leslie to be aggressive.

subcategorise for two complements: an object NP (*Leslie*) and a phrasal complement (*to be aggressive*) where the object NP is also the specifier of the phrasal complement. This identity between the object NP and the specifier of the phrasal complement is achieved by making the two feature structures reentrant. Thus, the feature structure of the type *object-raising verb lexeme* (*orv-lxm*) is:

$$_{orv\text{-}lxm}\begin{bmatrix} \mathrm{SPR} & \langle NP \rangle \\ \mathrm{COMPS} & \left\langle \boxed{1}, {}_{phrase}\begin{bmatrix} \mathrm{SPR} & \langle \boxed{1} \rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

The reentrant feature structure may look as follows:



In the flattened representation of the type *orv-lxm* there will be several pairs of path-value pairs with identical suffixes (shown in bold face) e.g.:

COMPS|FIRST|**SYN**|**HEAD**|**FORM**:***form-cat***
COMPS|REST|FIRST|SPR|FIRST|**SYN**|**HEAD**|**FORM**:***form-cat***

However, unlike M 4 (Proportion of Reentrancies) this measure is also sensitive to shared path-value pair suffixes that do not involve a reentrancy. These, too, usually occur in connection with subcategorisation constraints. For example, a transitive verb, which subcategorises for a subject and an object NP, will normally contain two path-value pairs that end in SYN|HEAD:*noun*, one for the subject (i.e. SYN|SPR|FIRST|SYN|HEAD:*noun*) and another one for the object (i.e. SYN|COMPS|FIRST|SYN|HEAD:*noun*), despite the fact that there is no reentrancy between subject and object NP.

Furthermore the measure is sensitive to the proportion of shared attributes, i.e. the length of the shared suffix. For reentrancies the length of the shared suffix sometimes indicates where in an attribute-path the reentrancy occurs: reentrancies that occur early in a path often result in longer repeated path-value pair suffixes than reentrancies that occur late in a path.

Whether this measure contributes much to determining the pruning probability of a node is perhaps questionable but since additional measures do not do much harm under the maximum entropy framework the measure was included.

Path Suffix Coherence (M 8):

• how similar the endings of the node's paths are

This measure is similar to the previous measure but it disregards the value of a path and only takes the attribute-path portion of path-value pairs into account.

An example is given below. The calculation of the maximum is the same as for Path Prefix Coherence. This time the first attribute in two otherwise identical paths has to be different.

$$\text{M 8}\left(\begin{bmatrix} \\ \text{X}|\text{Z}|\text{A}|\text{B}|\text{C} & v1 \\ \text{U}|\underline{\text{V}}|\underline{\mathbf{A}}|\underline{\mathbf{B}}|\underline{\mathbf{C}} & v1 \\ \text{Y}|\underline{\mathbf{A}}|\underline{\mathbf{B}}|\underline{\mathbf{C}} & v2 \\ \text{Q}|\underline{\text{R}}|\underline{\text{S}}|\underline{\text{T}} & v3 \\ \text{V}|\underline{\text{W}}|\underline{\mathbf{S}}|\underline{\mathbf{T}} & v4 \\ \text{H}|\underline{\text{I}}|\underline{\text{J}}|\underline{\text{K}} & v6 \\ \text{O}|\underline{\text{I}}|\underline{\text{J}}|\underline{\text{L}} & v6 \\ \\ \end{bmatrix}\right) = 8/19 \approx 0.4211$$

This measure covers all cases covered by the previous measure, such as reentrancies, but it also takes cases into account where two identical path suffixes differ in their values. This sometimes happens with subcategorisation restrictions. For example a verb is specified as HEAD:*verb* but requires its subject to be HEAD:*noun*. The intension of the verb node will therefore contain two attribute paths that end in HEAD but with two different values. Another example is case selection in languages which mark grammatical function by inflection. For example, a transitive verb may require its subject to be HEAD|CASE:*nom* and its direct object to be HEAD|CASE:*acc*. Again, a high degree of Path Suffix Coherence will often indicate the presence of subcategorisation constraints.

As with the previous measure it remains to be seen whether this measure contributes much to assessing the pruning probability of a node.

Average Mutual Information (M 9):

• the average mutual information between a node and its path-value pairs

The pointwise mutual information between two values x and y of two random variables X and Y measures how much uncertainty about one of them is decreased by knowledge about the other. It is thus essentially a measure of independence between two variables. Mutual information

MI(x,y) is defined as follows:

$$MI(x,y) = log\frac{p(x \wedge y)}{p(x)p(y)}$$

Transferred to path-value pairs and nodes the pointwise mutual information between a node *n* and a path-value pair pvp_i which is contained in *n*'s overt intension is:[11]

$$\begin{aligned} MI(n, pvp_i) &= log\frac{P(n \wedge pvp_i)}{P(n) \times P(pvp_i)} \\ &= log\frac{P(pvp_i|n) \times P(n)}{P(n) \times P(pvp_i)} \\ &= log\frac{P(pvp_i|n)}{P(pvp_i)} \end{aligned}$$

The probability of the path-value pair given the node is 1, thus:

$$MI(n, pvp_i) = log\frac{1}{P(pvp_i)}$$

The probability of a path-value pair is calculated from its frequency in the (compiled out) lexicon, i.e. it is defined as the number of times it occurs in the intension of a lexical entry in the lexicon divided by the number of lexical entries.

The average mutual information between a node and its overt intension of size *k* is then defined as:

$$\text{M } 9 = \sum_{i=1}^{k} \frac{1}{k}\, log\frac{1}{P(pvp_i)}$$

The mutual information between a node and a path-value pair is high if the path-value pair has a low probability, i.e. only occurs in a few entries. Average mutual information between a node and its path-value pairs is thus a measure of the specificity of a node: a node whose intension contains a lot of general path-value pairs that occur in many entries will have a low Average Mutual Information. Intuitively, a low Average Mutual Information should increase the pruning probability of a node.

Variance of Average Mutual Information (M 10):

• the variance of the average mutual information

The mutual information between a node and a path-value pair may vary for different path-value pairs and the degree of this variation may be important. Therefore the variance of the mutual

[11] It has been suggested that mutual information is not very reliable for low counts as it tends to overestimate the significance of rare events (see Dunning 1994). This means that mutual information may not be the ideal choice here.

information is implemented as an additional measure:

$$\text{M } 10 = \frac{\sum_{i=1}^{k}(MI(n, pvp_i) - \text{M } 9)}{k}$$

Maximal Mutual Information (M 11):

• the maximal mutual information for any path-value pair

It is possible that a very high mutual information between a node and one of the path-value pairs in its overt intension is a good indicator that the node should be retained. However, a high mutual information with one path-value pair might be "watered down" by low mutual information with other path-value pairs. Thus Maximal Mutual Information has been defined as an additional measure that captures the maximal value for mutual information:

$$\text{M } 11 = \underset{pvp_i}{\text{argmax}}\, MI(n, pvp_i)$$

Average Path-Value Pair Dependency (M 12):

• what proportion of a node's path-value pairs are in a dependency relation to each other[12]

Four different kinds of path-value pair dependency have been defined. These are discussed below. Whether there is a dependency between paths or path-value pairs is determined by looking at the (compiled out) input lexicon. This is done before the pruning is started. When considering a node in the Galois lattice for pruning, the proportion of the path-value pairs in its overt intension which are in some kind of dependency relation is computed.

The four kinds of dependencies are:

- **mutual path-value pair dependency:** one path-value pair implies (i.e. always co-occurs with) another path-value pair in the input lexicon:
  [A:x] $\leftrightarrow$ [B:y]
  That is, every entry in the input lexicon that contains the path-value pair [A:x] also contains the path-value pair [B:y] and *vice versa*. For this dependency to hold it is by definition enough if one value of A co-occurs with one value of B. It is not necessary that all values of A enter into a dependency relation with a value of B.[13]

[12] This measure is defined with respect to path-value pairs rather than attribute-value pairs. This allows a more fine-grained model of dependency. Thus if two attribute-value pairs are dependent all paths which end in these attribute-value pairs will also be dependent. However it is possible that two attribute-value pairs are only dependent if they are prefixed by a certain path. Defining dependency with respect to path-value pairs allows one to capture this kind of dependency as well.

[13] A relation where each value of one attribute implies a particular value for the other and *vice versa* is also interesting but this is a special case of the fourth dependency measure (path-value pair set dependency), which is discussed below.

Example: It may be that suffixes in a particular language are stressed if they are non-native and unstressed otherwise, so that:[14]
[NAT:+] ↔ [STR:-] and [NAT:-] ↔ [STR:+]

- **path-value pair attribute dependency:** a path-value pair implies an attribute path:
  [A:*x*] ↔ [B:*any value*]
  Every entry that contains the path-value pair [A:*x*] also contains the (complete) path B and *vice versa*.
  Example: The presence of this dependency often indicates an appropriateness condition. Thus the fact that the attribute CASE is appropriate for a feature structure of type *noun* will show up as [HEAD|TYPE:*noun*] ↔ [HEAD|CASE:*any value*]

- **mutual path dependency:** a path A implies another path B:
  [A:*any value*] ↔ [B:*any value*]
  Every entry that contains the (complete) path A also contains the (complete) path B and *vice versa*. This dependency often indicates the existence of a third path-value pair, C:*z* which "causes" the presence of both A and B, i.e.:
  (C:*z* ↔ A:*any value*) ∧ (C:*z* ↔ B:*any value*)
  Example: Nouns are often not only marked for case but also for whether or not they are anaphoric (ANA:+/-). Thus reflexive pronouns like *themselves* are ANA:+, non-reflexive pronouns and ordinary, non-pronominal nouns are ANA:-.[15] Thus, the path HEAD|CASE will always co-occur with the path HEAD|ANA.

- **path value pair value set dependency:** each value of an attribute-path A implies that the value of attribute-path B is taken from a particular subset of B's values. The value subsets do not overlap (i.e. no value of B occurs with more than one value of A) and exhaust the complete value set for B:
  $\forall x_i \in X, v_j \in V'$ ([A:x_i] ↔ [B:v_j])
  where X is the set of all values appropriate for A and V' is the set of k mutually exclusive subsets of the set V of values that are appropriate for B.
  That is, every entry that contains the (complete) path A with a value x_i also contains the complete path B with a value v_j where v_j is drawn from the subset of values of B which

[14] Note, the dependency between non-native prefixes and stress in the prefix data discussed in Chapter 3 is not an example of mutual path-value pair dependency since:[NAT:-] → [STR:+] but [STR:+] ↛ [NAT:-].
[15] Cf. Sag and Wasow 1999, p. 149f.

co-occur with the path-value pair [A:x_i] and *vice versa*.

Example: There may be an attribute INFL which is appropriate for both nouns and verbs but for nouns the value of INFL is selected from the subset of nominal inflectional endings whereas for verbs it is selected from the subset of verbal inflectional endings. In that case:
[HEAD|POS:*noun*] $\leftrightarrow$ ([INFL:*noun-ending1*] $\vee$ [INFL:*noun-ending2*] $\vee \ldots$)
and:
[HEAD|POS:*verb*] $\leftrightarrow$ ([INFL:*verb-ending1*] $\vee$ [INFL:*verb-ending2*] $\vee \ldots$)

Nodes usually have more than two path-value pairs. While it would be possible to extend the dependency definitions to sets of attribute-value pairs this would considerably increase computational complexity as dependencies between subsets would have to be determined. Therefore only pairwise dependency is calculated rather than overall dependency. Different kinds of dependency are not distinguished when it comes to calculating the proportion of dependent path-value pairs because, intuitively, it is the presence of some kind dependency between path-value pairs rather than the presence of a particular kind of dependency that will influence the pruning probability of a node. Thus the proportion of path-value pair dependency for a node is defined as the number of pairs of path-value pairs that enter in any kind of dependency divided by the overall number of pairs of path-value pairs in the overt intension. If the node's overt intension contains *k* path-value pairs the number of pairs of path-value pairs is:

$$\left(\begin{array}{c} k \\ 2 \end{array} \right) = \frac{k!}{2!(k-2)!} = \frac{k!}{2(k-2)!}$$

The Average Path Value Pair Dependency is then:

$$\text{M } 12 = |\textit{dependent pvp pairs}| \times \frac{2(k-2)!}{k!}$$

Intuitively, the presence of interdependencies between attributes of a node *n* should increase the probability of *n* being retained. An interdependence may also hint at the relative importance of the involved attribute paths. For example, it can be argued that the attribute path HEAD|TYPE is relatively important because it encodes part-of-speech information and this in turn determines the presence of other paths (e.g. HEAD|CASE) and/or other path-value pairs (e.g. INFL:*noun-ending1*). While causality cannot be captured directly, the above dependency measures help to capture it indirectly.

Relative level of node (M 13):

• the relative level of the node in the lattice

The level of a node in a lattice is defined as the level of its highest immediate descendant plus 1, i.e. if $D(n)$ is defined to be the set of immediate descendants of a node *n* and *Terminals* is defined to be the set of terminal nodes then:

$$level(n) = \begin{cases} 0 & \text{if } n \in Terminals \\ (\underset{d \in D(n)}{\text{argmax}}\ \text{level(d)}) + 1 & \text{otherwise} \end{cases}$$

The relative level of a node is then defined to be the absolute level divided by the number of levels that the lattice contains.

Information about the relative level of a node is interesting because it seems to be the case that most nodes in a manual hierarchy have a relatively low level in the corresponding Galois lattice. That is, nodes at high levels in the lattice occur only infrequently in manual hierarchies. Hence, if the relative level of a node is high the pruning probability should increase.

It seems that linguists are more interested in relatively low level generalisations than in high level generalisations. This could be due to the fact that low level generalisations, i.e. generalisations over a fairly small proportion of lexical entries, are linguistically more meaningful but to some extent it could also be due to cognitive constraints, i.e. humans find it more difficult to generalise over a set of heterogeneous entities than over a set of relatively homogeneous entities.

Overt Intension + Parents (M 14):

• a variation of Light's measure

This measure[16] implements the goodness criterion that has been used by Light (1994) to decide where in an existing hierarchy a new lexical entry *n* should be inserted. Light assumes that the best insertion place is the one that minimises the following sum:

$$\text{M 14} = |\textit{overt intension of } n| + |\textit{parents of } n|$$

Light uses this criterion in a different context in that he tries to determine the best set of parents for a terminal node while the maximum entropy modelling aims at determining how likely it is that a node should be pruned. However this measure might still be useful.

Node Globally Introduces Path-Value Pair (M 15):

• Petersen's measure

[16] This measure is a case where it is difficult to distinguish between inter- and intra-node measures, i.e. one could as well group it under the latter category.

Petersen (2001) only retains those nodes in the Galois lattice that introduce new path-value pairs, i.e. whose intension contains path-value pairs that cannot be inherited from any of their ancestors in the lattice. This measure implements Petersen's criterion. It is boolean and returns *true* if a node contains path-value pairs that it cannot inherit from any of its ancestors in the Galois lattice and *false* otherwise. Note that unlike other measures in this section, this measure is defined in relation to all nodes in the Galois lattice not only in relation to the positive nodes.

5.2.2 Inter-Node Measures

Inter-node measures compute the relations between a node *n* and a set of other nodes or the relations within a set of other nodes that are all in the same relation to *n*. Since it is computationally too expensive to consider *n*'s relation to all other nodes, only a fairly local context is taken into account. This context consists of:

- *n*'s parents
- *n*'s children
- *n*'s siblings, i.e. nodes that share a parent with *n*
- *n*'s co-parents, i.e. nodes that share a child with *n*

These are probably the kinds of nodes which are most useful for determining the pruning probability of *n*. Looking at parents is important because they determine *n*'s overt intension. For example, if most path-value pairs in *n*'s underlying intension can be inherited from its parents then *n*'s overt intension will be relatively small and this may make it more likely that *n* should be pruned. Looking at children is motivated for similar reasons: if *n* is very similar to its children, pruning *n* will be less harmful because there is still a chance of keeping its children which express similar concepts. Siblings and co-parents give some indication of the presence or absence of a *dimension* (see page 31 for a definition) in this area of the hierarchy. If *n* and (most of) its siblings contain an attribute *A* but differ in their values for this attribute, this may indicate that *A* represents such a dimension. For example, a the node *verb* in a hierarchy may have several children each belonging to a different inflectional class, with the intension of each child containing an attribute INFL which states their inflectional class. The children together then partition the dimension VERBAL INFLECTION and this should reduce the pruning probability of each individual child. Co-parents can also give cues about *orthogonality* (see page 34).

It is possible that nodes other than these would also provide some cues but it is unlikely that *all* other nodes do so. For example, it is not very probable that grand-parents or grandchildren provide much information about the pruning probability of a given node. Furthermore, as mentioned before (cf. page 86), using a larger context renders the approach intractable and is also not advisable because the probabilistic modelling would soon run into sparse data problems.

Inter-node context can be defined with respect to the manual hierarchy or with respect to the Galois lattice. Defining it with respect to the manual hierarchy means that only positive nodes (nodes labelled as *retain*) are taken into account when computing *n*'s parents etc. The Galois lattice in Figure 5.5 illustrates this point. Positive nodes are shown as filled circles, negative nodes as unfilled circles. If inter-node relations for *Node 10* are defined with respect to the lattice there are three parents (nodes 5, 6 and 7), three children (nodes 15, 16 and 17), three siblings (nodes 9, 11 and 12) and three co-parents (nodes 9, 11 and 8).[17] If, however, inter-node relations are defined on the basis of the manual hierarchy there are two parents (nodes 5 and 4),[18] two children (nodes 16 and 17), one sibling (node 8) and one co-parent (node 8).

Computing context purely with respect to positive nodes works well for training the model but when the model is applied to the Galois lattice of a new lexicon it has the disadvantage that not all positive nodes are known when a node *n* is considered for pruning. However, since pruning is done top-down all nodes at levels above *n*'s level have already been classified when *n* is considered (see page 120). For example, when *Node 10* in the hierarchy in Figure 5.5 is considered the classifications of all nodes at levels $k+1$ to $k+4$ are known. Thus it is possible to determine positive parents (assuming the previous classifications have been correct) but it is not possible to determine positive children or positive co-parents and it may or may not be possible to determine positive siblings, depending on whether or not all potential siblings have already been classified.

On the other hand, computing context purely with respect to the Galois lattice has the serious drawback that it increases the computational complexity of the modelling since there are many more inter-node relations if positive and negative nodes are taken into account. In addition to complexity issues, a context that is defined with respect to all nodes is also less

[17] An alternative would be not to put all co-parents into one set but to split them into different sets according to the children by virtue of which they are co-parents. Thus for *Node 10* there would be two sets: the first would contain *Node 9* which is a co-parent for *Node 15*, the second would contain *Node 11* and *Node 8* which are co-parents for *Node 17*. While this may be a more accurate way to model inter-node relations it also increases the computational complexity of the maximum entropy modelling and thus has not been adopted here.

[18] Pruning the negative *Node 7* results in a new link between *Node 4* and *Node 10*. Thus *Node 4* is a parent of *Node 10* in the manual hierarchy. *Node 3* on the other hand is not a parent because no link is added between it and *Node 10* when *Node 6* is pruned since *Node 10* already inherits all of *Node 3*'s properties via *Node 5*. Consequently a link between *Node 10* and *Node 3* would be redundant.

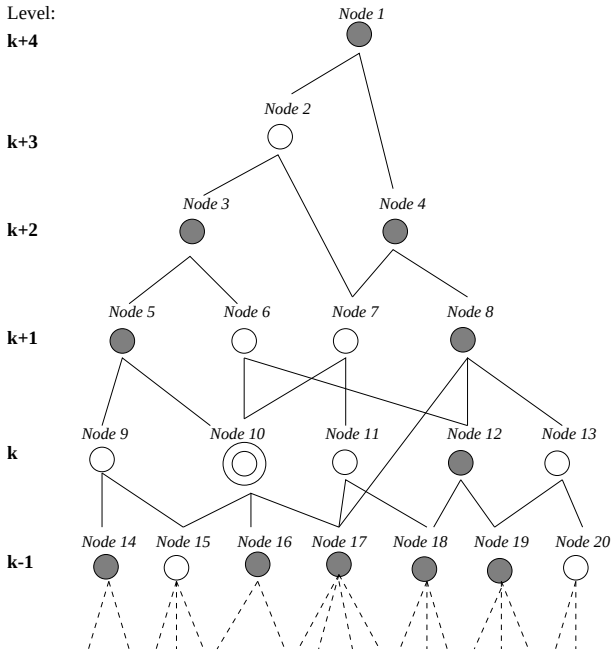


Figure 5.5: Inter-Node Relations

interesting than a context that is only defined with respect to positive nodes since the former is wholly determined by interdependencies between path-value pairs in the lexicon. For example, the number of parents of a node N in a Galois lattice is determined by the independence of the path-value pairs in N's intension since the existence of a parent A_i means that there is a sibling node S_j which overlaps with N in its intension (Figure 5.6, where $k-p$ are path-value pairs). The more parents there are the more nodes S_j exist with which N overlaps. Since all nodes A_i differ in their intensions, N has to overlap with all S_j's in different ways, i.e. for each S_j there must be one path-value pair which N shares with S_j but not with any of the other S's. For example, N shares the path-value pair k with $S1$ but not with $S2$ and $S3$. Likewise N shares l with $S2$ but not with any of its other siblings. This indicates that N's path-value pairs are fairly independent: they do not always co-occur in the lexicon but they freely co-occur with other path-value pairs that are not part of N's intension. For example, k does not always co-occur with l and m (as in N) but it also occurs with n (as in $S1$), l occurs with o as well as with k and m etc. The degree of independence between the path-value pairs of a node may be a useful piece of information in determining whether the node should be pruned but it is already covered by one of the intra-node measures (i.e. Average Path-Value Pair Dependency). Thus defining parenthood with respect to all nodes does not seem to add much information beyond what is

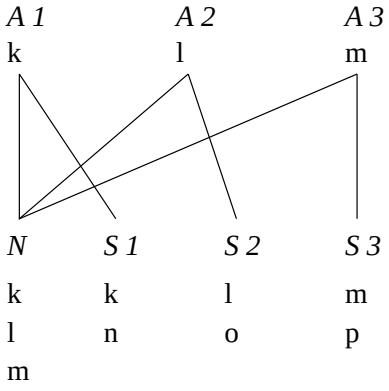


Figure 5.6: Parents in a Galois Lattice

already captured by intra-node measures and the same is true for other inter-node relations if they are defined with respect to the lattice.

Given that both of the above definitions of inter-node relations have disadvantages, a hybrid approach has been taken in which classifications which have already been made (i.e. classifications at a higher level) are taken into account but classifications at the current or a lower level are not. Thus if *Node 10* in Figure 5.5 is the first node to be classified at level $n+1$ this means that there are two parents (nodes 5 and 4), three children (nodes 15, 16 and 17), two siblings (node 9 and 8) and three co-parents (node 9, 11 and 8). To ensure consistency between training and testing the same approach is applied to both, e.g. even when the model is trained the inter-node measures for children take all children into account not only positive ones.

Of course pruning decisions made at higher levels could be wrong. Defining parents with respect to only retained nodes thus potentially introduces noise into the system, i.e. a mistake made at a higher level percolates down to lower levels because the context of lower level nodes is defined with respect to higher level nodes. However, given that the algorithm does not backtrack (which would be computationally expensive) one might as well optimise with respect to previously made decisions even if these decisions may have been wrong.

5.2.2.1 Similarity Measures

The main thing that can be measured between two nodes n and m is how similar they are. In general, the more similar two nodes are, the more likely it is that one of them should be pruned. Similarity can be measured with respect to a node's intension or with respect to its extension. Because of the duality between intension and extension, nodes that are similar in

their intension are also similar in their extension and *vice versa*. Thus it is probably enough to measure similarity with respect to either intension or extension. For inheritance hierarchies it is more usual to look at a node's intension. Therefore similarity will be measured with respect to intension rather than extension.

The most obvious way to measure similarity is by counting the number of path-value pairs in which the two nodes overlap. This is the so-called *matching coefficient* (Manning and Schütze, 1999). If $pvps(n)$ is a function that returns the set of n's path-value pairs, i.e. its intension, then the matching coefficient is defined as follows:

$$M_{match}(n,m) = |pvps(n) \cap pvps(m)|$$

The matching coefficient does not take into account the size of the intensions. The *Dice coefficient* relates overlap between two nodes to their combined intension sizes::

$$M_{dice}(n,m) = \frac{2|pvps(n) \cap pvps(m)|}{|pvps(n| + |pvps(m)|}$$

The *Jaccard coefficient* is similar but relates the size of the overlap to the size of the union of the two intensions which results in lower values for small overlaps:

$$M_{jacc}(n,m) = \frac{|pvps(n) \cap pvps(m)|}{|pvps(n) \cup pvps(m)|}$$

The above measures treat path-value pairs as atoms, however sometimes it is interesting to look at shared *parts* of path-value pairs. For this purpose four further similarity measures have been implemented. These are more fine-grained in that they compare elements of attribute paths.

