Machine Learning and Phonological Classification — *On the Induction of Paradigmatic Decision Trees*

Moritz Neugebauer
moritz.neugebauer@ucd.ie
Department of Computer Science
University College Dublin

ABSTRACT

In knowledge-based speech technology components, information on phonological classes is frequently employed to determine neighbouring units in the process of segmentation of a given phonetic utterance. In contrast to phonotactic implications of this kind, the focus in this paper is on dependencies within the segmental unit. Here, we investigate possible well-formed combinations of phonological features within a single speech sound based on the assumption that segments are composed of a finite set of phonological features and that feature combinations are partially predictable.

The notion of machine learning presented in this paper relies on an algorithmic method to induce sets of sounds and their set-theoretical relations. Despite its generic nature, the presentation of our algorithm to induce implications among sets of sounds will draw on articulatory-based features. In addition to the computation of set descriptions the presented algorithm provides for an automatically generated lattice representation visualizing inheritance relationships. Selected work on phonological feature theory is reviewed and an extension concerning typed feature structures is presented along with an implementation as provided in Neugebauer (2003).

1. INTRODUCTION - PRELIMINARIES OF PARADIGMATIC DECISION TREES

A prominent means to reduce the error-rate in the automatic segmentation of continuous speech consists of decision trees which are built on a set of linguistic questions concerning the immediate environment of a single speech sound. Drawing on an example from Hwang (1993), the knowledge-based tree is constructed from yes/no-questions of the kind displayed in (1):

(1)
- a. Is the left-phone voiced?
- b. Is the left-phone a retroflex?
- c. Is the right-phone a nasal?
- d. Is the right-phone a velar stop?

This method intends to determine well-formedness of the potential output string and the decisions made in trees of this kind are based on *syntagmatic* relations. The short extract of linguistic questions

given above suffices to show that, while the atomic unit of linguistic description in these decision trees is the phonological segment, a subsegmental layer in terms of articulatory feature structures is assumed to account for a more detailed distinction among segmental lexical entries. Although this approach also refers to classes of sounds by reference to feature labels, intelligent decision-making below the segment is not envisaged. This is due to a system architecture which aims to extract segments from a given signal rather than features. In contrast to systems built on segment models, this paper pursues another line of research which inherently builds on feature models. In the following we will argue that if articulatory features are the primary information achieved in the recognition process, another field of application for decision trees emerges. Instead of segmental syntagmatic constraints, featural paradigmatic constraints gain importance. Paradigmatic decision-making of this kind crucially builds on lexical information on well-formed versus ill-formed cooccurrences of features naturally in addition to constraints on the combinatorics of feature bundles (i.e. segments). Constraints of the latter kind are assumed to be accounted for in terms of phonotactic models which can be built efficiently using finite-state machines (cf. Carson-Berndsen 2000).
In this paper, classes of speech sounds are presented as the result of generalizations over complex segmental feature structures. It is assumed that logical interdependencies between subsegmental articulatory features such as set relations can be extracted from representations which are multilinear in nature; according to this view the examples in (1) deal with features concerning phonation (a.), place of articulation (b.), manner of articulation (c.) and combinations of those (d.). In multilinear representations each of these feature classes would be represented on an individual tier as visualized in Figure 1 below.

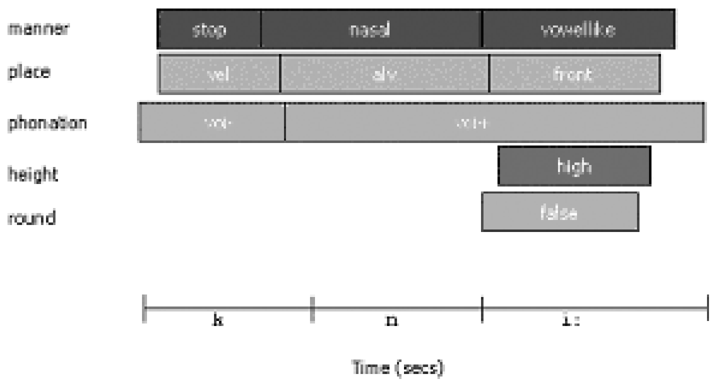

Figure 1: Multilinear Representation

The work presented in the following sections is loosely based on the concept of *feature geometry* in generative phonology (cf. Clements 1985) as we adhere to the general idea of natural classes of *features* (i.e. manner, place, etc. features) along with the claim that these features are highly dependent.[1] A type-theoretic interpretation of multilinearity is applied in section 3.2. building on Neugebauer (2003).
By constructing inheritance hierarchies over the set of given features, we acquire additional knowledge to our system of articulatory features since now underspecified feature slots can be overcome simply

