A Computational Treatment of Lexical Rules in HPSG as Covariation in Lexical Entries

W. Detmar Meurers*
University of Tübingen

Guido Minnen
University of Tübingen

This paper proposes a new computational treatment of lexical rules as used in the HPSG *framework. A compiler is described which translates a set of lexical rules and their interaction into a definite clause encoding, which is called by the base lexical entries in the lexicon. This way, the disjunctive possibilities arising from lexical rule application are encoded as systematic covariation in the specification of lexical entries. The compiler ensures the automatic transfer of properties not changed by a lexical rule. Program transformation techniques are used to advance the encoding. The final output of the compiler constitutes an efficient computational counterpart of the linguistic generalizations captured by lexical rules and allows on-the-fly application of lexical rules.*

1. Introduction

In the paradigm of HPSG, lexical rules have become one of the key mechanisms used in current linguistic analysis. Computationally, lexical rules have mainly been dealt with in two ways: On the one hand, lexical rules are used to expand out the full lexicon at compile-time. On the other hand, lexical rules are encoded as unary phrase structure rules. Both of these computational treatments of lexical rules, however, have significant shortcomings with respect to lexical rules as used in HPSG.

A computational treatment expanding out the lexicon cannot be used for the increasing number of HPSG analyses that propose lexical rules that would result in an infinite lexicon. Most current HPSG analyses of Dutch, German, Italian, and French fall into that category.[1] Furthermore, since lexical rules in such an approach only serve in a precompilation step, the generalizations captured by the lexical rules cannot be used at run-time. Finally, all such treatments of lexical rules currently available presuppose a fully explicit notation of lexical rule specifications that transfer properties not changed by the lexical rules to the newly created lexical entry. This conflicts with the standard assumption made in HPSG that only the properties changed by a lexical rule need be mentioned. As shown in Meurers (1994) this is a well-motivated convention since it avoids splitting up lexical rules to transfer the specifications that must be preserved for different lexical entries.

* The authors are listed alphabetically. SFB 340, Kleine Wilhelmstr. 113, D-72074 Tübingen, Germany. email: {dm,minnen}@sfs.nphil.uni-tuebingen.de URL: http://www.sfs.nphil.uni-tuebingen.de/sfb/b4home.html

1 This is, for example, the case for all proposals working with verbal lexical entries that raise the arguments of a verbal complement (Hinrichs and Nakazawa 1989) that also use lexical rules such as the Complement Extraction Lexical Rule (Pollard and Sag 1994) or the Complement Cliticization Lexical Rule (Miller and Sag 1993) to operate on those raised elements. Also an analysis treating adjunct extraction via lexical rules (van Noord and Bouma 1994) results in an infinite lexicon.

Treatments of lexical rules as unary phrase structure rules also require their fully explicit specification, which entails the last problem mentioned above. In addition, computationally treating lexical rules on a par with phrase structure rules fails to take computational advantage of their specific properties. For example, the interaction of lexical rules is explored at run-time, even though the possible interaction can be determined at compile-time given the information available in the lexical rules and the base lexical entries.[2]

Based on the research results reported in Meurers and Minnen (1995, 1996), we propose a new computational treatment of lexical rules that overcomes these shortcomings and results in a more efficient processing of lexical rules as used in HPSG. We developed a compiler that takes as its input a set of lexical rules, deduces the necessary transfer of properties not changed by the individual lexical rules, and encodes the set of lexical rules and their interaction into definite relations constraining lexical entries. Each lexical entry is automatically extended with a definite clause encoding of the lexical rule applications which the entry can undergo. The definite clauses thereby introduce what we refer to as *systematic covariation in lexical entries.*

Definite relations are a convenient way of encoding the interaction of lexical rules, as they readily support various program transformations to improve the encoding: We show that the definite relations produced by the compiler can be refined by program transformation techniques to increase efficiency. The resulting encoding allows the execution of lexical rules on-the-fly, i.e., coroutined with other constraints at some time after lexical lookup. The computational treatment of lexical rules proposed can be seen as an extension to the principled method discussed by Götz and Meurers (1995, 1996, 1997b) for encoding the main building block of HPSG grammars—the implicative constraints—as a logic program.

The structure of the paper is as follows: We start with a brief introduction of the formal background on which our approach is based in Section 2. We then describe (Section 3) how lexical rules and their interaction can be encoded in a definite clause encoding that expresses systematic covariation in lexical entries. We show how the encoding of lexical rule interaction can be improved by specializing it for different word classes and, in Section 4, focus on an improvement of this specialization step by means of program transformation techniques. A further improvement relevant to on-the-fly application of lexical rules is presented in Section 5. In Section 6, we discuss implementation results and illustrate the efficiency of the proposed encoding. A comparison with other computational approaches to lexical rules (Section 7) and some concluding remarks (Section 8) end the paper.

2. Background

In this section we introduce the formal setup of HPSG grammars that we assume and discuss two ways to formalize a lexical rule mechanism and their consequences for a computational treatment.

2.1 A Formal Setup for HPSG Grammars

An HPSG grammar formally consists of two parts (Pollard and Sag 1994): The *signature* defines the ontology of linguistic objects, and the *theory*, i.e., the usually implicative constraints encoding the grammatical principles, describes the subset of those linguistic

2 This is not to say that a special precompilation treatment along those lines would not be profitable for phrase structure rules. In fact, such a proposal is made by Torisawa and Tsuji (1996).

objects that are grammatical. The constraints constituting the theory are expressions of a formal language that define the set of grammatical objects, in the sense that every grammatical object is described by every principle in the theory.

The signature consists of the type hierarchy defining which types of objects exist and the appropriateness conditions specifying which objects have which features defined on them to represent their properties.[3] A signature is interpreted as follows: Every object is assigned exactly one most specific type, and in case a feature is appropriate for some object of a certain type, then it is appropriate for all objects of this type.[4]

A logic that provides the formal architecture required by Pollard and Sag (1994) was defined by King (1989, 1994). The formal language of King allows the expression of grammatical principles using type assignments to refer to the type of an object and path equalities to require the (token) identity of objects. These atomic expressions can be combined using conjunction, disjunction, and negation. The expressions are interpreted by a set-theoretical semantics.

2.2 Lexical Rules in HPSG

While the setup of King provides a clear formal basis for basic HPSG grammars, nothing is said about how special linguistic mechanisms like lexical rules fit into this formal setup. Two formalizations of lexical rules as used by HPSG linguists have been proposed, the meta-level lexical rules (MLRs; Calcagno 1995; Calcagno and Pollard 1995) and the description-level lexical rules (DLRs; Meurers 1995).[5]

2.2.1 Meta-Level Lexical Rules. The MLR approach sees lexical rules in the more traditional sense as relations between lexical entries, i.e., descriptions of word objects. The set of lexical entries constituting the lexicon is closed under the application of lexical rules, which results in a (possibly infinite) set of lexical entries. In order to be grammatical, every word object occurring in a sentence has to be described by one of the descriptions in this expanded lexicon set. In the MLR setup, lexical rules are thus external to the rest of the theory, they only serve to provide an expanded lexicon set. Licensing grammatical words is then done by this set—the lexical rules play no direct role. Externalizing the lexicon and lexical rule application from the theory in such a way has an interesting consequence, namely that the lexical entries serving as input to a lexical rule are not tested for grammaticality.

A computational treatment of lexical rules that expands out the lexicon at compile-time closely resembles the MLR interpretation of lexical rules. The work on MLRs can therefore be seen as providing a semantics for such a computational treatment. It also allows a clear view of its restrictions: First, no restrictions on lexical entries serving as input to a lexical rule can be enforced that cannot be executed on the basis of the information present in the lexical entry alone,[6] and second, grammars including lexical rules that, under the MLR formalization, result in an infinite lexicon, can only

3 The terminology used in the literature varies. Types are also referred to as sorts, appropriateness conditions as feature declarations, and features as attributes. To avoid confusion, we will only use the terminology introduced in the text.

4 This interpretation of the signature is sometimes referred to as closed world (Gerdemann and King 1994; Gerdemann 1995).

5 An in-depth discussion including a comparison of both approaches is provided in Calcagno, Meurers, and Pollard (in preparation).

6 The Partial-VP Topicalization Lexical Rule proposed by Hinrichs and Nakazawa (1994, 10) is a linguistic example. The in-specification of this lexical rule makes use of an append relation to constrain the valence attribute of the auxiliaries serving as its input. In the lexicon, however, the complements of an auxiliary are uninstantiated because it raises the arguments of its verbal complement.

$$\begin{array}{lll} \textit{simple-word} & \rightarrow & LE_1 \vee \cdots \vee LE_n \\ \textit{derived-word} & \rightarrow & ([\text{IN } LR_1\text{-}in] \wedge LR_1\text{-}out) \vee \cdots \vee ([\text{IN } LR_m\text{-}in] \wedge LR_m\text{-}out) \end{array}$$

Figure 1
The extended lexicon under the DLR approach.

partially be dealt with, for example, by using a depth bound on lexical rule application to ensure that a finite number of lexical entries is obtained.[7]

2.2.2 Description-Level Lexical Rules. The DLR approach formalizes lexical rules as relations between word objects. Lexical rules under this approach are part of the theory, just like any other constraint of the grammar, and they relate the word objects licensed by the base lexical entries to another set of well-formed word objects. Thus, under the DLR approach, no new lexical entries are created, but the theory itself is extended in order to include lexical rules. One possibility for extending the theory is to introduce two subtypes of *word*, i.e., *simple-word* and *derived-word*, and define an additional feature IN with appropriate value *word* for objects of type *derived-word*. The principles encoding the extended lexicon in such an approach are shown in Figure 1. Each basic lexical entry is a disjunct *LE* in an implicative constraint on *simple-word*. This disjunction thus constitutes the base lexicon. The disjuncts in the constraint on *derived-word*, on the other hand, encode the lexical rules. The in-specification of a lexical rule specifies the IN feature, the out-specification, the derived word itself. Note that the value of the IN feature is of type *word* and thus also has to satisfy either a base lexical entry or an out-specification of a lexical rule. While this introduces the recursion necessary to permit successive lexical rule application, it also grounds the recursion in a word described by a base lexical entry. Contrary to the MLR setup, the DLR formalization therefore requires all words feeding lexical rules to be grammatical with respect to the theory.

