EXPRESSING PROBABILISTIC CONTEXT-FREE GRAMMARS IN THE RELAXED UNIFICATION FORMALISM

Tony Abou-Assaleh

Nick Cercone

Vlado Kešelj

Faculty of Computer Science, Dalhousie University,
Halifax, Nova Scotia, B3H 1W5, Canada
{taa,nick,vlado}@cs.dal.ca

The Theory of Relaxed Unification is a newly proposed theory that extends the power of classical unification. The theory relaxes the rigid constraints of requiring a perfect match between the terms being unified to allow multi-valued attributes. The Relaxed Unification Inference System is an implementation of an inference engine and an interpreter, which uses the relaxed unification mechanism in resolving the rules. We give an overview of the System's capabilities and demonstrate how it can be used to express Probabilistic Context-Free Grammars.

Key words: Probabilistic Context-Free Grammar, Definite Clause Grammar, Theory of Relaxed Unification

1. INTRODUCTION

The classical unification function takes two terms as input and produces a boolean value indicating whether the unification can be performed successfully. In case of a result of true, the function also returns a substitution that unifies these two terms. The unification fails if the same feature is assigned different values in the objects being unified. This process places rigid constraints on the data requiring it to be correct and consistent. Since real-world data is seldom perfect, classical unification fails when it encounters the slightest error. Erroneous data often contains enough information that one can exploit to overcome the errors. In other cases, it is possible to draw approximate or uncertain conclusions.

Relaxed unification provides a method for extracting information from imperfect data. To achieve this functionality, we relax the strict true/false result of classical unification and replace it by a real number in the range $(0, 1]$ that indicates the correctness of the unification. A correctness value of 1 would represent a success under the classical unification; any other value would represent a failure. Relaxed unification does not includes a notion of failure; the unification always succeeds and returns a substitution.

We show how Probabilistic Context-Free Grammars (PCFG) can be naturally expressed in the Relaxed Unification formalism, and how to construct a correctness function that outputs the probabilities of the PCFG parses.

2. RELATED WORK

The Theory of Unification is well formalized and understood. Robinson [Robinson1965] was the first to introduce unification in 1965. He set forth the basic definitions of the theory, presented a straightforward recursive algorithm for unifying two terms, and proved some theorems and lemmas that are fundamental to the Theory of Unification. Knight [Knight1989] provided an extensive survey of representations, algorithms, and applications of unification. More recently, Keselj [Kešelj and Cercone] introduced an efficient general-purpose graph unification algorithm and discussed the low-level details of its implementation.

Although classical unification has witnessed a great success, its assumption that the data is absolutely true isolated it from any real-world problems that involve uncertainty.

Relaxed unification [Abou-Assaleh and Cercone2002, Abou-Assaleh and Cercone] was the first attempt to formalize the concept of unifying sets of values. Further work has led to proposing the Theory of Relaxed Unification [Abou-Assaleh2003], which is the first coherent and complete formalization of the theory along with an implementation of a relaxed unification system.

PCFG are widely used in natural language processing, especially to help resolve language ambiguities that appear when Context-Free Grammars are used. Charniak [Charniak1993] has an excellent overview of PCFG, their algorithms, and their applications.

3. RELAXED UNIFICATION INFERENCE SYSTEM

We implemented an inference system based on relaxed unification (*RU System*). The RU System allows the user to assert relaxed predicates and execute relaxed queries. Relaxed predicates and queries are similar to the first-order logic predicates that are used in logic programming languages such as PROLOG with the exception that the terms in relaxed predicates and queries are relaxed terms. In addition, each relaxed predicate is assigned a degree of belief β. The particular assignment of β's depends on the probabilistic models that the user wants to impose on the knowledgebase.

The RU System is composed of two modules: the interpreter (*RU Interpreter*) and the inference engine (*RU Engine*). The interpreter is the interface between the user and the inference engine.

RU Engine is an inference engine that is based on relaxed unification. It differs from a classical inference engine in a number of ways. Since unification always succeeds, there is no notion of backtracking. When a query is executed, all possibilities are explored and a list of possible answers is created. Of course, not all the possibilities have semantic relevance to the user's query. A correctness function is used to compute the relevance of predicates after unifying them with the query. The user sets a threshold to prune branches of the search tree that have low relevance or low accuracy.

The query is a relaxed term, which usually contains a variable as a placeholder for the information we are querying the knowledgebase for. A query without variables (or empty sets) can be used to verify the consistency of the query term with the knowledgebase. When a query is executed, an implicit search tree is built. The tree begins with a branch for each predicate in the knowledge base. If the predicate is a fact then it is unified with the query. Otherwise the query is unified with the head giving a unifier σ, a subquery is generated with the first term of the tail as head and the rest of the tail as tail, and σ is applied to the head of the subquery before executing it. When a subquery returns, its substitution is unified with the head to propagate information. Thus, information is passed from the query term, to the head of a predicate, to the tail of the predicate, and then back to the query term.

