Towards Probabilistic Extensions of Constraint-Based Grammars

Andreas Eisele*

May 19, 1998
Contribution to DYANA-2 Deliverable R1.2.B

1 Abstract

We motivate the need for probabilistic extensions of constraint-based grammar formalisms. We take CUF ([DD93]) as an example to discuss some of the necessary design decisions. We sketch a formal semantics for P-CUF, a simple probabilistic extension of CUF, and compare it to more complex alternatives.

We describe some implementation techniques for efficient search strategies for P-CUF and discuss some questions concerning the automatic acquisition of probabilistic parameters from training material.

2 Introduction

Whereas logic-based grammar formalisms provide powerful languages for the declarative description of linguistic competence, their suitability for the description of linguistic performance and for the application in practical systems has often been questioned, since such frameworks make it difficult to describe gradual distinctions, which however seem essential in practice. The need for coverage and robustness usually introduces large amounts of ambiguity which cannot be resolved by rule-based reasoning alone.[1]

On the other hand, probabilistic models of linguistic performance which have so far been used in applications such as speech recognition (see e.g. [Jel90] for an

*The research described in this report has been sponsored in part by the German BMFT within the Verbundprojekt VERBMOBIL *"Entwicklung eines mobilen Systems zur Übersetzung von Verhandlungsdialogen in face-to-face Situationen"*, Förderkennzeichen 01 IV 101 U

[1]Even if we could capture all relevant linguistic knowledge in a purely symbolic description, there are important real-world applications involving speech recognition, document scanning, typing errors etc. where considerable uncertainty or ambiguity is already present in the input data.

overview) are quite simplistic (e.g. based on word trigrams or hidden markov models) and have obvious difficulties even for simple phenomena like phrase structure and agreement.

It seems clear that a combination of symbolic and probabilistic approaches has much to offer. An incomplete list of phenomena, for which only such a combination of rule-based and preference-based reasoning seems to be powerful enough includes such diverse things as ambiguities of attachment, word sense, scope, anaphora reference and translation, and the description of partially free word order.

Although ambiguity resolution in some cases requires quite deep understanding of an utterance, we feel that many cases can already be resolved on the basis of relatively shallow criteria. Instead of waiting for a grand unified theory of all necessary levels, we aim at an approximation of some levels which so far have not undergone a comprehensive theoretical explanation. Naturally, an approximation is bound to give wrong predictions in some cases, but we expect it to behave correctly at least in typical cases. We also expect that an approximative treatment of additional phenomena can ease the development of grammars with broader coverage and thus will help to highlight phenomena where additional theoretical work is required.

Whereas our work is not the first one that tries to combine constraint-based descriptions of grammar with probabilistic preferences (see e.g. [BC93, Wu90, BJL$^+$92] for other attempts), our approach is new as far as we integrate probabilistic preferences into a powerful deductive formalism, whose application is not restricted to the description of phrase-structure-based grammars, but supports the modular description of general grammatical principles and language specific parameters which is popular in recent linguistic theories such as HPSG [PS87] and GB [Cho81].

An extension of constraint-based grammar formalism should provide means to express grades in grammaticality, in order to rank competing analyses according to their quality. Given a formalism that allows for preferential distinctions between possible analyses, the overall picture of grammar development will be that the grammar writer will provide a symbolic, constraint-based description of fundamental linguistic principles that have to be obeyed by wellformed utterances, whereas the details of the preferential distinctions will be as much as possible induced automatically from training examples by suitable estimation methods.

It is clear that stochastic preferences can only distinguish the quality of different *possible* analyses, but cannot restore analyses that are rejected by symbolic constraints. Hence the sybolic part of the description should be as "liberal" as possible, and in cases where grammar writers must chose – due to limited resources – between overgeneration (not enough constraints) and undergeneration (too strong constraints), they should always go for the former alternative. See [BJL$^+$92] for a similar view.

In the sequel, we discuss some of the design decisions which are required for a preference-based extension of a grammar formalism and try to motivate our choice. Some of the decisions are grouped together, since one argumentation applies to several questions.

Numerical vs. qualitative preferences ?

User-specified vs. derived from training material ?

We think that it should be at least possible to derive preference information from training material, since preferences are much more sensitive to domain specific use of language than grammatical rules and principles and hence the specification of such information might otherwise have to be repeated quite often. Also, it is more difficult for a grammar writer to specify preference information manually than symbolic information concerning yes/no distinctions. The idea is therefore to provide a general grammar with some unspecified parameters, which are added automatically according to a given set of examples from a limited domain.

Probabilistic vs. possibilistic or fuzzy ?

We prefer a probabilistic approach, since probability theory is well understood and can serve as a sound theoretical basis for our application. One of the main advantages of probabilistic approaches is the possibility to define training procedures to adapt the parameters to examples. Approaches based on probability theory have been quite successful in speech recognition and other applications of pattern recognition. Since there exists a rapidly growing stock of work on probabilistic description of natural language ([HR93, CM93] and many others), it is a very gratifying perspective if the results of such approaches can easily be integrated into a general linguistic description.