Since lexical rules are expressed in the theory just like any other part of the theory, they are represented in the same way, as unary immediate dominance schemata.[8] This conception of lexical rules thus can be understood as underlying the computational approach that treats lexical rules as unary phrase structure rules as, for example, adopted in the LKB system (Copestake 1992). Both the input and output of a lexical rule, i.e., the mother and the daughter of a phrase structure rule, are available during a generation or parsing process. As a result, in addition to the information present in the lexical entry, syntactic information can be accessed to execute the constraints on the input of a lexical rule. The computational treatment of lexical rules that we propose in this paper is essentially a domain-specific refinement of such an approach to lexical rules.[9]

2.2.3 Lexical Rule Specification and Framing. An important difference between unary immediate dominance schemata and lexical rules, however, is that immediate dominance schemata are fully specified in the linguistic theory and can thus be directly interpreted as a relation on objects. Lexical rules, on the other hand, are usually not

7 This approach is, for example, taken in the ALE system. See Section 7 for more discussion of different computational approaches.
8 Elaborating this analogy, the IN feature of derived words can be understood as the DTRS feature of a phrase.
9 See Section 7 for a more detailed discussion of the relation between our approach and this perspective on lexical rules.

$$\left[\begin{array}{ll}\text{S}\,|\,\text{LOC}\,|\,\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & {}_{verb}\left[\text{VFORM}\ \ psp\right] \\ \text{VAL} & \left[\begin{array}{ll}\text{SUBJ} & \left\langle\left[\text{LOC}\,|\,\text{CONT}\,|\,\text{INDEX}\ \boxed{1}\right]\right\rangle \\ \text{COMPS} & \left\langle\left[\text{LOC}\,|\,\text{CONT}\,|\,\text{INDEX}\ \boxed{2}\right] \,|\, \boxed{3}\right\rangle\end{array}\right]\end{array}\right]\end{array}\right]$$

$$\mapsto$$

$$\left[\begin{array}{ll}\text{S}\,|\,\text{LOC}\,|\,\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & \left[\text{VFORM}\ \ pas\right] \\ \text{VAL} & \left[\begin{array}{ll}\text{SUBJ} & \left\langle\left[\text{LOC}\,|\,\text{CONT}\,|\,\text{INDEX}\ \boxed{2}\right]\right\rangle \\ \text{COMPS} & \boxed{3} \oplus \left\langle\left(\left[\text{LOC}\ \left[\begin{array}{ll}\text{CAT}\,|\,\text{HEAD} & {}_{prep}\left[\text{PFORM}\ \ by\right] \\ \text{CONT}\,|\,\text{INDEX} & \boxed{1}\end{array}\right]\right]\right)\right\rangle\end{array}\right]\end{array}\right]\end{array}\right]$$

Figure 2
A passivization lexical rule.

written as fully specified relations between words, rather, only what is supposed to be changed is specified.

Consider, for example, the lexical rule in Figure 2, which encodes a passive lexical rule like the one presented by Pollard and Sag (1987, 215) in terms of the setup of Pollard and Sag (1994, ch. 9). This lexical rule could be used in a grammar of English to relate past participle forms of verbs to their passive form.[10] The rule takes the index of the least oblique complement of the input and assigns it to the subject of the output. The index that the subject bore in the input is assigned to an optional prepositional complement in the output.

Only the verb form and some indices are specified to be changed, and thus other input properties, like the phonology, the semantics, or the nonlocal specifications, are preserved in the output. This is so since the lexical rule in Figure 2 "(like all lexical rules in HPSG) preserves all properties of the input not mentioned in the rule." (Pollard and Sag [1994, 314], following Flickinger [1987]). This idea of preserving properties can be considered an instance of the well-known **frame problem** in AI (McCarthy and Hayes 1969), and we will therefore refer to the specifications left implicit by the linguist as the *frame specification*, or simply *frame*, of a lexical rule. Not having to represent the frame explicitly not only enables the linguist to express only the relevant things, but also allows a more compact representation of lexical rules where explicit framing would require the rules to be split up (Meurers 1994).

One thus needs to distinguish the lexical rule specification provided by the linguist from the fully explicit lexical rule relations integrated into the theory. The formalization of DLRs provided by Meurers (1995) defines a formal lexical rule specification language and provides a semantics for that language in two steps: A rewrite system enriches the lexical rule specification into a fully explicit description of the kind shown in Figure 1. This description can then be given the standard set-theoretical interpretation of King (1989, 1994).[11]

10 Note that the passivization lexical rule in Figure 2 is only intended to illustrate the mechanism. We do not make the linguistic claim that passives should be analyzed using such a lexical rule. For space reasons, the SYNSEM feature is abbreviated by its first letter. The traditional ⟨*First* | *Rest*⟩ list notation is used, and the operator ⊕ stands for the append relation in the usual way.

11 Manandhar (1995) proposes to unify these two steps by including an update operator in the

The computational treatment we discuss in the rest of the paper follows this setup in that it automatically computes, for each lexical rule specification, the frames necessary to preserve the properties not changed by it.[12] We will show that the detection and specification of frames and the use of program transformation to advance their integration into the lexicon encoding is one of the key ingredients of the covariation approach to HPSG lexical rules.

3. Lexical Covariation: Encoding Lexical Rules and their Interaction as Definite Relations

Having situated the computational approach presented in this paper as a computational treatment of DLRs that emphasizes their domain-specific properties, we now turn to the compiler that realizes this approach. We describe four compilation steps that translate a set of lexical rules, as specified by the linguist, and their interaction into definite relations to constrain lexical entries. To give the reader a global idea of our approach, we focus on those aspects of the compiler that are crucial to the presented conception of lexical rules. The different steps of the compiler are discussed with emphasis on understandability and not on formal details.[13]

Figure 3 shows the overall setup of the compiler. The first compilation step, discussed in Section 3.1, translates lexical rules into a definite clause representation and derives, for each lexical rule, a frame predicate that ensures the transfer of properties that remain unchanged. In the second compilation step (Section 3.2), we determine the possible interaction of the lexical rules. This results in a finite-state automaton representing global lexical rule interaction, i.e., the interaction of lexical rules irrespective of the lexical entries in the lexicon. In the subsequent step of word class specialization (Section 3.3) this finite-state automaton is fine-tuned for each of the natural classes of lexical entries in the lexicon. In the fourth compilation step (Section 3.4) these automata are translated into definite relations and the lexical entries are adapted to call the definite relation corresponding to the automaton fine-tuned for the natural class to which they belong.

3.1 Lexical Rules as Definite Relations and the Automatic Specification of Frames

We start by translating each lexical rule into a definite clause predicate, called the *lexical rule predicate*. The first argument of a lexical rule predicate corresponds to the in-specification of the lexical rule and the second argument to its out-specification.

Assume the signature in Figure 4 on which we base the example throughout the paper and suppose the lexical rule specification shown in Figure 5.[14] This lexical rule applies to base lexical entries that unify[15] with the in-specification, i.e., lexical entries specifying B and Y as $-$. The derived lexical entry licenses word objects with $+$ as the value of X and Y, and *b* as that of A.

The translation of the lexical rule into a predicate is trivial. The result is displayed

description language.

12 In order to focus on the computational aspects of the covariation approach, in this paper we will not go into a discussion of the full lexical rule specification language introduced in Meurers (1995). The reader interested in that language and its precise interpretation can find the relevant details in that paper.

13 A more detailed presentation can be found in Minnen (in preparation).

14 We use rather abstract lexical rules in the examples to be able to focus on the relevant aspects.

15 Hinrichs and Nakazawa (1996) show that the question of whether the application criterion of lexical rules should be a subsumption or a unification test is an important question deserving of more attention. We here assume unification as the application criterion, which formally corresponds to the conjunction of descriptions and their conversion to normal form (Götz 1994). Computationally, a subsumption test could equally well be used in our compiler.

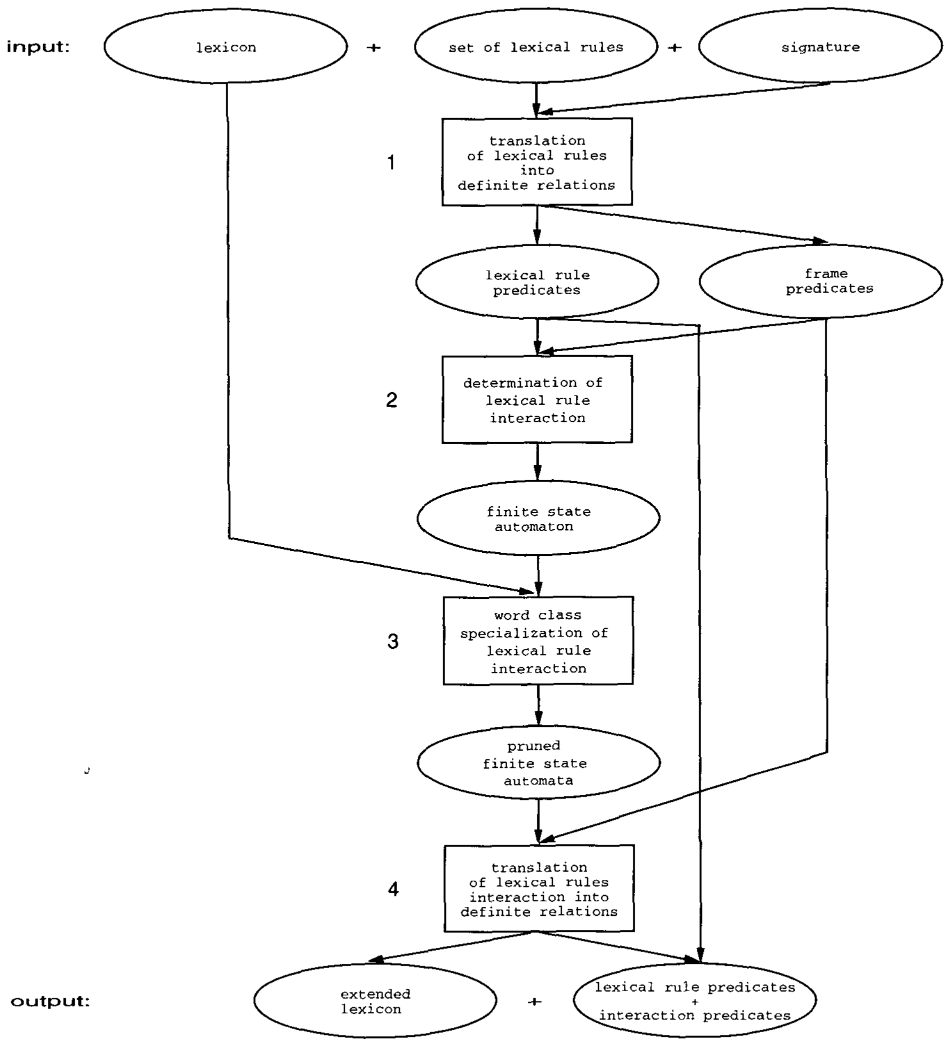
input:
lexicon
+
set of lexical rules
+
signature
1
translation of lexical rules into definite relations
lexical rule predicates
frame predicates
2
determination of lexical rule interaction
finite state automaton
3
word class specialization of lexical rule interaction
pruned finite state automata
4
translation of lexical rules interaction into definite relations
output:
extended lexicon
+
lexical rule predicates + interaction predicates

Figure 3
The compiler setup.

in Figure 6. Though this predicate represents what was explicitly specified in the lexical rule, it does not accomplish exactly what is intended. As discussed in Section 2.2.3, features specified in a lexical entry unifying with the in-specification of the lexical rule that are not specified differently in the out-specification of the lexical rule are intended to receive the same value on the derived word as on the input: The compiler implements this by enriching the lexical rule with type specifications and path equalities between the in- and the out-specification to arrive at an explicit representation of its frame.