[1]Note that natural classes of features are not identical to natural classes of *segments*.

by reference to our previously built knowledge base. Even in cases where no single solution to an underspecified representation can be delivered, our approach will still put us in a position where we can compute the set of possible sounds. By this method, we reduce the set of possible well-formed feature structures significantly. The potential usefulness of lattice theory for phonological classification is explored in this paper as well as its generic applicability to feature systems. Section 2 introduces necessary basics for the induction of sets and inheritance hierarchies along with an informal description of our underlying algorithm. Section 3 presents a case study provides and hints towards its application, section 4 concludes with an outline of directions for further research.

2. FOUNDATIONS FROM MATHEMATICAL INDUCTION

In this paper, familiarity with set-theoretical fundamentals is assumed while general ideas concerning the inductive method are explained. The following section sketches an efficient algorithm which computes set descriptions together with their explicit lattice representation. For our current purposes the presentation will be kept in an informal fashion, i.e. aspects like complexity analysis and running time evaluation will not be the center of investigation. We refer to Neugebauer (ms.) for a thorough treatment of these issues. In the next subsection we will present the nature of input data to the algorithm which is exemplified later in section 3.

2.1. Inductively Defined Sets

The method we will apply to deduce sets of speech sounds from a given number of segmental feature structures is described in the following, drawing on a proof technique known as mathematical induction. To be able to carry out deductive reasoning of this kind we rely on the (informal) definitions below.

An inductive definition of a set S is a definition that consists of a collection of rules. The rules are of two types. *Basis rules* are ones that state unconditionally that certain elements are in the set whereas *inductive rules* state that an element is in the set if certain other elements are in the set. The element that is put into the set by an inductive rule is called the conclusion of the rule. The elements that have to be in the set in order for the conclusion to be in the set are called the hypotheses of the rule. The elements of S are those objects that can be shown to be in S by a finite number of the rules.

There are two ways to show whether or not an object is an element of S; one way is with a line-by-line proof. Each line in the proof is either an element put into S by a basis rule or it is the conclusion of an inductive rule whose hypotheses appear as earlier lines in the proof. The last line of the proof is the element that the proof shows is in S.

The second way to show that an object is an element of S is with a labelled tree that shows how the object is built up using the rules. Each leaf is labelled with an element put into S by a basis rule. Each interior node is labelled with the conclusion of an inductive rule whose hypotheses are the labels of the children of the node. The label of the root of the tree is the element that is shown to be in S by the tree. In our approach, both basis and induction rules are generated automatically from the basic feature structures, i.e. our data (cf. section 3.1. below). Before we proceed to decribe the induction of lattices the following example is intended to demonstrate the two kinds of rules described above. Consider a subset S of the natural numbers by the following inductive definition

(2) a. 0 is $\in S$

b. If n $\in S$, then $n + 1 \in S$

The first rule is a basis rule, and the second one is an inductive one with one hypothesis. The fact that S is all of N is simply a restatement of the principle of induction for the natural numbers.
When we are dealing with more than one set as in the above example, we need to consider relations between sets such as subsets and supersets. Besides the computation of set relations another goal might be to construct a lattice visualizing the subsumption relationships among the sets and elements (or singleton sets). Just as for a single set defined inductively, there is a principle of structural induction for several sets defined by simultaneous induction. Let $\mathcal{S}$ be a collection of sets defined by simultaneous inductive definition, and for each $S \in \mathcal{S}$, let P_s be a property of the elements of S. Suppose that

(3) a. for every basis rule in the definition of $\mathcal{S}$, if x is put into S by the rule, then $P_s(x)$ is true;

b. for every inductive rule in the definition of $\mathcal{S}$, if each of the hypotheses of the rule has the property appropriate for the set of which it is a member, then the conclusion has the property appropriate for the set into which it is put.

Then, for all $S \in \mathcal{S}$ and $x \in S$, $P_s(x)$ is true.