Probabilities tightly or loosely coupled to grammar ?

Occasionally, loosely coupled combinations of symbolic and stochastic models have been used with some success. For instance, work by [Sch94] in the framework of parsing spontaneous speech uses a combination of a grammar and an independent stochastic language model to find a linguistically wellformed hypothesis for which the combined score of the speech recognizer and the stochastic language model has the best value. [Bre93] shows how a grammar in CUF can be combined with a stochastic language model used for part-of-speech prediction. Whereas such loosely coupled combinations of symbolic and stochastic models allow for a quick integration of already available modules, such approaches do not seem promising on the long run, since both components have obvious limitations which cannot be resolved without deeper interaction. More specifically, for none of the phenomena listed above, a loosely coupled combination seems powerful enough, since the module describing the stochastic preferences has no access to relevant structural information.

Fixed vs. user-defined calculation of quality measure ?

With a probabilistic interpretation in mind, there is usually not much choice concerning the method how the quality scores of partial results should be combined to get an overall score. Therefore this calculation scheme will be fixed in our implementation. While this might occasionally feel like a restriction[2], it has some important advantages. It simplifies the descriptive work in as much as the grammar writer has no need and no possibility to fiddle with combination schemes. Instead, a general mechanism provides support for general implementation techniques for efficient search, compile-time optimizations, training procedures and the like, which could not be done in general, if arbitrary schemes were allowed for preference calculation.

Where to attach the probabilities ?

How much context sensitivity ?

Whereas probability theory provides a sound theoretical basis for our approach, it has some severe drawbacks. Assume we have a reliable estimate for a conditional probability $p(e \mid c_1)$ for an event e given knowledge of some context c_1. Now in general, we will not observe e in a context exactly equal to c_1, the context for which the probability was estimated. Instead, we will usually have a context which is compatible with c_1, but we will have additional knowledge about the context, which we will write informally as c_2. We ought to know $p(e \mid c_1, c_2)$, which can in general not be derived from $p(e \mid c_1)$ nor from $p(e \mid c_2)$ nor from both of them. But it is impossible to obtain reliable estimates for all conditional probabilities which might appear in practice.

The usual way out of this dilemma is to impose structural restrictions on the influence of context that can be modeled in a description. E.g. in Markov-models a sequence of observations is produced by a sequence of internal states, where the probability of each observation is conditioned alone on its corresponding state. In stochastic context-free grammars (SCFG) [BT73, LY90], the probability distribution over the realizations of a given nonterminal symbol depends only on that symbol, and not on its context.

Although the assumptions of context-independence underlying such restrictions are usually too strong to be valid, the probability estimates obtained under such assumptions are based on larger sets of data and are therefore more reliable than estimates for more detailed models which suffer from the problem of sparse training data. To get useful results, it is important to find a good compromise between the conflicting goals of context sensitivity and reliable estimates of parameters.

The maxim of our extension to CUF will be to introduce as little additional machinery as possible. Hence we will attach probabilities not to every construct of the language, but select only one level for our extension. Since CUF is a relational extension of a simple language of constraints on feature structures using Horn clauses, we can simplify the difficulties of our probabilistic extension if we restrict it to one of

[2]E.g. when constructing a compound data structure such as a list, we have no freedom to interpret it as a disjunction of possibilities and add the probabilities of its elements. Since the parts of the description producing the elements are combined conjunctively, we are forced to multiply the scores.

the levels involved. One way of doing so is to leave the feature constraint language as it is and provide a probabilistic extension only to the relational language. We can do this by attaching probabilities to each rule in a CUF description. If we want to, we can restrict the scope of the probabilistic extension even further. If we decompose the relational extension of feature logic to CUF into several steps, we are free to assume a probabilistic interpretation only for some of these steps.

In the sequel, we assume that for a subset of the predicates in a CUF program, the defining clauses are annotated with probabilities. For each annotated predicate, we will require that the probabilities of all rules sum to 1.

The decision to base the granularity of the probabilistic description on the granularity of the predicate definitions is not a severe restriction, and hence the criticism against stochastic context free grammars based on their inability to express e.g. lexical information (see [BJL+92, Sch92, Res92] for a discussion) does not apply to our extension. Should it turn out that we need more fine-grained distinctions or more context dependencies than is possible within a given description, we can always add auxiliary definitions for a new predicate involving all relevant information, whose only purpose is to provide attachment points for additional probabilistic parameters.