4. EXPRESSING PROBABILISTIC CONTEXT-FREE GRAMMARS

4.1. Probabilistic Context-Free Grammars

PCFG is a Context-Free Grammar (CFG) that associates a probability with each of its rules. A context-free grammar is defined by a set of terminals, a set of nonterminals, a start symbol, and a set of production rules that expand a nonterminal symbol into a sequence of terminal and nonterminal symbols. PCFG generates the same set of parse trees for a text that the corresponding CFG generates. However, PCFG assigns a probability to each parse tree. Probabilities of the parse trees are computed by multiplying the probabilities of the rules that are used in generating the tree. A rule's probability is contributed to a tree's probability for as many times as the rule is used in generating the tree. Readers interested in a formal definition of PCFG are referred to [Charniak1993].

4.2. Probabilistic Definite Clause Grammars

Definite Clause Grammars (DCGs) [Pereira and Warren1980] is a notation that is developed in the natural language processing community. This notation facilitates writing CFGs in PROLOG, which automatically converts the DCG clauses into PROLOG's horn clauses. The general form of a DCG clause is:

```
HEAD --> BODY.
```

where both `HEAD` and `BODY` are comma-separated lists of terminals and nonterminals, which may also include PROLOG terms. PCFGs can be expressed in DCGs using these additional PROLOG terms, as demonstrated in the example below.

Example 1. Expressing PCFG in DCG

This example illustrates a known method for expressing a PCFG using a DCG and shows how it is translated into PROLOG clauses.

Consider the following PCFG.

```
NP  →  D N   /0.8
NP  →  N     /0.2
D   →  a     /0.6
D   →  the   /0.4
N   →  fox   /1.0
```

It is expressed in DCG as:

```
np(X) --> d(Y), n(Z), {X is 0.8*Y*Z}.
np(X) --> n(Y),       {X is 0.2*Y}.
d(X)  --> [a],        {X is 0.6}.
d(X)  --> [the],      {X is 0.4}.
n(X)  --> [fox],      {X is 1.0}.
```

PROLOG converts these DCG clauses into the PROLOG clauses:

```
np(X, S0, S) :- d(Y, S0, S1), n(Z, S1, S), X is 0.8*Y*Z.
np(X, S0, S) :- n(Y, S1, S), X is 0.2*Y.
d(X, S0, S) :- 'C'(S0, a, S), X is 0.6.
d(X, S0, S) :- 'C'(S0, the, S), X is 0.4.
n(X, S0, S) :- 'C'(S0, fox, S), X is 1.0.
```

where `'C'` is defined as,

```
'C'([X|S], X, S).
```

When a query is executed, the probabilities are automatically calculated and displayed to the user. Two sample queries and their results are show below.

```
?- np(Prob, [a, fox], []).
Prob = 0.48

?= np(Prob, [fox], []).
Prob = 0.2
```

While expressing CFGs as DCGs is straightforward and natural, expressing PCFGs as DCGs adds the complication of having to specify the product of probabilities for each rule as opposed to simply stating the probabilities. Therefore, in the RU System, we amend the DCGs to enable specifying a probability for each rule while preserving the simplicity of expressing the CFGs rules in DCGs. We simply prefix each DCG clause with a probability, and then use a correctness function to calculate the probability of a given query. Our approach provides the user with the added flexibility of specifying any correctness function that is desirable for a particular grammar or task. For instance, the correctness function could calculate the probability of a clause by averaging the probabilities of its terms and then multiplying the average by the probability of the rule itself. However, to express PCFGs accurately, the correctness function must compute the product of the probabilities of terms in the clauses. The next subsection uses an example to demonstrate how PCFGs can be expressed in the RU System using our simplified notation for probabilities in DCGs.

4.3. Example

We detail the process of parsing a sentence using a PCFG. We begin by motivating our task to use PCFG as opposed to CFG. Next, we show how the PCFG is presented to the RU System using a notation based on DCGs. We conclude by presenting the relaxed predicates that the system automatically generates and show a sample query execution.

Table 1. Sample Context Free Grammar

`S` $\rightarrow$ `NP VP`	`VP` $\rightarrow$ `V PP`	`D` $\rightarrow$ `an`
`NP` $\rightarrow$ `N`	`PP` $\rightarrow$ `P NP`	`V` $\rightarrow$ `like`
`NP` $\rightarrow$ `N N`	`N` $\rightarrow$ `time`	`V` $\rightarrow$ `flies`
`NP` $\rightarrow$ `D N`	`N` $\rightarrow$ `arrow`	`P` $\rightarrow$ `like`
`VP` $\rightarrow$ `V NP`	`N` $\rightarrow$ `flies`	

We wish to parse the sentence *time flies like an arrow* [Kešelj and Schuurmans2001]. According to the CFG listed in table 1, there are two possible parses. The first parse (see figure 1) interprets the sentence as having a special type of flies that like an arrow, i.e., *time flies* is a noun phrase and *like* is a verb. The second parse (see figure 2) implies that there is a likelihood between the way time and an arrow fly, i.e., *flies* is a verb and *like* is a proposition. It is conclusive from this example that a more powerful formalism than CFG is required in order to deal with ambiguous sentences. PCFG add the ability to distinguish between the appropriateness of a particular parse of a sentence based on a probabilistic model of the language constructs.

