A Soft Unification Method for Robust Parsing

Hideto Tomabechi
Tokushima University
Dept. of Information Science and Intelligent Systems
Minami-Josanjima, Tokushima 770 JAPAN
tomabech@is.tokushima-u.ac.jp tomabech+@cs.cmu.edu

Abstract

Given natural language input is not guaranteed to be well-formed as specified in the grammar, robust natural language systems would need a flexible method for handling illformed input. Otherwise, the systems would simply reject the input sentence altogether even if the ungrammaticality in the input is trivial. This ability to handle illformed input is essential especially with noisy speech input. This paper proposes a methodology to enhance robustness through an enhancement in unification algorithm. The robust unification is achieved through partially delaying the evaluations of some of the graph paths. The paths that are delayed are: 1) paths exceeding the depths of predetermined thresholds; 2) paths containing the predetermined labels. In both variations, delayed unifications will be forced after a complete parse. If they simply succeed, then the sentence was simply well-formed. If they fail, it means the input was illformed. The parser can either report the point of failure to the human user so that the user can intervene and examine the location of illformedness for further processing, or the parser can call other modules (such as semantic inference modules) for further automatic processing.

1 Motivation

Given practical natural language input is not guaranteed to be well-formed as specified in the grammar prepared in the NL systems, the NL systems need to be enhanced through methodologies for handling illformed input. Otherwise, the systems would simply reject the input sentence altogether even if the ungrammaticality in the system is trivial. This problem is strongly felt when a natural language system uses so called "unification-based grammar" for constraint postulation. It is because in these systems, if there is any failure at any depth of feature-structure graphs, the whole unification fails and hence the parse candidate is immediately rejected. The solution to this problem is essential especially with spoken language input since 1) speech recognition may not be accurate and 2) the spoken language itself is often ungrammatical.

Consider the simple example augmented CFG grammar:

```
S     ==>  VP
(x0 sem) == (x1 sem)

VP  ==> N V N
(x1 agr) == (x2 agr)
(x0 sem relation) == (x2 sem)
(x0 sem arg1) == (x1 sem)
(x0 sem arg2) == (x3 sem)

V  --> "study"
(x0 agr num) == plural
(x0 sem) == *STUDY

V  --> "studies"
(x0 agr num) == singular
(x0 sem) == *STUDY

N  --> "John"
(x0 agr num) == singular
(x0 sem) == *JOHN

N --> "Lisp"
```

```
(x0 sem) == *LISP
```

With the grammar above, *John studies Lisp* is grammatical and *John study Lisp* is not. In the spoken language situation, the last phoneme *s* may be unrecognized (due to inaccuracy of the recognition device or due to weak pronunciation), and therefore unless a spoken-input NL system adopts some kind of error recovery scheme, the correct input *John studies Lisp* may be recognized as *John study Lisp* and the sentence will be rejected altogether.

In the currently adopted schemes of unification-based context-free parsing[1], a context-free rule is accompanied with so called "augmentation" which is a group of feature path equations (which are converted to graphs). Graph unifications are used to determine the validity of accepting the rule based on the augmentation and if the unification fails then the rule is rejected. Thus, by definition, one obvious observation we can make about unification-based grammar is that:

The input is rejected when unification-fails.

In other words, if we want to make a natural language systems to accept illformed input, our options are rather limited. At least two methods have been proposed to do this. namely:

1) To fool the grammar by making illformed input well-formed – unifications will naturally succeed since the input is transformed into a grammatical one.

One method proposed at CMU during 1988 ([Saito and Tomita, 1988]) was by introducing the confusion matrix and trying every possible completion of the rules by supplying possibly confused phonemes (with weights based upon phonemic closeness distance). This scheme fits nicely with Tomita's LR parsing algorithm; however, one problem with this scheme is that since we are essentially considering every possible alterations of phonemes the parser generally over-generates and when a grammar is sufficiently large, the ambiguity of the sentences may become massive. For example, [Tomabechi and Tomita, 1988] reported over 100 ambiguities for a parse of a short sentence *atamagaitai* ('I have a headache') using the scheme.

[1]Such as in [Tomita and Carbonell, 1987], [Morimoto, *et al*, 1990].

2) To combine partially created structures by using separate mechanisms – results of successful unifications are combined to create a meaningful larger constituent.

For example, [Kirtner and Lytinen, 1991] suggests a scheme to combine pieces of partially created feature structures using semantic knowledge. This method seems generally adoptable in unification-based grammar processing since in many cases a parse of a sentence would have created many rather large partial constituents before detecting an ungrammaticality and failing before creating an S. Of course if unifications did not create sufficient enough constituents to create a meaningful larger constituent this scheme will not work. In both Saito and Tomita and Kirtner and Lytinen, improvements are made to the parsing methodologies. Another possibility is to improve unification algorithms. Our observation about parsing failure was that "a rule is rejected when unification-fails", thus by making unification successul with illformed input, we can in effect create a robust parsing system. Thus, our third option which we are proposing in this paper is below:

3) To make unification less strict to process illformed input.