To sum up our choices, we are aiming at a formalism with the following properties:

- Horn clauses over feature logic with numerical scores for some of the definitions
- The scores are interpreted as probabilities
- The scores can be estimated from training material
- Probabilities allow for a tight integration of structural and preferential description
- Fixed schema for the interpretation and combination of probabilities

3 Form and interpretation of probabilistically annotated definitions

Although CUF offers a convenient functional notation, the essence of CUF definitions can be expressed in a purely relational way. A clause has the form

$$r \leftarrow q_1 \wedge \ldots \wedge q_n \wedge \phi.$$

where the so-called relational atoms r and $q_1 \ldots, q_n$ are built only from a predicate name and distinct variables as arguments and where ϕ is a possibly complex constraint from a built-in constraint language ranging over those variables. An annotated clause has the form

$$r \leftarrow_p q_1 \wedge \ldots \wedge q_n \wedge \phi.$$

where p is a probability in the range $(0\ldots 1)$ and all probabilities attached to clauses of the same predicate sum to 1.

Our interpretation of a CUF program with probabilities is a stochastic procedure that constructs SLD proof trees according to the given probability distribution. Hence each proof tree has a certain probability of being constructed, which depends only on probabilities attached to the rules used in the proof. The tree construction process is quite analogous to the construction of derivation trees in a stochastic context-free grammar.

Since the same proof tree can be constructed in different orders, we have to fix the selection of the next goal to rewrite during the proof by introducing a selection function S. This is only a technical trick to avoid spurious nondeterminism in the construction of the proofs, and has no influence on the results.

Given a P-CUF program P, a selection function S, and a goal G_0, a proof of G_0 from P with answer constraint ϕ_n and probability p is a sequence

$$\langle G_0, \phi_0\rangle, \ldots \langle G_n, \phi_n\rangle$$

where $\phi_0 = \textsc{True}$, G_n is the empty conjunction of goals, and for each $\langle G_i, \phi_i\rangle, \langle G_{i+1}, \phi_{i+1}\rangle$ in the sequence, there is a variant of a clause from P of the form $H \leftarrow_{p_i} B \wedge \phi$ such that for some G , $G_i = H \wedge G$, $S(G_i) = H$, $G_{i+1} = B \wedge G$, $\phi_{i+1} = \phi_i \wedge \phi$, and $p = \prod_{i=0}^{n-1} p_i$.

We will write $p(\phi \mid G)$ for the probability that the prover rewrites a goal G to the answer constraint ϕ. In general, we will only be interested in proofs for which the answer constraint is satisfiable.

Consider for instance the following definitions[3]:

```
p(X,Y,Z) ←1 q(X,Y), r(Y,Z).

q(a,b) ←0.4 .
q(X,c) ←0.6 .

r(b,d) ←0.5 .
r(X,e) ←0.5 .
```

Confronted with a goal `?- p(A,B,C).` the prover can construct 4 different proof trees, which are identical in shape, but annotated with different constraints. We get the following probabilities and answer constraints:

$p = 0.2$: `A=a` $\wedge$ `B=b` $\wedge$ `C=d`
$p = 0.2$: `A=a` $\wedge$ `B=b` $\wedge$ `C=e`

[3]In the sequel we will freely intermix notations form CUF and Prolog, as long as no confusion can arise.

$p = 0.3$: `B=c` $\wedge$ `C=e`
$p = 0.3$: FALSE

Given a more constrained goal, such as `?- p(A,b,C).` , we will get a higher probability of failure. Since each proof tree possibly constrains the variables appearing in a goal, we get for each goal containing variables a probability distribution over constraints on these variables. However, the probabilities only sum to 1 if we also take the probability of failure into account[4]

There are several ways we could deal with the fact that there is a nonzero probability for invalid proof trees. These ways coincide with different ways of combining probability distributions over partially overlapping sets of variables. In the example given above, the predicates q and r in the body of the p-clause introduce a probability distribution on possible value-pairs for the respective variable pairs. It is not obvious how these should be combined, since both of them constrain the value of Y and are hence not independent.

In this situation, the simplest approach seems to be the assumption that proof trees for both the q and the r predicate are constructed entirely independent of each other, and the cases where both produce incompatible constraints on a shared variable increase the overall probability of producing an inconsistent proof.[5]

The fact that a (perhaps major) part of the probability mass is spent on proofs with inconsistent constraints is not a real problem, as long as the purpose of our probabilistic annotations is not primarily to estimate absolute probabilities for certain observations, but to compare different explanations of a given observation relative to

[4] In the presence of recursive definitions, additional probability mass can be lost for the construction of infinite proof trees. See [BT73] for the discussion of similar effects in SCFGs and for criteria that guarantee that a SCFG produces finite syntax trees with probability 1, which carry over to our generalized application.

[5] This way of dealing with feature- or constraint-based information is of course not the only possible one. A different approach would be to make the flow of information in the proof trees more explicit and view the probabilities of generating a certain part of the proof tree conditioned on information produced in other parts.