We wish to parse the sentence *time flies like an arrow* [Kešelj and Schuurmans2001]. According to the CFG listed in table 1, there are two possible parses. The first parse (see figure 1) implies that there is a likelihood between the way time and an arrow fly, i.e., *flies* is a verb and *like* is a proposition. The second parse (see figure 2) interprets the sentence as

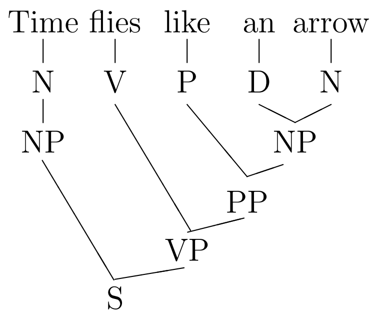

Figure 1. Parse 1—a possible parse tree for *time flies like an arrow*

having a special type of flies that like an arrow, i.e., *time flies* is a noun phrase and *like* is a verb. It is conclusive from this example that a more powerful formalism than CFG is required in order to deal with ambiguous sentences. PCFG add the ability to distinguish between the appropriateness of a particular parse of a sentence based on a probabilistic model of the language constructs.

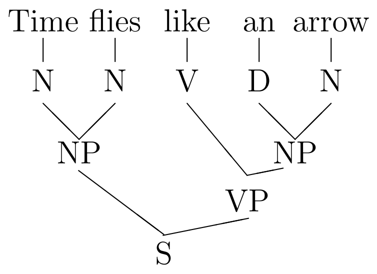

Figure 2. Parse 2—a possible parse tree for *time flies like an arrow*

Table 2. Sample Probabilistic Context-Free Grammar

S → NP VP	/1.0	VP → V PP	/0.5	D → an	/1.0
NP → N	/0.4	PP → P NP	/1.0	V → like	/0.3
NP → N N	/0.2	N → time	/0.5	V → flies	/0.7
NP → D N	/0.4	N → arrow	/0.3	P → like	/1.0
VP → V NP	/0.5	N → flies	/0.2		

We revisit the same sentence after transforming the CFG in table 1 into the PCFG listed in table 2 by assigning probabilities to the production rules. PCFGs allow us to compute the probability of the parse tree by multiplying the probabilities of the rules that are used in constructing the tree. Thus,

```
P(parse 1) = P(N → time)P(V → flies)P(P → like)P(D → an)
               P(N → arrow)P(NP → N)P(NP → D N)P(PP → P NP)
               P(VP → V PP)P(S → NP VP)
             = 0.0084

P(parse 2) = P(N → time)P(N → flies)P(V → like)P(D → an)
               P(N → arrow)P(NP → N N)P(NP → D N)P(VP → V NP)
               P(S → NP VP)
             = 0.00036
```

Since the probability of parse 1 (0.0084) is greater than the probability of parse 2 (0.00036), we conclude that parse 1 is the more likely correct interpretation of the sentence *time flies like an arrow*. Of course, one still has to deal with assigning the proper probabilities for each rule. Readers interested in more information regarding the assignment of probabilities in PCFGs are referred to [Charniak1993].

Table 3. Expressing PCFG in RU System Using DCG Notation

`(1.0) s --> np, vp.`	`(0.5) vp --> v, pp.`	`(1.0) d --> an.`
`(0.4) np --> n.`	`(1.0) pp --> p, np.`	`(0.3) v --> like.`
`(0.2) np --> n, n.`	`(0.5) n --> time.`	`(0.7) v --> flies.`
`(0.4) np --> d, n.`	`(0.3) n --> arrow.`	`(1.0) p --> like.`
`(0.5) vp --> v, np.`	`(0.2) n --> flies.`	