Path Similarity measures the relative number of complete attribute paths that are identical in both intensions. It is defined as the sum of identical attribute-paths multiplied by two and divided by the overall number of attribute-paths in both feature structures. That is, Path Similarity works like the Dice coefficient but the values of attribute-paths are disregarded:

$$\text{Path Similarity}\left(\begin{bmatrix} & \\ \mathbf{Z}|\mathbf{A}|\mathbf{B} & v1 \\ \text{C}|\text{D}|\text{F} & v3 \\ \text{G}|\text{H}|\text{I} & v4 \\ \mathbf{K}|\mathbf{L}|\mathbf{M}|\mathbf{N} & v5 \\ \mathbf{V}|\mathbf{W}|\mathbf{X}|\mathbf{Y}|\mathbf{Z} & v6 \end{bmatrix} \begin{bmatrix} & \\ \text{O}|\text{P}|\text{Q} & v0 \\ \mathbf{Z}|\mathbf{A}|\mathbf{B} & v2 \\ \text{O}|\text{D}|\text{F} & v3 \\ \text{H}|\text{I} & v4 \\ \mathbf{K}|\mathbf{L}|\mathbf{M}|\mathbf{N} & v5 \\ \mathbf{V}|\mathbf{W}|\mathbf{X}|\mathbf{Y}|\mathbf{Z} & v7 \end{bmatrix}\right) = 6/11 \approx 0.54$$

In monotonic hierarchies a measure for attribute-path similarity only makes sense between nodes that are not in a subsumption order (i.e. between siblings or co-parents but not between parents or children) because nodes which are in a subsumption order will never differ in their values for a given complete path, i.e. if Path Similarity is applied to nodes that are in a subsumption order it will return the same results as the Dice coefficient. Consequently, this measure is only used for siblings and co-parents.

The final three similarity measures deal with identical suffixes and prefixes. *Path Prefix Similarity* measures the relative number of identical attributes in path prefixes. It is defined as the number of shared attributes (shown in bold face) in identical prefixes multiplied by two and divided by the overall number of attributes:

$$\text{Path Prefix Similarity}\left(\begin{bmatrix} \mathbf{A}|\mathbf{B}|\mathbf{C} & v1 \\ \mathbf{E}|\mathbf{F}|\mathrm{G}|\mathrm{H} & v2 \\ \mathrm{K}|\mathrm{L}|\mathrm{M} & v4 \\ \mathbf{X}|\mathbf{Y}|\mathrm{Z} & v6 \end{bmatrix} \begin{bmatrix} \mathbf{A}|\mathbf{B}|\mathbf{C} & v1 \\ \mathbf{E}|\mathbf{F}|\mathrm{I} & v3 \\ \mathrm{L}|\mathrm{M} & v5 \\ \mathbf{X}|\mathbf{Y}|\mathrm{V}|\mathrm{W} & v7 \end{bmatrix}\right) = 14/25 = 0.56$$

The rationale for this measure is the same as for the intra-node measure Path Prefix Coherence. That is, Path Prefix Similarity measures to what degree two nodes deal with similar areas of linguistics.

Path Suffix Similarity is defined analogously for path suffixes:

$$\text{Path Suffix Similarity}\left(\begin{bmatrix} \mathrm{Z}|\mathrm{A}|\mathrm{B} & v0 \\ \mathrm{X}|\mathbf{A}|\mathbf{B} & v1 \\ \mathrm{X}|\mathrm{S}|\mathbf{C}|\mathbf{D}|\mathbf{E}|\mathbf{F} & v2 \\ \mathrm{Y}|\mathrm{T}|\mathrm{C}|\mathrm{D}|\mathrm{E}|\mathrm{Q} & v3 \end{bmatrix} \begin{bmatrix} \mathrm{K}|\mathbf{A}|\mathbf{B} & v7 \\ \mathrm{H}|\mathbf{C}|\mathbf{D}|\mathbf{E}|\mathbf{F} & v8 \\ \mathrm{I}|\mathrm{C}|\mathrm{D}|\mathrm{E}|\mathrm{P} & v9 \end{bmatrix}\right) = 12/31 \approx 0.39$$

Path-Value Pair Suffix Similarity behaves like path suffix similarity but also takes values

into account:

$$\text{Path-Value Pair Suffix Similarity}\left(\begin{bmatrix} \text{Z}|\text{A}|\text{B} & v0 \\ \text{X}|\text{A}|\text{B} & v1 \\ \text{X}|\text{S}|\mathbf{C}|\mathbf{D}|\mathbf{E}|\mathbf{F} & \mathbf{v2} \\ \mathbf{K}|\mathbf{L}|\mathbf{M} & \mathbf{v4} \\ \text{Y}|\text{T}|\text{C}|\text{D}|\text{E}|\text{Q} & v3 \end{bmatrix} \begin{bmatrix} & \\ \text{K}|\text{A}|\text{B} & v7 \\ \text{H}|\mathbf{C}|\mathbf{D}|\mathbf{E}|\mathbf{F} & \mathbf{v2} \\ \text{O}|\text{P}|\text{Q}|\mathbf{K}|\mathbf{L}|\mathbf{M} & \mathbf{v4} \\ \text{I}|\text{C}|\text{D}|\text{E}|\text{P} & v9 \end{bmatrix}\right) = 18/49 \approx 0.37$$

5.2.2.2 Relations with and between Parents

13 measures have been implemented that analyse relations between a node and its parents and relations between its parents. In the following definitions P is the set of (positive) parents of a node n, $p_i \in P$ is an individual parent and k is the number of (positive) parents of n.

For measures that are averaged by the number of parents, the variance of the measure is also calculated. Variance, σ^2, is defined as

$$\sigma^2 = \frac{\sum_{i=1}^{k}(x_{p_i} - \mu)^2}{|P|}$$

where x_{p_i} is the value of the measure when applied to parent p_i and μ is the calculated average.

Number of Positive Parents (M 16):

• the number of positive parents of the current node

The number of parents indicates how well a node is connected with higher parts of the hierarchy. In general, nodes in manual hierarchies have a relatively low number of parents. Hence if a node has exceptionally many parents this may indicate that the node should be pruned.[19] Of course, it is also possible that a high number of parents arises from a failure to prune parents which should have been pruned. For this reason it is also important to look in the other direction and take the number of co-parents into account (see page 146).

Average Number of Inherited Path-Value Pairs and Variance (M 17 & 18):

• how many path-value pairs a node inherits from a parent on average

This is effectively the matching coefficient between the node and each parent averaged by all

[19] Note, that the number of parents does not increase monotonically between ancestors and descendants. That is, the descendant of a node n may have fewer parents than n itself. Consequently, it is possible to reduce the average number of parents in a hierarchy by pruning nodes with unusually many parents.

parents. It is defined as:

$$\text{M } 17 = \sum_{i=1}^{k} \frac{1}{|P|} \times |\textit{underlying intension}(p_i)|$$

As with other similarity measures, if there is a high degree of similarity, i.e. a node inherits many path-value pairs on average, this may indicate that the pruning probability should be increased.

Average Proportion of New Path-Value Pairs and Variance (M 19 & 20):

• what proportion of path-value pairs cannot be inherited from a parent on average

$$\text{M } 19 = \sum_{i=1}^{k} \frac{1}{|P|} \times \frac{|\textit{overt intension(n)}|}{|\textit{underlying intension}(p_i)|}$$

This measure is a variation of the previous one (M 17): it measures how different a node is on average from its parents. While a high degree of similarity between a node and its parents may indicate that the node should be pruned, a high degree of dissimilarity may indicate that the node should be retained.

Average Path-Prefix Similarity with Parents and Variance (M 21 & 22):

• how similar are the path-prefixes between *n*'s overt intension and the underlying intensions of its parents on average

$$\text{M } 21 = \sum_{i=1}^{k} \frac{1}{|P|} \times \textit{path-prefix-similarity}(n,p_i)$$

A high Path Prefix Similarity means that *n*'s new path-value pairs (i.e. those it cannot inherit) add information that is similar to (i.e. falls into the same linguistic areas as) the information it already inherits. One could hypothesise that nodes often add new information that is similar to the one that they inherit and that a node which adds information that is very dissimilar to the information it inherits should therefore be pruned. However this will not be true in all cases. Despite this it seems that this measure may be useful in deciding whether or not a node should be pruned.

Average Path-Value Pair Suffix Similarity with Parents and Variance (M 23 & 24):

• how similar are the path-value pair suffixes of *n*'s overt intension and the underlying intensions of its parents on average

$$\text{M } 23 = \sum_{i=1}^{k} \frac{1}{|P|} \times \textit{path-value pair suffix-similarity}(n,p_i)$$

This measure assesses to what extent embedded feature structures of a parent node re-occur in the feature structure of the current node (see Path-Value Pair Suffix Coherence, page 124). This measure has been implemented in analogy with the corresponding intra-node measure.

Average Path-Suffix Similarity with Parents and Variance (M 25 & 26):

• how similar are the path-suffixes between *n* and its parents on average

$$\text{M } 25 = \sum_{i=1}^{k} \frac{1}{|P|} \times \textit{path-suffix-similarity}(n, p_i)$$

Again this measure was implemented in analogy with the corresponding intra-node measure and may not have a big effect. One situation where an embedded feature structure is repeated in this way is where a parent specifies an underspecified value for an attribute and the current node gives a more specific value. However, this case will not arise in the system described here because underspecified values are lost in inheritance paths where a more specific value exists, due to the pre-processing and compiling out of the training lexicon (see Section 6.2).

Average Proportion of Overlap between Parents and Variance (M 27 & 28):

• how much do the node's parents overlap

This is a measure *between* parents rather than a measure between *n* and its parents. It measures how much the underlying intensions of *n*'s parents overlap. The overlap between a parent p_i and all other parents is defined as the intersection between p_i's underlying intension with the union of the underlying intensions of all other parents. To get the average proportion of the overlap this is then divided by the size of p_i's intension and averaged over all parents:

$$\text{M } 27 = \sum_{i=1}^{k} \frac{1}{|P|} \times \left(\frac{|\textit{underlying intension}(p_i) \cap \bigcup_{p_j \in P/\{p_i\}} (\textit{underlying intension}(p_j))|}{|\textit{underlying intension}(p_i)|} \right)$$

This is effectively a measure of the degree of orthogonality between the parents. Intuitively, a high percentage of parental overlap is not desirable. If a node inherits from different parents it usually inherits different property sets from each of them, e.g. subcategorisation properties from one of them and inflectional properties from another.

5.2.2.3 Relations with Children

11 measures have been defined for a node's relation with its children. In the definitions, C is the set of (all) children of a node n, $c_i \in C$ is an individual child and k is the number of children of n.

Number of Children (M 29):

• how many children the current node has

This is the number of children in the Galois lattice, i.e. all children rather than only the positive children. One reason for including this measure is that how many children a node has indicates with how many other, mutually exclusive (i.e. not co-occurring) path-value pairs the elements in the node's intension can co-occur. This is illustrated by Figure 5.7. Each child in the Galois lattice will differ from the current node in at least on path-value pair and all children will contain at least on path-value pair that none of the other children contains (because the children are incompatible and not in a subsumption relation to each other). For example, *Node 4* differs from *Node 2* in that it contains the path-value pair f while *Node 5* differs from *Node 2* in that it contains the path-value pair d. Even where the intensions of two children are compatible in some cases, i.e. where two children have a common descendant (nodes 2 and 3 in Figure 5.7), there has to be at least one descendant (*Node 4*) of one child which is not compatible with at least one descendant (*Node 6*) of the other child.

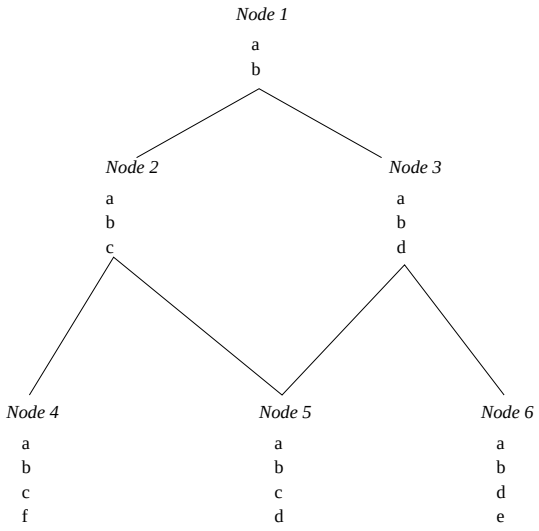


Figure 5.7: Number of Children in Galois Lattice

Average Path-Prefix Similarity with Children and Variance (M 30 & 31):

• how similar the path-prefixes between the node and its children are

These measures are defined in analogy to measures M 21 & 22 (Average Path-Prefix Similarity

with Parents and Variance):

$$\text{M } 30 = \sum_{i=1}^{k} \frac{1}{|C|} \times \textit{path-prefix-similarity(n,}c_i\textit{)}$$

Again this is a similarity measure which may be useful in determining whether the current node should be pruned.

Average Path-Value Pair Suffix Similarity with Children and Variance (M 32 & 33):

• how similar the path-value pair suffixes between the current node and its children are

These measures, too, are defined in analogy to the corresponding parent measure:

$$\text{M } 32 = \sum_{i=1}^{k} \frac{1}{|C|} \times \textit{path-value-pair-suffix-similarity(n,}c_i\textit{)}$$

Average Path-Suffix Similarity with Children and Variance (M 34 & 35):

• how similar the path endings between the current node and its children are

Again these measures are defined in analogy to the corresponding parent measure:

$$\text{M } 34 = \sum_{i=1}^{k} \frac{1}{|C|} \times \textit{path-suffix-similarity(n,}c_i\textit{)}$$

Average Proportion of Overlap between Children and Variance (M 36 & 37):

• how much the node's children overlap

These criteria are very similar to criteria M 27 & 28 (Average Proportion of Overlap between Parents and Variance). However, the path-value pairs inherited from *n* are not taken into account since all children will overlap in these path-value pairs.

In effect this is a measure of orthogonality between the children of a node. If two children overlap in the path-value pairs that they do not inherit from the current node, this is an indicator that they have another common parent, apart from the current node. The degree of overlap between the children may be a useful piece of information. For example, it could be that a high degree of overlap should increase the pruning probability of the current node because it means that many of its children have other common parents or that the children have few common parents but inherit a large amount of path-value pairs from these. Both scenarios intuitively lower the importance of the current node and thus may indicate that the current node should be pruned.

Average Number of New Path-Value Pairs of Children and Variance (M 38 & 39):

• the average number of non-inherited path-value pairs of the node's children

These criteria are very similar to the criteria Average Number of Inherited Path-Value Pairs and Variance (M 17 & 18). For each child the measure computes the proportion of path-value pairs that that child cannot inherit from n and averages by all children. If C is the set of n's children the criterion is defined as:

$$\text{M } 38 = \sum_{i=1}^{k} \frac{1}{|C|} \times |\textit{overt intension}(c_i)|$$

Again this is a measure of similarity between the current node and its children and as such may be important in determining whether the node should be pruned.

5.2.2.4 Relations with Siblings

15 measures for a node's relation with its siblings have been defined.

Number of Siblings (M 40):

• how many siblings the node has

Siblings considered by this measure are those that have been classified as positive and those that have not yet been considered by the testing or training algorithm (see page 136). Due to this ambiguous nature of why a node is counted as a sibling, this measure is a bit imprecise. The same is true for all other sibling measures. Nonetheless the number of siblings can say something about the (current) degree of inter-connectedness of the hierarchy and may therefore be important.

The remaining 14 sibling measures are similarity measures. In the definitions below, n is the current node, S is the set of siblings, $s_i \in S$ and $k = |S|$. The general motivation for including these measures is that a node that is very similar to its siblings may be more likely to be pruned.

Average Path-Prefix Similarity with Siblings and Variance (M 41 & 42):

• how similar the path-prefixes between the node and its siblings are

These measures have been defined in analogy to other Path Prefix Similarity measures (M 21 & 22, M 30 & 31), i.e.:

$$\text{M } 41 = \sum_{i=1}^{k} \frac{1}{|S|} \times \textit{path-prefix-similarity(n,}s_i\textit{)}$$

Like the corresponding parent measures they assess in how far nodes (in this case the current node and its siblings) deal with similar linguistic areas. One could hypothesise that siblings

usually deal with similar areas and that a failure to do so may be an indication that the current node should be pruned.

Average Path-Suffix Similarity with Siblings and Variance (M 43 & 44)

• how similar the path-suffixes between the node and its siblings are

Again these measures have been defined in analogy to other Path Suffix Similarity measures:

$$\text{M 43} = \sum_{i=1}^{k} \frac{1}{|S|} \times \textit{path-suffix-similarity(n,}s_i\textit{)}$$

Average Path-Value Pair Suffix Similarity with Siblings and Variance (M 45 & 46)

• how similar the path-value pair suffixes between the node and its siblings are

These measures too, have been defined in analogy to the corresponding Path-Value Pair Suffix Similarity measures:

$$\text{M 45} = \sum_{i=1}^{k} \frac{1}{|S|} \times \textit{path-value pair suffix-similarity(n,}s_i\textit{)}$$

Average Sibling Matching Similarity and Variance (M 47 & 48):

• the average matching coefficient between the node and its siblings

$$\text{M 47} = \sum_{i=1}^{k} \frac{1}{|S|} \times |\textit{overt intension}(n) \cap \textit{overt intension}(s_i)|$$

The matching coefficient is calculated with respect to overt rather than underlying intensions because siblings share at least some of the path-value pairs they inherit since they have a common parent.[20]

Average Sibling Dice Similarity and Variance (M 49 & 50):

• the Dice coefficient between *n* and its siblings

$$\text{M 49} = \sum_{i=1}^{k} \frac{1}{|S|} \times \frac{2|\textit{overt intension}(n) \cap \textit{overt intension}(s_i)|}{|\textit{overt intension}(n| + |\textit{overt intension}(s_i)|}$$

[20] Instead of disregarding all inherited path-value pairs it would also be possible – and possibly more accurate – to only disregard those path-value pairs that can be inherited from *common* parents.

Average Sibling Jaccard Similarity and Variance (M 51 & 52):

• the Jaccard coefficient between *n* and its siblings

$$\text{M } 51 = \sum_{i=1}^{k} \frac{1}{|S|} \times \frac{|\textit{overt intension}(n) \cap \textit{overt intension}(s_i)|}{|\textit{overt intension}(n) \cup \textit{overt intension}(s_i)|}$$

Average Path Similarity with Siblings and Variance (M 53 & 54):

• the path similarity between *n* and its siblings

$$\text{M } 53 = \sum_{i=1}^{k} \frac{1}{|S|} \times \textit{path-similarity}(n,s_i)$$

For parents and children this criterion did not make sense because nodes that are in a subsumption relation do not differ in the values of a given attribute-path. However for siblings and co-parents it may be useful. This measure is particularly interesting because it is sometimes the case that siblings specify different values for the same paths. For example, a verb node may have a set of children which all differ in their inflection class. In this case, the children will all contain the attribute INFL-CLASS but each of them will specify a different value for this attribute. While a high degree of similarity between two nodes may usually increase the pruning probability of one of them, a high Path Similarity between a node and its siblings could actually decrease the node's pruning probability.

5.2.2.5 Relations with Co-Parents

15 measures have been defined for co-parents.

Number of Co-Parents (M 55):

• the number of co-parents of the current node

Which nodes count as a co-parent is again a bit imprecise: all parents of all children of the current node count as co-parents unless they (the parents) have been classified as negative. That is, the set of co-parents may contain nodes that will be pruned at a later stage. For this reason the co-parent measures may not be very effective. But despite this the number of co-parents is an important piece of information. One could hypothesise that a large number of co-parents decreases the relative importance of the current node and should therefore increase its pruning probability.

The remaining co-parent measures are all similarity measures (similarity has been defined with respect to overt intensions). In the definitions, X is the set of co-parents of the current

node n, $x_i \in X$ and $k = |X|$. Again, the general motivation for including these measures is that a node that is very similar to its co-parents may be more likely to be pruned.

Average Co-Parent Matching Similarity and Variance (M 56 & 57)

• the average Matching coefficient between the node and its co-parents

$$\text{M } 56 = \sum_{i=1}^{k} \frac{1}{|X|} \times |\textit{overt intension}(n) \cap \textit{overt intension}(x_i)|$$

Average Co-Parent Dice Similarity and Variance (M 58 & 59)

• the average Dice coefficient between the node and its co-parents

$$\text{M } 58 = \sum_{i=1}^{k} \frac{1}{|X|} \times \frac{2|\textit{overt intension}(n) \cap \textit{overt intension}(x_i)|}{|\textit{overt intension}(n| + |\textit{overt intension}(x_i)|}$$

Average Co-Parent Jaccard Similarity and Variance (M 60 & 61)

• the average Jaccard coefficient between the node and its co-parents

$$\text{M } 60 = \sum_{i=1}^{k} \frac{1}{|X|} \times \frac{|\textit{overt intension}(n) \cap \textit{overt intension}(x_i)|}{|\textit{overt intension}(n) \cup \textit{overt intension}(x_i)|}$$

Average Path-Prefix Similarity with Co-Parents and Variance (M 62 & 63)

• the average path-prefix similarity between the node and its co-parents

$$\text{M } 62 = \sum_{i=1}^{k} \frac{1}{|X|} \times \textit{path-prefix-similarity}(n,x_i)$$

Average Path-Suffix Similarity with Co-Parents and Variance (M 64 & 65)

• the average path-suffix similarity between the node and its co-parents

$$\text{M } 64 = \sum_{i=1}^{k} \frac{1}{|X|} \times \textit{path-suffix-similarity}(n,x_i)$$

Average Path-Value Pair Suffix Similarity with Co-Parents and Variance (M 66 & 67)

• the average path-value pair suffix similarity between the node and its co-parents

$$\text{M } 66 = \sum_{i=1}^{k} \frac{1}{|X|} \times \textit{path-value pair suffix-similarity}(n,x_i)$$

Average Path Similarity with Co-Parents and Variance (M 68 & 69)

• the average path similarity between the node and its co-parents

$$\text{M } 68 = \sum_{i=1}^{k} \frac{1}{|X|} \times \textit{path-similarity(n,x}_i\textit{)}$$

The similarity measures are interesting because they again address orthogonality and dimensionality. Thus if a node and its co-parents are orthogonal one would expect low Matching, Dice and Jaccard similarities and also a low Path and Path Prefix Similarity. In general, one would expect co-parents to exhibit a certain degree of orthogonality. One would not expect a node to inherit from two relatively similar parents. Hence, if a node is very similar to its co-parents its pruning probability should increase.

5.2.3 From Measures to Contextual Predicates

The measures discussed in the previous sections model the context of a node but they cannot be used directly as contextual predicates in maximum entropy features because contextual predicates are boolean functions that determine the presence or absence of a property in a context whereas most of the above measures are real-valued. To turn real-valued measures into contextual predicates they first have to be converted into discrete functions, i.e. they have to be *quantised*. Once this has been done a measure which ranges over *k* discrete values can be turned into *k* contextual predicates, each of which encoding whether a particular value has been observed or not.

Quantisation is achieved by breaking up the range of a measure into a set of intervals. For example, the measure Average Sibling Dice Similarity (*avg-sibl-dice*) ranges from 0 to 1. This can be turned into five equal size intervals:

avg-sibl-dice: [0.0-0.2), [0.2-0.4), [0.4-0.6), [0.6-0.8), [0.8-1.0]

This would result in five contextual predicates, each measuring whether the value of the Dice Similarity Measure falls in one particular interval. For example, if a node has an Average Sibling Dice Similarity of 0.3, the contextual predicate that tests for the second interval would return *true* and the other four contextual predicates would return *false*.

The number of intervals can be fixed but it can also be chosen automatically by looking at the training data and recursively splitting an interval of values into sub-intervals at a point where the information gain is the largest. The minimum description length principle can be employed to determine when an interval should not be split any further (Fayyad and Irani, 1993).

A problem with this approach to quantisation is that information about the order of values is lost. However, order may be important. For example, the fact that the interval $[0.2 - 0.4)$ is closer to the interval $[0.4 - 0.6)$ than it is to the interval $[0.8 - 1.0]$ is potentially significant. All other things being equal, a node whose Average Sibling Dice Similarity is 0.3 is —all other things being equal— more likely to behave like a node whose Average Sibling Dice Similarity is 0.5 than like a node whose Average Sibling Dice Similarity is 0.9.

An alternative strategy which addresses this problem is to turn the range of a measure into $k-1$ binary measures, e.g.:

avg-sibl-dice-1: < 0.2, ≥ 0.2
avg-sibl-dice-2: < 0.4, ≥ 0.4
avg-sibl-dice-3: < 0.6, ≥ 0.6
avg-sibl-dice-4: < 0.8, ≥ 0.8

Preserving order information is particularly important if there is not a lot of training data since in that case it may be that not enough nodes have been seen whose Average Sibling Dice Similarity falls in a particular interval, say $[0.2 - 0.4)$, This may result in the classifier not being certain whether —all other things being equal— a Sibling Dice Similarity in that interval is a good indicator for pruning or for retaining a node. However, if the intervals $[0.0 - 0.2)$ and $[0.4 - 0.6)$ seem to be good indicators for retaining a node, one may conclude that the interval $[0.2 - 0.4)$ is also a good indicator for retaining. If order is not preserved it is impossible to infer this. But if Sibling Dice Similarity is encoded by four binary measures, lack of data in the interval $[0.2 - 0.4)$ will be balance out by sufficient data in neighbouring intervals. Thus the interval $[0.2 - 0.4)$ will be covered by the inequality < 0.6 and data for the interval $[0.0 - 0.2)$ will contribute to the importance assigned to the inequality < 0.4.