For instance, we could see the predicate q as a generator of instantiations which have to be taken as given in the definition of r and in other parts of the description. Then it seems motivated to describe the behavior of r in terms of conditional probabilities, which determine the value of Z, given knowledge of Y. Alternatively, we could also let r generate instantiations and let q condition on them. Both approaches would exhaust the overall probability better in the sense that the model would not produce invalid proof trees. However, it would require more descriptive work in order to make the flow of information explicit. A possible notation indicating the dependencies would be `p(X,Y,Z) :- q(X,Y), r(Z,|Y).`

Whereas usually the probabilities of all rules for a given predicate sum to 1, we would now have probabilities that sum to 1 for each instantiation of the input variables. It is not clear how cases should be handled where the input parameters are not fully instantiated. Such cases should therefore be excluded for clarity. A more severe problem lies in the fact that in a general CLP scheme there is no dichotomy between ground and non-ground terms and it is not clear what “instantiated” means. However, since consistent models based on conditional probabilities have certain advantages, this approach and its specific difficulties should also be further investigated.

each other.

However, the fact that probabilistic models can be "deficient" makes it more difficult to measure the performance of a model relative to a given set of examples using cross entropy as a criterion. Consider the comparison of a deficient model that describes the characteristics of the domain pretty well with a model that is less adequate, but manages to produce useful solutions with probability 1. It might well be that the deficient model is less likely to produce the examples due to the "wasted" probability and hence looks worse. In order to achieve a fair comparison, one has to normalize the probability for the test set with the overall probability that the model produces a consistent result. However, the exact computation of this normalizing constant is difficult in the general case.[6]

4 Procedural Interpretation

Our motivation for extending a constraint-based grammar formalism with numerical preferences was the need for a metric to select certain solutions from a set of possibilities in the case of ambiguity. Once suitable criteria for selecting solutions have been identified, it seems natural to use them not only *after* all results have been found, but already during the search.

For applications where the set of solutions to select from remains small, this does not seem essential. But if the aim is to leave toy examples and to describe linguistic reality in appreciable breadth, it will get more and more difficult to hold the balance between under- and overgeneration. Since in the case of undergeneration preference information will be of no help, in general, one will have to deal with hybrid descriptions, where the role of the symbolic, rule-based skeleton is not so much to identify the correct analysis from the incorrect, but to distinguish completely pointless analyses from alternatives, which – although most of them still incorrect – are somewhat less devious. In such a setting, we would run the risk of intractability due to massive ambiguity and overgeneration, if we exploit only symbolic constraints during processing.

Even if we restrict ourselves to limited domains, where it is possible to achieve satisfactory coverage without too much overgeneration by carefully handcrafted grammars, we will often still have the problem of defective input, such as typed text or the output of speech recognition etc. Given sufficiently robust mechanisms to correct or interpret the input, a complete enumeration of all possible readings is usually not feasible.[7] In cases like this, we cannot hope to find good solutions using naive search

[6]It is straightforward to give predicate definitions for which the estimation of the failure probability is an undecidable problem.

[7]The German VerbMobil project has defined word lattices as an interface between speech recognition and parsing. Typical results from speech recognizers based on current technology are graphs with an average branching factor from 5 up to 40 edges per node, which nevertheless often fail to include the correct reading.

strategies such as depth-first search. Instead, we have to apply strategies which satisfy as many as possible of the following requirements:

- Find the best solution efficiently
- Enumerate solutions in the order of quality (best first)
- Do not spend work on irrelevant subproblems
- Avoid repeated solution of identical subproblems

For each of these requirements, there are known methods or techniques for more specialized applications (such as context-free grammars or weighted graphs) that might contribute to a solution. The task is to combine them into a general procedural treatment of probabilistic CUF that scales well enough to be useful for real-world applications.

4.1 Searching for the best solution of a goal

Given a P-CUF program P and a goal G, a natural question to ask is that for the most likely proof of G from P. More exactly, among all constraints ϕ_i for which $P \models \phi_i \rightarrow G$ can be shown, we are looking for the ϕ_i for which this fact is derived most probably by our stochastic prover. This question is analogous to the notion of *Viterbi-paths* in hidden markov models.

The simplest approach is to use the standard CUF implementation to produce all solutions and to annotate each solution with the product of all probabilities of rules involved. The set of solutions can then be sorted according to their probabilities. This approach fails in cases where the enumeration of solutions does not terminate in finite time due to the depth-first search regime of the CUF interpreter, and even in cases where it works, much processing time will be wasted for the search for non-optimal solutions.

We can exploit the probabilistic annotations to guide the search in several ways. In each situation where the search tree branches, we can assess the local probabilities to search the best branch first. However, this local optimization will not guarantee that we will find the best solution first, since during the construction of a proof tree, we do not know in what ways a local decision influences the probability of the complete tree under construction.

We can however improve the strategy by precomputing at compile time, for each predicate in a CUF program, the minimal cost (maximal probability) of a proof tree for this predicate. Using this information, we can take the minimal costs of the unresolved goals on the stack as an additional criterion to guide the search. We can even exploit constraints already known at compile time to exclude inconsistent proof trees and obtain more fine grained distinctions. However, since the validity of a proof tree depends on information concerning variable bindings and constraints which are only known at runtime, we can not hope to precompute exact costs in all cases, only best-case estimates. This propagation of estimated costs will help to find a rather goal-directed search strategy, which is still based on local reordering of branches.