Table 4. PCFG Expressed in Relaxed Predicates

```
(1.0) s(X0, X, s(T1, T2)) :- np(X0, X1, T1), vp(X1, X, T2).
(0.4) np(X0, X, np(T)) :- n(X0, X, T).
(0.2) np(X0, X, np(T1, T2)) :- n(X0, X1, T1), n(X1, X, T2).
(0.4) np(X0, X, np(T1, T2)) :- d(X0, X1, T1), n(X1, X, T2).
(0.5) vp(X0, X, vp(T1, T2)) :- v(X0, X1, T1), np(X1, X, T2).
(0.5) vp(X0, X, vp(T1, T2)) :- v(X0, X1, T1), pp(X1, X, T2).
(1.0) pp(X0, X, pp(T1, T2)) :- p(X0, X1, T1), np(X1, X, T2).
(0.5) n(X0, X, n(time)) :- 'C'(X0, time, X).
(0.3) n(X0, X, n(arrow)) :- 'C'(X0, arrow, X).
(0.2) n(X0, X, n(flies)) :- 'C'(X0, flies, X).
(1.0) d(X0, X, d(an)) :- 'C'(X0, an, X).
(0.3) v(X0, X, v(like)) :- 'C'(X0, like, X).
(0.7) v(X0, X, v(flies)) :- 'C'(X0, flies, X).
(1.0) p(X0, X, p(like)) :- 'C'(X0, like, X).
(1.0) 'C'([X|Y],X,Y).
```

The next step is presenting the PCFG to the RU System. Table 3 lists the rules in the format that one may enter them into the RU System using a DCG-like notation. It is evident from the table that the DCG notation that we adapted has closer resemblance to the PCFG grammar notation than the DCG notation presented in example 1. Once the PCFG rules are entered into the RU System, the system converts them into relaxed predicated. Table 4 lists the relaxed predicates that are generated from the rules in Table 3. In addition to the probabilities associated with each possible parse tree, the system will also return the parse tree itself in the third argument of a requested query. To parse our sentence, the following query is executed:

```
?- s([time, flies, like, an, arrow], [], T).
(0.0084) T = s(np(n(time)), vp(v(flies), pp(p(like), np(d(an),
    n(arrow)))))
(0.00036) T = s(np(n(time), n(flies)), vp(v(like), np(d(an),
    n(arrow))))
```

We note that only two results are shown. Since relaxed unification always succeeds, there are other parses that are erroneous. These additional parses are not useful in our example. Therefore, we set the pruning threshold to allow only classical unification. The usefulness of such additional parses emerges when a grammar cannot generate any parse trees for a given sentence.

5. CONCLUSION

We present the RU System—an implementation of a unification engine based on the Theory of Relaxed Unification. We show that Probabilistic Context-Free Grammars (PCFGs) can be expressed in the Relaxed Unification Formalism, and illustrate this process using a detailed example. The RU System accepts a DCG-like notation for inputting PCFG rules. We show the final internal representation of the PCFG within the system. The main advantage of using the RU System to process PCFGs is the simplicity of the notation, the automatic generation of all possible parse trees for a given sentence, and the automatic computation of the probabilities associated with each one of these trees.

There is evidence in the Question Answering domain that the use of a unification based grammar with a relaxation of the classical unification is beneficial [Harabagiu et al.2000]. We plan to explore the appropriateness of using relaxed unification with the Head-driven Phrase Structure Grammar (HPSG). We intend to employ a relaxed unification based HPSG parser and answering engine to participate in the Question Answering Track of the Text REtrieval Conference [TRE2003].

ACKNOWLEDGMENTS

The financial support for this work was provided in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) *PGS B - 266950 - 2003* and Open Text Corporation.

REFERENCES

[Abou-Assaleh and Cercone] T. Abou-Assaleh and N. Cercone. Relaxed unification—proposal. *Appl. Math. Lett.* To appear.

[Abou-Assaleh and Cercone2002] T. Abou-Assaleh and N. Cercone. 2002. Relaxed unification—proposal. In R. Cohen and B. Spense, editors, *Lect. Notes Artif. Int.: 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002*. Springer.

[Abou-Assaleh2003] T. Abou-Assaleh. 2003. Theory of relaxed unification—proposal. Master's thesis, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[Charniak1993] E. Charniak. 1993. *Statisitical Language Learning.* The MIT Press.

[Harabagiu et al.2000] S. Harabagiu, D. Moldovan, M. Paşca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Gîrju, V. Rus, and P. Morărescu. 2000. Falcon: Boosting knowledge for answer engines. In *The Ninth Text REtrieval Conference (TREC 9)*.

[Kešelj and Cercone] V. Kešelj and N. Cercone. A graph unification machine for NL parsing. *Comput. Math. Appl.* To appear.

[Kešelj and Schuurmans2001]
V. Kešelj and D. Schuurmans. 2001. Course notes cs486/686:introduction to artificial intelligence.

[Knight1989] K. Knight. 1989. Unification: A multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124.

[Pereira and Warren1980] F.C.N. Pereira and D.H.D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artif. Intell.*, 13(3):231–278.

[Robinson1965] J.A. Robinson. 1965. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41.

[TRE2003] 2003. Text REtrieval Conference (TREC). WWW: http://trec.nist.gov/.