Both quantisation methods have been tested in the experiments in Chapter 7.[21]

Once the measures have been quantised every interval (or every binary measure if binary quantisation is used) corresponds to a contextual predicate. For example, there might be a contextual predicate (*avg-sibl-dice-2nd-interval*) which tests whether the Average Sibling Dice Similarity of the current node falls in the second interval, i.e. the interval [0.2-0.4):

$$\text{avg-sibl-dice-2nd-interval(n)} \rightarrow \{ \text{ true, false } \}$$

Theoretically, it is possible to combine different pieces of evidence into one predicate. For example, one could define a predicate which tests whether the Average Sibling Dice Similarity

[21] To quantise my data, I used the filtering methods provided by the WEKA machine learning software package: `http://www.cs.waikato.ac.nz/ml/weka/index.html` (3.7.03).

of the current node falls in the second interval *and* the Average Co-Parent Dice Similarity falls in the third interval. This predicate would only return true if both conditions were fulfilled. Combining evidence in this way increases the expressive power of the statistical modelling (McCallum 2003) and may lead to better results in cases where this increase in expressiveness is needed to separate the different classes. However, I did not make use of this here. The main reason for this is that there are many possible combinations and it is difficult to have an intuition about which pieces of evidence should be combined in this way. In the absence of such intuitions conjoined features are probably best restricted to cases where features are induced automatically.

5.3 Summary

This chapter gave an overview of the maximum entropy framework. Maximum entropy modelling is a supervised machine learning technique, which uses pre-classified training data to estimate the likelihood that an object (i.e. a node in the Galois lattice) belongs to a given class (i.e. *prune* or *retain*) given its context.

Maximum entropy modelling is well suited for the problem discussed here because it allows a very fine-grained modelling of node context. Furthermore it does not assume that different contextual properties are statistically independent and it has been applied successfully to a range of NLP tasks.

Context in the maximum entropy framework is modelled by a set of contextual features. It was argued that the features for the task discussed here need to be fairly abstract. In particular they should not refer directly to individual path-value pairs since different lexicons usually differ in most of their path-value pairs and mapping one set to another is very difficult if not impossible even if done by hand. Maximum entropy features that are too specific will therefore mean that the system cannot generalise across lexicons which renders it useless.

This chapter introduced 69 contextual measures which model the context of a node in the Galois lattice. The measures can be split into intra-node measures, which assess the properties of a node itself, and inter-node measures, which assess the relation between the node and other nodes. The measures assess properties such as the amount of interconnectedness in the hierarchy, the similarity between nodes and the degree of interdependence between the path-value pairs in a node's intension.

Chapter 6

The System

This chapter gives details of the system used in the experiments in Chapter 7. Section 6.1 introduces the lexical data that were used in the experiments. The system is implemented for LKB grammars (Copestake, 2002). However, the LKB lexicons have to be pre-processed before they can be used. This is discussed in Section 6.2. Once a lexicon has been pre-processed the Galois lattice can be built for it as described in Section 6.3. Since the training lexicon is too big to train on the complete set of nodes in the Galois lattice, the maximum entropy model is only trained on a subset of training examples and this subset is randomly sampled from the lattice as described in Section 6.4. Finally, Section 6.5 explains how the automatically derived hierarchies are evaluated.

6.1 Lexicons

At least two lexicons are required to test the maximum entropy pruning: one to train the model and one to test it. To ensure that there are sufficient training data, the training hierarchy should be relatively big or there should be several training hierarchies. One of the largest publically available lexical inheritance hierarchies is distributed with the LinGO English Resource Grammar (henceforth LinGO ERG),[1] a large-scale grammar for English that has been developed over several years and used in various projects. The version used in the experiments contains just under 7,000 lexical entries. This may sound small for a lexicon but is actually very big for a hierarchical lexicon as these tend to be relatively small.[2]

Given its size, LinGO ERG is an obvious choice for a training hierarchy. It has been

[1] See `http://lingo.stanford.edu/` (10.4.03).

[2] One reason for this is probably the amount of work required to build a hierarchical lexicon.

developed for the LKB grammar and lexicon development environment (Copestake, 2002). The LKB system is geared towards constraint-based grammars, such as HPSG. The lexical component in LKB grammars is always represented as an inheritance hierarchy and lexical and grammar rules are embedded in the hierarchy rather than being a separate component. Hence, the terms *lexicon* and *grammar* are more or less synonymous for the LKB system.

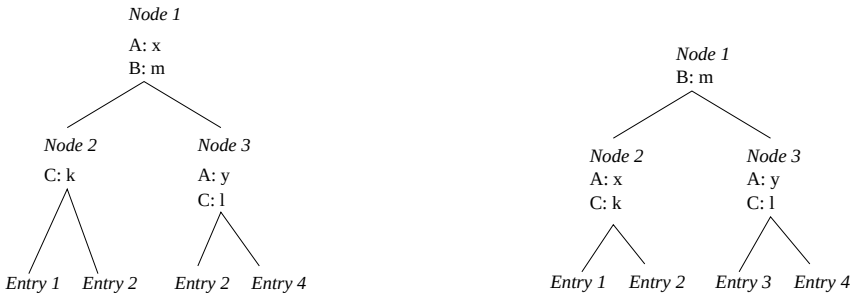
Theoretically any lexicon of reasonable size can serve as test data. However, a popular evaluation method for this task (used by both Barg 1996a and Light 1994) is the comparison of the automatically generated hierarchy to a manually built hierarchy. The latter is assumed to be the gold standard for that lexicon. There are some problems with this method, which are discussed in Section 6.5, but since most alternative evaluation methods are not practical, I have opted for the gold standard evaluation method as well. This means that a manually built hierarchy is required for the test data which can then be used as a gold standard. Since the system needs to be interfaced with the LKB anyway to enable the use of LinGO ERG as a training hierarchy it makes sense to use LKB grammars for testing, too. Apart from LinGO ERG several other grammars have been implemented for the LKB. Some of them are too small for the experiments described here but a couple are of the right size. The first of these is the so-called Textbook Grammar (henceforth Textbook).[3] This is an English grammar that has been developed for Sag and Wasow (1999). It contains 507 lexical entries. The second is a Spanish grammar (Quirino Simões, 2001) containing 405 entries.[4]

The Textbook hierarchy is actually non-monotonic. This means that it cannot be accurately re-created by the pruning method discussed here. However, the pruning method can usually create a hierarchy which contains the same nodes as a non-monotonic hierarchy. The difference will only lie in the intensions of the nodes. This is illustrated by Figure 6.1. The hierarchy fragment on the left is non-monotonic and the attribute-value pair A:*x* at *Node 1* is overridden by the conflicting attribute-value pair A:*y* at *Node 3*. The hierarchy fragment on the right is monotonic but it contains the same nodes (i.e. nodes with the same extensions). The difference lies only in the intensions of the nodes as the attribute-value pair A:*x* is now specified at *Node 2* rather than at *Node 1*.

The maximum entropy pruner would not be able to derive the fragment on the left but it could derive the fragment on the right. The evaluation method described in Section 6.5 evaluates a derived hierarchy by matching it to the original hierarchy. Two nodes are matched

[3] See `http://lingo.stanford.edu/teaching-grammars.html` (10.4.03).

[4] There is also a Japanese lexicon developed by Melanie Siegel (cf. Siegel 2000) but this is substantially bigger (nearly 3,000 entries) than the Spanish and Textbook lexicon and proved too big for the system discussed here.



(a) Non-monotonic (b) Monotonic

Figure 6.1: Non-monotonic vs. monotonic hierarchy fragments

if they have the same extension. Using this method a perfect match can be achieved between the two fragments in Figure 6.1. This means that —while it is not possible to re-create the original Textbook hierarchy with the pruning method discussed here— the Textbook hierarchy can nonetheless serve as a means of evaluating the pruning method. This is particularly true as non-monotonic inheritance is used sparsely in the Textbook lexicon. The only problem that could arise is that the monotonic hierarchy that corresponds most closely to the original, non-monotonic Textbook hierarchy is not as plausible as the non-monotonic version and this could bias the pruning method against it and cause it to derive another monotonic hierarchy.

There are two reasons to choose the Textbook hierarchy as a test lexicon despite the fact that it is non-monotonic. First, it is very difficult to find suitable monotonic hierarchies that have the right size for a test lexicon. Second, the Textbook lexicon is one of the most carefully constructed hierarchical lexicons that exist for the LKB. It has been developed over several years and by several people. Furthermore, it was developed as practical material for a linguistic textbook (Sag and Wasow 1999) and, consequently, it has been rigorously checked for consistency.

The fact that there are precious few suitable test lexicons currently available was also the reason for choosing a Spanish lexicon as the second test lexicon. Since there is only one suitable Spanish lexicon available for the LKB, the maximum entropy model that is used to derive Spanish hierarchy has to be trained on LinGO ERG (i.e. an English grammar). That is, the training and test data will be for different languages in this case. However this should not be a problem since the maximum entropy model defines context in a fairly abstract form. No reference is made to individual path-value pairs or individual linguistic analyses (see Section 5.2).

Instead features refer to properties such as the overlap parents or the relative level of a node. These properties should not be influenced by the language for which the grammar has been developed. For example, if a relative level of 0.9 (i.e. fairly high in the Galois lattice) makes it likely that a node is pruned this should hold no matter whether the lattice has been built for an English or a Spanish lexicon.

While it is unlikely that there are language specific idiosyncrasies in this area, it is possible that a contextual criterion like "relative level" does not generalise very well across different *lexicons*, for example because some hierarchical lexicons only capture fairly low-level generalisation while other hierarchies also capture high-level generalisations. The existence of many such idiosyncrasies in individual lexicons would mean that training on one lexicon and testing on another might be difficult even if both lexicons were for the same language. The way around such problems would be to train on several lexicons to average out any idiosyncrasies. At the moment this is not an option, though, given the lack of lexical hierarchies that are currently available.

In general, it is not possible to get around problems of idiosyncratic hierarchy design by splitting one grammar into two subsets and use one for training and the other for testing, since splitting a hierarchy usually changes the contexts of at least some of the nodes and these modifications might influence the structure of the hierarchy. For instance, some nodes may not make sense any more after the hierarchy has been split, e.g. because their extension set has been split in half and the smaller extension set does no longer justify the existence of the node in the hierarchy. Splitting a hierarchical lexicon into two sub-lexicons is only safe if no node other than the root node is split as a result. For this to be the case no pair of nodes in the two sub-lexicons should have a common ancestor or a common descendant in the original hierarchy. Even if it was possible to split a hierarchy the question is whether the resulting sub-hierarchies would be big enough for training and testing.

The Spanish lexicon was developed as part of a summer project and later formed the basis for a master's thesis on Spanish clitics (Quirino Simões 2001). This means that the development time that went into it was a lot less than for the Textbook lexicon. It does contain a few inconsistencies (e.g. the partial order is not observed in one case, see page 107) and —as will become evident later— it is also very sparse.

6.2 Pre-Processing

LKB hierarchies have certain properties that make them not immediately compatible with the learning algorithm proposed here. Therefore they need to be pre-processed. Three areas have to be addressed in the pre-processing. As was discussed in Section 3.1 the Galois lattice construction algorithm expects lexical entries to be represented by sets of atomic feature structures rather than by complex feature structures. Hence, feature structures have to be converted in this way. Reentrancies also have to be re-represented. In addition, LKB hierarchies are typed and enforce the principle of Minimal Introduction of attribute-value pairs; some nodes in the hierarchy represent types that are referred to within the feature structures of other types, i.e. some nodes correspond to embedded feature-structures. This is incompatible with Galois lattices. Therefore, feature structures have to be re-represented by expanding out the types in their definitions. Finally, LKB hierarchies also contain nodes that do not strictly belong to the lexicon but represent grammar or lexical rules and they also contain extra nodes to ensure boundedness. These two sets of nodes are removed.

6.2.1 Abandoning Minimal Introduction

LKB hierarchies are typed and follow the principle of Minimal Introduction of attribute-value pairs, which requires that every attribute-value pair is only introduced at a single point in the hierarchy (Copestake, 2002, p. 75).[5] For example, the (partial) type hierarchy in Figure 6.2 violates the constraint of Minimal Introduction because the attribute-value pairs AGR:*agr-cat* and CASE:*case-cat* occur at type *verb-lxm* and at type *prep-wd* but not at a common ancestor of these two types (e.g. *synsem-struc* or *feat-struc*).[6] Thus, these two attribute-value pairs are effectively introduced twice into the hierarchy.

The hierarchy can be transformed into a hierarchy which does obey the constraint by adding *noun* as a separate node as in Figure 6.3. Now the attributes AGR and CASE do not have to be mentioned at *verb-lxm* and *prep-wd* and thus the constraint is not violated.

Because they obey Minimal Introduction, LKB hierarchies contain many nodes that correspond to embedded feature structures, like the node *noun* in the hierarchy in Figure 6.3. These nodes function like a kind of macro that can be inserted in the appropriate place if necessary. For example, when the hierarchy in Figure 6.3 is compiled out all occurrences of the type *noun*

[5] Copestake (2002) calls this *Maximal Introduction*. The term *Minimal Introduction* has been taken from Keller (1993).

[6] The type hierarchy has been taken from one of the grammars supplied with the LKB (the so-called "Pacifier" grammar). Note, that only part of the features structures for *verb-lxm* and *prep-wd* is displayed.

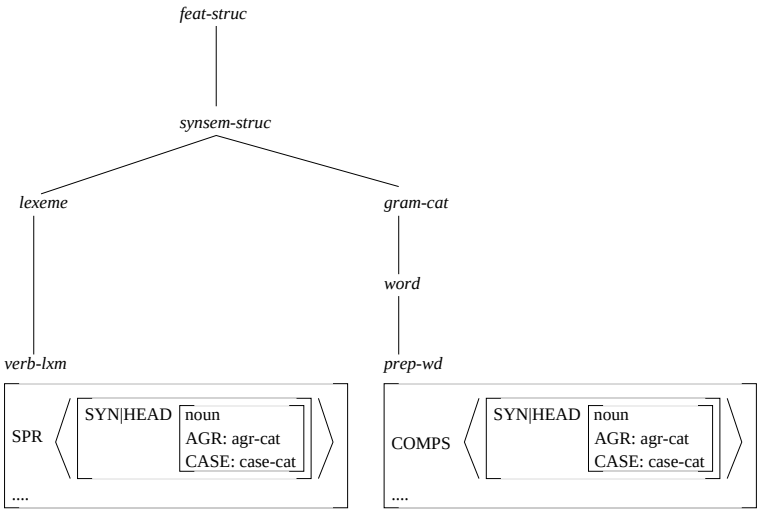


Figure 6.2: Attribute-value pairs not minimally introduced

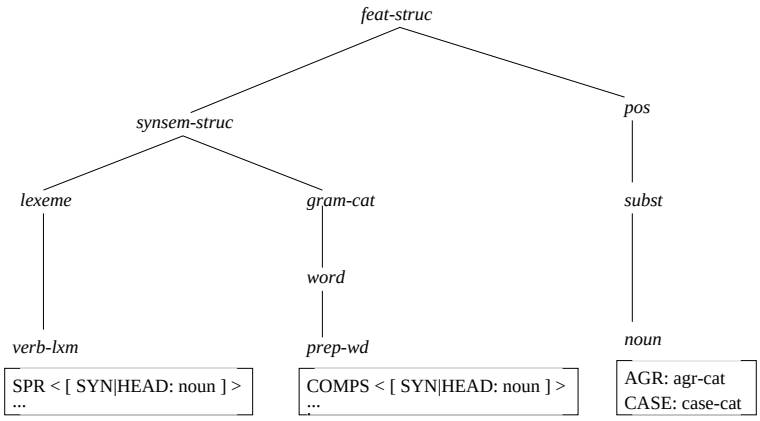


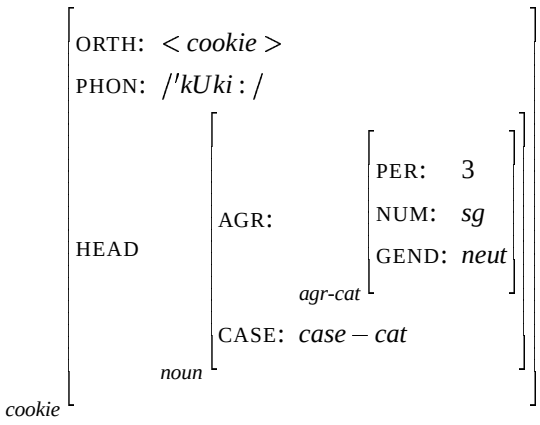
Figure 6.3: Minimal introduction of attribute-value pairs

are replaced by the corresponding feature-structure. Thus, whether a hierarchy obeys Minimal Introduction or not is a matter of implementation. It does not change the underlying lexicon. Galois lattices work on flat feature structures and consequently there is no concept of feature structure embedding; all nodes in the Galois lattice correspond to complete features structures rather than embedded ones. However the training data is obtained by matching a Galois lattice to the corresponding manually built hierarchy. To ensure that these two can be matched properly the pre-processing step transforms hierarchies like the one in Figure 6.3 into hierarchies like the one in Figure 6.2 by replacing value types which refer to embedded feature structures

by the embedded feature structure. For examples, all occurrences of the value type *noun* in Figure 6.3 will be replaced by the *noun* feature structure.

6.2.2 Feature Structure Flattening

The next step of the pre-processing "flattens" the feature structures, i.e. every complex feature structure is converted into a set of atomic feature structures as shown in Figure 6.4. Note, that complex types, like *noun* and *agr-cat*, are retained as the values of a new attribute TYPE. This has been done because some of them, such as *noun*, encode important information. Getting rid of the value type of HEAD would effectively remove all part-of-speech information from the lexicon.[7]



(a) Complex FS

ORTH :< *cookie* >
PHON : /'*kUki* : /
SYN|HEAD|TYPE : *noun*
SYN|HEAD|AGR|TYPE : *agr-cat*
SYN|HEAD|AGR|PER : 3
SYN|HEAD|AGR|NUM : *sg*
SYN|HEAD|AGR|GEND : *neut*
SYN|HEAD|CASE : *case-cat*

(b) Set of atomic FSs

Figure 6.4: Flattening feature structures

Knowledge of appropriateness conditions is lost at this point. However it would be possible to add appropriateness conditions explicitly, for example by adding attribute-value pairs such as SYN|HEAD|APPROP:*agr* to express the fact that the attribute AGR is appropriate for SYN|HEAD.

Converting all nodes to sets of atomic feature structures also involves the re-representation

[7] Types like *agr-cat* and *case-cat* are intuitively less important. One clue for the relative relevance of a complex value type is its position in the type hierarchy. The maximally general value type of an attribute is probably relatively unimportant. More specific types like *noun* (*head* $\sqsubset$ *noun*) carry more information. A refinement of the pre-processing would therefore reason about the relative importance of a complex type and remove the less important types.

of reentrancies. In the automatically constructed hierarchies the two pieces of information potentially contained in a reentrancy, i.e. the value of an attribute and the fact that it is reentrant, are teased apart. To represent the fact that two or more paths are reentrant a new attribute REENTR is introduced, which takes the set of reentrant paths as its value:

$$\begin{bmatrix} \text{PAST} & \boxed{1}\text{+ed} \\ \text{PAST PART} & \boxed{1} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{PAST} & \text{+ed} \\ \text{PAST PART} & \text{+ed} \\ \text{REENTR} & \{\text{PAST,PAST PART}\} \end{bmatrix}$$

Three or more reentrant values are represented as follows:

$$\begin{bmatrix} \text{F} & \boxed{1}\text{v1} \\ \text{G} & \boxed{1} \\ \text{H} & \boxed{1} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{F} & \text{v1} \\ \text{G} & \text{v1} \\ \text{H} & \text{v1} \\ \text{REENTR} & \{\text{F,G,H}\} \end{bmatrix}$$

There can be more than one REENTR attribute in a feature structure to represent the fact that there is more than one reentrancy:

$$\begin{bmatrix} \text{F} & \boxed{1}\text{v1} \\ \text{G} & \boxed{1} \\ \text{H} & \boxed{2}\text{v2} \\ \text{I} & \boxed{2} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{F} & \text{v1} \\ \text{G} & \text{v1} \\ \text{H} & \text{v2} \\ \text{I} & \text{v2} \\ \text{REENTR} & \{\text{F,G}\} \\ \text{REENTR} & \{\text{H,I}\} \end{bmatrix}$$

Representing reentrancies in this way has the consequence that path identity and path value do not have to occur at the same node; it is possible to generalise over entries which have a reentrancy on the same paths but state different values for these paths, as in Figure 6.5.

However, there is no mechanism in the implementation which allows splitting up a set of reentrant paths (i.e. the value of REENTR) into subsets, e.g.:

$$\text{REENTR:}\{\text{F,G}\} \sqsubseteq \text{REENTR:}\{\text{F,G,H}\}$$

This is an implementational choice and has nothing to do with Galois lattices as such. It would be possible to extent the implementation to cover this case. However this kind of generalisation is not normally used in manually built hierarchies.[8]

[8]No such case occurred in any of the manually-built hierarchies used here and it is difficult to imagine a scenario

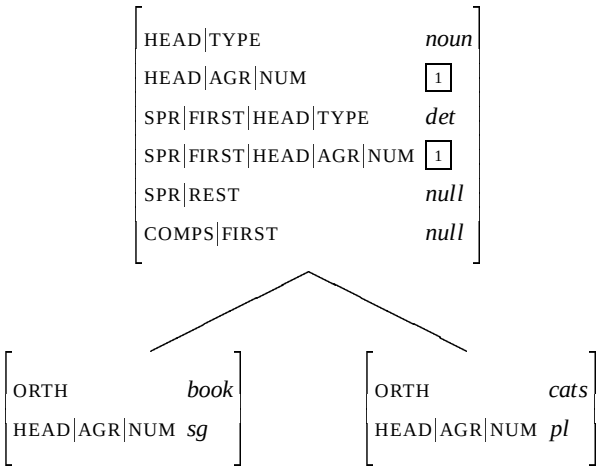


Figure 6.5: Splitting path identity and path value

6.2.3 Removing Rules and GLB-Types

A hierarchy pre-processed in this way still differs from an automatically constructed hierarchy in that the former may contain empty nodes which are inserted by the LKB to ensure that any pair of consistent nodes has a unique greatest lower bound. These nodes are called *glb-types*. The automatically constructed hierarchies do not enforce unique greatest lower bounds. Thus, to make the two hierarchies comparable the glb-types have to be removed from the manually built hierarchy. This is relatively straightforward since they are guaranteed not to contain any attribute-value pairs, so removing them does not remove any information.

Finally, lexical and grammar rules are removed. In constraint-based grammars, such as HPSG, it is relatively easy to incorporate rules in the hierarchy as one can view rules as constraints on types. For example, the *birule-head-first*, which specifies that a head which subcategorises for exactly one complement can combine with this complement to form a phrase, can be reformulated as the constraint in Figure 6.6. The type *birule-hf* can then be incorporated into the hierarchy, for example as a subtype of *synsem-struc* in the hierarchy fragment in Figure 6.3.

In the manually built hierarchy, rules are usually interwoven with the lexical hierarchy, i.e. a rule node may share an ancestor (*synsem-struc* in the example above) with a lexical entry. However, rules are easily identified as they never occur as ancestors of lexical entries. Hence, rules can be removed by deleting every node that is not an ancestor of a lexical entry.

where this kind of generalisation would be beneficial.

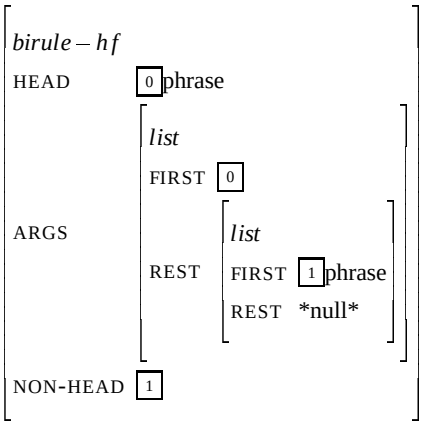


Figure 6.6: Birule-head-first

6.3 Galois Lattice Construction

After the hierarchies have been pre-processed and the lexicons have been extracted by compiling out the hierarchies, the Galois lattices can be built.[9] Table 6.1 shows the sizes of the four lexicons and their Galois lattices. The second column gives the number of lexical entries, i.e. the number of terminal nodes in the hierarchy. The third shows how many different path-value pairs occur in the lexicon. The fourth column shows the number of intermediate nodes (INs) in the manually built hierarchy.[10] Only intermediate nodes can be pruned. The root node has to be retained to ensure rootedness and the terminal nodes have to be retained to ensure that the derived hierarchy is sound (though path-value pairs are pruned from the terminal nodes if they can be inherited). For this reason training (and evaluation) only takes intermediate nodes into account. The number in brackets shows how many of these intermediate nodes are actually contained in the Galois lattice. Nodes that are not contained in the lattice correspond to generalisations that are not implicit in the lexicon but require background knowledge (see Section 4.2, page 103). The derived hierarchy will be evaluated with respect to the number of nodes in brackets, i.e. it is assumed that the manually built hierarchy contains only generalisations that are implicit in the lexicon. This is discussed in more detail below (see page 168).

[9] I am grateful to Steve Finch and James Curran for letting me use their implementations for Galois lattice construction.

[10] This is the number of intermediate nodes after the pre-processing has been done. Since pre-processing collapses some nodes and removes others the number of non-terminal nodes given in the table is lower than the number of non-terminals in the original LKB hierarchy.