For the purposes of this paper, the elements of description and the members of sets as described above are taken to be segmental units of spoken language. Such a segment is defined by a set of properties which are usually referred to as *features* in most work in the field of language technology.[2] The sets of segments which can be induced from those individual segmental descriptions are sometimes referred to as *natural classes*. The knowledge base - storing a finite set of segments and features - will be referred to as the lexicon. For a finite lexicon *(Seg,Feat,Lex)* all set descriptions can be computed by finding all set descriptions of sets which, of course, entails set relations as subsumption, union and disjunction. Starting at the smallest set description of a lattice, the algorithm recursively computes all set descriptions *SetDesc(Seg,Feat,Lex)*. Given a set description *(S,F)* that is distinct from $\top$ of *SetDesc*, the set E contains set descriptions greater than the initial description *(S,F)* where $S \in Seg$ and $F \in Feat$. The set E contains at most $|Seg|$ members; each member is greater than *(S,F)* but not necessarily an upper node in the lattice. The algorithm *SetDescSet* now captures these insights in that it computes the upper nodes of *(S,F)*. For every set description generated by $s \in Seg$ (where $s \neq S$) it tests all other elements with respect to two questions:

(4) a. Is the element different from s?

b. Does it itself generate an upper node?

If the answer to the second question is 'yes' then the actual extent may not belong to an upper node in which case s is removed.
Assume for instance that S_1 is the extent of an upper node of (S,F). Both x and y generate S_1 and are considered by the algorithm in that order. Initially, both x and y are members of *Up* which contains elements from $Seg \neq S$ that generate upper sets (at the end of the algorithm *Up* is a minimal complete set of upper nodes). First, all members of S_1 different from x - y is among them - are checked against *Up*. y is found in *Up* and so is x (falsely) assumed not to generate an upper node and is removed from *Up*. Next, all elements x different from y are checked: x is no longer in *Up* and thus the set description generated

[2]This holds at least for all levels of abstraction where some kind of *feature logic* is assumed.

by y is known to be an upper node. Whenever a (set of) node(s) is generated by a number of elements from $Seg \neq S$, only the last one considered by the algorithm is detected as set description-generating and therefore stays in *Up*.
Given such an automated method for the generation of descriptions, the following subsection introduces an extension towards lattice generation.

2.2. Induction of Lattices

The above *SetDescSet* algorithm can be employed to recursively compute all set descriptions *SetDesc* of a finite lexicon by starting from the smallest set description of a lattice. Lattices represent certain orderings between elements of systems or domains of objects and the order-theoretic or topological properties of such ordered structures. Traditionally, the label $\top$ (read *top*) is assigned to the most general description in a given context. In the graphical representation it is the top-most node. Daughters of the $\top$-node denote disjoint sets while the bottom node (written $\perp$) of the lattice denotes inconsistency. Lattices can (also) be read as paradigmatic decision trees (cf. section 1) since every arc that connects to nodes represents an implication for the feature system in question. Of course, these implications might hold for individual elements as well as for whole sets. An algorithm such as *SetDescSet* provides us with all sets present in a feature system and thus implements the first step in the task of lattice generation.
The previous section introduced a mathematical method to induce partial orderings on sets, i.e. sets ordered by a reflexive, anti-symmetric and transitive relation. Before we go on to the induction of lattices, we introduce the notion of *posets*. Any partially ordered set A together with its ordering, i.e. $\langle A, \leq \rangle$ is called a poset. Lattices are now special cases of posets and the route taken in this paper is to define lattices which rely on a given poset.
A lattice can be regarded as a special case of a poset $\langle A, \leq \rangle$ if a $\sigma\{$a,b$\}$ (called *supremum*) and $\iota\{$a,b$\}$ (called *infimum*) exist for all a, $b \in A$. With *meet* and *join* we introduce two lattice operations; these operations are always binary. This allows us to characterize a lattice as an algebra, i.e. a non-empty set with two operations with certain algebraic properties. The operations are the following: the meet of a and b is given by $a \wedge b = \iota\{$a,b$\}$, and $a \vee b = \sigma\{$a,b$\}$ is called the join. The first three out of four properties of lattice operations are: idempotent law, commutative law and associative laws. The fourth property connects the two operations and we will refer to it as *absorption laws*. Note that if $a \leq b$, then $\iota\{$a,b$\} = a$, i.e. $a \vee b = a$, and dually, if $a \geq b$, then $\sigma\{$a,b$\} = a$, i.e. $a \wedge b = a$. Since $a \leq a \wedge b$ by definition of $\sigma\{$a,b$\}$, we let $a \wedge b$ substitue for b in the first equations to derive $a \wedge (a \vee b) = a$. Similarly, since $a \geq a \wedge b$ by definition of $\iota\{$a,b$\}$, we derive from the second equations $a \vee (a \wedge b) = a$. Thus we have established the two absorption laws of logic as given in (5):