The detection of which additional specifications are intended by the linguist crucially depends on the interpretation of the signature assumed in HPSG, discussed in Section 2.1. This interpretation makes it possible to determine which kind of word objects (by ontological status fully specified) may undergo the rule. A type can always be replaced by a disjunction of its most specific subtypes and the appropriate features

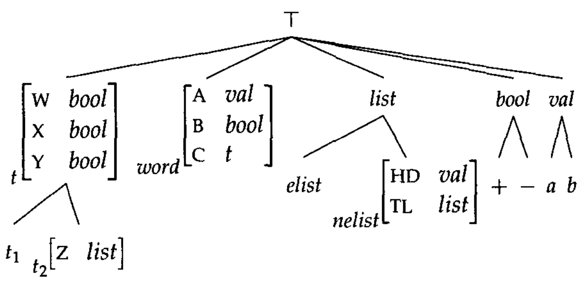

Figure 4
An example signature.

$$\begin{bmatrix}\text{B} & - \\ \text{C} & \begin{bmatrix}\text{Y} & -\end{bmatrix}\end{bmatrix} \mapsto \begin{bmatrix}\text{A} & b \\ \text{C} & \begin{bmatrix}\text{X} & + \\ \text{Y} & +\end{bmatrix}\end{bmatrix}$$

Figure 5
Lexical rule 1.

$$\text{lex\_rule\_1}(\begin{bmatrix}\text{B} & - \\ \text{C} & \begin{bmatrix}\text{Y} & -\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\text{A} & b \\ \text{C} & \begin{bmatrix}\text{X} & + \\ \text{Y} & +\end{bmatrix}\end{bmatrix})$$

Figure 6
Definite clause representation of lexical rule 1.

$${}_{\mathit{word}}\begin{bmatrix}\text{C} & t_1\end{bmatrix}$$

Figure 7
A sample lexical entry.

of each type are known. So, on the basis of the signature, we can determine which "appropriate" paths the linguist left unspecified in the out-specification of the lexical rule. For those appropriate paths not specified in the out-specification, one can then add path equalities between the in- and the out-specifications of the lexical rule to ensure framing of those path values.

Frame specification becomes slightly more difficult when one considers type specifications of those paths in words serving as input to a lexical rule that occur in the out-specification of the lexical rule but are not assigned a type value. For example, the lexical rule 1 of Figure 6 applies to word objects with t_1 as their C value and to those having t_2 as their C value. With respect to frame specification this means that there can be lexical entries, such as the one in Figure 7, for which we need to make sure that t_1 as the value of C gets transferred.[16]

One would think that the type information t_1, which is more specific than that

16 A linguistic example based on the signature given by Pollard and Sag (1994) would be a lexical rule deriving predicative signs from nonpredicative ones, i.e., changing the PRD value of substantive signs from − to +, much like the lexical rule for NPs given by Pollard and Sag (1994, p. 360, fn. 20). In such a Predicative Lexical Rule (which we only note as an example and not as a linguistic proposal) the subtype of the head object undergoing the rule as well as the value of the features only appropriate for the subtypes of *substantive* either is lost or must be specified by a separate rule for each of the subtypes.

$$\text{lex\_rule\_1}(\boxed{1}\left[\begin{array}{ll} \text{B} & - \\ \text{C} & \left[\begin{array}{ll} \text{Y} & - \end{array}\right] \end{array}\right], \boxed{2}\left[\begin{array}{ll} \text{A} & b \\ \text{C} & \left[\begin{array}{ll} \text{X} & + \\ \text{Y} & + \end{array}\right] \end{array}\right]) \text{:- frame\_1}(\boxed{1},\boxed{2}).$$

Figure 8
Lexical rule predicate representing lexical rule 1.

$$\text{frame\_1}(\left[\begin{array}{ll} \text{B} & \boxed{1} \\ \text{C} & \underset{t_1}{\left[\begin{array}{ll} \text{W} & \boxed{2} \end{array}\right]} \end{array}\right], \left[\begin{array}{ll} \text{B} & \boxed{1} \\ \text{C} & \underset{t_1}{\left[\begin{array}{ll} \text{W} & \boxed{2} \end{array}\right]} \end{array}\right]). \qquad \text{frame\_1}(\left[\begin{array}{ll} \text{B} & \boxed{1} \\ \text{C} & \underset{t_2}{\left[\begin{array}{ll} \text{W} & \boxed{2} \\ \text{Z} & \boxed{3} \end{array}\right]} \end{array}\right], \left[\begin{array}{ll} \text{B} & \boxed{1} \\ \text{C} & \underset{t_2}{\left[\begin{array}{ll} \text{W} & \boxed{2} \\ \text{Z} & \boxed{3} \end{array}\right]} \end{array}\right]).$$

Figure 9
Definition of the frame predicate for lexical rule 1.

given in the output of the lexical rule, can be specified on the out-specification of the lexical rule if the specification of C is transferred as a whole (via structure sharing of the value of C). This is not possible, though, since the values of X and Y are specified in the out-specification of the lexical rule. The problem seems to be that there is no notion of sharing just the type of an object. However, introducing such type sharing would not actually solve the problem, since one also needs to account for additional appropriate features. The subtypes of t have different appropriate features, the values of which have to be preserved. In particular, in case the lexical entry has t_2 as the value of C, we need to ensure that the value of the feature Z is transferred properly.

To ensure that no information is lost as a result of applying a lexical rule, it seems to be necessary to split up the lexical rule to make each instance deal with a specific case. In the above example, this would result in two lexical rules: one for words with t_1 as their C value and one for those with t_2 as their C value. In the latter case, we can also take care of transferring the value of Z. However, as discussed by Meurers (1994), creating several instances of lexical rules can be avoided. Instead, the disjunctive possibilities introduced by the frame specification are attached as a constraint to a lexical rule. This is accomplished by having each lexical rule predicate call a so-called *frame predicate*, which can have multiple defining clauses. So for the lexical rule 1, the frame specification is taken care of by extending the predicate in Figure 6 with a call to a frame predicate, as shown in Figure 8.[17]

On the basis of the lexical rule specification and the signature, the compiler deduces the frame predicates without requiring additional specifications by the linguist. The frame predicate for lexical rule 1 is defined by the two clauses displayed in Figure 9. The first case applies to lexical entries in which C is specified as t_1. We have to ensure that the value of the feature W is transferred. In the second case, when feature C has t_2 as its value, we additionally have to ensure that Z gets transferred. Note that neither clause of the frame predicate needs to specify the features A, X, and Y since these features are changed by *lex_rule_1*. Furthermore, filling in features of the structure below Z is unnecessary as the value of Z is structure shared as a whole. Finally, if a lexical entry specifies C as t, both *frame_1* clauses apply.[18]

17 We use indexing of predicate names to be able to indicate later on which lexical rule a frame predicate belongs to.

18 Since in computational systems, in contrast to the general theoretical case, we only need to ensure transfer for the properties actually specified in the lexical entries of a given grammar, some of the distinctions made in the signature can possibly be ignored. One could therefore improve the calculation of frame predicates by taking the base lexical entries into account at this stage of the

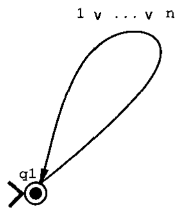

Figure 10
Finite-state automaton representing free application.

Summing up, we distinguish the lexical rule predicates encoding the specification of the linguist from the frame predicates taking care of the frame specification. Based on the signature, the frame predicates are automatically derived from the lexical rule predicates and they can have a possibly large number of defining clauses. In Section 4 we will show that the encoding can be advanced in a way that eliminates the nondeterminism introduced by the multiply defined frame predicates.

3.2 Determining Global Lexical Rule Interaction

In the second compilation step, we use the definite clause representation of a set of lexical rules, i.e., the lexical rule and the frame predicates, to compute a finite-state automaton representing how the lexical rules interact (irrespective of the lexical entries).

In general, any lexical rule can apply to the output of another lexical rule, which is sometimes referred to as *free application*. As shown in Figure 10, this can be represented as a finite-state automaton that consists of a single state with a cycle from/into this state for all lexical rules.[19] When looking at a specific set of lexical rules though, one can be more specific as to which sequences of lexical rule applications are possible. One can represent this information about the interaction of lexical rules as a more complex finite-state automaton, which can be used to avoid trying lexical rule applications at run-time that are bound to fail. To derive a finite-state automaton representing global lexical rule interaction, we first determine which lexical rules can possibly follow which other lexical rules in a grammar. The set of follow relationships is obtained by testing which in-specifications unify with which out-specifications.[20]

To illustrate the steps in determining global lexical rule interaction, let us add three more lexical rules to the one discussed in Section 3.1. Figure 11 shows the full set of four lexical rules.

Figure 12 shows the definite clause representations of lexical rules 2, 3, and 4 and the frame predicates derived for them. The definite clauses representing lexical rule 1 and its frame were already given in Figures 8 and 9. The follow relation obtained for the set of four lexical rules is shown in Figure 13, where *follow(LR,ListOfLRs)* specifies

compilation process.

19 We use the following conventions with respect to finite-state automata to represent lexical rule interaction: The state annotated with an angle bracket represents the initial state. All states (including the initial state) are final states. The labels of the transitions from one state to another are (disjunctions of) the lexical rule predicate indices, i.e., the lexical rule names constitute the alphabet of the finite-state automaton.