For training, however, *all* intermediate nodes are used, i.e. the set of positive training examples comprises all 608 intermediate nodes in the LinGO ERG hierarchy. Strictly speaking maximum entropy training should only be done on those 473 nodes that are actually contained in the Galois lattice. However, since there are so few positive training examples it was decided to train on all intermediate nodes contained in the manually built hierarchy even if that means introducing some noise. The next column shows how many lexical entries there are on average for each intermediate node in the manual hierarchy. This is interesting because it shows that the Spanish hierarchy is actually very "sparse" in its entries. While for the Textbook and LinGO ERG hierarchies the ratio of intermediate nodes to lexical entries is about 1:11 for the Spanish hierarchy it is much smaller, i.e. about 1:4. To some extent this can be explained by the way in which the lexicon was developed, namely as a tool to illustrate one aspect of Spanish (i.e. clitics), rather than as a fully fledged Spanish lexicon. Because the focus is on illustrating a particular linguistic analysis, intermediate nodes are more important than entries, i.e. each linguistic class is only represented by a few entries. The final column in the table shows the number of intermediate nodes in the Galois lattice.

Lexicon	lexical entries	path-value pairs	man. hierarchy INs		man. entries per IN	Galois lattice INs
Textbook	507	2,568	44	(28)	11.52	1,664
Spanish	405	4,726	108	(50)	3.75	16,544
LinGO ERG	6,897	21,064	608	(473)	11.34	$>$ 7.5 Mil.

Table 6.1: Hierarchy sizes

Trying to establish the exact number of intermediate nodes in the Galois lattice for the LinGO ERG lexicon failed because the construction algorithm ran out of memory after finding 7.5 million nodes,[11] Even if the Galois lattice for LinGO ERG could be built, the training examples extracted from it would be highly imbalanced, i.e. at least 7.5 million negative examples *versus* 608 positive examples. Learning from highly imbalanced data is problematic, thus I decided to undersample the negative examples. This is discussed in detail in the next section.

However, in general the Galois lattices turn out to be much smaller than the worst case predictions. The worst case complexity of a Galois lattice is $2^{|\text{lexical entries}|}$ (see page 99). For

[11]This experiment has been conducted by James Curran who implemented a memory efficient version of the Galois lattice construction algorithm and ran it on his own computer to establish a lower bound for the size of LinGO ERG's Galois lattice.

the Textbook lexicon this would mean that the Galois lattice could contain up to 2^{507} nodes. However, it actually contains between 2^{10} and 2^{11} nodes. This shows that lexicons have very sparse context tables, which means that their Galois lattices are a lot smaller than the worst case predictions.

6.4 Sampling

The Galois lattice for the LinGO ERG lexicon contains at least 7.5 million non-terminal nodes which means that the ratio of positive *versus* negative examples is at least 1:11,000. Learning from data that is imbalanced to such a degree is often very difficult because imbalanced data cause many classifiers to be biased against the minority class. Many strategies have been proposed to deal with class imbalance[12] but the most straightforward ones are minority class oversampling, i.e. adding additional copies of instances that belong to the minority class, and majority class undersampling, i.e. removing some instances that belong to the majority class from the training set. There is some evidence that undersampling may yield better results than oversampling (Japkowicz 2000b; Ling and Li 1998). Majority examples can be removed randomly from the training set or a more sophisticated selection strategy can be used. Since there is no strong evidence that sophisticated undersampling outperforms random undersampling (see Japkowicz 2000a for neural networks *vs.* Kubat and Matwin 1997 for memory-based learning), I chose random undersampling.

If creating the whole Galois lattice was possible, sampling could be done by randomly removing nodes from it. However, since creating the full lattice for LinGO ERG requires more memory than I had available,[13] this was not practical. Furthermore, the imbalance in the training set is quite severe and a lot of nodes would have to be removed to get a training set that is fairly balanced. Therefore, it seems better to randomly create negative nodes. Nodes can be created by intersecting sets of lexical entries. First, the number of nodes to be intersected, n, is chosen randomly, ranging from 2 to $|e| - 1$ (where $|e|$ is the number of lexical entries). Each n in the permissible range has equal probability. Then n nodes are chosen at random and intersected. If the intersection is non-empty and is not yet contained in the sampled lattice it is added. Otherwise a new n is chosen until a new node has been found.

Ideally, one would want the negative nodes to be normally distributed with few nodes at

[12] See v. Chawla *et al.* (2002) for a recent overview.

[13] Constructing the Galois lattice for LinGO ERG required in excess of 2 GB RAM, even with an implementation that was optimised for memory rather than speed.

lower and higher levels of the lattice and many nodes in the middle, as this is the distribution expected for a complete Galois lattice. One might think that the suggested sampling method is biased towards higher levels because intersecting a small number of nodes might still lead to a new node fairly high in the lattice if the intersected nodes do not have much in common, while intersecting a large number of nodes will never lead to a new node that is low in the lattice. Put differently, if n nodes are intersected the extension of the resulting node will contain at least n nodes but it might contain many more nodes. However, it turns out that the suggested sampling method leads to negative nodes that are more or less normally distributed. The reason for this is probably that a new n is chosen at every step and —since there are fewer negative nodes at the highest level— the algorithm is less likely to find any of them at first trial and then chooses another n. A small n is more likely to give rise to a node in the middle of the lattice and since there are more nodes in the middle it is less likely that that node is already contained in the training set. Hence, while choosing n and the nodes to be intersected randomly increases the probability that the intersection is small, a small intersection is less likely to lead to a new node. A consequence of this is that the algorithm is somewhat computationally expensive as most sets of nodes do not lead to a new node when intersected.

Choosing a good sampling rate can only be done by trial and error, as it seems that the ideal sampling rate may be problem-dependent (Estabrooks and Japkowicz, 2001). The experiments discussed in Chapter 7 mainly used a sampling rate of 1:1, i.e. perfectly balanced training data with 608 positive examples and 608 negative examples. However, Section 7.3.3 discussed how the learning algorithm performs if different sampling rates are chosen.

Undersampling of course introduces some noise. This affects in particular inter-node measures as it is no longer possible to accurately compute the children of a node since some of them may be missing from the sampled lattice. This in turn means that errors are introduced when co-parents and siblings are computed as these require a reliable computation of the children relation. However, the parent relation can still be computed accurately as it is defined with respect to positive nodes only and all positive nodes will be contained in the sampled lattice.

6.5 Evaluation

There are three basic ways in which the quality of an automatically built hierarchy can be assessed. First, it can be compared to a gold standard, where the comparison can either been done automatically or manually. Second, it can be manually evaluated by human subjects (without reference to a gold standard). Finally, its benefit for a given application can be assessed.

In previous approaches to lexical inheritance hierarchy learning, the derived hierarchies were either not evaluated at all (Petersen, 2001) or a gold standard in the form of a manually built hierarchy for the lexicon was assumed and the derived hierarchy was manually compared to it (Barg 1996a, and Light 1994). However, evaluating a hierarchy by comparing it to a gold standard is not without problems. As was discussed in Chapter 3, hierarchy construction is to some extent a subjective task; there is no such thing as a unique best hierarchy for a lexicon. Yet assuming a gold standard is based on the assumption that there is exactly one solution to the problem. This makes it very hard to obtain good results since it is theoretically possible that a perfectly plausible hierarchy is generated which nonetheless deviates in certain aspects from the manually built gold standard. However, while it is not entirely possible to assess the absolute merit of a hierarchy, one would expect plausible hierarchies to be closer to the manual hierarchy than implausible ones. Consequently, the gold standard approach should at least make it possible to *rank* a set of automatically generated hierarchies.

The evaluation problem faced by automatic hierarchy construction is shared by other NLP problems for which no absolutely best solution can be determined, such as sentence ordering in natural language generation. In these areas evaluation is often done by employing human subjects who manually rate the results. For sentence ordering this is fairly straightforward as it is not too difficult to rate the coherence of a text. However assessing an inheritance hierarchy manually is much harder. One reason for this is that hierarchies are two-dimensional and therefore less easy to process by humans than a one-dimensional sequence of sentences. Hierarchies are also typically fairly large, with hundreds of nodes and complex connections between nodes, which makes it often infeasible to even display the hierarchy let alone have it analysed by human subjects.[14] Finally, assessing a hierarchy usually requires good knowledge of the lexicon, such as the exact meaning of different attributes etc. This suggests that hierarchies can only really be assessed by somebody with a certain degree of linguistic expertise. Therefore a large scale evaluation with human subjects is not practical.[15]

A third evaluation strategy would be some kind of task-based evaluation. For example one could make the assumption that a good hierarchy makes it easier to deal with unknown words in NLP tasks such as parsing, as a hierarchy that generalises well should make it easier to infer

[14] Any attempt to do manual evaluation of complex hierarchies would probably have to make use of suitable data exploration and display tools to facilitate the evaluation and make it easy to navigate in the hierarchy.

[15] Barg (1996a) and Light (1994) evaluate their hierarchies themselves but their lexicons are fairly small. For example, Barg's lexicons contain around 15 entries and 10 attribute-value pairs per (compiled out) entry. Light does not give exact number but mentions that his lexicons contained "a small number of words". The test lexicons used in this thesis, on the other hand, contain 400 to 500 words and between 2,500 and 4,700 path-value pairs.

properties of new words that cannot be inferred on the basis of context alone. This has already been discussed in detail in Sections 1.1 (page 5f) and Section 1.2 (page 7f). Being able to infer additional properties from the lexicon should then result in an increased parsing performance. For example, verb subcategorisation pattern are often related in a predictable way (see Levin (1993)). Cooking verbs, for instance, exhibit a causative/inchoative alternation:

(6.1) a. Jennifer baked the potatoes.

b. The potatoes backed.

A hierarchy which allows one to infer that an unknown verb *boil* belongs to this class and thus can be used transitively as well as intransitively is surely beneficial for parsing. Hence, hierarchies that allow one to infer additional properties for a new word should increase parsing performance. However, due to time constraints it was not possible to pursue this approach here.

There are a few other problems with this approach, too. First, the fact that hierarchies are not evaluated directly but indirectly by their effect on parsing performance means that bad absolute results may be due factors other than the hierarchy itself. For example, the heuristics for inserting a new word into the hierarchy may be flawed.[16] However, provided that all hierarchies use the same insertion strategy, it should at least be possible to rank them. Another potential problem is that a task-based evaluation may not be fine-grained enough. For example, a parser would probably be able to parse the sentence *The potatoes boiled* even if it had only seen *boil* in a transitive context before. Hence, whether one can infer the causative/inchoative alternation from the hierarchy may actually be immaterial for parsing performance and as a consequence, task-based evaluation may not be able to differentiate between hierarchies which allow the inference of this alternation and those which do not. Similarly, many inferences may be of a nature which does not have direct influence on parsing performance. For example, a hierarchy may capture the generalisation that nouns that end in *-ee* are animate (see page 8) but this may not have a direct influence on the parsing performance for sentences containing an unknown noun ending in *-ee*. In this context, it may also be important how the data for parsing is chosen. Ideally, one would want to choose texts which allow the parser to benefit from good generalisations over the existing lexicon. However, this may mean that one has to know which generalisation a good hierarchy for the input lexicon should capture. But if one

[16] If the hierarchy is monotonic, the most straightforward way for inserting a new entry is by making it a subtype of the most specific, compatible type. This strategy is pursued by Barg (1996b). However, for real-world applications a more complex strategy may be necessary. Barg and Walther (1998) and Barg and Kilbury (2000), for example, propose a distinction between *generalisable* and *specialisable* information. It may also be necessary to employ some kind of non-monotonic reasoning, e.g. insert the new entry under the class that is considered the most likely given the current knowledge and later revising this decision if conflicting information is encountered.

knows this already, a more direct evaluation would be possible, such as simply checking which hierarchies contain these generalisations.

An alternative to the parsing-based evaluation method outlined above would be to take the parser out of the equation and simply evaluate hierarchies in terms of how much information can be inferred about partly specified new words. This would mean that the inferred information does not have to have an effect on parsing – the mere fact that it *can* be (correctly) inferred would be taken as an indication of hierarchy quality. Since it is also be possible to infer wrong information, e.g. because a hierarchy captures spurious generalisations, it would be necessary to assess the correctness of the inferred information and trade off correct and incorrect inferences. In a way, this strategy could be seen as evaluating how easy it is to extend the lexicon (manually or automatically). That is, a hierarchy that allows many correct inferences for new words is easier to extend than a hierarchy which allows only few or incorrect inferences.

On balance a gold standard approach was adopted for evaluation. While it does have problems it is more practical than the alternatives. In contrast to Barg and Light, I will opt for an automatic evaluation method, which makes it possible to evaluate large lexicons. This requires a graph matching algorithm to map the manually built hierarchy to the derived one and a similarity measure to assess the distance between those two. The matching algorithm matches nodes in one hierarchy to nodes in the other. It is described in more detail below but I will start by describing the similarity measure.

The similarity between two hierarchies can be assessed by using the standard measures of **precision** (*P*) and **recall** (*R*). Precision and recall are defined in terms of the number of **true** and **false positives** (*TP* and *FP*, respectively) and **true** and **false negatives** (*TN* and *FN*, respectively). For automatic hierarchy construction these sets are defined as follows:

	Manually built	
automatically built	retained	pruned
retained	TP	FP
pruned	FN	TN

Precision is a measure of what proportion of objects selected by the system are correct. It is defined as the number of true positives divided by the sum of true positives and false positives, i.e. the number of correctly retained nodes divided by the overall number of retained nodes:

$$precision = \frac{TP}{TP + FP}$$

Recall is a measure of what proportion of objects that should have been selected were selected. It is defined as the number of true positives divided by the sum of true positives and

false negatives, i.e. the number of correctly retained nodes divided by the number of nodes that should have been retained:

$$recall = \frac{TP}{TP+FN}$$

There is usually a trade-off between these two measures. For example one can vary the decision threshold and only prune nodes if $P(prune) > 0.7$ instead of pruning nodes if $P(prune) > 0.5$. This will typically lead to more nodes being retained and a higher recall. But it will probably also mean a lower precision as it is likely that some of the nodes with a pruning probability between 0.5 and 0.7 are false positives.

Because of this trade-off it is useful to combine precision and recall into a single measure so that systems with different recall and precision values can be compared. This is achieved by the so-called **f-score** (F), which is defined as follows:

$$F = \frac{1}{\alpha \times \frac{1}{P} + (1-\alpha) \times \frac{1}{R}}$$

The parameter α is a weight that determines the relative importance of precision and recall. In general recall is more important than precision if a semi-automatic approach is chosen (see below, page 183). However, it is difficult to quantify just how much more important it is. Therefore, I will assume equal weights. The formula for the f-score then becomes:

$$F = \frac{2PR}{P+R}$$

The graph matching can be strict, i.e. two nodes are matched if they have the same extension (or if they have the same underlying intension), or it can be based on similarity rather than identity, i.e. two nodes are matched if their similarity is above a user-set threshold. The latter is sometimes also called *error-correcting* graph matching (Messmer, 1996).

Originally, I opted for error-correcting matching to take account of the fact that some pruning errors are worse than others and a pruning algorithm should be given some credit if it wrongly retained a node that is very similar to a node that should have been retained. That is, even if few nodes are correctly retained, the derived hierarchy may still be relatively good if its nodes are very similar to the nodes in the manually built hierarchy. Error-correcting graph matching makes a difference between pruning algorithms that retain nodes that are very similar to the ones that should have been retained and algorithms that retain nodes that are very different. In this respect error-correcting matching is also more fine-grained than strict matching.

Error-correcting matching requires a measure of node similarity and a threshold. If the similarity of two nodes is above the threshold they are matched. In my implementation, node

similarity was defined as the proportion of overlap in intension and extension between two nodes. Intensional overlap was defined as the number path-value pairs that occurred in the (underlying) intension of both nodes times 2 and divided by the sum of path-value pairs in both nodes. Extensional overlap was defined similarly.

Two nodes were matched if they overlapped in both intension and extension and if their combined intensional-extensional overlap was at least 50%. For example, two nodes could be matched if there was a 50% overlap in both intension and extension. But they could also be matched if the overlap in extension was small but the overlap in intension was nearly 100%. A greedy algorithm was then used to optimise the hierarchy matching globally.

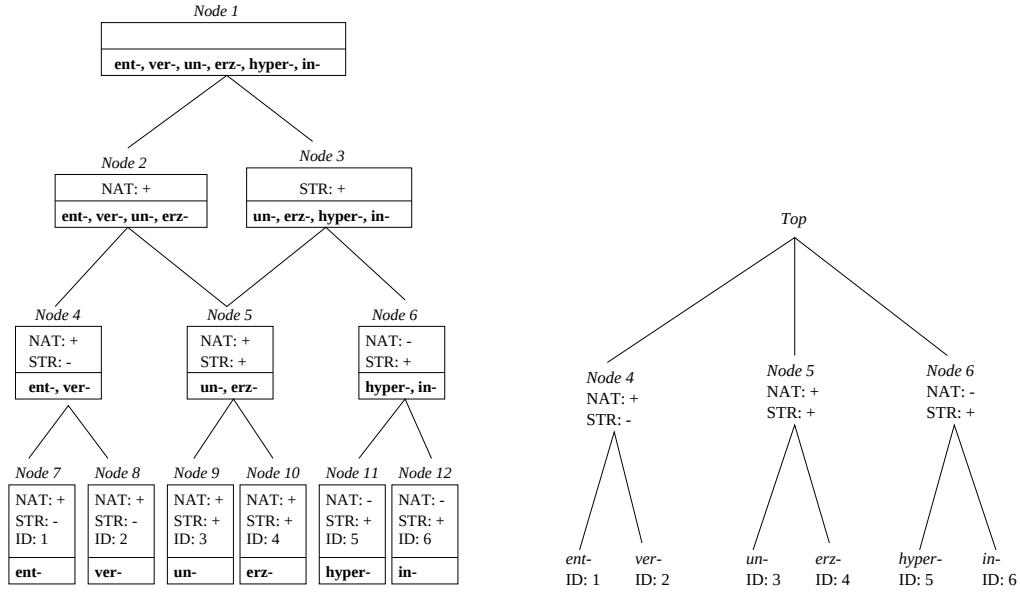
However, I later abandoned error-correcting matching because it has several disadvantages. For a start it is difficult to determine a good value for the threshold. If the threshold is too high error-correcting matching will behave like strict matching, if it is too low nodes will be matched even if they are only marginally similar. Ideally the threshold would have to be determined empirically by trying different thresholds and assessing which of them performs best. However this would involve the use of another evaluation method to evaluate the merit of different thresholds. The use of a threshold also waters down the precision-recall tradeoff, especially if the threshold is relatively low. This means that the performance of a pruner is very difficult to interpret. For these reasons, I decided to use strict matching instead.

Strict matching does not require a similarity threshold and consequently a variation in the decision threshold has more immediate effects (i.e. the recall-precision tradeoff is not watered down). This makes it much easier to interpret the results. The matching itself is also more straightforward and arguable more objective.

However, it is also a much more conservative measure as two nodes are only matched if their extensions are identical.[17] Furthermore, it is based on a very coarse similarity definition; only pruning algorithms that retain correct nodes score high. No distinction is made between a pruning algorithm that retains nodes that are very similar to the nodes in the original hierarchy and a pruning algorithm that retains nodes that are very different. This means that it is extremely difficult to score high on this evaluation criterion (see Section 7.1 for a discussion of upper and lower bounds).

Once two hierarchies have been matched using strict matching, they are evaluated by calculating precision, recall and f-score as defined above. The set of true positives is calculated with respect to those intermediate nodes of the manual hierarchy that are actually contained in

[17] This is equivalent to matching two nodes if their underlying intensions are identical.



(a) Galois (semi-)lattice

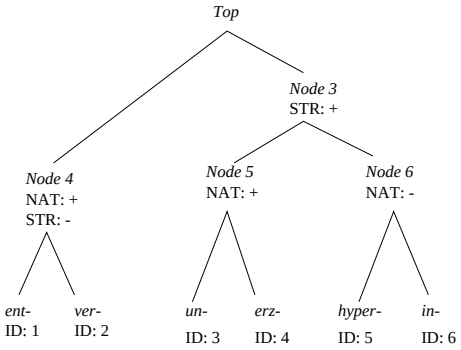
(b) Manually built hierarchy

Figure 6.7: Galois lattice and manually built hierarchy

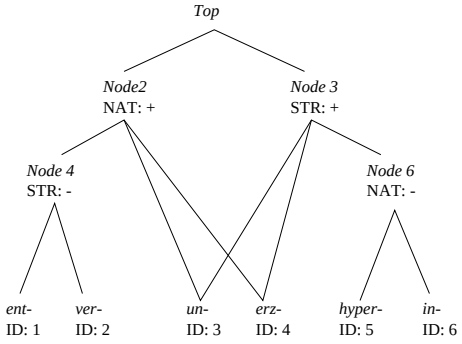
the Galois lattice. For example, for the Textbook hierarchy there are 28 true positives (rather than 44) and for the Spanish hierarchy there are 50 true positives (rather than 108). This means that a pruning algorithm can in theory achieve an f-score of 100% on every lexicon, which makes it easier to compare the performance for different lexicons.

As an illustrative example of how this evaluation method works for different derived hierarchies consider the prefix lexicon (introduced in Section 3.3) again. Figure 6.7(a) shows the Galois (semi-)lattice for the lexicon. In Section 3.3 it was argued that a good manually built hierarchy for the data should represent the fact that three distinct prefix classes can be identified for German (see the discussion on pages 68ff), hence a manually built hierarchy could look like the one shown in Figure 6.7(b). This can be derived form the Galois lattice by retaining nodes 4, 5, and 6 and pruning all other intermediate nodes. Section 3.3 also gave some examples of sound hierarchies that can be built for the lexicon (page 71). Three of them are shown in Figure 6.8.

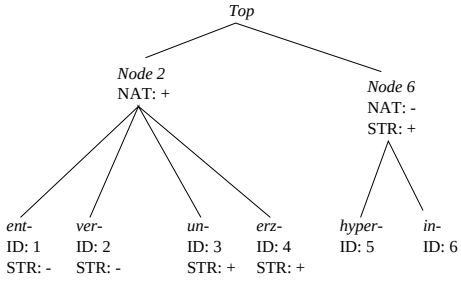
According to the criterion outlined above, the four hierarchies will be evaluated as follows: Hierarchy 5 contains three true positives (nodes 4, 5, and 6) and one false positive (Node 3). Hence, recall is $\frac{3}{3}$, precision is $\frac{3}{4}$ (i.e. three true positives out of four retained intermediate



(a) Hierarchy 5



(b) Hierarchy 1



(c) Hierarchy 6

Figure 6.8: Derived hierarchies

nodes) and the f-score is 0.8571 (i.e. 85.71%). Hierarchy 1 contains only two true positives and also two false positives, resulting in a recall of $\frac{2}{3}$, a precision of $\frac{2}{4}$ and an f-score of 57.14%. Hierarchy 6 contains one true positive and one false positive which means a recall of $\frac{1}{3}$, a precision of $\frac{1}{2}$ and an f-score of 40%. According to this criterion, Hierarchy 5 would be considered the best and Hierarchy 6 the worst of the three hierarchies. This correlates very well with the intuitive ranking of the three hierarchies (cf. the discussion on page 70f).

There is one objection one can have to either of the matching strategies (i.e. strict matching and error-correcting matching), namely that they have not been empirically validated. This is particularly true for error-correcting matching. Strictly speaking, it would be necessary to show empirically that hierarchies that score high according to a evaluation measure are hierarchies that are also judged as fairly good by humans (or that at least are judged to be fairly similar

to the gold standard hierarchy). To prove that this is the case one would have to conduct an experiment where some automatically generated hierarchies are evaluated according to the evaluation criterion and then compare this to an evaluation by a human subject (or a set of human subjects). However, as was argued above, human evaluation of automatically generated hierarchies is difficult in practice because the hierarchies can be very large and evaluating them involves a certain degree of linguistic expertise.

The objection is less true for the strict matching approach as it does not involve a special purpose similarity measure for nodes. Instead similarity is defined as identity of extensions. This is justified if one views a manually built hierarchy as being derived from an underlying Galois lattice (by a human rather than by a machine). The extension of a node indicates from which node in the Galois lattice it was derived. If a node in the manually built hierarchy and a node in the automatically constructed hierarchy have identical extension they will have been derived from the same node in the underlying Galois lattice. In this respect matching on the basis of extensional identity is a very plausible and objective criterion.

However, while the matching itself is fairly objective, it is still possible that retaining some nodes is more important for the overall quality of the hierarchy than retaining others. This is not captured by the strict matching method; it might still be that a human arrives at a different assessment for a hierarchy than this matching algorithm. Likewise it is possible that some combinations of nodes are better than others and this again is not captured as the evaluation method defines hierarchy similarity in terms of node similarity. But overall, strict matching is a fairly objective evaluation criterion.

6.6 Summary

This chapter presented the system used in the experiments described in the next chapter.

Figure 6.9 gives an overview of how the system is applied to new data and tested. The original, manually built hierarchy is first pre-processed. Pre-processing flattens the feature structures, re-represents reentrancies, abandons Minimal Introduction and removes rules and glb-types. The pre-processed hierarchy is then compiled out and a Galois lattice is constructed for the compiled out lexicon. After that, a pruning algorithm is applied to the lattice and the lattice is pruned into an (untyped) inheritance hierarchy. Pruning is not restricted to the maximum entropy pruner described in the previous chapter, other pruning algorithms can be tested with this set-up as well. Section 7.2 in the next chapter, for example, tests Petersen's 2001 pruning algorithm.