As long as we are interested only in the best solution, we can always take the quality of the best solution found so far to cut off branches of the search tree on which the sum of already encountered costs plus the estimation of remaining costs are already higher. First practical investigations have shown that this pruning technique, in combination with a relatively accurate estimation of remaining costs, will lead to considerable reduction in search effort, comparable to the related technique of α-β-pruning in the search of game trees [KM75].

When processing grammars with many sources of ambiguity, the techniques sketched above will not suffice to avoid exponential overhead during search. In such cases, we can try to reorder the branches of the search tree globally, i.e. apply a global best-first strategy. This has the advantage that we are guaranteed to produce the results in the order of their quality, and if we need only the best solution, we can abort the search after the first one has been found. A certain disadvantage of global best-first search is the need to maintain different sets of constraints for several active branches of the search tree simultaneously. Furthermore, a breadth-first or best-first traversal of the search tree does not guarantee that equivalent goals appearing in different branches of the tree are solved only once. But a scheme that avoids recomputation in such cases is mandatory if one wants to avoid exponential overhead and get the benefits of chart parsing.

Ways to generalize Earley deduction to be used in a general CLP framework have been described in [Dör93]. A quite similar scheme is also presented in [Joh93], in a somewhat more abstract fashion. These algorithms can be extended to do a global best-first search. In both cases, there is a global agenda of partial results that require further processing. Instead of using lists or the goal stack of the implementation language to maintain the agenda, we can employ a priority queue, i.e. a data structure where entries are entered together with a numerical key and can be accessed in the order of increasing keys.

A natural choice for the key to use for the entry of a derived rule instance `H :- B` is the quality of the best solution of the original goal that can be derived using this rule. This criterion has to take into account the costs of the rules already used when predicting `H` triggered by the original goal `G`, and also the estimated rest costs of all goals that are introduced on the path from `G` to `H`. It is intuitively clear that ordering processing steps according to this key will lead to an enumeration of solutions in the order of quality. Details of the implementation are currently under investigation and will lead to an extended CUF implementation in the near future.

5 Training of probabilistic parameters

As already stated above, an important advantage of our way to extend CUF is the ability to optimize the probabilistic parameters of a program, given a set of queries.

5.1 Supervised vs. unsupervised learning

In principle, there are two modes of "learning" these parameters: unsupervised and supervised. In the unsupervised case, we only give the program $P(\pi)$, where π is some initial seed of parameter values, and a set of queries $G_1, \ldots, G_n$, which can be collapsed to one conjunctive query G[8]. We are searching for a set of parameter values $\hat{\pi}$, for which the probability of producing a valid proof for G gets maximal.

In the supervised case, we do not only have a set of queries, but we also have a set of constraints on the solutions we are interested in. E.g. we do not only want to maximize the probability that a set of sentences is analyzed by a grammar, but we want to give (partial) analyses for the sentences and try to maximize the probability that the grammar assigns analyses compatible to the given ones. A nice property of our approach is the fact that we do not need an extra training algorithm for the supervised learning of parameters. We only add the constraints we want to impose on the solutions to the query G and maximize the probability to prove G, as in the unsupervised case.

5.2 Applying the EM-algorithm

For the estimation of parameters for Markov-models, it is standard practice to apply the EM-algorithm [Bau71, DLR77] to improve an initial estimation of the model iteratively. We can do the same for our application, by computing expected values for the counts of rule applications, given some initial model $P(\pi)$. For a given Program P with initial parameters π, and a given query G, we compute the expected number of times a certain rule is used in a proof of G. If we take the estimated frequency of a rule relative to the estimated frequency of all rules for the same predicate as a new estimate of the rule's probability, we get an improved set of parameter values. For this reestimation, it is important that we take the expected number of rule invocations under the condition that there is a succesful proof of G, i.e. in general we have to normalize the counts by the factor $1 - p(\textsc{False} \mid G)$

In the simple case where there is only one result for a goal, the estimated frequency for the used rules can be trivially counted. In the case of ambiguity, we can take the probability of each of the results as a factor to compute a weighted average of rule applications.

Specifically, assume that P contains I predicates, where the ith predicate has J_i clauses and let $p_{i,j}$ be the probability of the jth clause of the ith predicate. Assume

[8]It is important to assume that a conjunction of goals is always seen as a *multiset*, i.e. we do not make use of the fact that conjunction is logically an idempotent operation.

furthermore that there are K distinct proof trees $t_1, \ldots, t_K$ for the goal G and let $c_{i,j,k}$ the number of occurences of the jth clause of the ith predicate in the kth proof tree.