20 For the computation of the follow relationships, the specifications of the frame predicates are taken into account. In case the frame relation called by a lexical rule has several defining clauses, the generalization of the frame possibilities is used.

$$\text{Rule 1: } \begin{bmatrix}\text{B} & - \\ \text{C} & \begin{bmatrix}\text{Y} & -\end{bmatrix}\end{bmatrix} \mapsto \begin{bmatrix}\text{A} & b \\ \text{C} & \begin{bmatrix}\text{X} & + \\ \text{Y} & +\end{bmatrix}\end{bmatrix} \qquad \text{Rule 2: } \begin{bmatrix}\text{A} & b \\ \text{B} & - \\ \text{C} & \begin{bmatrix}\text{W} & -\end{bmatrix}\end{bmatrix} \mapsto \begin{bmatrix}\text{C} & \begin{bmatrix}\text{W} & +\end{bmatrix}\end{bmatrix}$$

$$\text{Rule 3: } \begin{bmatrix}\text{C} & {}_{t_2}\begin{bmatrix}\text{W} & + \\ \text{X} & + \\ \text{Z}\,|\,\text{TL} & \boxed{1}\end{bmatrix}\end{bmatrix} \mapsto \begin{bmatrix}\text{C} & \begin{bmatrix}\text{Y} & + \\ \text{Z} & \boxed{1}\end{bmatrix}\end{bmatrix} \qquad \text{Rule 4: } \begin{bmatrix}\text{B} & - \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & + \\ \text{X} & + \\ \text{Z} & \langle\rangle\end{bmatrix}\end{bmatrix} \mapsto \begin{bmatrix}\text{B} & + \\ \text{C} & \begin{bmatrix}\text{X} & -\end{bmatrix}\end{bmatrix}$$

Figure 11
A set of four lexical rules.

$$\text{lex\_rule\_2}(\boxed{1}\begin{bmatrix}\text{A} & b \\ \text{B} & - \\ \text{C} & \begin{bmatrix}\text{W} & -\end{bmatrix}\end{bmatrix}, \boxed{2}\begin{bmatrix}\text{C} & \begin{bmatrix}\text{W} & +\end{bmatrix}\end{bmatrix}) \text{:- frame\_2}(\boxed{1},\boxed{2}).$$

$$\text{lex\_rule\_3}(\boxed{1}\begin{bmatrix}\text{C} & {}_{t_2}\begin{bmatrix}\text{W} & + \\ \text{X} & + \\ \text{Z}\,|\,\text{TL} & \boxed{3}\end{bmatrix}\end{bmatrix}, \boxed{2}\begin{bmatrix}\text{C} & \begin{bmatrix}\text{Y} & + \\ \text{Z} & \boxed{3}\end{bmatrix}\end{bmatrix}) \text{:- frame\_3}(\boxed{1},\boxed{2}).$$

$$\text{lex\_rule\_4}(\boxed{1}\begin{bmatrix}\text{B} & - \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & + \\ \text{X} & + \\ \text{Z} & \langle\rangle\end{bmatrix}\end{bmatrix}, \boxed{2}\begin{bmatrix}\text{B} & + \\ \text{C} & \begin{bmatrix}\text{X} & -\end{bmatrix}\end{bmatrix}) \text{:- frame\_4}(\boxed{1},\boxed{2}).$$

$$\text{frame\_2}(\begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_1}\begin{bmatrix}\text{X} & \boxed{3} \\ \text{Y} & \boxed{4}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_1}\begin{bmatrix}\text{X} & \boxed{3} \\ \text{Y} & \boxed{4}\end{bmatrix}\end{bmatrix}). \qquad \text{frame\_2}(\begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5}\end{bmatrix}\end{bmatrix}).$$

$$\text{frame\_3}(\begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & \boxed{3} \\ \text{X} & \boxed{4}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & \boxed{3} \\ \text{X} & \boxed{4}\end{bmatrix}\end{bmatrix}). \qquad \text{frame\_4}(\begin{bmatrix}\text{A} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & \boxed{2} \\ \text{Y} & \boxed{3} \\ \text{Z} & \boxed{4}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\text{A} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix}\text{W} & \boxed{2} \\ \text{Y} & \boxed{3} \\ \text{Z} & \boxed{4}\end{bmatrix}\end{bmatrix}).$$

Figure 12
The definite clause encoding of lexical rules 2, 3, and 4.

```
follow(1, [2, 3, 4]).    follow(2, [1, 3, 4]).    follow(3, [3, 4]).    follow(4, []).
```

Figure 13
The follow relation for the four lexical rules of the example.

that only the lexical rules in *ListOfLRs* can possibly be applied to a word resulting from the application of lexical rule *LR*.

Once the follow relation has been obtained, it can be used to construct an automaton that represents which lexical rule can be applied after which *sequence* of lexical rules. Special care has to be taken in case the same lexical rule can apply several times in a sequence. To obtain a *finite* automaton, such a repetition is encoded as a transition cycling back to a state in the lexical rule sequence preceding it.

In order to be able (in the following steps) to remove a transition representing a certain lexical rule application in one sequence without eliminating the lexical rule application from other sequences, every transition, except those introducing cycles, is taken to lead to a new state. The finite-state automaton in Figure 14 is constructed on the basis of the follow relation of Figure 13.

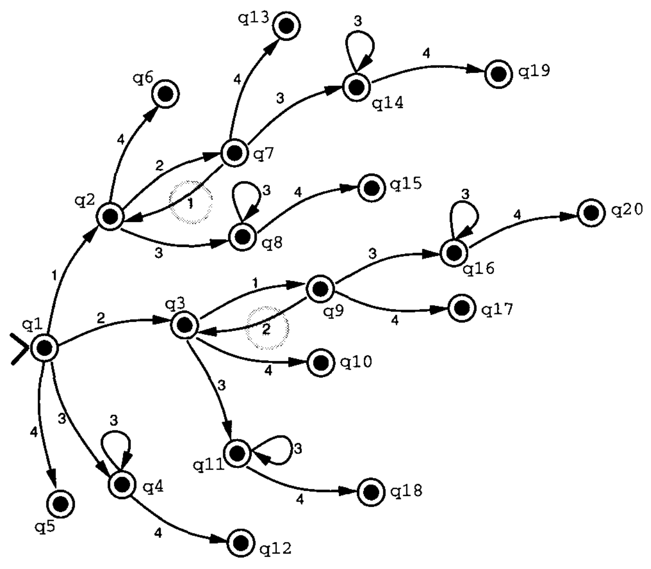

Figure 14
Finite-state automaton representing global lexical rule interaction.

The finite-state automaton representing global lexical rule interaction can be used as the backbone of a definite clause encoding of lexical rules and their interaction (see Section 3.4). Compared to free application, the finite-state automaton in Figure 14 limits the choice of lexical rules that can apply at a certain point. However, there still are several places where the choices can be further reduced. One possible reduction of the above automaton consists of taking into account the *propagation of specifications* along each possible path through the automaton. This corresponds to actually unifying the out-specification of a lexical rule with the in-specification of the following lexical rule along each path in the automaton, instead of merely testing for unifiability, which we did to obtain the follow relation.[21] As a result of unifying the out-specification of a lexical rule in a path of the finite-state automaton with the in-specification of the following lexical rule, the out-specification of the second rule can become more specific. This is because of the structure sharing between the second lexical rule's in- and out-specifications, which stem from the lexical rule and its frame specification. This makes it possible to eliminate some of the transitions that seem to be possible when judging on the basis of the follow relation alone.[22]

For example, solely on the basis of the follow relation, we are not able to discover the fact that upon the successive application of lexical rules 1 and 2, neither lexical rule 1 nor 2 can be applied again. Taking into account the propagation of specifications, the result of the successive application of lexical rule 1 and lexical rule 2 in any order (leading to state $q7$ or $q9$) bears the value + on features W and Y. This excludes lexical

21 The reason for first determining the automaton on the basis of the follow relation alone, instead of taking propagation of specifications into account right from the start, is that the follow relations allow a very simple construction of a finite-state automaton representing lexical rule interaction. Using unification right away would significantly complicate the algorithm, in particular for automata containing cycles.

22 Note that in the case of transitions belonging to a cycle, only those transitions can be removed that are useless at the first visit and after any traversal of the cycle.

$$\text{lex\_entry}\left(\begin{bmatrix} \text{A} & b \\ \text{B} & - \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & - \\ \text{X} & - \\ \text{Y} & - \\ \text{Z} & \langle a,b \rangle \end{bmatrix} \end{bmatrix}\right).$$

Figure 15
A lexical entry.

rules 1 and 2 as possible followers of that sequence since their in-specifications do not unify with those values. As a result, the arcs $1(q7, q2)$ and $2(q9, q3)$, which are marked with grey dots in Figure 14, can be removed.

Two problems remain: First, because of the procedural interpretation of lexical rules, duplicate lexical entries can possibly be derived. And second, relative to a specific lexical entry, many sequences of lexical rules that are bound to fail are tried anyway. We tackle these problems by means of word class specialization, i.e., we prune the automaton with respect to the propagation of specifications belonging to the base lexical entries.

3.3 Word Class Specialization of Lexical Rule Interaction

In the third compilation step, the finite-state automaton representing global lexical rule interaction is fine-tuned for each base lexical entry in the lexicon. The result is a *pruned* finite-state automaton. The pruning is done by performing the lexical rule applications corresponding to the transitions in the automaton representing global lexical rule interaction. To ensure termination in case of direct or indirect cycles, we use a subsumption check. If the application of a particular lexical rule with respect to a lexical entry fails, we know that the corresponding transition can be pruned for that entry. In case of indirect or direct cycles in the automaton, however, we cannot derive all possible lexical entries, as there may be infinitely many. Even though one can prune certain transitions even in such cyclic cases, it is possible that certain inapplicable transitions remain in the pruned automaton. However, this is not problematic since the lexical rule application corresponding to such a transition will simply fail at run-time.