The automatically constructed hierarchy then has to be evaluated. Several different evaluation strategies would be possible. It was argued that a task-based evaluation required too much extra machinery and may not be fine-grained enough. An evaluation strategy that makes use of human subjects also poses several difficulties. Evaluating big hierarchies manually is a difficult task due to the inherent complexity of inheritance hierarchies and it requires at least a suitable display-tool and possibly also a data exploration tool. It also requires subjects with a certain degree of linguistic expertise. A third evaluation strategy involves evaluating the derived hierarchy against a gold standard. This strategy is a bit artificial as it assumes that there is one best hierarchy for a given lexicon which is clearly not the case. Consequently, it is very difficult to achieve high scores if a gold standard based evaluation method is used. Despite this, it is the most feasible evaluation strategy and has been used in previous approaches (Barg 1996a, Light 1994). However, unlike previous approaches I proposed an automated evaluation method, in which the derived hierarchy is automatically matched to the original hierarchy. Using an automated strategy makes it possible to evaluate hierarchies that are much large than those discussed by Barg and Light. Once two hierarchies have been matched, their similarity is assessed by counting the number of nodes that occur in both hierarchies (i.e. nodes that have identical extensions) and calculating precision, recall and the f-score accordingly.

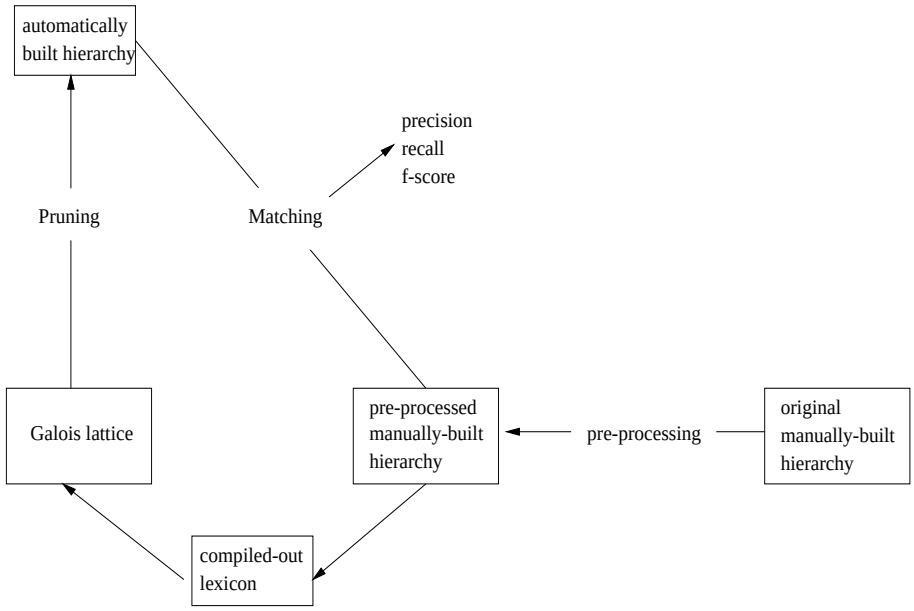


Figure 6.9: Testing

Figure 6.10 gives an updated overview of how the maximum entropy model is trained. The Galois lattice of the training hierarchy is very large. However, most nodes contained in it are negative training examples (i.e. do not occur in the corresponding manually constructed hierarchy). This means that the complete training set is highly imbalanced. Learning from imbalanced data, however, is difficult as it causes the learner to be biased against the minority class. Therefore it was decided to incorporate only a subset of negative nodes in the training set. These are sampled by randomly intersecting a set of terminal nodes and checking whether the intersection gives rise to a new node. It turns out that this sampling strategy leads to negative examples that are approximately normally distributed, i.e. most negative nodes will be found at intermediate levels in the lattice.

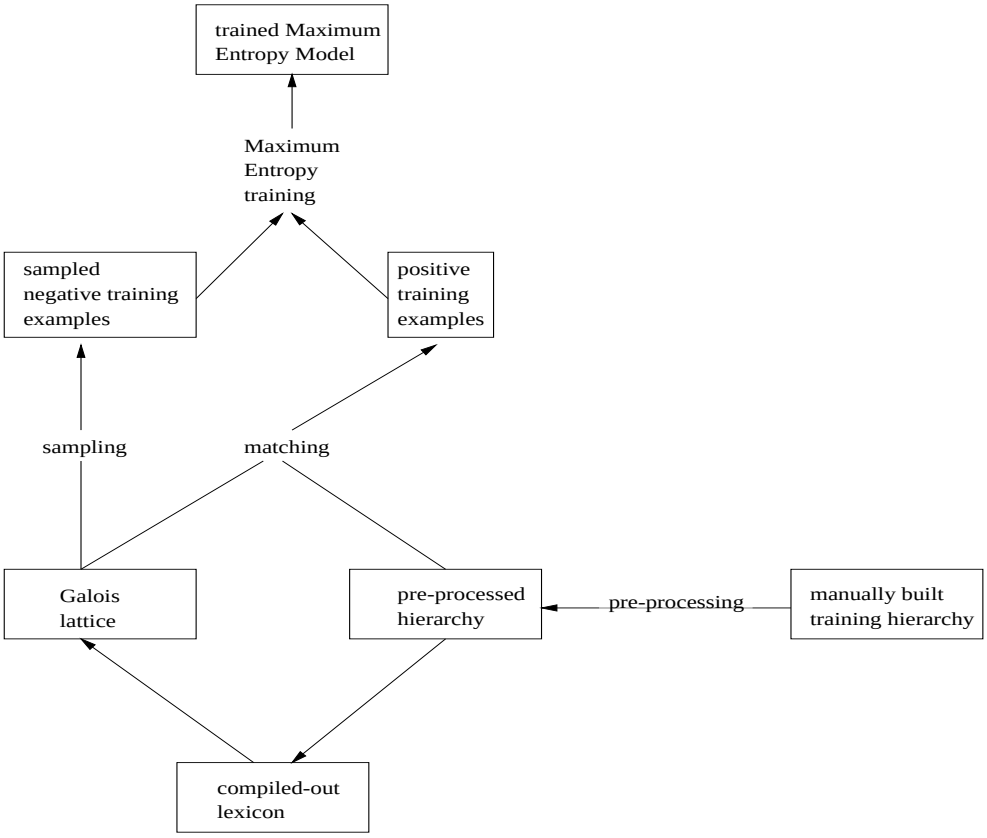


Figure 6.10: Training with sampling

Chapter 7

Experiments

To test the maximum entropy pruner, three different pruning architectures have been implemented. Two of these are semi-automatic, i.e. the pruner does only part of the pruning and leaves the rest to a linguist. Using a semi-automatic architecture is motivated by the fact that the task of constructing a lexical inheritance hierarchy is quite subjective and does not have a unique solution (see Chapter 3). There are usually several equally good solutions to the problem. Which one is chosen by a linguistic expert is influenced by many factors such as personal taste and the task for which the hierarchy will be used. Hence, while it may be possible for an automatic system to derive a reasonably good hierarchy, it is impossible for such a system to derive exactly *the* hierarchy that the user would have constructed if she had done it manually. This can only be achieved by a semi-automatic pruning architecture.

However, there may be situations where semi-automatic construction is not an option, for example because speed is important or because no linguistic expert is available to do the semi-automatic construction. In these situations, a fully automatic approach is the better option. Therefore, the maximum entropy pruner has also been tested as part of an automatic architecture.

For comparison, Petersen's (2001) pruning method has also been tested. Petersen uses a symbolic pruning method, i.e. the hierarchy is pruned using a pre-defined minimality criterion (see Section 2.5). It would have been interesting to also compare the maximum entropy pruner with the system proposed by Barg (1996a). Unfortunately, this is not an option since the latter is implemented for DATR hierarchies and these are formally very different from the hierarchies discussed here (see Section 2.5). Therefore Barg's system cannot be applied to LKB hierarchies.

This chapter starts with an overview of upper and lower bounds of performance for the task (Section 7.1). Section 7.2 discusses the results achieved by Petersen's method. Semi-automatic maximum entropy pruning is discussed in Section 7.3: Section 7.3.1 discusses the first of the two semi-automatic architectures. This architecture is non-interactive; the pruner produces a partially pruned lattice which can then be manually post-processed by a linguist. The pruner is conservative about the nodes it prunes, i.e. it leaves in nodes which are not clear cases for pruning. Ideally, the partly pruned lattice should contain all nodes that a linguist might want to retain and few additional nodes.

Sections 7.3.2 to 7.3.5 discuss four further experiments that have been conducted for this architecture. Section 7.3.2 investigates how varying the pruning threshold influences the results. The higher the threshold, the higher the pruning probability of a node has to be before the node can be pruned, i.e. the fewer nodes will be pruned. Hence the threshold influences the precision-recall tradeoff; a high threshold will typically lead to a higher recall but a lower precision.

Section 7.3.3 investigates the role of the sampling rate for training the maximum entropy pruner. Remember, that the negative training data have to be sampled because the complete set is too big to train on (see Section 6.4). Finding the best sampling rate is not straightforward as it depends on the learning problem and method (Estabrooks and Japkowicz 2001, p. 79). Section 7.3.3 reports an experiment in which different sampling rates are used.

The next section (Section 7.3.4) explores the different effects of intra- and inter-node measures. Inter-node measures are generally more noisy than intra-node measures (see Section 5.2). Computing them also tends to be computationally more expensive. Hence, it would be nice to know how much they do contribute to the overall performance of the maximum entropy pruner and whether they could be abandoned.

Section 7.3.5 discusses how much a different quantisation method for the contextual predicates influences the results of Maximum Entropy Pruning. As discussed in Section 5.2.3, the maximum entropy pruner uses an interval-based quantisation method (Fayyad and Irani, 1993), i.e. real-valued maximum entropy measures are split into intervals. This quantisation method has the disadvantage that order information is lost. Section 7.3.5 explores whether an order-preserving binary quantisation method leads to better results.

Section 7.3.6 introduces the second semi-automatic architecture. Instead of separating automatic pruning and manual post-processing, the two are interleaved, i.e. the maximum entropy pruner goes through the lattice level by level and prunes nodes which have a fairly high prun-

ing probability, while nodes with a lower pruning probability are shown to a linguist who can then decide whether to prune or retain the node. Interleaving the automatic pruning and the manual post-processing should lead to better results, as it means that some mistakes (i.e. false positives) are corrected immediately and thus cannot percolate down the hierarchy.

An automatic architecture is discussed in Section 7.4. Building a hierarchy automatically may mean sacrificing some amount of hierarchy quality but it does not require any human intervention.

7.1 Upper and Lower Bounds

The absolute performance of a new algorithm is often not particularly informative, since some tasks are inherently less complex than others. A high accuracy in a task that is is not very complex is often less of an achievement than a lower accuracy in a task that is highly complex. For example, it is not too difficult to achieve a high accuracy in part-of-speech tagging where state-of-the-art systems achieve accuracies around 95% to 97%[1] and a simple algorithm which assigns the most frequent tag for a given lexeme already achieves 90% accuracy.[2] However, it is much more difficult to achieve a good result in machine translation or natural language generation. Establishing an upper and a lower bound for a task allows one to estimate its inherent difficulty and then assess the performance of a system relative to this difficulty.

The upper bound is usually taken to be the performance humans can achieve. To establish an upper bound for automatic hierarchy construction one would need to give a flat lexicon to several human experts and ask them to build an inheritance hierarchy for it. However, building an inheritance hierarchy manually is time-consuming and involves a lot of expertise (see 164) and conducting such an experiment was beyond the scope of this thesis.

Another way of establishing an upper bound would be to look for two different manually built hierarchies for the same lexicon and compare these. However, it is highly unlikely that one would be able to find two different hierarchies for the same lexicon, as hierarchies are usually built at the same time as the lexicon is developed and it is difficult to imagine a scenario where two different hierarchies for one lexicon would exist. Comparing two hierarchies for two different lexicons would not work either, even if the lexicons were quite similar in terms of coverage etc., because even small differences in the attribute-value pair sets can make two hierarchies incomparable (see the discussion about count and mass nouns in Section 5.2).

[1] Manning and Schütze 1999, p. 371.
[2] Charniak 1993, p. 49.

While it is difficult to determine an exact upper bound for automatic hierarchy construction one can assume that it is considerably lower than 100% due to the fact that there are usually several equally good solutions. In this respect automatic hierarchy construction differs from tasks such as part-of-speech tagging. While humans typically agree on the correct tag sequence for a sentence (if the tag set is fixed in advance), two linguists will not normally agree on the best hierarchy for a lexicon. Regarding its inherent difficulty, automatic hierarchy construction is therefore much more on a par with tasks such as machine translation or natural language generation.

The lower bound, also called the *baseline*, is the performance one can achieve with the most naive algorithm. In many NLP classification tasks the baseline is taken to be the performance achieved by always assigning the class which is most likely for a given instance. For example, a baseline for a part-of-speech tagger can be established by assigning each word in the test corpus its most frequent part-of-speech tag in the training corpus. The lexeme *corner*, for instance, can be either a noun or a verb, as in example (7.1). However, it is more often used as a noun. A baseline part-of-speech tagger would therefore assign the tag *noun* to all occurrences of the lexeme *corner*.

(7.1) a. The police waited at the corner of the hotel.

b. The police tried to corner the thief.

For Galois lattice pruning the most probable class for a node is *prune* as a Galois lattice generally contains many more nodes that should be pruned than nodes that should be retained (see Galois lattice size vs. lexicon size in Table 6.1, page 161). However, pruning every node would result in no true positives and consequently in an f-score of 0. This is therefore not a useful baseline for the task.

Another way of establishing a baseline is by assigning a class randomly, where each class is equally likely to be chosen. This baseline is sometimes used in tasks such as stochastic parse selection (e.g. Riezler *et al.* 2002) and text summarisation (e.g. Burstein *et al.* 2001). Transferring this to Galois lattice pruning means that every node will be assigned a class randomly, where the two classes have equal probabilities, i.e. $P(prune) = P(retain) = .5$. The expected performance that can be achieved by applying this strategy can be determined mathematically. For example, there are 1,664 intermediate nodes in the Textbook lattice. Since the random pruner assigns both classes with the same probability, on average half of the nodes will be retained, i.e. 832 nodes. Likewise half of the 28 positive nodes in the lattice will be retained on

average, i.e. 14. From these figures one can calculate the values for precision, recall and the f-score. The results for the Textbook lexicon and the Spanish lexicon are shown in Table 7.1. In the table, the second column shows the expected average number of retained nodes, the third column shows the expected average number of true positives and the remaining three columns give the values for recall, precision and the f-score, respectively.

Lexicon	retained	TPs	R %	P %	F %
Textbook	832	14	50.00	1.68	3.25
Spanish	8,272	25	50.00	0.30	0.60

Table 7.1: Expected values for Random Uniform Pruning

The figures show that automatic inheritance hierarchy induction is a difficult task. It is even more difficult for the Spanish lexicon than for the Textbook lexicon because for the former the proportion of positive (intermediate) nodes to all (intermediate) nodes in the Galois lattice is much smaller. But in many respects this baseline is arbitrary. For example, using a uniform probability for assigning classes is a bit counter-intuitive as it is known that the class *prune* is much more common. An alternative would be to assign the class *prune* with a higher probability. Ideally one would want to choose a probability that reflects its frequency relative to the overall number of nodes in the Galois lattice. However, this relative frequency varies greatly across lexicons. In the Textbook lattice, the class *prune* occurs 59 times more often than the class *retain*, while in the Spanish lattice, it occurs 330 times more often. The ratio for a new lexicon will not normally be known and using the ratio of the training lexicon instead is again a bit arbitrary[3] given the large variation in ratios. It might be possible to find an average ratio on the basis of several lexicons/hierarchies but to find a reliable average one would need more lexicons than are currently available.

The situation is further complicated by the fact that several maximum entropy models need to be trained because the sampling of the training set (see Section 6.4) introduces some randomness and one would expect some models to underperform due to a non-representative training set. Different models will usually differ in the number of nodes they retain. Therefore it is not possible to establish, for example, a general baseline for the number of true positives and evaluate each model's performance against it since the number of true positives will be influenced by the number of retained nodes, i.e. models which retain more nodes are more likely to

[3]Establishing the ratio for the training lexicon used here is difficult anyway because the Galois lattice for it could not be generated due to insufficient memory resources.

retain more true positives. However, it is possible to establish a unique baseline for each model by estimating how significant the number of true positives found by the model is in relation to the number of retained nodes. In other words, if a model retains n nodes, among which l are true positives, is this a significant result? Or is it likely that the same result could be obtained by retaining n nodes randomly? If it is highly unlikely that the results could be obtained by a random selection it can be concluded that the model has learned something from the training data and that this enables it to make a selection that is better than chance.

The average number μ of true positives one would expect if n nodes were selected can again be estimated. It is given by:

$$\mu = n \times \frac{k}{m}$$

where k is the number of positive intermediate nodes in the Galois lattice (28 in the case of the Textbook lexicon), and m is the number of all intermediate nodes in the Galois lattice (1,664 for the Textbook lexicon). To estimate how likely it is that the number of true positives for a model is due to chance one also needs to calculate the variance v for a random selection of n nodes. This is given by:

$$v = \mu \times (1 - \frac{k}{m})$$

It is then possible to calculate the z-score that the number of true positives x selected by a given model belongs to a distribution with mean μ and variance v:

$$z = \frac{x - \mu}{\sqrt{v}}$$

The probability that the performance of the model is due to chance can then be found by looking up the z-score in a standard normal distribution table.

For example, if a pruner retains 1,310 nodes, among which are 27 true positives, is this a significant result? If 1,310 nodes were retained randomly the expected number of true positives is 22.04 and the expected variance is 21.67. While the model performs better than this, this difference turns out to be not statistically significant, since:

$$z = \frac{27 - 22.04}{\sqrt{21.67}} \approx 1.07 \qquad p = 0.1423$$

i.e. there is still a one in seven chance that the performance of the model is due to chance.

It should be stressed here that the z-score only gives an indication about the performance of *one* particular model, i.e. the z-score allows one to estimate how likely it is that the performance of a given model when applied to a given lexicon is due to chance. This is different from *significance testing* as it is usually done in NLP. Ideally one would want to apply the system

to a large number of test lexicons and then estimate how likely it is that a good performance across these lexicons is due to chance. That way it would be possible to show whether the performance of the system is significant in general. Unfortunately, this was not possible here because only two test lexicons were available. Instead the argumentation is more indirect: models that performed well on one lexicon were applied to the other lexicon and if they also performed well on the second lexicon, this was taken as some evidence that that particular model performs well across lexicons.

7.2 Minimal Path-Value Pair Pruning

Petersen (2001) proposed a pruning method (henceforth *Minimal Path-Value Pair Pruning*) that does not make use of machine learning but takes a rule-based approach: a Galois lattice is pruned into an inheritance hierarchy by only retaining nodes which introduce new path-value pairs. This pruning method results in a hierarchy that is minimal with respect to path-value pairs, i.e. every path-value pair occurs exactly once. The motivation for employing this method is that lexical inheritance hierarchies are supposed to reduce redundancy (see the discussion in Chapter 3) and hierarchies derived by Minimal Path-Value Pair Pruning are minimally redundant with respect to path-value pairs.

The same pruning method has been suggested by Godin *et al.* (1998) for the related task of automatically building class hierarchies in object oriented design. However, while conciseness may be the most important criterion for the quality of a class hierarchy in object oriented programming, lexical inheritance hierarchies should also be linguistically plausible. In Chapter 3 it was argued that hierarchies that are minimally redundant with respect to path-value pairs are often not particularly plausible. One reason for this is that they often contain many nodes whose overt intension consists of only one path-value pair and these nodes usually do not correspond to linguistically meaningful generalisations.

To investigate empirically whether aiming for hierarchies which minimise the number of path-value pairs is a good strategy, the lattices for the Textbook and the Spanish lexicon have been pruned using Petersen's method and then evaluated against the original hierarchies. The results are shown in Table 7.2.

The results are quite different for the two lexicons. The derived hierarchy for the Textbook lexicon is not very good: Only 6 of the 273 nodes retained by the algorithm are true positives. This means that 267 nodes that introduce new path-value pairs in the lattice do not occur in the original hierarchy. Furthermore, 22 nodes occur in the hierarchy but do not introduce

Lexicon	retained	TPs	expected TPs	R %	P %	F %	p
Textbook	273	6	4.59	21.43	2.20	3.99	.2546
Spanish	293	37	0.89	74.00	12.63	21.58	$< .0001$

Table 7.2: Minimal Path-Value Pair Pruning on Textbook and Spanish lattices

path-value pairs in the lattice. If the Textbook hierarchy follows a principle of minimality it is clearly not minimality with respect to path-value pairs. The creators of the Textbook hierarchy chose to occasionally repeat path-value pairs for the benefit of keeping the number of (intermediate) nodes small (22 vs. 273). Consequently Petersen's pruning method leads to a fairly low precision and recall for the Textbook lexicon and only performs insignificantly better than what one would expect if 273 nodes were retained randomly.

The second line in Table 7.2 gives the results for the Spanish lexicon. Here Minimal Path-Value Pair Pruning performs much better. 37 out of the 50 intermediate nodes in the manual hierarchy introduce a new path-value pair, resulting in a recall of 74%. Precision, too, is higher than for the Textbook lexicon at 12.63%. The difference between Minimal Path-Value Pair Pruning and randomly retaining 293 nodes is very big (statistically significant at $p < 0.0001$). If 293 nodes were retained randomly one would expect that —on average— less then one of them is a true positive.

Hence, it seems that Minimal Path-Value Pair Pruning is a reasonably good strategy for some lexicons, like the Spanish lexicon, but not so good for others, like the Textbook lexicon. One reason for this difference may be that it is to some extent a matter of personal taste whether the creator of a hierarchy prefers hierarchies with a lot of multiple inheritance and little repetition of path-value pairs or whether he prefers to use multiple inheritance sparingly at the expense of more repetitions of path-value pairs. Minimal Path-Value Pair Pruning creates hierarchies which use a lot of multiple inheritance and is therefore better suited for the first of these scenarios. It is impossible to decide on the basis of two lexicons which case is more common. Further experiments are required for this. However, the Spanish lexicon was not as carefully constructed as the Textbook lexicon (see the discussion in Section 6.1, page 154) and therefore it may be less representative of hierarchical lexicons in general.

7.3 Semi-Automatic Maximum Entropy Pruning

7.3.1 Non-Interactive Pruning

The first semi-automatic architecture for Maximum Entropy Pruning assumes that the lattice is partially pruned by the maximum entropy pruner, i.e. the pruner removes nodes which have a relatively high pruning probability but leaves in those with a lower probability. The partially pruned lattice can then be post-processed by a linguist who looks at all the remaining nodes in the lattice and decides whether or not to retain them. The pruning step and the post-processing step are completely separate, i.e. the *complete* lattice is (partially) pruned before the linguist looks at it. Essentially the maximum entropy pruner only makes decisions about nodes where it is fairly certain that they should be pruned and leaves decisions about other nodes to the linguist. It was argued in Chapter 3 that some nodes in the lattice will be clearly implausible while others are plausible or marginally plausible. Ideally the maximum entropy pruner should prune nodes which fall into the first group and retain all others. This would then save the linguist time, because she does not have to consider all possible generalisations, while still making it possible to build the hierarchy according to her wishes.

The post-processing phase should allow re-introduction as well as removal of nodes.[4] However, the former is much more expensive because it is much easier to make a decision about whether to retain an existing node than to re-create from scratch a node which is no longer contained in the partly pruned lattice. This means that false negatives (nodes which were pruned but should not have been) are more harmful than false positives. Consequently, a high recall is more important than a high precision; ideally one would like 100% recall while still reducing the number of nodes the linguist has to look at as much as possible.

With automatic pruning it often makes sense to prune nodes with an empty intension straightaway, without calculating their pruning probability, as there is little reason for retaining an empty node unless one wants to ensure that the hierarchy is bounded and the node is necessary for this.

With semi-automatic pruning the situation is different. Since the emptiness of a node's overt intension depends on retain decisions that have been made higher up in the hierarchy, and since nodes that have been retained can be pruned during post-processing, the fact that a node has an empty overt intension should not *per se* lead to its pruning.[5] Therefore in the non-

[4] Introducing nodes is necessary for cases where nodes are not contained in the original Galois lattice, e.g. for nodes that generalise beyond the lexicon (see Section 4.2, page 103).

[5] The number of nodes with an empty overt intension depends on how many nodes are retained by the pruner:

interactive semi-automatic architecture *all* nodes are assessed by the maximum entropy model and only pruned if their pruning probability was above the threshold. While wrong decisions at one level will still influence the context of their descendants and possibly cause them to have an empty overt intension and thereby increase their pruning probability, this effect can be overridden if other aspects in a node's context strongly suggest that it should be retained. This happened on several occasions in the experiments and some true positives do in fact have an empty overt intension but were still retained.