Then the probability to generate a valid proof tree for G is

$$p(\textsc{Consistent} \mid G) = \sum_{k=1}^{K} p(t_k \mid G) = \sum_{k=1}^{K} \prod_{i=1}^{I} \prod_{j=1}^{J_i} p_{i,j}^{c_{i,j,k}}$$

Given initial parameters $\pi = (p_{i,j})$, we estimate the optimized values $\pi' = (p'_{i,j})$ as

$$p'_{i,j} = p_{i,j} \frac{\frac{\partial p(\textsc{Consistent}|G)}{\partial p_{i,j}}}{\sum_{j=1}^{J_i} \frac{\partial p(\textsc{Consistent}|G)}{\partial p_{i,j}}}$$

This reestimation is guaranteed to converge towards an optimum, which is a (local) maximum likelihood estimate of the parameter values. Although we are not guaranteed to find a global maximum this way, we can assume that this is the case if the grammar provides enough information to guide the training procedure away from unreasonable analyses.

It is an interesting fact that this training procedure, while trying to maximize the probability of a set of training examples, will also try to reduce the probability of inconsistent proofs. This can lead to somewhat surprising results if the probability of failure differs significantly for different parameter values. In such cases, the ratios of probabilities for various outcomes with the optimized parameters will not reflect the respective ratios in the training data.

To make this point clearer consider the following program:

```
s(X) ←1 p(X), q(X).


p(a)←pa .
p(b)←pb .
p(c)←pc .


q(a)←qa .
q(b)←qb .
q(c)←qc .
```

and the training data

```
?- s(a). (9 times)
?- s(b). ( once )
```

The training algorithm will try to maximize $(p_a * q_a)^9 * (p_b * q_b)$ under the constraint that the p_i and q_i sum to 1 respectively. The optimal solution (without smoothing) to this will be $p_a = q_a = 0.9, p_b = q_b = 0.1, p_c = q_c = 0$, and that is what the training procedure sketched above will converge to. But this leads to a probability ratio between s(a) and s(b) of 81:1, instead of 9:1, as might perhaps be expected.

One could argue that the right thing to maximize is $(p_a * q_a/(p_a * q_a + p_b * q_b + p_c * q_c))^9 * (p_b * q_b/(p_a * q_a + p_b * q_b + p_c * q_c))$, i.e. maximize the probability of the training data relative to the cases where a consistent proof tree is built. This leads to different optimal values, e.g. $p_a = q_a = 0.75, p_b = q_b = 0.25, p_c = q_c = 0$, which reproduce the right probability ratio for s. Unfortunately, this solution is not unique, any values with $p_a/p_b * q_a/q_b = 9$ will do. I currently do not know of a training procedure that finds an optimal assignment of probabilities according to this criterion in the general case.

One can look at this problem as follows: Our model allows both predicates `p` and `q` to influence the probability distribution of the common variable `X`. Hence multiplying the probabilities leads to a model which, informally speaking, counts the evidence given in the training data twice. One way of coping with the situation heuristically would be to take the (suitably renormalized) geometric mean instead of the product in such cases, but this would introduce additional complications into the processing strategy and it is not clear if and under what conditions the EM-algorithm remains applicable.

Although this is a problem in our method that requires further attention, we assume that our current treatment nevertheless helps to find good approximations for practical applications. Further work on this point should be guided by investigations based on real data.

5.2.1 Approximative Training under Massive Ambiguity

The processing techniques needed for finding best solutions and for training parameters are quite different in their use of the weights given in the grammatical description. When looking for optimal or near optimal solutions, we can exploit the annotations of the solutions found so far in order to cut down the search space considerably. Once we know that a certain partial result cannot contribute to an (near) optimal solution, we can ignore this result completely. On the other hand, for the training of parameters according to the EM-algorithm, we cannot ignore such results, since the rules used for suboptimal solutions still have to be used for obtaining an updated estimate of the weights. Hence, we cannot fully exploit the optimizations sketched above in parameter training.

For simplified tasks, such as parsing and parameter estimation for SCFGs, this is not an essential difficulty. Even in cases where the number of possible analyses for a given grammar G and training text T grows exponentially with the size of T, it is still possible to obtain exact estimates for new weights in polynomial time by the use of the inside-outside algorithm [LY90]. This is a generalization of the well-known forward-backward algorithm for estimating parameters in Hidden Markov Models and relies on the fact that the relevant information concerning each rule invocation (i.e. probability and first partial derivative relative to each parameter) can be computed in cubic time. This simplification relies ultimately on the context-free nature of the formalism.

In a more general setting, where structural details of an intermediate result might influence the applicability of this result in arbitrary ways, we cannot hope to find a similar simplification. In contrary, it is relatively easy to find definitions P and a goal G such that the computation of the relative probability of a certain solution for G from P is a $\#P$-complete problem [GJ79].[9] But since the application of the EM-algorithm is itself only a step in an iterative approximation scheme, we do not really need full accuracy. Instead, we expect it to be satisfactory if we are able to estimate significant contributions of rules to the derivation of the training examples.

Given a solution algorithm capable to enumerate the solutions in the order of their quality, we can run this algorithm until further solutions get bad enough to be ignored as insignificant. This is a generalization of an idea that is sometimes also called "Viterbi Training", in which only the most likely solution (called Viterbi-solution) is considered in the reestimation process. Instead of taking only one solution, we are free to collect evidence from alternative solutions as long as the results justify further computation.