Consider the base lexical entry in Figure 15. With respect to this base lexical entry, we fine-tune the finite-state automaton representing global lexical rule interaction by pruning transitions. In the automaton of Figure 14, we can prune the transitions $\{3(q2, q8), 4(q2, q6), 3(q3, q11), 4(q3, q10), 3(q1, q4), 4(q1, q5)\}$, because the lexical rules 3 and 4 can not be applied to a (derived) lexical entry that does not have both W and X of value $+$. As a consequence, the states $q8$, $q15$, $q11$, $q18$, $q4$, and $q12$ are no longer reachable and the following transitions can be eliminated as well: $\{3(q8, q8), 4(q8, q15), 3(q11, q11), 4(q11, q18), 3(q4, q4), 4(q4, q12)\}$. We can also eliminate the transitions $\{4(q7, q13), 4(q9, q17)\}$, because the lexical rule 4 requires the value of Z to be empty list. Note that the lexical rules 3 and 4 remain applicable in $q14$ and $q16$.

Furthermore, due to the procedural interpretation of lexical rules in a computational system (in contrast to the original declarative intention), there can be sequences of lexical rule applications that produce identical entries.[23] To avoid having arcs in the pruned automaton leading to such identical entries, we use a tabulation method

23 Note that the order in which two lexical rules are applied is immaterial as long as both rules modify the value of different features of a lexical entry.

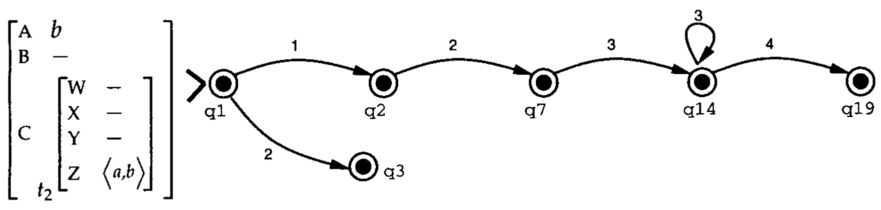

Figure 16
Pruned finite state automaton representing lexical rule interaction for a lexical entry.

during word class specialization that keeps track of the feature structures obtained for each node. If we find a feature structure for a node qn that is identical to the feature structure corresponding to another node qm, the arc leading to qn or the arc leading to qm is discarded.[24] In the example, $q7$ and $q9$ are such identical nodes. So we can discard either $2(q2, q7)$ or $1(q3, q9)$ and eliminate the arcs from states that then become unreachable. Choosing to discard $1(q3, q9)$, the pruned automaton for the example lexical entry looks as displayed in Figure 16.[25]

Note that word class specialization of lexical rule interaction does not influence the representation of the lexical rules themselves. Pruning the finite-state automaton representing global lexical rule interaction only involves restricting lexical rule interaction in relation to the lexical entries in the lexicon.

The fine-tuning of the automaton representing lexical rule interaction results in a finite-state automaton for each lexical entry in the lexicon. However, identical automata are obtained for certain groups of lexical entries and, as shown in the next section, each automaton is translated into definite relations only once. We therefore automatically group the lexical entries into the natural classes for which the linguist intended a certain sequence of lexical rule applications to be possible.[26] No additional hand-specification is required. Moreover, the alternative computational treatment to expand out the full lexicon at compile-time is just as costly and, furthermore, impossible in case of an infinite lexicon.

An interesting aspect of the idea of representing lexical rule interaction for particular word classes is that this allows a natural encoding of exceptions to lexical rules. More specifically, the linguist specifies exceptions as a special property of either a lexical rule or a lexical entry. During word class specialization, the compiler then deals with such specifications by pruning the corresponding transitions in the finite-state automaton representing global lexical rule interaction for the particular lexical entry under consideration. This results in an encoding of exceptions to a lexical rule in the interaction predicate called by the irregular lexical entries. An advantage of the setup presented is that entries that behave according to subregularities will automatically be grouped together again and call the same interaction predicate. The final representa-

24 In general, there is not always enough information available to determine whether two sequences of lexical rule applications produce identical entries. This is because in order to be able to treat recursive lexical rules producing infinite lexica, we perform word class specialization of the interaction predicate instead of expanding out the lexicon.

25 Note that an automaton can be made even more deterministic by unfurling instances of cycles prior to pruning. In our example, unfurling the direct cycle by replacing $3(q14, q14)$ with $\{3(q14, q14'), 3(q14', q14'), 4(q14', q19')\}$ would allow pruning of the cyclic transition $3(q14', q14')$ and the transition $4(q14, q19)$. Note, however, that unfurling of the first n instances of a cycle does not always allow pruning of transitions, i.e., reduce nondeterminism.

26 The pruned finite-state automaton constitutes valuable feedback, as it represents the interaction of the set of lexical rules possible for a word class in a succinct and perspicuous manner.

$$\text{lex\_entry}(\boxed{Out})\text{:- q\_1}(\left[\begin{array}{ll} \text{A} & b \\ \text{B} & - \\ \text{C} & {}_{t_2}\left[\begin{array}{ll} \text{W} & - \\ \text{X} & - \\ \text{Y} & - \\ \text{Z} & \langle a,b \rangle \end{array}\right] \end{array}\right], \boxed{Out}).$$

Figure 17
An extended lexical entry.

tion of the lexical rules and the lexical entries remains, without a special specification of exceptions.[27]

3.4 Lexical Rule Interaction as Definite Relations

In the fourth compilation step, the finite-state automata produced in the last step are encoded in definite clauses, called *interaction predicates*. The lexical entries belonging to a particular natural class all call the interaction predicate encoding the automaton representing lexical rule interaction for that class. Figure 17 shows the extended version of the lexical entry of Figure 15. The base lexical entry is fed into the first argument of the call to the interaction predicate *q_1*. For each solution to a call to *q_1* the value of $\boxed{Out}$ is a derived lexical entry.

Encoding a finite-state automaton as definite relations is rather straightforward. In fact, one can view the representations as notational variants of one another. Each transition in the automaton is translated into a definite relation in which the corresponding lexical rule predicate is called, and each final state is encoded by a unit clause. Using an accumulator passing technique (O'Keefe 1990), we ensure that upon execution of a call to the interaction predicate *q_1* a new lexical entry is derived as the result of successive application of a number of lexical rules. Because of the word class specialization step discussed in Section 3.3, the execution avoids trying out many lexical rule applications that are guaranteed to fail.

We illustrate the encoding with the finite-state automaton of Figure 16. As the lexical rules themselves are already translated into a definite clause representation in the first compilation step, the interaction predicates only need to ensure that the right combination of lexical rule predicates is called. The interaction predicate encoding the finite-state automaton of Figure 16 is shown in Figure 18.[28]

We now have a first complete encoding of the lexical rules and their interaction represented as covariation in lexical entries. The encoding consists of three types of definite clause predicates:

1. Lexical rule predicates representing the lexical rules;
2. Frame predicates specifying the frame for the lexical rule predicates; and
3. Interaction predicates encoding lexical rule interaction for the natural classes of lexical entries in the lexicon.

The way these predicates interconnect is represented in Figure 19.

27 Briscoe and Copestake (1996) argue that semi-productivity of lexical rules, which can be understood as a generalization of exceptions to lexical rules, can be integrated with our approach by assigning probabilities to the automaton associated with a particular lexical entry.

28 In order to distinguish the different interaction predicates for the different classes of lexical entries, the compiler indexes the names of the interaction predicates. Since for expository reasons we will only discuss one kind of lexical entry in this paper, we will not show those indices in the examples given.

q_1($\boxed{In}$,$\boxed{Out}$):- lex_rule_1($\boxed{In}$,$\boxed{Aux}$), q_2($\boxed{Aux}$,$\boxed{Out}$).

q_1($\boxed{In}$,$\boxed{Out}$):- lex_rule_2($\boxed{In}$,$\boxed{Aux}$), q_3($\boxed{Aux}$,$\boxed{Out}$).

q_2($\boxed{In}$,$\boxed{Out}$):- lex_rule_2($\boxed{In}$,$\boxed{Aux}$), q_7($\boxed{Aux}$,$\boxed{Out}$).

q_7($\boxed{In}$,$\boxed{Out}$):- lex_rule_3($\boxed{In}$,$\boxed{Aux}$), q_14($\boxed{Aux}$,$\boxed{Out}$).

q_14($\boxed{In}$,$\boxed{Out}$):-lex_rule_3($\boxed{In}$,$\boxed{Aux}$), q_14($\boxed{Aux}$,$\boxed{Out}$).

q_14($\boxed{In}$,$\boxed{Out}$):-lex_rule_4($\boxed{In}$,$\boxed{Aux}$), q_19($\boxed{Aux}$,$\boxed{Out}$).

q_1($\boxed{In}$,$\boxed{In}$). q_2($\boxed{In}$,$\boxed{In}$). q_3($\boxed{In}$,$\boxed{In}$). q_7($\boxed{In}$,$\boxed{In}$). q_14($\boxed{In}$,$\boxed{In}$). q_19($\boxed{In}$,$\boxed{In}$).

Figure 18
The definite relations representing the pruned finite state automaton of Figure 16.

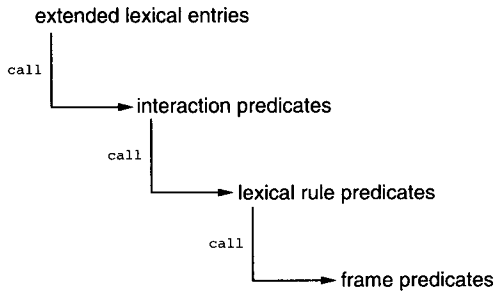

Figure 19
Schematic representation of definite clause encoding of lexical rules and their interaction.

4. Partial Unfolding of Frame Predicates

The automata resulting from word class specialization group the lexical entries into natural classes. In case the automata corresponding to two lexical entries are identical, the entries belong to the same natural class. However, each lexical rule application, i.e., each transition in an automaton, calls a frame predicate that can have a large number of defining clauses. Intuitively understood, each defining clause of a frame predicate corresponds to a subclass of the class of lexical entries to which a lexical rule can be applied. During word class specialization, though, when the finite-state automaton representing global lexical rule application is pruned with respect to a particular base lexical entry, we know which subclass we are dealing with. For each interaction definition we can therefore check which of the frame clauses are applicable and discard the non-applicable ones. We thereby eliminate the redundant nondeterminism resulting from multiply defined frame predicates.

The elimination of redundant nondeterminism is based on Unfold/Fold transformation techniques (Tamaki and Sato 1984).[29] The *unfolding* transformation is also referred to as partial execution, for example, by Pereira and Shieber (1987). Intuitively understood, unfolding comprises the evaluation of a particular literal in the body of a clause at compile-time. As a result, the literal can be removed from the body of

29 This improvement of the covariation encoding can also be viewed as an instance of the program transformation technique referred to as *deletion of clauses with a finitely failed body* (Pettorossi and Proietti 1994).