The experiments below use a sampling rate of 1:1. That is, a model was trained on an equal number of positive and negative examples. There are 608 positive examples in LinGO ERG. 608 negative examples were sampled randomly in the way described in Section 6.4. A maximum entropy model was then trained on the 1216 training examples. Because the quality of a maximum entropy model depends very much on the training set and the training set is selected randomly by the sampling algorithm, it is to be expected that some models trained in this way underperform due to an unrepresentative training set. This is particularly true if a low sampling rate is used since a low number of negative examples in the training set makes it more likely that they are not representative of negatives in general. In practice this is not a problem provided that (i) one has development data available to which a model can be applied to see whether it performs well or not, and (ii) it can be shown that the performance of a model generalises across lexicons, i.e. a model that performs well on one lexicon also performs well on another. If these two conditions are met it is possible to identify a good model using the development set and then use this model on new data. In the present case, the models will be selected on the basis of their performance on the Textbook lexicon and some of them will be applied to the Spanish lexicon to see if the results carry over.

Ten different training sets where sampled, each containing the same 608 positive examples but different negative examples. For each training set a maximum entropy model was trained using a cut-off of 50, i.e. features that occurred less then 50 times in the training data were discarded. The motivation for using a feature cut-off is that low frequency features can be unreliable (see the discussion in Ratnaparkhi (1998), Section 10.2.5). Having a cut-off is therefore a way of smoothing maximum entropy models to avoid overfitting.[6] Ratnaparkhi recommends

the more nodes are retained, the more have an empty intension. For example, the (complete) Galois lattice of the Textbook lexicon contains 273 nodes with a non-empty overt intension and 1,391 with an empty overt intension. The more nodes are removed from the lattice the more likely it becomes that nodes further down will not be empty.

[6] Apart from cut-offs there are other smoothing methods, for example Gaussian priors (Chen and Rosenfeld 1999), feature merging (Mullen and Osborne 2000), and incremental feature selection (Della Pietra *et al.* 1997). Curran and Clark (2003) provide some evidence that Gaussian priors lead to small performance gains over feature cut-offs.

a cut-off of 5 or 10. The cut-off of 50 used here was optimised using a small lexicon that was not used as a test lexicon.[7] The reason why a higher cut-off seems to work better for this task may be that there is more potential for noise in the features due to noisy inter-node relations. The small training set also means that there is more risk of overfitting.

For the experiments in this section, the pruning threshold was set to .50, i.e. a node was pruned if its pruning probability was higher than .50. The next section investigates higher thresholds. It turns out that a threshold of .50 already leads to a fairly high recall due to the fact that relatively many nodes are retained with this threshold. This is discussed in more detail below (page 189). For each model, the lattice of the Textbook lexicon (i.e. the test data) was then pruned into a hierarchy and this hierarchy was matched to the original hierarchy as described in Section 6.5 and the number of true positive nodes was calculated. For each model it was then determined whether its performance was significantly better than if the same number of nodes were retained randomly (i.e. significantly better than the individual baseline introduced on page 180).

Of the ten models one performed worse than random selection, five performed insignificantly better than random selection and four models performed significantly better (three at $p < 0.01$ and one at $p < 0.05$). The results for these four models are given in Table 7.3. The first column gives the model number. The second gives the number of retained intermediate nodes. The third column shows the number of matched intermediate nodes (i.e. the number of true positives). The forth column gives the expected number of true positives if the retained nodes where chosen at random. Columns five, six and seven give the values for recall, precision and the f-score, respectively, and the final column gives the probability that this performance is due to chance.

Model	retained	TPs	expected TPs	R %	P %	F %	p
1	790	22	13.29	78.57	2.78	5.36	.0080
2	1015	26	17.08	92.86	2.56	4.98	.0146
4	803	24	13.51	85.71	3.00	5.80	.0020
8	823	25	13.85	89.29	3.04	5.88	.0013

Table 7.3: Non-interactive, semi-automatic pruning (Textbook)

[7] The lexicon used for the optimisation was one of the lexicons ("Pacifier") developed as teaching material for the ESSLLI'98 course on "Practical HPSG Grammar Engineering". This lexicon contains only 18 entries and was therefore considered too small for the experiments reported in this chapter. The cut-off was optimised by training a model on LinGO ERG and then applying it to the Pacifier lexicon.

The f-score is very low compared to the results typically achieved for some other machine learning tasks (e.g. part-of-speech tagging). But it has to be remembered that the task of automatic inheritance hierarchy construction is relatively hard and the evaluation method used here (i.e. strict matching) is very conservative. More importantly, the fact that the four models achieve statistically significant results compared to retaining the same number of nodes randomly means that some useful facts have been learned about the data. This is despite the fact that only a very limited amount of training data was available to the maximum entropy learner.

The recall is also significantly better than the recall achieved by Random Uniform Pruning (recall=50%, see Table 7.1). The only exception is Model 1 where the difference is marginally significant ($\chi^2 = 3.81, p \leq .0509$). While the precision is also better (Random Uniform Pruning, avg. precision = 1.68%) this difference is not significant.

Maximum Entropy Pruning also leads to better results than Minimal Path-Value Pair Pruning. The f-score achieved by the latter is lower (3.99%, see Table 7.2). So are the values for recall and precision. The difference in recall is statistically significant for all four models. However, the difference in precision is not.

The high recall achieved by the four models means that the hierarchies derived by them are well suited for manual post-processing. For example, if Model 8 —the best performing model in terms of the f-score— is applied to the Textbook Galois lattice, one can achieve more than 50% reduction in workload, i.e. the linguist only has to look at 823 nodes instead of the 1,664 nodes in the complete lattice, while still ensuring nearly 90% recall.

While looking at 823 nodes is still a lot of work (the manually built hierarchy only has 28 intermediate nodes), this can be reduced further. For example, if the nodes are considered in a top-down fashion during post-processing each decision to retain a node potentially leads to empty nodes further down and these could be pruned automatically without ever being shown to the linguist. If the number of nodes that have to be looked at is still large, it might be useful to use a data exploration system to make the post-processing task easier. Formal Concept Analysis (Ganter and Wille, 1999) may be a good candidate for this.

While the results for the Textbook lexicon look reasonable they are only useful if it can be shown that the performance of the models carries over to other lexicons. It is particularly important to show that a model that performs well on one lexicon also performs well on another lexicon because this indicates that a good model can be selected on the basis of its performance on a development data set and can then be applied to a new data set. For example, if a model that achieved significant results on the Textbook lexicon did not achieve significant results on

the Spanish lexicon, the maximum entropy pruner would be of limited use as this would mean that it is not possible to predict the performance of a model on a new lexicon.

Unfortunately, it was not possible to apply all ten models to the Spanish lexicon because the Spanish Galois lattice is very big —nearly ten times as big as the Galois lattice for the Textbook lexicon— and this leads to very long running times for the pruning algorithm.[8] Therefore, only Models 1 and 2 were applied to the Spanish lexicon. The results are given in Table 7.4.

Model	retained	TPs	expected TPs	R %	P %	F %	p
1	4,182	46	12.64	92.00	1.10	2.17	$<.0001$
2	8,764	47	26.49	94.00	0.54	1.07	$<.0001$

Table 7.4: Non-interactive semi-automatic pruning (Spanish)

As can be seen both models are significantly better than random selection. Furthermore, Model 1, which led to a higher f-score than Model 2 when applied to the Textbook lexicon, also achieves a higher f-score when applied to the Spanish lexicon. Hence, the performance of a model on one lexicon seems to be a good indicator for its performance on another lexicon.

While the f-score for the two models drops when they are applied to the Spanish lexicon compared to their performance on the Textbook lexicon, the significance of the performance achieved by the maximum entropy models is actually larger for the Spanish lexicon because it is much easier to achieve good results for the Textbook lexicon, since the proportion of positive intermediate nodes to intermediate nodes in the Galois lattice is higher for the Textbook lexicon than for the Spanish lexicon. For the Textbook lexicon there is one chance in 59 to select a true positive; for the Spanish lexicon this drops to one in 330.

The Random Uniform Pruning baseline, too, is lower for the Spanish lexicon. The expected f-score for Random Uniform Pruning is only 0.60% compared to 3.25% for the Textbook lexicon. Furthermore, while for the Textbook lexicon only the difference in recall was statistically significant, for the Spanish lexicon the difference in precision is also significant for the two maximum entropy models. This means that the performance of the models relative to the baselines is actually better for the Spanish lexicon than for the Textbook lexicon.

Minimal Path-Value Pair Pruning, however, outperforms Maximum Entropy Pruning on the Spanish lexicon with respect to f-score and precision. But Maximum Entropy Pruning achieves a higher recall (74% for Minimal Path-Value Pair Pruning). And this difference is statistically

[8] For the current implementation, pruning the Spanish Galois lattice took around 3 weeks on a Dell 450.

significant (for Model 2: $\chi^2 = 4.54$, p$\leq$.0331; for Model 2: $\chi^2 = 6.03$, p $\leq$.0141). Hence, for semi-automatic construction Maximum Entropy Pruning may still be the better choice.

To sum up, out of ten maximum entropy models trained on ten different training sets with randomly sampled negative examples, four performed significantly better than their individual baseline on the Textbook lexicon. Three of these were significantly better for recall than the Random Uniform Baseline; the fourth narrowly missed significance. All four had a significantly better recall than Minimal Path-Value Pair Pruning. Two of the models were then applied to the Spanish lexicon. Both of them were significantly better than their individual baselines. In addition, both were significantly better than Random Uniform Pruning in both precision and recall. The model that performed better on the Textbook lexicon also performed better on the Spanish lexicon. This was taken as evidence that the performance of a model on a development set can predict its performance on a new lexicon. While Maximum Entropy Pruning is outperformed by Minimal Path-Value Pair Pruning with respect to precision and f-score when applied to the Spanish lexicon, Maximum Entropy Pruning achieves a significantly better recall.

7.3.2 The Role of the Pruning Threshold

A pruning threshold of .50 already results in a fairly high recall for both the Spanish and the Textbook lexicon. The next experiment investigates whether it is possible to get 100% recall by increasing the pruning threshold. This means that more nodes will be retained, increasing the workload for manual post-processing but in some situations (i.e. if quality is more important than time) this may be a price worth paying as it ensures that there will be no false negatives.

An inspection of the pruning probabilities assigned by Model 1 to the positive nodes in the Textbook lattice shows that there is an "outlier", i.e. a positive node with a very high pruning probability of .7873. This suggests that the pruning threshold has to be raised to at least .80 (i.e. only nodes with $P(\mathit{prune}) > .80$ will be pruned) to ensure 100% recall.

Table 7.5 shows the results of pruning the Textbook lattice with Model 1 and gradually increasing the pruning threshold. It turns out that a threshold of .80 is not enough to ensure 100%. Instead the threshold has to be raised to .85. The reason for this is that changes in the pruning threshold usually have an effect on the pruning probability of individual nodes. For example, the more nodes are retained at higher levels of the hierarchy, the more likely it is that nodes at lower levels are left with an empty overt intension. Under the assumption that the model has learned to prefer nodes with a non-empty overt intension, this leads to a higher

pruning probability for those nodes whose overt intension became empty due to a changed pruning threshold. Nodes which already have a high pruning probability are often particularly affected in this way: increasing the pruning threshold will increase their pruning probability even more. Consequently, it will often be impossible to raise the pruning threshold to a level where 100% recall is achieved while still pruning enough nodes to make the use of a semi-automatic technique —as opposed to manual construction— worthwhile. This is the case in the present example, where only 46 nodes (out of 1664) get pruned at threshold .85. This means that the workload for the linguist is only insignificantly reduced by the semi-automatic technique.

threshold	retained	TPs	expected TPs	R %	P %	F %	p
.55	1,038	23	17.47	82.14	2.22	4.32	.0901
.60	1,089	25	18.32	89.29	2.30	4.48	.0582
.65	1,136	25	19.12	89.29	2.20	4.29	.0869
.70	1,316	26	22.14	92.86	1.98	3.88	.2033
.75	1,384	26	23.29	92.86	1.88	3.69	.2843
.80	1,505	27	25.32	96.43	1.79	3.51	.3669
.85	**1,618**	**28**	**27.23**	**100.00**	**1.73**	**3.40**	**.4404**

Table 7.5: Varying the threshold for higher recall (Textbook, Model 1)

Note also that increasing the threshold to .55 or higher means that the results are no longer statistically significant. The more the threshold is raised the more likely it is that the same result can be obtained by random pruning. For example, if 1,618 nodes are retained randomly the number of true positives will already be 27 on average.

Why then are so many nodes retained at a threshold of .50? There are two obvious explanations. First it could be that this has something to do with the sampling rate, i.e. a training set that has been sampled at a rate of 1:1 does not contain enough negative examples to reliably determine all negative nodes in the Galois lattice of the test lexicon. In other words, the negative training nodes may not be representative of negative nodes generally. The role of the sampling rate is discussed in Section 7.3.3 and there is some evidence that the sampling rate does indeed play some role here.

A second explanation for the high proportion of nodes retained by the maximum entropy pruner is that it is easy to determine nodes which express very implausible generalisations but

it is difficult to distinguish marginally plausible from plausible nodes. Thus the pruner prunes mainly implausible nodes but retains all nodes that are of at least marginal plausibility. However, this is difficult to prove without human evaluation of the retained and pruned nodes and human evaluation is impractical (see page 164). One way to shed some light on the situation is by looking at the amount of overlap between the nodes that have been pruned by different models. In other words: do most models prune the same nodes? Or is there a lot of variation with respect to which nodes get pruned? If there is a lot of overlap this indicates that some nodes are indeed easier to identify as negatives and that this is independent of the model (and of the negative nodes in the training set).

Table 7.6 shows how much the four models overlap in the nodes they prune. The first column indicates how many nodes are pruned by at least one of the four models. The next columns show how many nodes have been pruned by how many models. It can be seen that 928 nodes have been pruned at least once. More than half of these (506) have been pruned by all four models. A further 265 nodes have been pruned by three of the models and a further 103 have been pruned by two of them. Only 54 have been pruned by one model alone. The latter were assigned fairly small pruning probabilities, i.e. these were nodes that the model was not too sure about. Hence, it seems that there is substantial overlap between models with respect to pruned nodes. Most of the 506 nodes that were pruned by all models are true negatives (505 out of 506). One is a false negative that is assigned a relatively high pruning probability by all models. This is the "outlier" that was mentioned above (page 188). This suggests that some nodes (i.e. the 506 retained by all models) are relatively easy to identify as negatives, independent of the model and training set, possibly because they are clearly implausible. On the other hand it is more difficult to distinguish plausible from marginally plausible nodes.

all	4 models	3 models	2 models	1 model
928	506	265	103	54

Table 7.6: Overlap between models with respect to pruned nodes

7.3.3 The Role of the Sampling Rate

The training sets for the maximum entropy models discussed above have been sampled by randomly selecting the same number of negative and positive examples. This leads to a fairly small training set of 1,216 examples. The number of positive examples is fixed by the number

of (intermediate) nodes in the training hierarchy but the number of negative examples could theoretically be increased up to the number of (negative) nodes in the Galois lattice for the training lexicon, i.e. up to 7.5 million. There are practical and theoretical reasons for not using the complete set of negative examples (e.g. computational cost and problems with learning from imbalanced data, see Section 6.4). However, it is quite likely that a slightly increased sampling rate leads to better results because it increases the likelihood that the negative examples in the training set are representative of negative examples in general. This may lead to fewer false positives being retained.

Sampling is quite expensive, especially at higher rates. The reason for this is that negative nodes are selected by randomly intersecting lexical entries and then adding the resulting node if it is a negative node and not yet contained in the training set (see Section 6.4). The majority of intersections, however, will either be empty or the resulting node will already be contained in the training set. This means that finding negative nodes is quite expensive, despite the fact that there are more than 7.5 million of them in the lattice. Because of this computational cost the sampling rates used in the following experiments were kept fairly low at 1:2, 1:3 and 1:4.

For each of the three sampling rates ten training sets were selected, i.e. ten sets with 608 positive examples and 1,216 negative examples (sampling rate 1:2), ten sets with 608 positives and 1,824 negatives (sampling rate 1:3) and ten sets with 608 positives and 2,432 negatives (sampling rate 1:4). For each of the training sets a maximum entropy model was trained and applied to the Textbook lattice, using a non-interactive semi-automatic pruning architecture. The resulting hierarchies were then evaluated and compared to their individual baselines (see Section 7.1). The statistically significant models are shown in Tables 7.7 to 7.9.

Model	retained	TPs	expected TPs	R %	P %	F %	p
2	829	22	13.95	78.57	2.65	5.13	.0150
4	920	26	15.48	92.86	2.83	5.49	.0035
10	787	24	13.24	85.71	3.05	5.89	.0014

Table 7.7: Textbook, Sampling rate 1:2

It looks like higher sampling rates lead to fewer nodes being retained which leads to a higher f-score but a lower recall. If only those models are taken into account which lead to statistically significant results, the average number of nodes retained at sampling rate 1:1 is 857.75. This drops to 545.25 for sampling rate 1:4. The retained nodes for sampling rates

Model	retained	TPs	expected TPs	R %	P %	F %	p
1	944	24	15.88	85.71	2.54	4.93	.0202
8	763	25	12.84	89.29	3.28	6.33	.0003

Table 7.8: Textbook, Sampling rate 1:3

Model	retained	TPs	expected TPs	R %	P %	F %	p
2	721	18	12.13	64.29	2.50	4.81	.0446
4	560	21	9.42	75.00	3.75	7.14	.0001
6	689	21	11.59	75.00	3.05	5.86	.0026
7	211	10	3.55	35.71	4.74	8.37	.0003

Table 7.9: Textbook, Sampling rate 1:4

1:2 and 1:4 are in the middle (845.33 and 853.50, respectively). The values for true positives do not show such a clear trend: the average number of true positives at rate 1:1 is 24.25, at sampling rate 1:2 it is 24, at sampling rate 1:3 it is 24.50 but it then drops to 17.50 at sampling rate 1:4. However, it could be that results for sampling rate 1:4, which show the trend most clearly compared to rate 1:1, are exceptional in this respect.

Having said that, it would be plausible if higher sampling rates lead to less nodes being retained, as the training sets that were sampled with a higher rate contain more negatives which may make it easier to identify negative nodes in the lattice. This could mean, for example, that models trained at higher sampling rates are more certain about which class a node belongs to, i.e. instead of assigning pruning probabilities of around .50 to many nodes they may assign higher probabilities and this may lead to more nodes being pruned. Since higher sampling rates also seem to lead to lower recall this would mean that the pruning threshold would have to be adjusted for semi-automatic pruning to ensure a high enough recall. However, it is possible that the recall could also be increased if more positive training data were available. If more negative examples help the pruner to identify negative nodes better, more positive examples are likely to have the a similar effect on positive examples. It has to be kept in mind, however, that a larger set of training data only helps to some extent as the task will always be difficult due to the fact that there is no unique best hierarchy and due to the lack of extraneous linguistic knowledge available to the system.

7.3.4 Inter- vs. Intra-Node Measures

The next experiment investigates the role of inter- and intra-node measures. Inter-node measures potentially introduce noise because they are heavily influenced by previous pruning decisions and these may have been false. Intra-node measures are also influenced by previous pruning decisions (insofar as they take a node's overt intension into account) but probably not to the same extent. Inter-node measures are also computationally more expensive than intra-node measures because a node has to be compared to several other nodes. Hence, it would be interesting to know how much inter-node measures contribute to the overall performance of a model and whether they introduce so much noise that it might be better to abandon them.

To test this, a training set was created at sampling rate 1:1 and used to train three different maximum entropy models. The first model (Model A) was trained using both inter- and intra-node measures. The second (Model B) was trained using only intra-node measures and the third (Model C) was trained using only inter-node measures. The three models were then applied to the Textbook lattice, using a pruning threshold of .50. The results achieved by Model A were statistically significant. Those achieved by the other two models were not but this is not surprising given that they only use a partial feature set.

I then looked at the influence the three models had on the retained nodes, in particular on the true positives. Are there any true positive nodes which are retained by one of the models but not by the other? The focus was on true positives rather than true negatives because the former have to be maximised to ensure a high recall and for semi-automatic pruning recall is more important than precision.

Eleven of the 28 positive nodes were misclassified by at least one of the models. Table 7.10 shows the pruning probabilities that were assigned to these eleven nodes by the three models. Pruning probabilities above .50, i.e. those that lead to a misclassification for the node are highlighted. The final column shows whether the node introduces new path-value pairs in the Galois lattice. The last line in the table gives the number of true positives for each model.

It turns out that the inter-node measures are not redundant but rather complement the intra-node measures. For example, five nodes which are pruned by Model B (603, 490, 38, 1153, and 84) are given a fairly low pruning probability by Model C which means that they are retained if both inter- and intra-node measures are used. Likewise, three nodes which are pruned by Model C (1298, 892, and 925) are assigned fairly low pruning probabilities by Model B. Two nodes (Node 665 and Node 360) are assigned a high pruning probability by Model B and C and these node are —predictably and in this case falsely— pruned by the combined model.

Node	P(prune) inter & intra (A)	intra (B)	inter (C)	new PVPs in GL?
603	.2396	**.5625**	.6060	yes
665	**.5208**	**.7411**	**.5393**	no
490	.1844	**.7405**	.8620	no
1298	.1843	.6610	**.8995**	yes
38	.2911	**.8468**	.1387	no
892	.1597	.4064	**.5644**	yes
1153	.2462	**.7300**	.1123	no
925	.1281	.2160	**.6547**	yes
360	**.8373**	**.7301**	**.8455**	no
456	**.5754**	.4402	**.5983**	yes
84	.1849	**.5186**	.2267	yes
TPs	25	19	22	

Table 7.10: False positives for different models

Comparing the pruning probabilities assigned by Model B with the final column it becomes evident that intra-node measures tend to assign relatively low pruning probabilities (i.e. below .6) to all nodes which introduce new path-value pairs in the lattice. Closer inspection of all true positives and the pruning probabilities assigned to them by Model B revealed that most nodes that would be retained by Minimal Path-Value Pair Pruning are also retained by Model B. However Model B also retains further nodes which means that it leads to more true positives than Minimal Path-Value Pair Pruning (19 vs. 6). This is due to the fact that intra-node measures are not only concerned with whether or not a node introduces new path-value pairs in the lattice but also with other aspects, such as the proportion of path-value pair dependencies in a node's overt intension.

However, there are still some positive nodes which Model B misclassifies. Most of these are retained if inter-node measures are taken into account. This demonstrates why Maximum Entropy Pruning performs better than Minimal Path-Value Pair Pruning on the Textbook lexicon: Maximum Entropy Pruning takes inter-node relations into account and it seems that despite the problems of reliably determining inter-node relations they are actually quite useful and provide important cues about whether or not a node should be pruned.

7.3.5 Binary Quantisation

The final experiment for the non-interactive semi-automatic architecture investigates whether binary quantisation leads to better results than the interval-based quantisation method used so far.

Remember that most maximum entropy measures defined in Section 5.2 are real-valued. Contextual predicates, however, are boolean functions. To transform real-valued measures into boolean functions a quantisation step is needed (Section 5.2.3). The approach taken so far was to automatically break up the range of a measure into intervals, using the method proposed by Fayyad and Irani (1993).[9] However, this approach causes information about the order of different values to be lost. For example, it is not possible to determine that the interval [0.0-0.2) is closer to the interval [0.2-0.4) than to the interval [0.6-0.8). If order is important, measures can be quantised using a binary approach. This turns the range of a measure into binary predicates of the form $< 0.2, \geq 0.2$ (see Section 5.2.3).

To investigate whether binary quantisation leads to better results, models 1, 4, and 8 (sampling rate 1:1) were retrained using binary quantisation and then applied to the Textbook lexicon. Table 7.11 shows the results. For Model 1 binary quantisation leads to a better f-score, for the other two models non-binary quantisation does. However, the differences are small. For all three models binary quantisation leads to a higher recall but the difference is not significant. So it seems that binary quantisation, surprisingly, does not lead to significant improvements over non-binary quantisation. It was argued in Section 5.2.3 that binary quantisation can be particularly important for sparse training sets because missing binary predicates can be "covered" by other predicates. One reason why binary quantisation does not lead to significant improvements could therefore be that data sparseness does not play such a big role. This may be due to the relative high feature cut-off point, which means that features that occurred less than 50 times in the training set are not considered. This means that only high frequency features are used which may make it less likely that a feature occurs in the test set but not in the training set. It is also possible that the Textbook lexicon is too small for differences between quantisation methods to manifest themselves in the results.

[9] The quantisation was done using the WEKA machine learning software: `http://www.cs.waikato.ac.nz/ml/weka/` (27.6.03) (see also Witten and Frank 1999).

Model	retained	TPs	expected TPs	R %	P %	F %
1 1:1 binary	818	26	13.76	89.66	3.18	6.14
1 1:1 non-binary	790	22	13.29	78.57	2.78	5.36
4 1:1 binary	1035	27	17.42	93.10	2.61	5.08
4 1:1 non-binary	803	24	13.51	85.71	3.00	5.80
8 1:1 binary	1082	27	18.21	93.10	2.50	4.86
8 1:1 non-binary	823	26	13.85	89.29	3.04	5.88

Table 7.11: Binary vs. Non-Binary Quantisation

7.3.6 Interactive Pruning

The second approach to semi-automatic pruning is interactive: automatic and manual pruning are interleaved. The pruning algorithm again traverses the lattice top-down. At each level the pruning probabilities for the nodes at that level are calculated and nodes whose probabilities are below the threshold are pruned but the remaining nodes are immediately presented to the linguist who can then decide whether to prune or retain them. This has the advantage that mistakes do not percolate down the hierarchy as much as in the previous architecture because it is guaranteed that all nodes that have been retained at higher levels have been retained correctly since all higher levels have already been checked by a linguist. This also makes it possible to prune nodes with an empty overt intension immediately at the beginning of a level without calculating their pruning probability. However, there is still a small possibility that errors accumulate because wrong negative decisions are not rectified unless the linguist re-introduces or compensates for falsely pruned nodes.