(5) a. $a \wedge (a \vee b) = a$

b. $a \vee (a \wedge b) = a$

Any algebra with two binary operations that has these four properties (i.e. idempotent law, commutative law, associative and absorption laws) constitutes a lattice. The figure below shows the lattice L=$\langle\{1,2\},\leq\rangle$.

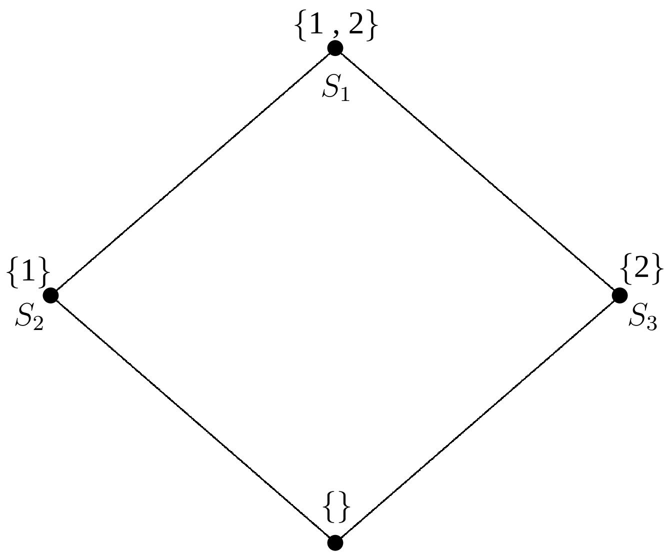

Figure 2: Example Lattice

We can see that the two pictured subsets are labelled with S_2 and S_3, respectively. These arbitrary labels give a hint as to the computation of sets from feature descriptions in our approach. It is assumed that the above lattice corresponds to a description in which a set description including two elements (here $\{1,2\}$) is further described by the assignment of subset relations which we might interpret as features. Following this interpretation, the element $\{1\}$ carries the feature labels S_1 and S_2 while differing from $\{2\}$ with respect to the feature assignment of S_3.
Coming back to our algorithm which we will call *Solus* (which stands for *Sorted-order Logic underlying Speech* and is a reformulation of our algorithm *SetDescSet*), we are already at a stage where we know that every set description *s* has two lists associated with it: the list s^* of its upper nodes and the list s_* of its lower nodes.
One set description may be shared by two different set descriptions as their upper nodes (a set description on a set). While the algorithm processes each of the two set descriptions, their shared upper node must be detected in order to get the correct relationship. For this purpose, all set descriptions are stored in a search tree *SetDesc*. Each time the algorithm finds a node, it searches (via lookup) in the tree *SetDesc* to find previously inserted instances of that set decription. In case the set description is found the existing lists of nodes are updated. Otherwise, the previously unknown set description is inserted into the tree. Due to lack of space we will not investigate further important issues regarding lattice generation concerning lattice order and complexity issues. As mentioned before we direct the reader to Neugebauer (ms.) for these details.
To sum up our final algorithm *Solus* has two algorithms as its basis which are employed to facilitate the following:

(6) a. calculate all set descriptions contained in a given lexicon
b. draw a sorted lattice corresponding to a given lexicon

Equipped with these automated methods the next section focusses on the specific problem of lattice generation from a set of feature structures. We consider the following to present an automated means for the acquisition of inheritance relationships as an extension to a linguistic knowledge base. Although the focus in the remaining section will be on speech related information, it should be clear that the algorithm presented above is generic in its architecture.

3. APPLICATION IN THE FIELD OF PHONOLOGICAL CLASSIFICATION: A CASE STUDY

The specific knowledge required to process speech automatically can grow immensely especially taking into account multiple languages, speakers and domains. The strategy in this paper so far has been to account for efficient knowledge acquisition by calculating all dependencies even in a complex feature system with the aim of inducing inheritance hierarchies. While the next section will exemplify the process of automated lattice building for a small vowel set, we will then proceed with an integration of our sorted lattice with a type discipline.