2 Our Scheme

Our principle for modifying graph unification to process illformed input is as follows:

- **Partially delay evaluations of graph paths.**
- **Perform delayed unifications after the parse.**

In order to delay evaluations of some of the paths, our method has two options:

- **Delay evaluation of graph path exceeding a predetermined threshold depth.**
- **Delay evaluation of graph path with predetermined labels.**

In the first method, we provide a predetermined depth of paths, and any unification after that depth is delayed until after the parse. With the

subsumption order assumption of constraints postulated in directed graph paths, a constraint is considered more specific if the paths leading to the constraint is longer. Thus, by delaying the evaluation of long paths, we are essentially ignoring the application of *very specific* constraints during parsing. In the second method, we designate certain path such as 'syntax head agreement' so that during a parse, these features are ignored (i.e., unifications of feature-structures for 'agreement' is delayed).

In both methods, delayed unifications will be forced after a complete parse. If they succeed, then the sentence was simply well-formed. If they fail, it means the input was illformed. The parser can report the point of failure to the user so that the user can intervene and examine the location of illformedness for further processing. Also, possibly, the parser can call other modules such as proposed by Kirtner and Lytinen and perform remedy measures.

3 Algorithm

Below is the algorithm description for the delayed failure detection. The algorithm is based on Quasi-Destructive Graph Unification ([Tomabechi, 1991] abd [Tomabechi, 1993]); however, the method should be adoptable to other graph unification algorithms[2]

The function unify1 assumes the top-level functions unify-dg and unify0. Also dereference functions and quasi-destructive copying functions are needed. The descriptions of these functions are provided in [Tomabechi, 1993].

```
Function unify1(dg1-underef,dg2-underef)
  dg1 <-- dereference-dg(dg1-underef);
  dg2 <-- dereference-dg(dg2-underef);
  IF (dg1 == dg2) THEN
      return(*T*);
  ELSE IF (dg1.type == :bottom) THEN
      forward-dg(dg1,dg2,:temporary);
      return(*T*);
 ELSE IF (dg2.type == :bottom) THEN
      forward-dg(dg2,dg1,:temporary);
      return(*T*);
 ELSE IF (either or both dg1,dg2  or their copies are delayed
          objects)  THEN
       Further delay evaluations.
```

[2]Such as [Emele, 1991], [Karttunen, 1986], [Kogure, 1990], [Wroblewski, 1987].

```
 ELSE IF (dg1.type == :atomic AND dg2.type == :atomic) THEN
      IF (dg1.arc-list == dg2.arc-list) THEN
           forward-dg(dg2,dg1,:temporary);
           return(*T*);
      ELSE throw with keyword 'unify-fail;
 ELSE IF (dg1.type == :atomic OR dg2.type == :atomic) THEN
      throw with keyword 'unify-fail;
 ELSE shared <-- intersectarcs(dg1,dg2);
      FOR EACH arc IN shared DO
         IF eval-feature-query(arc)  returns *T* THEN
               dg1.copy <--
                    delay(unify1(dest of the shared arc for dg1,
                                 dest of the shared arc for dg2));
                   dg1.copy.generation <-- *unify-global-counter*;
           ELSE unify1(dest of the shared arc for dg1,
                                   dest of the shared arc for dg2);
      forward-dg(dg2,dg1,:temporary);
      new <-- complementarcs(dg2,dg1);
      IF (dg1.comp-arc-list is non-empty) THEN
          IF (dg1.generation == *unify-global-counter*) THEN
               FOR EACH arc IN new DO
                    push arc to dg1. comp-arc-list;
          ELSE dg1.comp-arc-list <-- nil;
      ELSE dg1.generation <-- *unify-global-counter*;
                dg.comp-arc-list <-- new;
      return(*T);
END;
```

Dereference-dg(dg) performs the dereferencing by recursively following the forwarding pointers (cf. [Pereira, 1985], [Wroblewski, 1987], [Tomabechi, 1991]). Intersectarcs(dg1,dg2) returns the arcs with labels that exist both in dg1 and dg2 (set-intersection). Complementarcs(dg2,dg1) returns the arcs with labels that exist in dg2 but not in dg1 (set-difference). Eval-feature-query(arc) returns non-nil if the path leading to the arc exceeds the predetermined depth or the arc has the predetermined label (to be delayed).

4 Discussion:

The advantage of the proposed scheme is that we will not need to consider all possible completions of the grammar rules which would be required if we modified the illformed input itself. Also, since the delayed unifications are performed after the parse, either a user of the system or other module can determine the validity of a specific forced unification failure. The hypothesis

that would have been lost during a parse due to the unification failure can be considered at the end of the parse since the unification is delayed until after the parse. If the failure is determined to be unimportant at the end of the parse, then the parse can be accepted.