To test this approach the Textbook lattice has been pruned in an interactive fashion using Model 1. The role of the linguist has been simulated by an oracle, i.e. instead of asking a human whether a node selected for retention should indeed be retained, this information was looked up in a list of all positive nodes in the original Textbook hierarchy. A node was retained if its pruning probability was below the threshold and it also occurred in the original hierarchy. Nodes for which this is true are true positives. Nodes whose pruning probability was above the threshold but which did occur on the list of positives were pruned and recorded as false negatives. These nodes would never have been shown to the linguist. Nodes whose pruning probability was below the threshold but which did not occur on the list were also pruned. These are false positives and they would have been pruned by the linguist.

Note that this is only an approximation of what a linguist would do, in that a linguist would probably try to compensate for false negatives, possibly by retaining nodes which would not otherwise have been retained. For example, if the ideal hierarchy should contain node X but this was not offered to the linguist because its pruning probability was below the threshold then the linguist might compensate for this by retaining some of X's descendants which would not have been retained otherwise. Simulating the role of the linguist by an oracle does not provide this flexibility. Despite this, using an oracle seems a fairly good approximation of what would happen if the system interacted with a human and it avoids problems that arise if human subjects are used (see page 164).

Model 1 from the previous experiments was used to test Interactive Pruning. The results are shown in Table 7.12. The column entitled "decision steps" shows the number of nodes that were originally retained by the pruner and compared with the oracle. The next two columns show the true positives and the expected number of true positives if 892 nodes were retained randomly. The final four columns show, respectively, the recall, precision, f-score and the probability of obtaining these results by retaining 892 nodes randomly. Recall and precision have been calculated with respect to the performance of the maximum entropy pruner not with respect to the combined performance of the maximum entropy pruner and the oracle. For example, the precision has been calculated by dividing the number of true positives by the number of decision steps and then transforming the value to a percentage.

Model	decision steps	TPs	expected TPs	R %	P %	F %	p
1	892	15	15.01	53.57	1.68	3.26	.50

Table 7.12: Interactive Pruning (Textbook)

The maximum entropy pruner does not achieve good results for this task. Compared to its performance for non-interactive semi-automatic pruning it retains more nodes (892 *vs.* 790) but despite this it leads to fewer true positives. Essentially, the performance of the maximum entropy pruner in this task leads to results that could also be achieved by retaining 892 nodes randomly (random mean = 15.01).

This is surprising since interactive semi-automatic pruning should be easier than non-interactive semi-automatic pruning because in the former case there is less room for errors to accumulate by percolating down the hierarchy. It seems that the fact that relatively few nodes are retained at higher levels by the oracle "messes up" the pruning probabilities assigned

to nodes further down. For example, retaining fewer nodes leads to fewer nodes with an empty overt intension further down the hierarchy. Since a node with a non-empty overt intension is more likely to be retained this could explain why the number of decision steps increases compared with non-interactive pruning. While this explains the increased number of decision steps it does not explain the increase in the number of false negatives.

In a variation of the interactive architecture, the nodes at each level are ranked according to their retain probability and presented to the linguist in that order, i.e. the node with the highest retain probability is shown first. Once the linguist has made a decision about a node, the probabilities for the remaining nodes at the current level are re-calculated. This variation should lead to gains in performance provided that the pruning probabilities of some nodes are strongly influenced by decisions about their siblings. In this case it may also reduce the number of decision steps. For example, it is possible the a node *n* has a pruning probability which is slightly below the threshold, i.e. the pruner would retain the node but is not very certain about this decision. If there are other nodes about which the pruner is more certain, i.e. which have a lower pruning probability, these would be presented first. With every decision the user makes about these nodes, the pruner gets an additional piece of information which may influence the retain probability of *n* and eventually help to make a more informed decision about whether to prune *n* or present it to the user. For instance, if the user retains many of *n*'s siblings the pruning probability for *n* is likely to increase.

For this architecture, a new maximum entropy feature was implemented. The feature measures the average proportion of positive nodes at a given level in relation to the number of nodes at that level in the Galois lattice. The rationale for adding this feature was to make it easier for the pruner to estimate the pruning probability of a node on the basis of knowledge about retained and pruned siblings. In particular this measure should enable the pruner to decide to prune a node if many of its sibling had already been retained by the user.

A new model was trained on LinGO ERG, using the same set of training examples that were used to train Model 1. That is, this new model only differs from Model 1 in that it contains an additional feature. The model was then applied to the Textbook lattice as part of the re-ranking architecture described above.

As the re-ranking approach is computationally much more expensive than the non-re-ranking approach because the probabilities for every node have to be calculated several times, the number of re-ranking cycles was limited to ten. That is, on every level the nodes were only re-ranked 10 times at most. After that the probabilities were calculated without taking any

further decisions into account.

The results were similar to the ones reported for the non-re-ranking approach since the pruning probabilities changed only marginally from one re-ranking cycle to the next, i.e. user decisions about one node did not seem to have a big effect on the probabilities of its siblings. This may be due to the maximum entropy features, i.e. maybe the features are not sensitive enough to interactions between siblings. It may also be that the amount of training data is too limited for such interactions to be established. Finally it may be that there are not many interactions between siblings and that a re-ranking approach generally does not lead to improvements over a non-re-ranking approach.

Another variation of the interactive architecture would be to enable the maximum entropy pruner to learn from previous decisions of the user. For example, each decision of the user could be added as a new training example and the model could be re-trained at regular intervals. This way it would be possible to fine-tune the model to the needs and tastes of a user. This would be particularly useful because hierarchy construction is to some degree influenced by factors such as personal taste. For example, one user might prefer hierarchies which make a lot of use of cross-classification and employ multiple inheritance wherever possible to avoid repetitions of path-value pairs while other users may prefer to use multiple inheritance sparsely. While the maximum entropy training is quite fast because the number of features used here is fairly small compared to other tasks, this approach will only be feasible if a user has to construct a very big hierarchy or has to construct several hierarchies.

7.4 Automatic Maximum Entropy Pruning

There may be situations where semi-automatic hierarchy construction is not an option, for example because of time constraints or because no linguistic expert is available to do the semi-automatic construction. In these situations an automatic approach is needed.

The automatic system differs in two aspects from the non-interactive semi-automatic system described in Section 7.3.1. First, nodes with an empty overt intension will be pruned straightaway without calculating their pruning probability because there is no reason for keeping an empty node in the derived hierarchy (unless one wants to ensure boundedness and the node is necessary for that). Pruning empty nodes without calculating their context also increases the pruning speed. Second, in an automatic system, recall is no longer more important than precision. Rather the aim should be to find a good trade-off between the two by aiming to maximise the f-score. The trade-off between precision and recall can be influenced by

changing the pruning threshold. A maximum entropy pruner with a pruning threshold of .50 retains proportionally many nodes, i.e. favours recall at the expense of precision. For a higher f-score the pruning threshold therefore has to be decreased which means that more nodes will be pruned.

To test the effect of varying the pruning threshold, the Textbook lattice was automatically pruned using Model 1 with a varying pruning threshold, ranging from .50 (i.e. nodes are pruned if $P(prune) > .50$) to .05 (i.e. nodes are pruned if $P(prune) > .05$). Table 7.13 shows the results.

The first thing that can be observed is that the f-score of automatic pruning with a threshold of .50 is higher than the f-score that has been achieved with non-interactive semi-automatic pruning (8.86% vs. 5.36%, see Table 7.3, page 185). This difference is due to the fact that empty nodes are always pruned in the automatic approach. This nearly halves the number of retained nodes from 790 to 356. It also reduces the number of true positives (17 vs. 22) and leads to a lower recall but the reduction in recall is more than compensated by the increase in precision.

Looking at the effects of decreasing the pruning threshold, it can be seen that the highest f-score is achieved with a threshold of .20 (highlighted in the Table 7.13). If this threshold is used 258 nodes are retained. 14 of those also occur in the original hierarchy. This leads to a recall of 50% and a precision of 5.43%. A further decrease of the threshold to .15 leads to a further increase in precision but this is cancelled out by a significant drop in recall and the f-score decreases.

Note that the f-score does not rise monotonically to a threshold of .20. There are two local maxima, at thresholds .50 and .40. This is largely due to the fact that a decrease in the number of retained nodes sometimes leads to an increase in the number of true positives. For example, decreasing the pruning threshold from .45 to .40 causes the pruner to retain 330 instead of 355 nodes but these 330 nodes contain one more true positive than the 355 nodes at threshold .45. Likewise, decreasing the threshold from .30 to .25 leads to three additional true positives. This means that some true positives are assigned a higher retain probability due to the fact that fewer nodes are retained overall. This demonstrates how dependent the retain probability of a node is on the pruning decisions made about other nodes. It is not surprising that pruning more nodes can increase the retain probability of the remaining nodes. For example, the more nodes are pruned at higher levels in the hierarchy the less likely it is that nodes at lower levels will have an empty overt intension. If the pruner has learned to prefer nodes with a non-empty

Threshold	retained	TPs	expected TPs	R %	P %	F %	p
.50	356	17	5.99	60.71	4.78	8.86	$< .0001$
.45	355	16	5.97	57.14	4.51	8.36	$< .0001$
.40	330	17	5.55	60.71	5.15	9.49	$< .0001$
.35	313	14	5.27	50.00	4.47	8.21	.0001
.30	296	12	4.98	42.86	4.05	7.40	.0008
.25	284	15	4.78	53.57	5.28	9.61	$< .0001$
.20	**258**	**14**	**4.34**	**50.00**	**5.43**	**9.80**	$< .0001$
.15	163	9	2.74	32.14	5.52	9.42	.0001
.10	113	4	1.90	14.29	3.54	5.67	.0630
.05	72	2	1.21	7.14	2.78	4.00	.2358

Table 7.13: Varying the pruning threshold for automatic pruning

overt intension this leads to a higher retain probability for those nodes. What is surprising is the magnitude of the increase in true positives between thresholds .30 and .25: retaining twelve nodes less leads to 3 additional true positives.

The fact that reducing the pruning threshold to .20 leads to a higher f-score is only useful if it carries over to other lexicons. If it does carry over it would be possible to fine-tune the threshold on a lexicon for which a hierarchy already exists (i.e. a development data set) and then transfer it to a new lexicon. It is unlikely that a threshold of .20 works best for all lexicons but it is possible that a threshold in that *region* works reasonably well for most lexicons. However, additional experiments are required to confirm this.[10]

7.5 Summary

This chapter discussed several experiments which tested the performance of the maximum entropy pruner. A different pruning method, Minimal Path-Value Pruning (Petersen 2001), was also tested.

The results showed that, for many training sets Maximum Entropy Pruning leads to results that are significantly better than the results that would be achieved if the same number of nodes were retained randomly. For some training sets the results are not significantly better than

[10]Unfortunately, it was not yet possible to test this on the Spanish lexicon due to the long running times of the pruning method when applied to this lexicon.

random pruning but this is probably due to the fact that the negative examples in the training sets are randomly sampled and present only a small proportion of all negative examples in the training hierarchy. This can lead to unrepresentative training sets which in turn lead to maximum entropy models that underperform. The fact that a bad performance is due to the training set and not due to other aspects of the maximum entropy pruner is supported by the fact that good maximum entropy models lead to significant results for two different lexicons. Furthermore, the model that performed better on one lexicon also performed better on the other. This means that it is possible to identify good models by applying them to a development data set; a model which performs well on the development set should also lead to good results on a new data set.

Three different pruning architectures were discussed in detail. The first architecture assumed that the lattice is partially pruned by the maximum entropy pruner and then post-processed by a linguist. Compared to the number of nodes a linguist would have to look at if he had to process the complete lattice, the maximum entropy pruner achieves a reduction in workload that lies between one third (Textbook, Model 2) and three quarters (Spanish, Model 1) while still achieving a recall between 79% and 94%. Furthermore, Maximum Entropy Pruning is particularly useful for larger lexicons, which are very time consuming to create manually. However, the number of nodes that the linguist has to look at after the hierarchy has been partially pruned is still fairly large. This suggests that manual post-processing is best combined with a suitable support tool. One possibility would be to combine it with a data exploration tool.

Interestingly the number of retained nodes is fairly high even for a pruning threshold of .50. To some extent this seems to be due to the fact that a fairly low sampling rate was used and only a small proportion of the negative examples was used for training the maximum entropy model (see Section 7.3.3). This means that the model can only identify some negative nodes. However, there is also a large overlap between the nodes pruned by different models. This suggests that some nodes are easier to identify as negatives than others which makes sense as some nodes may be clearly implausible while other nodes may be marginally plausible and could occur in a manually built hierarchy. The former should be easier to identify while some human intervention is probably needed to differentiate between plausible and marginally plausible nodes as these decisions are often subjective.

In the second architecture maximum entropy pruning and manual post-processing are interleaved. This has the advantage that mistakes made by the pruner are corrected at an earlier

stage and cannot percolate down the hierarchy. Only one experiment has been conducted with this architecture but this, surprisingly, showed results that were worse than those achieved for the first architecture. One explanation for this is that the high proportion of nodes normally retained by the maximum entropy pruner together with the fact that in interactive semi-automatic pruning relatively few nodes are retained at higher levels "messes up" the pruning probabilities of nodes further down the hierarchy and ultimately leads to worse results. But clearly some more research into this is needed.

The third architecture investigated the use of maximum entropy pruning as part of a fully automated system. In an automatic architecture, nodes with an empty overt intension are always pruned. This leads to fewer retained nodes and a lower recall but higher precision and f-score. The f-score can be further increased by decreasing the pruning threshold and thereby retaining fewer nodes.

In a further set of experiments the effects of using Minimal Path-Value Pair Pruning were investigated. This pruning method leads to hierarchies that are minimal with respect to the number of path-value pairs. The results are somewhat inconclusive. While the pruning method leads to statistically insignificant results when applied to the first test lexicon, it leads to excellent results when applied to the second, where it outperforms Maximum Entropy Pruning by a wide margin. This difference may be due to the influence that personal taste plays when constructing a hierarchy. Some linguists may prefer hierarchies which use a lot of multiple inheritance while others may prefer to use multiple inheritance sparsely. Minimal Path-Value Pair Pruning is better at creating the first type of hierarchy. However, it has to be said that neither of the test hierarchies made use of multiple inheritance wherever possible. Consequently, Minimal Path-Value Pair Pruning only achieved an f-score of 22% when applied to the second lexicon. This suggests that minimal redundancy (at least when measured as the number of path-value pairs) is not a strategy normally followed in manually built hierarchies. At best a manually built hierarchy may follow a strategy of *reduced* path-value pair redundancy. Minimal Path-Value Pair Pruning also resulted in a lower recall than Maximum Entropy Pruning. And this was true for both lexicons. This suggests that it is less suited for a semi-automatic system.

The reason why Maximum Entropy Pruning performs better on the first test lexicon than Minimal Path-Value Pair Pruning becomes evident when one looks at the different effects inter- and intra-node measures have. It seems that the two sets of measures complement each other. Consequently, pruning methods which are based on *one* particular property of the hierarchy,

such as Minimal Path-Value Pair Pruning, which only looks at whether or not a node in the Galois lattice introduces new path-value pairs, do not lead to very good results, at least for some lexicons. It seems that one has to take several properties of a node and its context into account to make a relatively accurate decision about whether or not to prune it. Hence, the fine-grainedness offered by maximum entropy models may improve the performance of a hierarchy induction algorithm.

Chapter 8

Conclusion

8.1 Results and Contribution

This thesis proposed a new approach to automatic inheritance hierarchy construction. Instead of starting with a formal criterion of hierarchy quality (or a set of those), as has been done in previous approaches (Petersen 2001; Barg 1996a; Light 1994), the approach suggested here uses supervised machine learning to stochastically model hierarchy quality, using a manually built hierarchy as training data. This approach has two main advantages:

1. Different contextual measures are automatically combined into one model. This allows the combination of a large number of contextual measures (nearly 70 in the system used here) and permits the use of a wide variety of measures, taking into account interactions between path-value pairs, overlap between nodes etc. This leads to a much more fine-grained model of hierarchy quality.

2. Using a manually built hierarchy as the basis for a model of hierarchy quality makes it much more likely that the model will lead to hierarchies which are *both* concise and linguistically plausible or insightful. Approaches that start with a set of formal criteria and then test whether these lead to good hierarchies, on the other hand, tend to focus on conciseness while disregarding plausibility, which is difficult to formalise.

The approach proposed in this thesis combines Galois lattices and maximum entropy modelling. The former are used to define the search space as the (partially ordered) set of intersections over the lexicon, where each intersection can be seen as a potential generalisation over the lexicon, i.e. as a potential node in the hierarchy. Maximum entropy models are trained in a

supervised machine learning step to estimate the likelihood of each node given its context. A manually built hierarchy serves as training data. To construct a hierarchy for a new lexicon, its Galois lattice is built and the trained model is applied to each intermediate node in the lattice. Nodes whose probability lies beneath a user set threshold are removed. Path-value pairs which can be inherited are also removed.

To evaluate a derived hierarchies, I proposed an automatic evaluation method which matches the derived hierarchy to the manually built hierarchy for that lexicon and then measures the similarity between the two. An automatic evaluation method is better suited for large hierarchies than the previously suggested manual evaluation methods.

To test the approach, an experimental system was implemented. The system supports one automatic and two semi-automatic construction architectures. The maximum entropy pruner was tested with both architectures and its performance was compared to two random baselines and to the rule-based approach suggested by Petersen (2001), which derives hierarchies which are minimally redundant with respect to path-value pairs.

For the experiments, the negative training examples had to be randomly sampled because the complete set was too big to be trained on. Some of the training sets led to models that performed only insignificantly better than their individual random baselines. This is probably due to unrepresentative training sets. However, the experiments suggested that those models that lead to significant results on one lexicon also lead to significant results on another. Hence it should be possible to select a good model on the basis of a development set. These models also achieved a recall that was significantly better than that achieved by the two random baselines and also significantly better than that of Petersen's method. This was true for both test lexicons and for two of the three pruning architectures (Interactive Semi-Automatic Pruning being the exception). This suggests that the approach is well suited for semi-automatic construction: it can significantly reduce the search space (by 33% to 75%) while still achieving a recall that is significantly better than that achieved by any of the other methods (at 79% to 94%). This effect was particularly noticeable for the larger of the two test lexicons. This is encouraging because semi-automatic construction is particularly useful for big lexicons, while hierarchies for very small lexicons can probably be faster constructed by hand.

Regarding precision, the results were inconclusive. The maximum entropy pruner outperformed Petersen's approach on one lexicon (though the difference in precision was not significant) while for the other lexicon the results were reversed and Petersen's approach achieved very good results. The fact that Petersen's approach achieved very different results for the two

lexicons indicates that manually built hierarchies differ with respect to how much path-value pair redundancy they contain. However, her approach only achieved a maximal f-score of 22%. That is, non of the hierarchies was actually minimally redundant with respect to path-value pairs; both hierarchies still contained some redundancy.

Several things can be learned from the work presented here, both from a theoretical as well as from a practical perspective. First, while lexical inheritance hierarchies do generally *reduce* some of the redundancy contained in a flat lexicon they do not *minimise* it. It seems that repeating some information, e.g. repeating attribute-value pairs is often deemed appropriate if it allows one to capture generalisations better. The initial hypothesis that one has to look beyond simple definitions of minimality thus has been confirmed by the findings.

Second, the fact that intra- and inter-node measures were found to complement each other, points to the fact that one needs a fine-grained criterion of hierarchy quality. It seems that the usefulness of a node is determined by both, its own properties (such as the size of its intension) and by its relation to nodes in its local context. While this is not particularly surprising, it is also bad news, in that it renders the task more complex: instead of looking at a node in isolation one has to look at local areas of a hierarchy. This may also mean that, ideally, one should aim to optimise a combination of decisions within an area rather than optimise each decision only with respect to previous decisions, as was done by the system proposed here. That is, it may be necessary to provide some kind of backtracking mechanism to improve the results. This conclusion is also supported by the fact that the pruning probabilities of a node were found to be quite sensitive to previous pruning decisions. Unfortunately, globally optimising pruning decisions, e.g. searching all *combination* of nodes and trying to find the best one, is probably intractable for a decent-sized lexicon. But it is possible that a locally defined backtracking mechanism can already result in improved performance.

The fact that it is a combination of factors which seems to determine the quality of a hierarchy, i.e. that one needs a complex criterion also suggests that one needs an automatic method of combining them. This suggests that a machine learning approach is indeed more suitable than hand-crafted rules. The results also showed that it is quite difficult to get good results automatically, so the best way forward may be a semi-automatic system which combines a machine-learning component with the possibility to intervene manually. For example, the automatic component could suggest likely classes and the user could then decide to accept them or not. Some suggestions of how automatic and manual methods could be interleaved have been made in this thesis.

It has to be stressed that the two components suggested here, i.e. Galois lattices and maximum entropy modelling, are independent of each other. That is, it is entirely possible to combine Galois lattices with another machine learning component or to combine maximum entropy modelling with another way to define the search space. If this is done, it may still be possible to re-use some of the contextual features suggested here. This is particularly true if another learning component is used, as none of the features is dependent on using maximum entropy modelling.[1] The features are a bit more dependent on using Galois lattices as a search space but they, too, could be used with another search space provided it is still possible to identify local node contexts, e.g. parents, siblings etc., and to identify basic set-theoretic constructs, such as intensions. As these are pretty basic constructs for the task, it is likely that they can indeed be identified even if the search space is defined differently.

On a practical level, the research done here provided an automatic evaluation method which can be re-used for any kind of automatically derived hierarchy.

8.2 Future Work

There are several areas for future research. One possible direction would be to look at the performance of the maximum entropy pruner in more detail than was possible here. For example, one could test it on more lexicons and explore the effects of more training data and higher sampling rates. The system could be extended to ALE grammars if no further suitable LKB grammars become available.

It would also be interesting to investigate how other machine learning techniques perform. The general set-up is quite modular and allows the use of pruning systems other than the maximum entropy pruner. For example, it would be interesting to experiment with k-Nearest Neighbour techniques (see e.g. Daelemans *et al.* (1996)) where a node would be pruned if the most similar node in the training data was.

At the moment the system is also very much experimental. To turn it into something of practical use several points have to be addressed. One potential problem is the space complexity of Galois lattices. While the Galois lattice for a lexicon is typically much smaller than the worst case predictions, due to the fact that lexicons have sparse context tables, it can still grow to a size that is no longer practical. If the Galois lattice of the training lexicon is too big for

[1] Remember, however, that maximum entropy modelling was chosen because it does not assume that the features are independent. As it is likely that the features are not, it is possible that a method which does make an independence assumption, such as Naive Bayes or Decision Tree Learning, performs less well.

the available hardware the problem can be addressed by undersampling the data. However, a bigger problem arises if the Galois lattice of the input lexicon is too big. This can happen if the lexicon itself is relatively large. But large lexicons are where a (semi-)automatic hierarchy construction method is of most use because manual construction becomes more and more impractical with the size of the lexicon. While one can create a partial Galois lattice for the training lexicon without too many problems, doing the same for the input lexicon can seriously affect the performance of the system as it can no longer be guaranteed that the search space is complete (i.e. the partial lattice may not contain all true positives). One way to address this problem would be by interleaving lattice construction and pruning. For example, one could construct a partial lattice, prune it and then construct the rest of the lattice. However, this is would mean that the pruner potentially only sees part of the context of a node. This is particularly true since there is no algorithm that can construct Galois lattices top-down (or bottom-up); Galois lattices are typically constructed by incrementally adding new objects, intersecting them with each of the existing objects and then inserting new intersections in the appropriate place in the lattice. This means that new objects can lead to new intersections at any level of the lattice and the pruner can no longer proceed in a strict top-down fashion. Instead it would have to be able to backtrack or at least traverse the (partial) lattice more than once. One possible extension would be to make the system incremental, i.e. whenever a new entry (or set of entries) is added to an existing hierarchy, the new intersections could be generated and added to the hierarchy and the pruner could then be applied to the hierarchy to prune it back.

For practical purposes, the system should also be extended to non-monotonic inheritance hierarchies. The reason why it cannot deal with those at the moment is that Galois lattices are inherently monotonic. For example, the lexicon in Table 8.1 gives rise to the Galois (semi-)lattice in Figure 8.1. It is impossible to prune this into a non-monotonic hierarchy in which *x* is the default value for the attribute A.