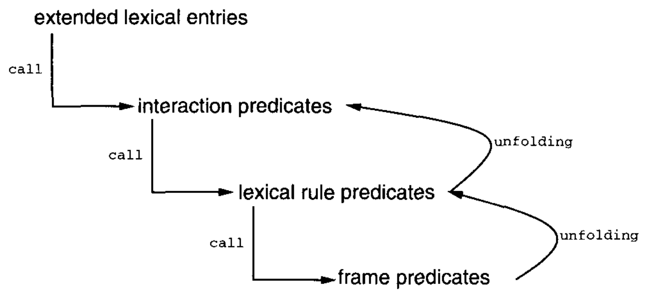

Figure 20
Schematic representation of the successive unfolding transformation.

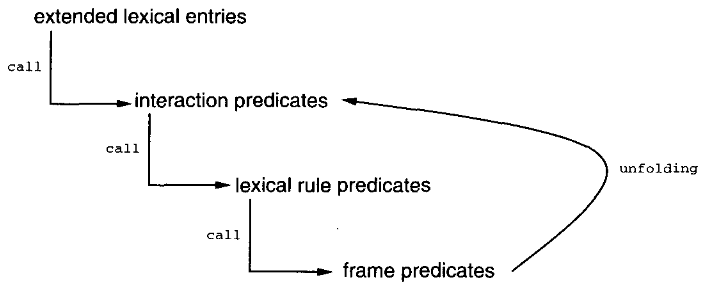

Figure 21
Schematic representation of the partial unfolding transformation.

the clause. Whereas unfolding can be viewed as a symbolic way of going forward in computation, *folding* constitutes a symbolic step backwards in computation.

Given a lexical entry as in Figure 15, we can discard all frame clauses that presuppose t_1 as the value of C, as discussed in the previous section. To eliminate the frame predicates completely, we can successively unfold the frame predicates and the lexical rule predicates with respect to the interaction predicates.[30] The successive unfolding steps are schematically represented in Figure 20.

Such a transformation, however, would result in the loss of a representation of the lexical rule predicates that is independent of a particular word class, but an independent representation of lexical rules constitutes an advantage in space in case lexical rules can be applied across word classes. Our compiler therefore performs what can be viewed as "partial" unfolding: it unfolds the frame predicates directly with respect to the interaction predicates, as shown in Figure 21.

One can also view this transformation as successive unfolding of the frame predicates and the lexical rule predicates with respect to the interaction predicates followed by a folding transformation that isolates the original lexical rule predicates. The definite clause encoding of the interaction predicates resulting from unfolding the frame predicates for the lexical entry of Figure 15 with respect to the interaction predicate of Figure 18 is given in Figure 22. The lexical rule predicates called by these interaction predicates are defined as in Figures 8 and 12, except that the frame predicates are no longer called.

30 Note that it is only possible to eliminate the frame predicates, since they are never called independently of the covariation encoding.

$$\text{q\_1}(\boxed{In}, \begin{bmatrix} \text{B} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{2} \\ \text{Z} & \boxed{3} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{ :- lex\_rule\_1}(\boxed{In}, \boxed{Aux}), \text{ q\_2}(\boxed{Aux} \begin{bmatrix} \text{B} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{2} \\ \text{Z} & \boxed{3} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_1}(\boxed{In} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{ :- lex\_rule\_2}(\boxed{In}, \boxed{Aux}), \text{ q\_3}(\boxed{Aux} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_2}(\boxed{In} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{ :- lex\_rule\_2}(\boxed{In}, \boxed{Aux}), \text{ q\_7}(\boxed{Aux} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{X} & \boxed{3} \\ \text{Y} & \boxed{4} \\ \text{Z} & \boxed{5} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_7}(\boxed{In} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{3} \\ \text{X} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{ :- lex\_rule\_3}(\boxed{In}, \boxed{Aux}), \text{ q\_14}(\boxed{Aux} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{3} \\ \text{X} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_14}(\boxed{In} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{3} \\ \text{X} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{ :- lex\_rule\_3}(\boxed{In}, \boxed{Aux}), \text{ q\_14}(\boxed{Aux} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{B} & \boxed{2} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{3} \\ \text{X} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_14}(\boxed{In} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{2} \\ \text{Y} & \boxed{3} \\ \text{Z} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}) \text{:- lex\_rule\_4}(\boxed{In}, \boxed{Aux}), \text{ q\_19}(\boxed{Aux} \begin{bmatrix} \text{A} & \boxed{1} \\ \text{C} & {}_{t_2}\begin{bmatrix} \text{W} & \boxed{2} \\ \text{Y} & \boxed{3} \\ \text{Z} & \boxed{4} \end{bmatrix} \end{bmatrix}, \boxed{Out}).$$

$$\text{q\_1}(\boxed{In}, \boxed{In}). \quad \text{q\_2}(\boxed{In}, \boxed{In}). \quad \text{q\_3}(\boxed{In}, \boxed{In}). \quad \text{q\_7}(\boxed{In}, \boxed{In}). \quad \text{q\_14}(\boxed{In}, \boxed{In}). \quad \text{q\_19}(\boxed{In}, \boxed{In}).$$

Figure 22
Unfolding the frame predicates for the example entry with respect to the interaction predicate.

5. On-the-fly Application of Lexical Rules

We want our compiler to produce an encoding of lexical rules that allows us to execute lexical rules on-the-fly, i.e., at some time after lexical lookup. This is advantageous because postponing the execution of the interaction predicates allows more constraints on the word to be collected. When the interaction predicate is finally called, as a result of syntactic information being present, many of its possible solutions simply fail. The search tree that would have resulted from pursuing these possibilities at the beginning of processing does not have to be explored.[31]

As it stands, our encoding of lexical rules and their application as covariation in lexical entries does not yet support the application of lexical rules on-the-fly. With respect to processing, the extended lexical entry of Figure 17 is problematic because before execution of the call to *q_1*, it is not known which information of the base lexical entry ends up in a derived lexical entry, i.e., tag $\boxed{Out}$ is completely uninstantiated. This means that there is no way of indexing the lexical entries according to what kind of

31 According to Pollard and Sag (1987) on-the-fly application of lexical rules is also well-suited to playing a role in a model of language use.

derived entry one is looking for. As a result, it is necessary to execute the call to *q_1* immediately when the lexical entry is used during processing. Otherwise, there would be no information available to restrict the search-space of a generation or parsing process.

Flickinger, Pollard, and Wasow (1985) solve this problem using additional specifications: "By providing with each lexical rule a generic class frame which specifies the general form and predictable properties of the rule's output, we avoid unnecessary work when the lexical rule applies" (p. 264). In the following, we show that the additional specifications on the extended lexical entry needed to guide processing can be deduced automatically.

5.1 Constraint Propagation

The intuitive idea behind this improvement of the covariation encoding is to lift into the extended lexical entry the information that is ensured after all sequences of possible lexical rule applications for a particular base lexical entry have occurred. Note that this is not an unfolding step. Unfolding the interaction predicates with respect to the lexical entries basically expands out the lexicon off-line. Instead, what we do is factor out the information common to all definitions of the called interaction predicate by computing the most specific generalization of these definitions.

The most specific generalization does not necessarily provide additional constraining information. However, usually it is the case that lexical entries resulting from lexical rule application differ in very few specifications compared to the number of specifications in a base lexical entry. Most of the specifications of a lexical entry are assumed to be passed unchanged via the automatically generated frame specification. Therefore, after lifting the common information into the extended lexical entry, the out-argument in many cases contains enough information to permit a postponed execution of the interaction predicate. When C is the common information, and $D_1, \ldots, D_k$ are the definitions of the interaction predicate called, we use distributivity to factor out C in $(C \wedge D_1) \vee \cdots \vee (C \wedge D_k)$: We compute $C \wedge (D_1 \vee \cdots \vee D_k)$, where the D are assumed to contain no further common factors. Once we have computed C, we use it to make the extended lexical entry more specific. This technique closely resembles the off-line constraint propagation technique described by Marriott, Naish, and Lassez (1988). The reader is referred to Meurers and Minnen (1996) for a more detailed discussion of our use of constraint propagation.[32]

We illustrate the result of constraint propagation with our example grammar. Since the running example of this paper was kept small, for expository reasons, by only including features that do get changed by one of the lexical rules (which violates the empirical observation mentioned above), the full set of lexical rules would not provide a good example. Let us therefore assume that only the lexical rules 1 and 2 of Figure 11 are given. We then only obtain seven of the clauses of Figure 22: those calling *lex_rule_1* or *lex_rule_2*, as well as the unit clauses for *q_1*, *q_2*, *q_3*, and *q_7*. Applying constraint propagation to the extended lexical entry of Figure 17 yields the result shown in Figure 23. The information common to all solutions to the interaction call is lifted up into the lexical entry and becomes available upon lexical lookup.

32 In certain cases an extension of the constraint language with named disjunctions or contexted constraints (Maxwell and Kaplan 1989; Eisele and Dörre 1990; Griffith 1996) can be used to circumvent constraint propagation. Encoding the disjunctive possibilities for lexical rule application in this way, instead of with definite clause attachments, makes all relevant lexical information available at lexical lookup. For analyses proposing infinite lexica, though, a definite clause encoding of disjunctive possibilities is still necessary and constraint propagation is indispensable for efficient processing.

$$\text{lex\_entry}(\boxed{Out}\left[\begin{array}{ll}\text{A} & b\\ \text{B} & \boxed{2}\\ \text{C} & {}_{t_2}\!\left[\text{Z}\ \boxed{3}\right]\end{array}\right]) \text{:- q\_1}(\left[\begin{array}{ll}\text{A} & b\\ \text{B} & \boxed{2} -\\ \text{C} & {}_{t_2}\!\left[\begin{array}{ll}\text{W} & -\\ \text{X} & -\\ \text{Y} & -\\ \text{Z} & \boxed{3}\langle a,b\rangle\end{array}\right]\end{array}\right], \boxed{Out}).$$

Figure 23
An entry suitable for on-the-fly application (lexical rules 1 and 2 only).

5.2 Dynamic and Static Coroutining

Even though we see on-the-fly application as a prerequisite of a computational treatment of lexical rules, it is important to note that a postponed evaluation of lexical rule application is not always profitable. For example, in the case of generation, underspecification of the head of a construction can lead to massive nondeterminism or even nontermination when not enough restricting information is available to generate its complements (Martinović and Strzalkowski 1992; Minnen, Gerdemann, and Götz 1995). Criteria to determine when it is most profitable to execute calls to an interaction predicate are required.