3.1. A lattice for a small vowel set

According to the format of multilinear representations introduced earlier in this paper, we assume that individual phonological segments are characterized by - in this case - five articulatory features which can be interpreted each as instances of five classes of features. Each of these feature classes is represented on a separarte tier. Set descriptions are now computed by $Seg \times Feat$, i.e. all binary relations defined in the lexical knowledge base. As a working example we define the five vowel system given in (7):

(7) a. *ah* : {voi,voc,semilo,cen,nrd}
b. *eh* : {voi,voc,semilo,frt,nrd}
c. *ih* : {voi,voc,semihi,frt,nrd}
d. *oh* : {voi,voc,semihi,bak,rd}
e. *uh* : {voi,voc,semihi,cen,rd}

For our example, the algorithm delivers in a first step the set descriptions starting with the ones displayed above. Further steps include the set description for $\top$ which is computed as: ({*ah, eh, oh, ih, uh*}; {voc, voi}). Here we learn that all segments of our sample lexicon share the two features denoting vocalic manner and voiced phonation. Contrastively, $\perp$ is imposed by our algorithm as consisting of the empty set description ({},{}). In the previous section we referred to $\top$ as *supremum* and to $\perp$ as *infimum*. Two interesting cases which result from the computation of all set descriptions should be mentioned additionally. Apart from featural information that is relevant for all segments and can in consequence be considered as being redundant (such as *voc* and *voi*, some features or feature (sub)sets refer to intermediate nodes in the sorted lattice. As a first example we consider the feature {*bak*}. A brief look at the above vowel set reveals that only one segment carries this particular feature. Accordingly, the algorithm computes a set description which shows that we can induce the vowel *oh* given only the featural information {*bak*}. As a second case we focus on the feature set {*voc,voi,cen*}. The set of segments which corresponds to this (underdetermined) feature structure consists in the segments {*ah,uh*} (for a more precise treatment of feature constraint inheritance refer to section 3.2.). Once all set descriptions are obtained, lattice generation is possible. If we assume that for each segment and for each feature one node is created in addition to $\perp$ and $\top$, we expect a maximum of sixteen nodes. For our current example thirteen nodes are created which can be easily confirmed by hand in this case as we are dealing with a very restricted lexicon: joint nodes are created for ({oh}; {bak}) and ({}; {voc, voi}) while the latter also labels the top node. This turns out to be the reason for the three nodes missing from the maximal number. All nodes created are listed below.

node $\perp$
node ({ah}; {})
node ({eh}; {})
node ({oh}; {bak})
node ({ih}; {})
node ({uh}; {})
node ({}; {voc,voi})
node ({}; {cen})
node ({}; {semilo})
node ({}; {nrd})
node ({}; {frt})
node ({}; {semihi})
node ({}; {rd})
node $\top$

Figure 3: Lattice node generation for the sample lexicon

The above minimal complete set of upper nodes is achieved at the end of the algorithm and has been introduced earlier as *Up*. From the induction of sets we now turn to the induction of lattices. Since we kept track of the overall structure by means of our search tree *SetDesc* all nodes can be created at the appropriate position in the lattice. Consequently, the set relations among the descriptions of sound classes can be visualized with the following lattice.

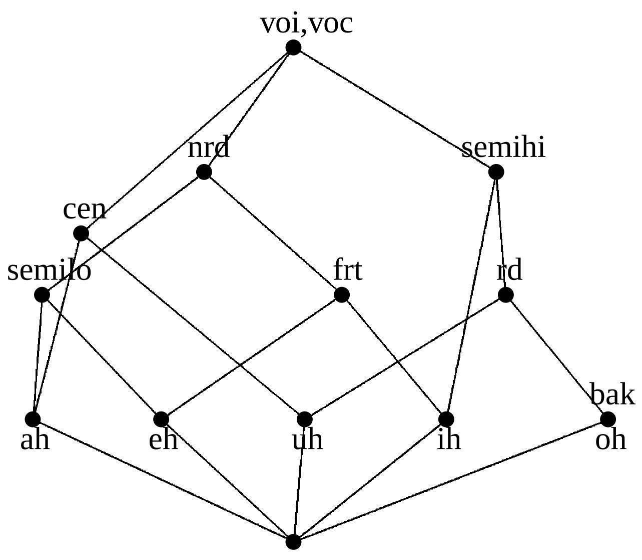

Figure 4: Lattice for the five vowels {ah,eh,ih,oh,uh}

Regarding an application of this represention as a paradigmatic decision tree (cf. section 1), we notice that we would need at least {cen,rd} as the minimal featural information to deduce a fully specified vocalic segment *uh*. The remaining features {voc,voi,semihi} are inherited from upper nodes. The topic of inheritance is now formulated applying a typed feature structure logic (cf. Carpenter 1992).