Entry 1	Entry 2	Entry 3	Entry 4	Entry 5	Entry 6
A:*x*	A:*x*	A:*x*	A:*x*	A:*y*	A:*y*

Table 8.1: Original lexicon

One solution would be to add the default, i.e. A:*x*, to all lexical entries as in Table 8.2. This would then give rise to the semi-lattice in Figure 8.2 which could be pruned into the desired non-monotonic hierarchy by removing *Node 2* and removing the attribute-value pair A:*x* from

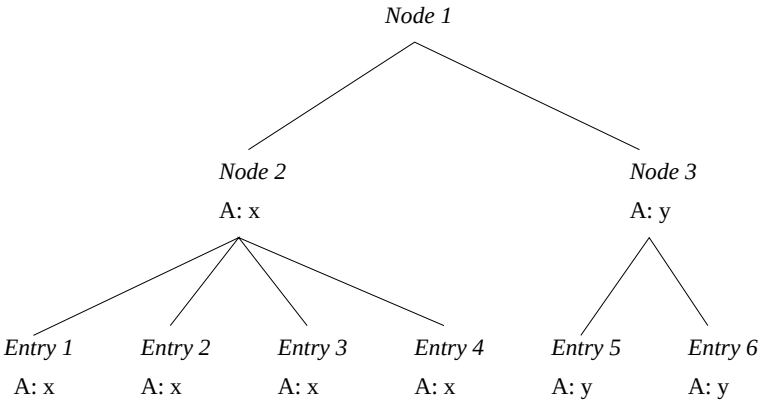


Figure 8.1: Galois (semi-)lattice for the data in Table 8.1

Node 3 and its descendants (Figure 8.3).[2]

Entry 1	Entry 2	Entry 3	Entry 4	Entry 5	Entry 6
A:*x*	A:*x*	A:*y*	A:*y*	A:*z*	A:*z*
A:*x*	A:*x*	A:*x*	A:*x*	A:*x*	A:*x*

Table 8.2: Lexicon for non-monotonic case, A:*x* is the default

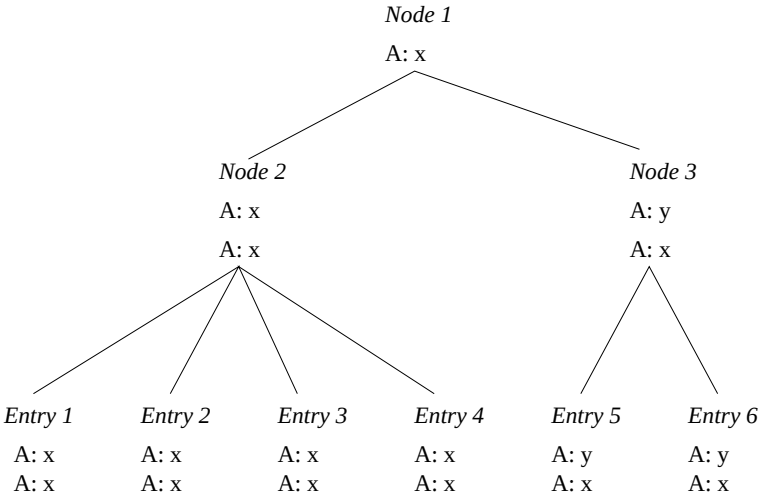


Figure 8.2: Galois (semi-)lattice for the data in Table 8.2

[2] I am grateful to Steve Finch for bringing this possibility to my attention.

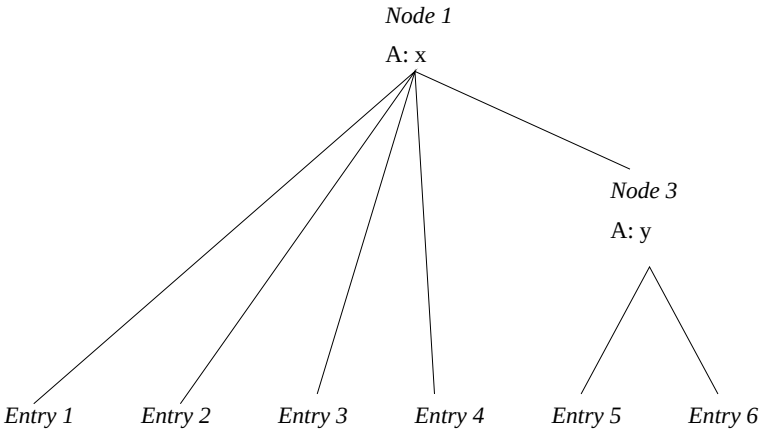


Figure 8.3: Non-monotonic inheritance hierarchy

Several issues need to be addressed for this approach to work. First, one needs to decide how to identify a possible default value for an attribute. One could simply choose the value that occurs most frequently in the lexicon. However, this does not work in all cases (see the example on page 62). A more sophisticated approach would be to use a supervised machine learning technique and a non-monotonic training hierarchy to try to find formal criteria for identifying the default. Second, the maximum entropy pruner needs to be extended to take defaults into account. For example whether of not one wants to keep *Node 2* in Figure 8.2 depends on whether or not one wants to make A:*x* a default in the hierarchy. Presumably one does not want use a default for all attributes so the pruner has to assess whether having a default in a particular situation makes sense or not. Third, to guarantee that the derived hierarchy is sound, one needs to keep track of the original values. When pruning the lattice it has to be checked whether each entry inherits the correct attribute-value pair set. If an entry inherits a wrong value for one of the attributes, the correct value has to be kept in the entry's overt intension to override the default value.

However, the most promising direction for future research is the combination of the approach suggested in this thesis with a powerful data exploration tool for semi-automatic hierarchy construction. One possibility would be to use ideas from Formal Concept Analysis (FCA, Ganter and Wille 1999) for the data exploration part. FCA uses Galois lattices for the manual exploration and analysis of data. Several methods have been developed to navigate Galois lattices and to find pattern and regularities in it and research is currently underway to develop FCA

techniques which support the manual construction of lexical inheritance hierarchies.[3] However, Galois lattices are typically very large, even for relatively small lexicons. Manually exploring a lattice which contains more than 16,000 nodes (as does the lattice for Spanish lexicon used here) will be very time consuming, even if a powerful FCA tool is available. The approach suggested in this thesis, on the other hand, can reduce the Galois lattice for the Spanish lexicon to 4,182 nodes while still achieving 92% recall. But manually post-processing more than 4,000 nodes is still a lot of work. However, if insights from FCA could be applied to a lattice that has already been partially pruner, using the approach proposed in this thesis, semi-automatic construction of lexical inheritance hierarchies could become relatively fast and use-friendly.

[3]The idea of applying Formal Concept Analysis to the task of (semi-)automatic inheritance hierarchy construction is due to Wiebke Petersen, who is researching this possibility as part of her PhD (personal communication, July 2003).

Bibliography

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, **23**(4), 597–618.

Ait-Kaci, H. (1984). *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures*. Ph.D. thesis, University of Pennsylvania.

Andry, F., Fraser, N., McGlashan, S., Thornton, S., and Youd, N. J. (1992). Making DATR work for speech: Lexicon compilation in SUNDIAL. *Computational Linguistics*, **18**(3), 245–267.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Barg, P. (1994). Automatic acquisition of DATR theories from observations. Technical report, Universität Düsseldorf.

Barg, P. (1996a). *Automatischer Erwerb von linguistischem Wissen. Ein Ansatz zur Inferenz von DATR-Theorien*. Max Niemeyer, Tübingen.

Barg, P. (1996b). Erstellung von Lexikoneinträgen aus Merkmalsstrukturen. In D. Gibbon, editor, *Proceedings of KONVENS 1996*, pages 249–254.

Barg, P. and Kilbury, J. (2000). Incremental identification of inflectional types. In *Proceedings of COLING 2000*, pages 49–54.

Barg, P. and Walther, M. (1998). Processing unknown words in HPSG. In *Proceedings of COLING/ACL 1998*, pages 91–95.

Basili, R., Pazienza, M. T., and Vindigni, M. (1997). Corpus-driven unsupervised learning of verb subcategorization frames. In *Proceedings of the 5th Conference of the Italian Association for Artificial Intelligence*, pages 159–170.

Berger, A. L., Pietra, S. A. D., and Pietra, V. J. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, **22**(1), 39–71.

Bloomfield, L. (1933). *Language*. Henry Holt and Company, New York.

Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. (1998). Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the 6th Workshop on Very Large Corpora*, pages 152–160.

Boudelaa, S. and Gaskell, M. G. (2000). In search of the minority default: the case of Arabic plurals. In *Proceedings of CogSci 2000*, pages 48–53.

Bouma, G. (1992). Feature structures and nonmonotonicity. *Computational Linguistics*, **18**, 183–203.

Briscoe, E. J., Copestake, A., and de Paiva, V., editors (1993). *Inheritance, Defaults and the Lexicon*. Cambridge University Press, Cambridge.

Briscoe, T. and Carroll, J. (1997). Automatic extraction of subcategorization from corpora. In *Proceedings of ANLP 1997*, pages 356–363, Washington, DC.

Briscoe, T., Copestake, A., and Lascarides, A. (1995). Blocking. In P. Saint-Dizier and E. Viegas, editors, *Computational Lexical Semantics*, pages 271–302. Cambridge University Press, Cambridge.

Burstein, J., Marcu, D., Andreyev, S., and Chodorow, M. (2001). Towards automatic classification of discourse elements in essays. In *Proceedings of ACL 2001*, pages 90–97.

Cahill, L. J. (1993). Some reflections on the conversion of the TIC lexicon into DATR. In T. Briscoe, V. de Paiva, and A. Copestake, editors, *Inheritance, Defaults, and the Lexicon*. Cambridge University Press, Cambridge.

Cahill, L. J. (1994). An inheritance-based lexicon for message understanding systems. In *Proceedings of ANLP 1994*, pages 211–212.

Cahill, L. J. (1998). Automatic extension of a hierarchical multilingual lexicon. In *Proceedings of the ECAI 1998 Workshop on Multilinguality in the Lexicon*, pages 16–23.

Carpenter, B. (1992). *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Cambridge University Press, Cambridge.

Carpenter, B. and Penn, G. (2001). *The Attribute Logic Engine User's Guide*. `http://www.cs.toronto.edu/~gpenn/ale/files/aleguideA4.ps.gz` (16.5.03).

Carpenter, R. (1993). Skeptical and credulous default unification with application to templates and inheritance. In Briscoe *et al.* (1993), pages 13–37.

Carpineto, C. and Romano, G. (1996). A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, **24**, 95–122.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, MA.

Chen, S. F. and Rosenfeld, R. (1999). A Gaussian prior for smoothing maximum entropy models. Technical Report CMUCS-99-108, Carnegie Mellon University.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris, Dordrecht.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row, New York.

Clahsen, H. (1999). Lexical entries and rules of language: A multidisciplinary study of German inflection. *Behavioral and Brain Sciences*, **22**, 991–1060.

Copestake, A. (1990). An approach to building the hierarchical element of a lexical knowledge base from a machine readable dictionary. Technical report, ACQUILEX WP NO. 8.

Copestake, A. (1992). *The Representation of Lexical Semantic Information*. Ph.D. thesis, University of Sussex.

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, CA.

Curran, J. R. and Clark, S. (2003). Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of EACL 2003*, pages 91–98.

Daelemans, W. and Smedt, K. D. (1994). Default inheritance in an object-oriented representation of linguistic categories. *International Journal of Human-Computer Studies*, **41**, 149–177.

Daelemans, W. and van der Linden, E.-J. (1991). Evaluation of lexical representation formalisms. In J. van Eijck and W. Meyer-Viol, editors, *Computational Linguistics in the Netherlands 1991. Papers from the Second CLIN Meeting*, pages 54 – 67, Utrecht. University of Utrecht.

Daelemans, W., Smedt, K. D., and Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics*, **18**(2), 205–218.

Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). MBT: A memory-based part of speech tagger. In *Proceedings of the 4th Workshop on Very Large Corpora*, pages 14– 27.

Darroch, J. N. and Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, **43**(5), 1470–1480.

deGroot, M. and Schervish, M. J. (2002). *Probability and Statistics*. Addison-Wesley, 3rd edition.

Della Pietra, S., Pietra, V. D., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions Pattern Analysis and Machine Intelligence*, **19**(4), 380–393.

Dunning, T. (1994). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, **19**(1), 61–74.

Emele, M. G. and Zajac, R. (1990). Typed unification grammars. In *Proceedings of COLING 1990*, pages 293–298.

Estabrooks, A. and Japkowicz, N. (2001). A mixture-of-experts framework for text classification. In *Proceedings of CoNLL 2001*, pages 76–83.

Evans, R. and Gazdar, G. (1989a). Inference in DATR. In *Proceedings of EACL 1989*, pages 66–71.

Evans, R. and Gazdar, G. (1989b). The semantics of DATR. In A. G. Cohn, editor, *Proceedings of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 79–87, London. Pitman/Morgan Kaufmann.

Evans, R. and Gazdar, G. (1996). DATR: a language for lexical knowledge representation. *Computational Linguistics*, **22**(2), 167–216.

Evans, R., Gazdar, G., and Weir, D. (1994). Using default inheritance to describe LTAG. In *3e Colloque International sur les grammaires d'Arbres Adjoints (TAG+3)*, Technical Report TALANA-RT-94-01, Paris. Université Paris 7, TALANA.

Evans, R., Gazdar, G., and Weir, D. (1995). Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. In *Proceedings of ACL 1995*, pages 77–84.

Ewing, G. (1997). An evaluation of Eiffel's inheritance mechanism. `http://www.elj.com/elj/v1/n1/gew/` (10.3.2000).

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of IJCAI 1993*, pages 1022–1027.

Fisher, D. and Langley, P. (1985). Approaches to conceptual clustering. In *Proceedings of IJCAI 1985*, pages 691 – 697.

Flickinger, D. and Nerbonne, J. (1992). Inheritance and complementation: A case study of *easy* adjectives and related nouns. *Computational Linguistics*, **18**(3), 269–310.

Flickinger, D. P. (1987). *Lexical Rules in the Hierarchical Lexicon*. Ph.D. thesis, Stanford University.

Fraser, N. M. and Hudson, R. A. (1992). Inheritance in word grammar. *Computational Linguistics*, **18**(2), 133–158.

Ganter, B. and Wille, R. (1998). Applied lattice theory: Formal concept analysis. In G. Grätzer, editor, *General Lattice Theory*, pages 591–605. Birkhäuser Verlag, Basel.

Ganter, B. and Wille, R. (1999). *Formal Concept Analysis. Mathematical Foundations*. Springer Verlag, Berlin.

Gibbon, D. (1992). ILEX: A linguistic approach to computational lexica. In U. Klenk, editor, *Computatio Linguae: Zeitschrift für Dialektologie & Linguistik*, pages 32–53. Franz Steiner Verlag, Stuttgart. Beiheft 73.

Girard, R. and Ralambondrainy, H. (1997). Computing a concept lattice from structured and fuzzy data. In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems*, volume 1, pages 135–142.

Godin, R., Missaoui, R., and Alaoui, H. (1995a). Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, **11**(2), 246–267.

Godin, R., Mineau, G., Missaoui, R., and Mili, H. (1995b). Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. *Revue d'Intelligence Artificielle*, **9**(2), 105–137.

Godin, R., Mili, H., Mineau, G. W., Missaoui, R., Arfi, A., and Chau, T.-T. (1998). Design of class hierarchies based on concept (Galois) lattices. *Theory and Practice of Object Systems*, **4**(2), 117–134.

Grover, C., Brew, C., Mananhar, S., and Moens, M. (1994). Priority union and generalisation in discourse grammars. In *Proceedings of ACL 1994*, pages 17–24.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING 1992*.

Hudson, R. (1984). *Word Grammar*. Blackwell, Oxford.

Jackendoff, R. (1977). *X' Syntax: A Study of Phrase Structure*. MIT Press, Cambridge, MA.

Japkowicz, N. (2000a). The class imbalance problem: Significance and strategies. In *Proceedings of IJCAI 2000*, pages 111–117.

Japkowicz, N. (2000b). Learning from imbalanced data sets: A comparison of various strategies. In *Papers from the AAAI Workshop on Learning from Imbalanced Data Sets. Tech. Rep. WS-00-05*, pages 10–15, Menlo Park, CA. AAAI Press.

Joshi, A. and Shabes, Y. (1991). Tree-adjoining grammars and lexicalized grammars. Technical report, University of Pennsylvania.

Kasper, R. and Rounds, W. (1986). A logical semantics for feature structures. In *Proceedings of ACL 1986*, pages 257–266.

Keller, B. (1993). Feature logics, infinitary descriptions and grammar. CSLI Lecture Notes no. 44, CSLI, Stanford, CA.

Kilbury, J., Barg, P., and Renz, I. (1994). Simulation Lexikalischen Erwerbs. In S. Felix, C. Habel, and G. Rickheit, editors, *Kognitive Linguistik: Repräsentation und Prozesse*, pages 251–271. Westdeutscher Verlag, Wiesbaden.

Kilgarriff, A. (1993). Inheriting verb alternations. In *Proceedings of EACL 1993*, pages 213–221.

Koeling, R. (2000). Chunking with maximum entropy models. In *Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal*, pages 139–141.

Koenig, J.-P. (1999). *Lexical Relations*. Stanford Monographs. CSLI Publications, Stanford University, C.A.

Koenig, J.-P. and Jurafsky, D. (1994). Type underspecification and on-line type construction in the lexicon. In *Proceedings of the 13th West Coast Conference on Formal Linguistics*.

König, E. (1999). LexGram. A practical categorial grammar formalism. *Journal of Language and Computation*, **1**(1), 33–52.

Kourie, D. G. and Oosthuizen, G. D. (1998). Lattices in machine learning: Complexity issues. *Acta Informatica*, **35**, 269–292.

Krieger, H.-U. and Nerbonne, J. (1993). Feature-based inheritance networks for computational lexicons. In Briscoe *et al.* (1993), pages 90 –136.

Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets. In *Proceedings of ICML 1997*, pages 179–186.

Lapata, M. (2000). *A Corpus-based Investigation of Systematic Polysemy*. Ph.D. thesis, University of Edinburgh.

Lascarides, A. and Copestake, A. (1999). Default representation in constraint-based frameworks. *Computational Linguistics*, **25**(1), 55–105.

Lascarides, A., Briscoe, E. J., Asher, N., and Copestake, A. (1996). Order independent and persistent typed default unification. *Linguistics and Philosophy*, **19**, 1–89.

Levin, B. (1993). *English Verb Classes and Alternations*. The University of Chicago Press, Chicago.

Light, M. (1994). Classification in feature-based default inheritance hierarchies. In *Proceedings of KONVENS 1994*.

Light, M. (1996). *Morphological Cues for Lexical Semantics*. Ph.D. thesis, Department of Computer Science, University of Rochester.

Lindig, C. (2000). Fast concept analysis. In *Proceedings of the 8th International Conference on Conceptual Structures*, pages 152–161.

Ling, C. X. and Li, C. (1998). Data mining for direct marketing: Problems and solutions. In *Proceedings of KDD 1998*, pages 73–79, New York.

Lüngen, H. (2002). *A Hierarchical Model of Word Formation in Spoken German*. Ph.D. thesis, Universität Bielefeld.

Lüngen, H. and Sporleder, C. (1999). Automatic induction of lexical inheritance hierarchies. In J. Gippert, editor, *Multilinguale Corpora. Codierung, Strukturierung, Analyse (11. Jahrestagung der Gesellschaft für linguistische Datenverarbeitung)*, pages 42–52. Enigma Corporation, Prague.

Malouf, R. (1998). *Mixed Categories in the Hierarchical Lexicon*. Ph.D. thesis, Stanford University.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL 2002*, pages 49–55.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. In *Proceedings of UCAI 2003*.

McCarthy, J. and Prince, A. (1990). Foot and word in prosodic morphology: The Arabic broken plural. *Natural Language and Linguistic Theory*, **8**, 209–283.

Messmer, B. T. (1996). *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. Ph.D. thesis, Uni Bern.

Meurers, W. D. (2001). On expressing lexical generalizations in HPSG. *Nordic Journal of Linguistics*, **24**(2), 161–217.

Michalski, R. S. and Stepp, R. E. (1983). Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, pages 331–363. Morgan Kaufmann, Los Altos, CA.

Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition without gazetteers. In *Proceedings of EACL 1999*, pages 1–8.

Mullen, T. and Osborne, M. (2000). Overfitting avoidance for stochastic modelling of attribute-value grammars. In *Proceedings of CoNLL 2000*.

Nigam, K., Lafferty, J., and McCallum, A. (1999). Using maximum entropy for text classification. In *Proceedings of the IJCAI 1999 Workshop on Machine Learning for Information Filtering*, pages 61–67.

Njiwoua, P. and Nguifo, E. M. (1997). A parallel algorithm to build concept lattice. In *Proceedings of the 4th Groningen International Information Technology Conference for Students*, pages 103–107.

Osborne, M. (2002). Using maximum entropy for sentence extraction. In *Proceedings of the ACL 2002 Workshop on Automatic Summarization*.

Petersen, W. (2001). A set-theoretic approach for the induction of inheritance hierarchies. In *Proceedings of the Joint Conference on Formal Grammar and Mathematics of Language*. (Electronic Notes in Theoretical Computer Science 53).

Poesio, M. and Mikheev, A. (1998). The predictive power of game structure in dialogue act recognition: Experimental results using maximum entropy estimation. In *Proceedings of the 5th International Conference on Spoken Language Processsing*, pages 405–408, Sydney, Australia.

Pollard, C. and Sag, I. A. (1987). *Information-Based Syntax and Semantics*, volume 1: Fundamentals. (CSLI Lecture Notes 13). CSLI, Stanford, CA.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Priss, U. E. (1998). The formalization of WordNet by methods of relational concept analysis. In C. Fellbaum, editor, *WordNet: An Electronic Lexical Database and Some of its Applications*, pages 179–96. MIT Press, Cambridge, MA.

Quillian, M. (1968). Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, Cambridge, MA.

Quirino Simões, A. P. (2001). *Spanish Clitics. A Computational Model*. Master's thesis, Universität Bielefeld.

Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, Computer and Information Science, University of Pennsylvania.

Reinhard, S. and Gibbon, D. (1991). Prosodic inheritance and morphological generalisations. In *Proceedings of EACL 1991*, pages 131–136.

Riehemann, S. Z. (1998). Type-based derivational morphology. *Journal of Comparative Germanic Linguistics*, **2**, 49–77.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell, J. T., and Johnson, M. (2002). Parsing the Wall Street Journal using lexical functional grammar and discriminative estimation techniques. In *Proceedings of ACL 2002*, pages 271–278.

Russell, G., Ballim, A., Carroll, J., and Warwick-Armstrong, S. (1992). A practical approach to multiple default inheritance for unification-based lexicons. *Computational Linguistics*, **18**, 311–337.

Sag, I. A. and Wasow, T. (1999). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.

Sahami, M. (1995). Learning classification rules using lattices. In *Proceedings of ECML 1995*, pages 343–346, Berlin. Springer Verlag.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. CSLI, Stanford, CA.

Siegel, D. (1979). *Topics in English Morphology*. Garland Publishing, New York/London.

Siegel, M. (2000). HPSG analysis of japanese. In W. Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer Verlag, Berlin.

Sporleder, C. (1999). *Learning Lexical Generalisations. An Operational Evaluation of Current Machine Learning Methods*. Master's thesis, Universität Bielefeld.

Sporleder, C. (2002a). A Galois lattice based approach to lexical inheritance hierarchy learning. In *Proceedings of the ECAI 2002 Workshop on Machine Learning and Natural Language Processing for Ontology Engineering*, pages 23–28.

Sporleder, C. (2002b). Learning lexical inheritance hierarchies with maximum entropy models. In *Proceedings of the ESSLLI 2002 Workshop on Machine Learning Approaches in Computational Linguistics*, pages 56–72.

Sporleder, C. (2002c). Machine learning of lexical inheritance hierarchies: Linguistic plausibility vs. minimal redundancy. In *Proceedings of the ACL 2002 Student Research Workshop*, pages 66–71.

Steinbrecher, D. (1995). MSEG: Morphologische Segmentierung deutscher Wortformen. Technical report, Verbmobil Memo 99. Universität Bielefeld.

Stumme, G. (1996). Exploration tools in Formal Concept Analysis. In E. Diday, Y. Lechevallier, and O. Opitz, editors, *Ordinal and Symbolic Data Analysis*, pages 31–44. Springer Verlag, Berlin.

Touretzky, D. S. (1986). *The Mathematics of Inheritance Systems*. Morgan Kaufmann, Los Altos, CA.

v. Chawla, N., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, **16**, 321–357.

Valtchev, P., Missaoui, R., and Lebrun, P. (2000). A fast algorithm for building the Hasse diagram of a Galois lattice. In *Proceedings of the LaCIM 2000 Conference on Combinatorics, Computer Science and Applications*, pages 293–306.

van der Merwe, F. J. and Kourie, D. G. (2001). A lattice-based data structure for information retrieval and machine learning. In *Proceedings of the International Workshop on Concept Lattice-Based Theory, Methods and Tools for Knowledge Discovery in Databases*, pages 77–90.

Vijay-Shanker, K. and Schabes, Y. (1992). Structure sharing in lexicalized tree-adjoining grammars. In *Proceedings of COLING 1992*, pages 205–211.

Villavicencio, A. (2002). *The Acquisition of a Unification-Based Generalised Categorial Grammar*. Ph.D. thesis, University of Cambridge.

Wegner, P. (1990). Concepts and paradigms of OO programming. *OOPS Messenger*, **1**(1), 8–87.

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman, San Francisco, CA.

Wood, M. (1993). *Categorial Grammars*. Linguistic Theory Guides. Routledge, London and New York.

Yang, G.-C. and Oh, J. (1993). Knowledge acquisition and retrieval based on conceptual graphs. In *Proceeding of the 1993 ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice*, pages 476–481.

Young, M. and Rounds, B. (1993). A logical semantics for nonmonotonic sorts. In *Proceedings of ACL 1993*, pages 209–215.

Zernik, U. (1987). Language acquisition: Learning a hierarchy of phrases. In *Proceedings of IJCAI 1987*, pages 125–132.

Zernik, U. and Dyer, M. G. (1986). Disambiguation and language acquisition through the phrasal lexicon. In *Proceedings of COLING 1986*, pages 247–252.