One possibility is to annotate the lexical rule encoding with such criteria by means of delay statements, as, for example, suggested by van Noord and Bouma (1994). While we consider this kind of control facility (Naish [1986] and references therein) to be, in general, indispensable for efficient processing, it also has disadvantages that make it desirable to search for alternative or additional mechanisms: Delay statements presuppose the procedural annotation of an otherwise declarative specification. Substantial computational expertise is required to provide restrictions on the instantiation status of a goal, which must be fulfilled before the goal can be executed. Furthermore, the computational bookkeeping necessary for the delaying mechanism is very expensive. An interesting alternative, therefore, is to automatically determine certain control problems and deal with them in an off-line fashion along the lines of Minnen, Gerdemann, and Götz (1995) and Minnen, Gerdemann, and Hinrichs (1996). They describe the use of a dataflow analysis for an off-line improvement of grammars that determines automatically when a particular goal in a clause can best be executed.

6. Efficiency Evaluation

The computational treatment of lexical rules as covariation in lexical entries was implemented in Prolog by the authors in cooperation with Dieter Martini for the ConTroll system (Gerdemann and King 1994; Götz and Meurers 1997a). We tested the covariation approach with a complex grammar implementing an HPSG analysis covering the so-called aux-flip phenomenon, and partial-VP topicalization in the three clause types of German (Hinrichs, Meurers, and Nakazawa 1994). This test grammar includes eight lexical rules; some serve syntactic purposes, like the Partial-VP Topicalization Lexical Rule, others are of morphological nature as, for example, an inflectional lexical rule that relates nonfinite verbs to their finite form. Our compiler distinguished seven word classes. Some nouns and most verbal lexical entries fed lexical rules, and a single base lexical entry resulted in up to 12 derivations.

6.1 Time Efficiency

To evaluate the time efficiency of the covariation encoding, we compared the parse times for our test grammar with three different computational encodings of the lexicon:

the expanded out lexicon, the basic covariation encoding, and the covariation encoding improved by constraint propagation.[33]

As discussed in Section 5.1, the parsing times with a covariation lexicon without constraint propagation suffer significantly from the lack of information directly available upon lexical lookup. For the test grammar, the resulting extended search-space of parsing with the basic covariation encoding leads to a performance that is, on average, 18 times slower than that with the expanded out lexicon.

The use of constraint propagation, however, makes it possible to exploit the covariation encoding of lexical rule application such that it results in an increase in speed. Parsing with the test grammar using the constraint propagated covariation lexicon is, on average, 25 percent faster than the performance with the expanded out lexicon. The representation of lexical information in a constraint propagated covariation lexicon makes the maximum information available at lexical lookup while requiring a minimum number of nondeterministic choices to obtain this information.

Summing up, the relation between parsing times with the expanded out (EXP), the covariation (COV), and the constraint propagated covariation (IMP) lexicon for the test grammar can be represented as IMP : EXP : COV = 0.75 : 1 : 18. With respect to our test grammar, the constraint propagated covariation lexicon thus is the fastest lexical encoding.

6.2 Space Efficiency

Besides the effect of requiring a minimum of nondeterministic choices and thereby reducing the number of resolution steps to increase time efficiency, the covariation encoding of lexical rules can result in an additional speedup since it reduces the space requirements of large grammars.

A comparison of space efficiency between an expanded out and a covariation lexicon needs to compare two different encodings. The expanded out lexicon consists solely of lexical entries, whereas the covariation lexicon is made up of three different data structures: the extended base lexical entries, the interaction predicates, and the lexical rule predicates. We focus on a qualitative evaluation of space efficiency, rather than on providing results for the test grammar, since the space efficiency of the covariation encoding relative to the expanded out lexicon is dependent on several properties of the grammar: the number of lexical entries in the lexicon that can undergo lexical rule application, the size of the lexical entries, and the number of lexical entries belonging to a word class.

Since only base lexical entries that feed lexical rules are modified by the lexical rule compiler, the covariation encoding naturally only results in space savings for those lexical entries to which lexical rules apply.

The space efficiency is dependent on the size of the lexical entries since in the covariation encoding much of the lexical information that is specified in a base lexical entry is not duplicated in the lexical entries that can be derived from it, as is the case for an expanded lexicon. Thus, the more information represented in a base lexical entry, the greater the space saving achieved by the covariation encoding. In lexically oriented grammar formalisms like HPSG, the lexical entries are highly information rich. A covariation treatment of HPSG lexica therefore can be particularly profitable.

The number of lexical entries belonging to a word class is relevant since the interaction predicates are identical for all lexical entries belonging to the same word class.

33 The lexicon of the test grammar can be expanded out off-line since the recursive Complement Extraction Lexical Rule applies only to full verbs, i.e., lexical entries with a complement list of finite length. As a result, the grammar does not have an infinite lexicon.

This means that the more lexical entries in a word class, the greater the saving in space. The covariation approach therefore is particularly attractive for grammars with a large lexicon.

7. Related Work

The powerful mechanism of lexical rules (Carpenter 1991) has been used in many natural language processing systems. In this section we briefly discuss some of the more prominent approaches and compare them with the treatment proposed in this paper.

7.1 Off-line Expansion of Lexical Rules

A common computational treatment of lexical rules adopted, for example, in the ALE system (Carpenter and Penn 1994) consists of computing the transitive closure of the base lexical entries under lexical rule application at compile-time. While this provides a front-end to include lexical rules in the grammars, it has the disadvantage that the generalizations captured by lexical rules are not used for computation. We mentioned in Section 2.2 that eliminating lexical rules in a precompilation step makes it impossible to process lexical rules or lexical entries that impose constraints that can only be properly executed once information from syntactic processing is available. A related problem is that for analyses resulting in infinite lexica, the number of lexical rule applications needs to be limited. In the ALE system, for example, a depth bound can be specified for this purpose. Finally, as shown in Section 6, using an expanded out lexicon can be less time and space efficient than using a lexicon encoding that makes computational use of generalizations over lexical information, as, for example, the covariation encoding.

7.2 Lexical Rules as Unary Phrase Structure Rules

Another common approach to lexical rules is to encode them as unary phrase structure rules. This approach is taken, for example, in LKB (Copestake 1992) where lexical rules are introduced on a par with phrase structure rules and the parser makes no distinction between lexical and nonlexical rules (Copestake 1993, 31). A similar method is included in PATR-II (Shieber et al. 1983) and can be used to encode lexical rules as binary relations in the CUF system (Dörre and Eisele 1991; Dörre and Dorna 1993b) or the TFS system (Emele and Zajac 1990; Emele 1994). The covariation approach described in this paper can be viewed as a domain-specific refinement of such a treatment of lexical rules.

The encoding of lexical rules used in the covariation approach is related to the work of van Noord and Bouma (1994), who describe the hand-encoding of a single lexical rule as definite relations and show how these relations can be used to constrain a lexical entry. The covariation approach builds on this proposal and extends it in three ways: First, the approach shows how to detect and encode the interaction of a set of lexical rules. Second, it provides a way to automatically obtain a definite clause encoding of lexical rules and their interaction. Finally, it automatically derives the frame specification for lexical rules such that, following standard HPSG practice, only the information changed in a lexical rule needs to be specified.

7.3 Alternative Ways to Express Lexical Generalizations

Lexical rules have not gone unchallenged as a mechanism for expressing generalizations over lexical information. In a number of proposals, lexical generalizations are captured using lexical underspecification (Kathol 1994; Krieger and Nerbonne 1992;

Riehemann 1993; Oliva 1994; Frank 1994; Opalka 1995; Sanfilippo 1995). The lexical entries are only partially specified, and various specializations are encoded via the type hierarchy, definite clause attachments, or a macro hierarchy.

These approaches seem to propose a completely different way to capture lexical generalizations. It is therefore interesting that the covariation lexical rule compiler produces a lexicon encoding that, basically, uses an underspecification representation: The resulting definite clause representation after constraint propagation represents the common information in the base lexical entry, and uses a definite clause attachment to encode the different specializations.

8. Summary

We presented a new computational treatment of HPSG lexical rules by describing a compiler that translates a set of lexical rules as specified by a linguist into definite relations, which are used to constrain lexical entries. The frame of a lexical rule and lexical rule interaction is automatically determined and the interaction is represented as a finite-state automaton. The automaton allows us to encode lexical rule interaction without actually having to apply lexical rules a possibly infinite number of times. Word classes relevant to lexical rule application are automatically detected and the corresponding finite-state automata are refined in order to avoid lexical rule applications that are guaranteed to fail. The refined automata are encoded as definite relations and each base lexical entry is extended to call the relation corresponding to its class. Finally, the encoding of lexical rules and their interaction is advanced using constraint propagation to allow coroutining of its execution with other grammar constraints. This reduces the number of nondeterministic choices related to lexical lookup, and, more importantly, allows syntactic information to be used to ensure termination of the covariation encoding of lexical rules. Finally, we discussed implementation results and illustrated the improvement in time and space efficiency resulting from the covariation encoding.

Acknowledgments
The research reported here was supported by Teilprojekt B4 "From Constraints to Rules: Efficient Compilation of HPSG Grammars" of SFB 340 "Sprachtheoretische Grundlagen für die Computerlinguistik" of the Deutsche Forschungsgemeinschaft. The authors wish to thank Thilo Götz and Dale Gerdemann, Erhard Hinrichs, Paul King, Suresh Manandhar, Dieter Martini, Bill Rounds, and the anonymous reviewers. Of course, the authors are responsible for all remaining errors.

References

Briscoe, Ted and Ann Copestake. 1996. Controlling the application of lexical rules. In *Proceedings of the SIGLEX Workshop on Breadth and Depth of Semantic Lexicons*, Santa Cruz, CA.

Briscoe, Ted, Ann Copestake, and Valeria de Paiva, editors. 1992. *Default Inheritance Within Unification-Based Approaches to the Lexicon*. Cambridge University Press, Cambridge, UK.

Calcagno, Mike. 1995. Interpreting lexical rules. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Also in: Proceedings of the ACQUILEX II Workshop on Lexical Rules, 1995, Cambridge, UK.

Calcagno, Mike, Detmar Meurers, and Carl Pollard. In preparation. On the nature of lexical rules in head-driven phrase structure grammar. Unpublished manuscript, Ohio State University and University of Tübingen.

Calcagno, Mike and Carl Pollard. 1995. Lexical rules in HPSG: What are they? Unpublished manuscript, Ohio State University, Columbus, OH.