3.2. An integration with Typed Feature Structures

Taking the generated sorted lattice as a starting point, explicit type labels and constraints will now be assigned to the given structure. A previous implementation of a type discipline which models an inheritance relationship between phonological classes has been carried out earlier with a focus on constraint inheritance in phonological type hierarchies. But while Neugebauer (2003) leaves the actual acquisition of type hierarchies to future research, this paper attempts to fill this gap employing the induction algorithm introduced in section 2. The type assignment to the sorted lattice simply takes twelve labels *type_1*. . . *type_12* while we start the assignment on the $\top$ node of the lattice as visualized below.

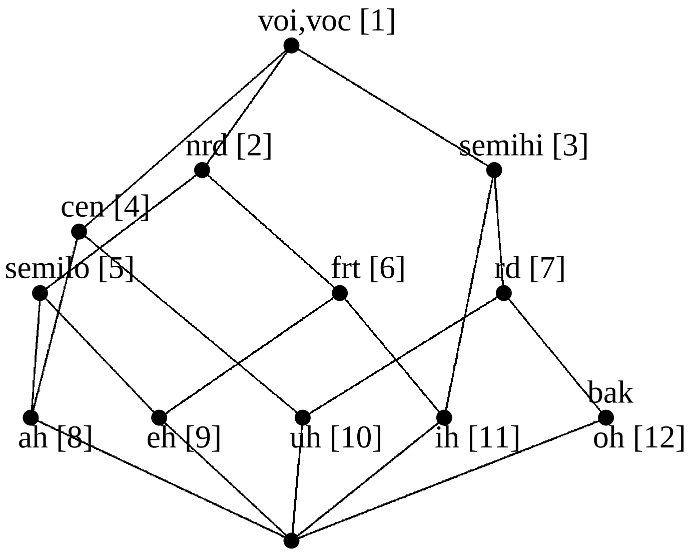

Figure 5: Type assignment for the vowel lattice

Porting this hierarchical structure to the *LKB* system for lexicon development (cf. Copestake 2002), a type hierarchy of the following kind can be created. Features in the above lattice will now be interpreted as constraints which can then be inherited among our set of types while respecting the hierarchical order. Segments are atomic types, i.e. *type_8* to *type_12*. However, we still have to investigate question of constraint inheritance in more detail. The sorted lattice induced by the *Solus* algorithm provides us with information on dependencies among articulatory features - here reinterpreted as feature constraints - which can be associated with our recently introduced type labels in a straightforward fashion. To start with, we consider the top node (labelled *type_1*) which has to carry two feature constraints shared by all five segments in our lexicon. While the values for manner and phonation attributes are common to all individual segments, the attributes concerning place, height and rounding information are introduced but not sufficiently specified yet. The values *pla, hei, rou* - as given in the Figure 7 below - do thus not refer to atomic types but to whole feature classes.

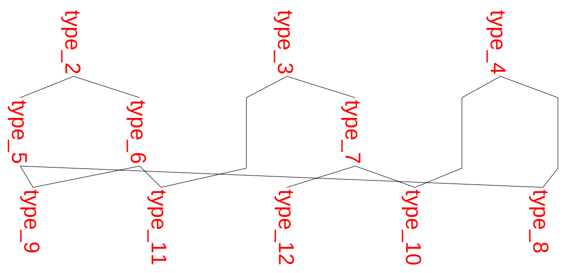

Figure 6: Generated type hierarchy (type_1 omitted)

This means, we assume a type labelled *pla* which dominates all possible values for the place attribute, i.e. {frt,cen,bak}. Before we point out further examples of constraint inheritance we provide a synopsis of all type labels and type definitions as used in our implementation. In addition, the column on the left hand side lists (sub)sets of segments corresponding to individual type labels.