5.2.2 Smoothing Techniques

A difficulty with the estimation of probabilistic parameters from training examples lies in the fact that a maximum likelihood estimation based on a training set is often not what is needed, since the purpose of the grammar is to generalize to unseen examples. E.g. if a rule happens not to be useful to describe the examples, or if other rules describe them more easily, the estimated probability of this rule will converge to 0 during training, even if the rule is necessary for examples that happen to be missing from the training set. This is aggravated by the fact that our formalism supports the statement of arbitrarily fine-grained and high-dimensional probability distributions, for which parameters can not be estimated by simple-minded techniques. Here we will need some additional techniques to "smooth" the parameter values found during

[9] Consider e.g the following definition, where annotations on the predicates `match/2` and `select_match/3` have been omitted, since they do not influence the relative probability of the various solutions.

```
match([],[]).
match([F|R],L) :- select_match(F,L,R2), match(R,R2).

select_match(F1,[F2|R],R) :- pair(F1,F2).
select_match(F,[F2|R2],[F2|R3]) :- select_match(F,R2,R3).

pair(I,J) ←p_{I,J} .
...
```

Given a query `?- match([1,...,n],[1,...,n])` there are $n!$ different proof trees, according to the $n!$ permutations of a list of length n. Summing over the probabilities of all proof trees is equivalent to the computation of the permanent of the matrix $(p_{i,j})$, which is known to be $\#P$-complete.

training, where the amount of smoothing should be dependent on the number of parameters, the number of training examples, the diversity of the training set, etc.

In work on speech recognition and statistical language modeling, a rapidly growing set of methods have been proposed to deal with the problem of sparsity of training data. A widespread practice in this framework is to assume that useful approximations of parameter values can be found by mixing models of various granularities. E.g. in the framework of n-gram modeling, where even for quite modest sizes of n similar problems arise, one can easily verify that a mixture of some models with n in the range 1 to 3 gives better prediction of unseen events than each of the individual models. This can be explained by the observation that for small n the model cannot express enough details to be accurate, whereas for larger values of n, many of the events that are structurally possible do not appear in the training data, hence their probability is underestimated. A linear mixture of various such models is a simple, yet effective way to compensate the weaknesses of the individual models.

Whereas the parameters in the individual models are set to their maximum-likelihood values (computed as relative frequencies), the mixing factors cannot be ascertained in this way, since this would always favor the most detailed of the models. Instead, one can employ unseen cross-validation data to find the optimal mixing factor. If the amount of available data is very limited, one can exploit the given data more efficiently by the leaving-one-out (sometimes also called jackknife) technique [NEK94], where each of the training samples is used as a singleton cross-validation set for the rest of the samples, in order to get a good estimate for the necessary mixing factors. The mixing is finally applied to the values obtained from the full training set. Whereas the latter technique exploits the training data most effectively, this has its price a considerable computational overhead, which might not be feasible under all circumstances.

It should be noted that these techniques can be easily adapted to constraint based descriptions in CUF. Assume, for instance, that we want to model a binary relation `r` between some linguistic entities, such as stems of verbs and head nouns of the associated objects. Given enough training material, we could enumerate a set of relevant entries in the form

$\mathtt{r(v_1,n_1)} \leftarrow_{p_{1,1}}$

$\vdots$

$\mathtt{r(v_k,n_m)} \leftarrow_{p_{k,m}}$

Of course such a list will get impractically large, and it is impossible to get enough training material to obtain reliable estimates for all the probabilities $p_{i,j}$. It has been shown ([PTL93, Roo94]) that one can get estimates that generalize better to unseen examples if one assumes that the selection of a verb-object pair depends on a *latent class* c, such that the probability $p(i,j)$ is decomposed into a product of the probabilities $p_c(c)p_v(i|c)p_n(j|c)$, where p_c models the probability that a pair from class c is found, whereas p_v and p_n model the probabilities for verbs and nouns, given the class. The classification could be linguistically motivated, based on taxonomic representation of the application domain, or it can be derived automatically from training data using suitable clustering algorithms. Once we have the classification, we can write our definition as

$\texttt{r(V,N)} \leftarrow_{p_c(1)} \texttt{v\_1(V), n\_1(N)}$

$\vdots$

$\texttt{r(V,N)} \leftarrow_{p_c(C)} \texttt{v\_C(V), n\_C(N)}$

$\texttt{v\_1(v}_1\texttt{)} \leftarrow_{p_v(1|1)}$

$\vdots$

$\texttt{v\_C(v}_j\texttt{)} \leftarrow_{p_v(j|C)}$

$\texttt{n\_1(n}_1\texttt{)} \leftarrow_{p_n(1|1)}$

$\vdots$

$\texttt{n\_C(n}_j\texttt{)} \leftarrow_{p_n(j|C)}$

In this encoding[10], we do not have to enumerate all possible pairs of verbs and nouns, but for each verb and each noun, we have to list only classes that can produce this word with a significant probability.