Carpenter, Bob. 1991. The generative power of categorial grammars and Head-Driven Phrase Structure Grammars with lexical rules. *Computational Linguistics*, 17(3):301–314.

Carpenter, Bob and Gerald Penn. 1994. ALE—The Attribute Logic Engine, User's

Guide, Version 2.0.1, December 1994. Technical report, Computational Linguistics Program, Philosophy Department, Carnegie Mellon University, Pittsburgh, PA.

Copestake, Ann. 1992. *The Representation of Lexical Semantic Information.* Cognitive science research paper CSRP 280, University of Sussex, Sussex, UK.

Copestake, Ann. 1993. The Compleat LKB. Technical report 316, University of Cambridge Computer Laboratory, Cambridge, UK.

Dörre, Jochen and Michael Dorna, editors. 1993a. *Computational Aspects of Constraint-Based Linguistic Description I.* University of Stuttgart, Stuttgart, Germany.

Dörre, Jochen and Michael Dorna. 1993b. CUF—A formalism for linguistic knowledge representation. In Dörre and Dorna (1993a).

Dörre, Jochen and Andreas Eisele. 1991. A Comprehensive Unification Based Formalism. DYANA Deliverable R3.1.B, University of Stuttgart, Stuttgart, Germany.

Eisele, Andreas and Jochen Dörre. 1990. Disjunctive Unification. Technical Report 124, IBM Wissenschaftliches Zentrum, Institut für Wissensbasierte Systeme.

Emele, Martin. 1994. The typed feature structure representation formalism. In *Proceedings of the International Workshop on Sharable Natural Language Resources,* Nara, Japan.

Emele, Martin and Rémi Zajac. 1990. Typed unification grammars. In *Proceedings of the 13th Conference on Computational Linguistics (COLING),* Helsinki, Finland.

Flickinger, Daniel. 1987. *Lexical Rules in the Hierarchical Lexicon.* Ph.D. thesis, Stanford University, Stanford, CA.

Flickinger, Daniel, Carl Pollard, and Thomas Wasow. 1985. Structure-sharing in lexical representation. In *Proceedings of the 23rd Annual Meeting,* pages 262–267, Chicago, IL. Association for Computational Linguistics.

Frank, Annette. 1994. Verb second by underspecification. In *Proceedings of KONVENS,* Berlin. Springer-Verlag.

Gerdemann, Dale. 1995. Open and closed world types in NLP systems. In *Proceedings of the DGfS Fachtagung Computerlinguistik,* Düsseldorf, Germany.

Gerdemann, Dale and Paul King. 1994. The correct and efficient implementation of appropriateness specifications for typed feature structures. In *Proceedings of the 15th Conference on Computational Linguistics (COLING),* Kyoto, Japan.

Ginsberg, Matthew L., editor. 1987. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann.

Götz, Thilo. 1994. A Normal Form for Typed Feature Structures. Arbeitspapiere des SFB 340 no. 40, University of Tübingen, IBM, Heidelberg, Germany.

Götz, Thilo and Detmar Meurers. 1995. Compiling HPSG type constraints into definite clause programs. In *Proceedings of the 33rd Annual Meeting,* Boston, MA. Association for Computational Linguistics.

Götz, Thilo and Detmar Meurers. 1996. The importance of being lazy—Using lazy evaluation to process queries to HPSG grammars. In *Proceedings of TALN 96 (Joint Session with the Third International Conference on HPSG),* Marseille, France.

Götz, Thilo and Detmar Meurers. 1997a. The ConTroll system as large grammar development platform. In *Proceedings of the ACL/EACL Post-Conference Workshop on Computational Environments for Grammar Development and Linguistic Engineering,* Madrid, Spain.

Götz, Thilo and Detmar Meurers. 1997b. Interleaving universal principles and relational constraints over typed feature logic. In *Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the EACL,* Madrid, Spain.

Griffith, John. 1996. Modularizing contexted constraints. In *Proceedings of the 16th Conference on Computational Linguistics (COLING),* Copenhagen, Denmark.

Hinrichs, Erhard, Detmar Meurers, and Tsuneko Nakazawa, editors. 1994. *Partial-VP and Split-NP Topicalization in German—An HPSG Analysis and its Implementation.* Number 58.

Hinrichs, Erhard and Tsuneko Nakazawa. 1989. Flipped out: Aux in German. In *Papers from the 25th Regional Meeting,* pages 193–202, Chicago. Chicago Linguistic Society.

Hinrichs, Erhard and Tsuneko Nakazawa. 1994. Partial-VP and split-NP topicalization in German: An HPSG analysis. In Hinrichs, Meurers, and Nakazawa (1994).

Hinrichs, Erhard and Tsuneko Nakazawa. 1996. Applying lexical rules under subsumption. In *Proceedings of the 16th Conference on Computational Linguistics (COLING),* pages 543–549, Copenhagen, Denmark.

Kathol, Andreas. 1994. Passive without lexical rules. In John Nerbonne, Klaus Netter, and Carl Pollard, editors, *HPSG for*

German. CSLI Lecture Notes, Stanford University, Stanford, CA.

King, Paul. 1989. *A Logical Formalism for Head-driven Phrase Structure Grammar*. Ph.D. thesis, University of Manchester, Manchester, UK.

King, Paul. 1994. An Expanded Logical Formalism for Head-driven Phrase Structure Grammar. Arbeitspapiere des Sonderforschungsbereich 340 no. 59, University of Tübingen, Tübingen, Germany.

Krieger, Hans-Ulrich and John Nerbonne. 1992. Feature-based inheritance networks for computational lexicons. In Briscoe, Copestake, and de Paiva (1992).

Manandhar, Suresh. 1995. The Update Operation in Feature Logic. Unpublished Manuscript, HCRC at University of Edinburgh, UK.

Marriott, Kim, Lee Naish, and Jean-Louis Lassez. 1988. Most specific logic programs. In *Proceedings of 5th International Conference and Symposium on Logic Programming*.

Martinović, Miroslav and Tomek Strzalkowski. 1992. Comparing two grammar-based generation algorithms: A case study. In *Proceedings of the 30th Annual Meeting*, Newark, DE. Association for Computational Linguistics.

Maxwell, John and Ronald Kaplan. 1989. An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27.

McCarthy, John and Patrick Hayes. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer and Michie (1969). Reprinted in Ginsberg (1987).

Meltzer, Bernard and Donald Michie, editors. 1969. *Machine Intelligence 4*. Edinburgh University Press, Edinburgh, UK.

Meurers, Detmar. 1994. On implementing an HPSG theory: Aspects of the logical architecture, the formalization and the implementation of Head-driven Phrase Structure Grammars. In Hinrichs, Meurers, and Nakazawa (1994).

Meurers, Detmar. 1995. Towards a semantics for lexical rules as used in HPSG. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Also in *Proceedings of the ACQUILEX II Workshop on Lexical Rules*, 1995, Cambridge, UK.

Meurers, Detmar and Guido Minnen. 1995. A computational treatment of HPSG lexical rules as covariation in lexical entries. In *Proceedings of the Fifth International Workshop on Natural Language Understanding and Logic Programming*, Lisbon, Portugal.

Meurers, Detmar and Guido Minnen. 1996. Off-line constraint propagation for efficient HPSG processing. In *Proceedings of TALN 96 (Joint Session with the Third International Conference on HPSG)*, Marseille, France.

Miller, Philip and Ivan Sag. 1993. French Clitic Climbing Without Clitics or Climbing. Unpublished Manuscript, University of Lille and Stanford University.

Minnen, Guido. In preparation. *Natural Language Processing with Constraint-Logic Grammars: Grammar Compilation for Declarative Under-determination*. Ph.D. thesis.

Minnen, Guido, Dale Gerdemann, and Thilo Götz. 1995. Off-line optimization for earley-style HPSG processing. In *Proceedings of the 7th Conference of the EACL*, Dublin, Ireland.

Minnen, Guido, Dale Gerdemann, and Erhard Hinrichs. 1996. Direct automated inversion of logic grammars. *New Generation Computing* 14(2):131–168.

Naish, Lee. 1986. *Negation and Control in Prolog*. Springer Verlag, New York.

O'Keefe, Richard. 1990. *The Craft of Prolog*. MIT Press, Cambridge, MA.

Oliva, Karel. 1994. HPSG lexicon without lexical rules. In *Proceedings of the 15th Conference on Computational Linguistics (COLING)*, Kyoto, Japan.

Opalka, Annette. 1995. Statische Programmtransformationen zur effizienten Verarbeitung constraintbasierter Grammatiken. Diplomarbeit, University of Stuttgart, Stuttgart, Germany.

Pereira, Fernando and Stuart Shieber. 1987. *Prolog and Natural Language Analysis*. CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University, Stanford, CA.

Pettorossi, Alberto and Maurizio Proietti. 1994. Transformations of logic programs: Foundations and techniques. *Journal of Logic Programming 19/20*:261–320.

Pollard, Carl and Ivan Sag. 1987. *Information-based Syntax and Semantics, Vol. 1*. Number 13 of CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University, Stanford, CA.

Pollard, Carl and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, IL.

Riehemann, Susanne. 1993. Word Formation in Lexical Type Hierarchies: A Case Study of *bar*-Adjectives in German. Master's thesis, University of Tübingen, Tübingen, Germany. Also published as SfS-Report-02-93, Seminar für Sprachwissenschaft, University of Tübingen.

Sanfilippo, Antonio. 1995. Lexical polymorphism and word disambiguation. In *Proceedings of the American Association for Artificial Intelligence (AAAI)*, Stanford University, Stanford, CA.

Shieber, Stuart, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and Mabry Tyson. 1983. The formalism and implementation of PATR II. In *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, CA, pages 39–79.

Tamaki, Hisao and Taisuke Sato. 1984. Unfold/Fold transformation of logic programs. In *Proceedings of the 2nd International Conference on Logic Programming*, Uppsala, Sweden.

Torisawa, Kentaro and Jun'ichi Tsuji. 1996. Off-line raising, dependency analysis and partial unification. In *Proceedings of TALN 96 (Joint Session with the Third International Conference on HPSG)*, Marseille, France.

van Noord, Gertjan and Gosse Bouma. 1994. The scope of adjuncts and the processing of lexical rules. In *Proceedings of the 15th Conference on Computational Linguistics (COLING)*, Kyoto, Japan.

Some of the above papers can be obtained electronically through the URL provided on the first page.