set description	type label	type definition
ah,eh,ih,oh,uh	type_1	MANNER: voc PLACE: *pla* PHON: voi HEIGHT: *hei* ROUNDING: *rou*
ah,eh,ih	type_2	parent type_1 ROUNDING: nrd
ih,oh,uh	type_3	parent type_1 HEIGHT: semihi
ah,uh	type_4	parent type_1 PLACE: cen
ah,eh	type_5	parent type_2 HEIGHT: semilo
eh,ih	type_6	parent type_2 PLACE: frt
oh,uh	type_7	parent type_3 parent type_4 ROUNDING: rd
ah	type_8	parents type_4 and type_5

set description	type label	type definition
eh	type_9	parent type_5 parent type_6
uh	type_10	parents type_4 and type_7
ih	type_11	parent type_3 parent type_6
oh	type_12	parent type_5 PLACE: bak

Figure 7: Synopsis of set descriptions, type labels and type definitions

The intermediate types in our type hierarchy - that is *type_2* to type *type_7* as displayed in Figure 6 - each carry exactly one feature constraint. It should be clear that these constraints can only refer to attribute values which have not been instantiated thus far since it would otherwise lead to type clashes. Except for one segment, all atomic segmental types inherit combinations of only these types or more precisely the constraints those types introduce. For example, the type labelled *type_9* corresponds to the segment *eh* since all required feature specifications are unified in this type: *type_1* constrains its MANNER value to *voc* and its PHON value to *voi*, *type_2* contributes the necessary information for the ROUNDING attribute while *type_5* and *type_6* set the HEIGHT value to *semilo* and the PLACE specification to *frt*, respectively. As a special case, the atomic type *type_12* carries a unique constraint for PLACE specification since it is the only back vocalic segment in our lexicon.
In this subsection we hope to have shown that the sorted lattices computed by our algorithm can be interpreted as typed feature structures without any fundamental modifications. Furthermore, we would like to stress that the major task in designing large-coverage typed feature structure grammars lies in the classification of types and the development of type constraints. It is this specific task of deductive reasoning which can be automated by the *Solus* algorithm as it delivers all classes present in a given lexicon and also provides for a sorted lattice.

4. Further Research

In this paper a generic algorithm for the induction of inheritance relations between sets of speech sounds has been presented. Although the presentation was kept informal we hope to have shown the strengths of our approach which consists of an automated method for deductive reasoning over feature structures. An integration with type discipline has been explored, building on a case study of the task of phonological classification. The purpose of this case study was to establish the notion of a paradigmatic decision tree and its formal properties cast in the typed feature structure formalism. We consider the work presented in this paper to extend on that presented in Neugebauer (2003) where lexical knowledge representation in computational phonology is reviewed and phonological type hierarchies for subsegmental structure are mentioned. The encoding of this information in terms of types and type constraints is extended here by means of automatically acquired knowledge.
We are aware that the following important questions have to be investigated in future research. We consider questions concerning the algorithm and its application in the field of spoken language technology:

(8) a. How does the algorithm perform on larger lexica (larger sets of features and segments)?

b. How can it be employed on lower levels of spoken language?

Both questions are topics of current research while results concerning specifically the question of complexity and runtime evaluation are documented in Neugebauer (ms.).

ACKNOWLEDGEMENTS
This material is based upon works supported by the Science Foundation Ireland under Grant No. 02/IN1/I100.
The opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Science Foundation Ireland.

References

Carpenter, B. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press.

Carson-Berndsen, J. 2000. Finite State Models, Event Logics and Statistics in Speech Recognition. In: Sparck Jones, K.; G. J. M. Gazdar & R. M. Needham (eds.): Computers, Language and Speech: formal theories and statistical data. Philosophical Transactions of the Royal Society, Series A, Volume 358, issue no. 1769: 1255-1266.

Copestake, A. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications.

Hwang, M. 1993. *Subphonetic Acoustic Modeling for Speaker-Independent Continuous Speech Recognition*. PhD thesis. Carnegie Mellon University.

Neugebauer, M. 2003. Computational Phonology and Typed Feature Structures. Proceedings of the First CamLing Postgraduate Conference on Language Research. Cambridge. University of Cambridge. [http://cspeech.ucd.ie/~cliste/group/publications/2003CPandTFSCamLing.pdf]

Neugebauer, M. ms. Computational Phonology with Feature Terms, Set Descriptions and Lattice Algorithms. Department of Computer Science, University College Dublin.