The purpose of this simplified example is to show that smoothing techniques described elsewhere can easily be applied within the framework of probabilistic CUF. However, since CUF supports more abstract, feature based descriptions, it might be more natural to decompose a probability distribution on a lexical domain into a combination of probability distributions on a set of features.

[10]The introduction of auxiliary predicates $\texttt{n}_i$ and $\texttt{v}_i$ is a technical trick to enable the specification of conditional probabilities. Given the notation introduced in Footnote 5, we could also write

$\texttt{r(V,N)} \leftarrow_1 \texttt{c(C), v(V,|C), n(N,|C)}$

$\texttt{c(1)} \leftarrow_{p_c(1)}$

$\vdots$

$\texttt{c(C)} \leftarrow_{p_c(C)}$

$\texttt{v(v}_1\texttt{,1)} \leftarrow_{p_v(1|1)}$

$\vdots$

$\texttt{v(v}_j\texttt{,C)} \leftarrow_{p_v(j|C)}$

$\texttt{n(n}_1\texttt{,1)} \leftarrow_{p_n(1|1)}$

$\vdots$

$\texttt{n(n}_j\texttt{,C)} \leftarrow_{p_n(j|C)}$

6 Conclusion

We have presented a simple, yet powerful extension to CUF that allows for the specification of probabilistic preferences. The extended formalism supports the formulation of hybrid descriptions where symbolic constraints and probabilistic preferences complement each other to form a much more effective synthesis than what would be possible in one of the paradigms alone.

One of the main advantages of the extended formalism is the possibility to acquire probabilistic parameters automatically from training examples. Hence, the grammar writer can concentrate on the creative parts of the work, namely to provide a framework that introduces the essential descriptive devices and reflects the principal regularities of the domain, without having to specify all the details by hand. However, the manner in which probabilities and constraints interact in our framework is not the only possible one, and there is need for more theoretical and practical investigation in order to decide whether the scheme proposed here can serve as a basis for large scale grammar development.

References

[Bau71] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process. In Shisha and Qved, editors, *Inequalities III*, pages 1–8. Academic Press, New York, 1971.

[BC93] Ted Briscoe and John Carroll. Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, 19(1), 1993.

[BJL+92] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings, DARPA Speech and Natural Language Workshop*. 1992.

[Bre93] Chris Brew. Adding preferences to CUF. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I, DYANA-2 deliverable R1.2.A*. ESPRIT, Basic Research Project 6852, July 1993.

[BT73] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, 1973.

[Cho81] Noam Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, 1981.

[CM93] Kenneth W. Church and Robert L. Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1), 1993.

[DD93] Jochen Dörre and Michael Dorna. CUF — a formalism for linguistic knowledge representation. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I, DYANA-2 deliverable R1.2.A.* ESPRIT, Basic Research Project 6852, July 1993.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B. 39, 1977.

[Dör93] Jochen Dörre. Generalizing Earley deduction for constraint-based grammars. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I, DYANA-2 deliverable R1.2.A*. ESPRIT, Basic Research Project 6852, July 1993.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, Cal., 1979.

[HR93] Donald M. Hindle and Mats Rooth. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1), 1993.

[Jel90] Fred Jeline. Self-organized language modeling for speech recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*, pages 450–506. Morgan Kaufmann, San Mateo, CA, 1990.

[Joh93] Mark Johnson. Memoization in constraint logic programming. ms., Department of Cognitive Science, Brown University, 1993. Presented at the 1st International Conference on Constraint Programming, Newport, Rhode Island; to appear in the Proceedings.

[KM75] D. E. Knuth and R. W. Moore. An analysis of alpha-beta-pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[LY90] K. Lari and S.J. Young. The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech & Language*, 4(1):35–56, 1990.

[NEK94] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilisitic dependences in stochastic language modeling. *Computer Speech & Language*, 8(1):1–38, 1994.

[PS87] Carl J. Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Vol.1 Fundamentals*, volume 13 of *CSLI Lecture Notes*. CSLI/Chicago Univ. Press, Stanford, 1987.

[PTL93] Fernando C. N. Pereira, Naftali Z. Tishby, and Lillian Lee. Distributional clustering of English words. In *Proceedings of the 31th ACL*, pages 183–190, 1993.

[Res92] Philip Resnik. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings COLING 92, Nantes*, 1992.

[Roo94] Mats Rooth. Two-dimensional clusters in grammatical relations. unpublished manuscript, 1994.

[Sch92] Yves Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proceedings COLING 92, Nantes*, 1992.

[Sch94] Ludwig A. Schmid. Parsing word graphs using a linguistic grammar and a statistical language model. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 2–41–2–44. Adelaide, 1994.

[Wu90] Dekai Wu. Probabilistic integration of syntax and semantics. In *Proceedings of COLING 13th, Helsinki*, pages 413–418, 1990.