The proposed scheme can also be considered as a method to speed up unification-based parsing. The speed gain is possible by not spending time on unimportant features. If we make less violated features to be delayed, then constraint violations can be detected faster than without delaying. As demonstrated in [Tomabechi, 1991], efficiency gain can be significant if we can make unifications detect failures as soon as possible since over 60 percent of unifications are normally failures during a parse of a typical large scale grammar. Any activity performed for failed unifications are wastes. By delaying the evaluation of features with less frequency of failures, we can essentially accelerate the timing of evaluation for more frequently violated features. This way, a failure can be found quicker using the delaying method.

In general, delaying some of the unification evaluation until the end of a parse should speed up the parsing process. It is because by delaying the unification until the end of the parse, the unnecessary unifications for feature structures that are eventually rejected by context free grammar rules are avoided. However, it is important to note that the nature of the proposed scheme is different from so called 'lazy unification' schemes (such as [Emele, 1991], [Godden, 1990], [Kogure, 1990]). It is because in the 'lazy unification' schemes, unification operations are delayed within the same top-level unification (in order to avoid so called 'early copying'). In the proposed scheme some unifications are delayed until the end of the parse and not just until the later point within the same top-level unification. The advantage of the proposed scheme is that we can effectively avoid the combinatorial explosion associated with the schemes such as using confusion matrix to make input sentences arbitrarily well-formed. For example with *John studies Lisp* verses *John study Lisp*. The schemes that would consider all possible alterations of phonemes (morphemes) would consider missing of 's' after *study* as well as considering all other possible phonemic alterations. This clearly over-generates. In the proposed scheme alterations (and missing/adding) of phonemes are not directly considered. Instead the feature paths for singular/plural agreement is delayed until the after the parse. This way, massive over-generation due to the consideration of all possible alterations of phonemes can be avoided. It is important to note here that the proposed scheme does not add new hypothetical inputs (source of combinatorial explosion), instead it considers the single input many ways by loosening some

of the unification-based constraints.

Also, the algorithm is designed as a scheme for unification augmentation to context-free parsing. In other words, even if the looser application of unification operations during a parse generates hypothesis that is normally rejected by the standard unification, if the hypothesis is ill-formed based on context-free grammar rules, then it is rejected later anyway. Also, the delayed unifications are forced at the end of the parse. In other words the constraint applications are not 'ignored', it is 'delayed'. One way to view the proposed scheme is that it is the method to avoid premature application of unification constraints that are too strict. By avoiding the premature application we can 1) avoid performing unnecessary unifications for hypotheses that will be rejected later by the context-free parser; 2) some hypothesis that are rejected altogether during a parse can be considered at the end of the parse at which point strict unification operations are forced.

5 Conclusion

Our proposal can be summarized as based upon two observations about natural language parsing using augmented context-free grammar parsing using graph unification as a method to apply augmentation constraints.

Unification results may be eventually rejected and wasted by CFG rules.

Early application of strict constraints makes recovery of rejected hypothesis difficult.

The first claim is essentially about the efficiency and the second claim is about the robustness. In a typical CFG-based parsing using unification-based grammar, many hypotheses produced during a parsing are eventually rejected by subsequent applications of the context-free grammar rules. Since the intermediate hypotheses are created (constituents are built-up) using graph unification operations, the unifications for eventually rejected hypothesis are wastes. Since unification operations are very expensive, this can be a significant waste factor. Because standard unifications are strict regardless of the points in parsing (whether beginning of a sentence or at the end of the sentence), strictest constraint application with the given grammar is performed throughout the parsing. At the beginning of the parsing, most

CFG rules are yet to be fired, therefore, there could be a massive amount of hypotheses most of which are to be eventually rejected. It seems a waste to perform full unification operations for all of them. Thus, ideally, if we can make unification application to become gradually stricter as the parse progresses and more CFG constraints become available, we could effectively avoid such wastes. In the proposed scheme, we make the unification loose until the end of the parse and provide the strict application at the end of the parse. (We may be able to make applications gradually stricter in our future implementations). At the end of the parse, if we force all the delayed application of unification constraints then the parse results would be the same as the ordinally strict unification-based grammar parsing. Only difference is that the burden of constraint application would be partially shifted from unification-operations to CFG parsing. Since parsing operations (such as LR parsing) are significantly more efficient than graph unification operations, this would result in more efficient overall parsing with the same results.

When we force the delayed unification after the parse, if a human user examines the hypothesis that are rejected with the forced strict application, then there will be a possibility that the hypothesis that is rejected may be accepted if the user decides that the original rejection was due to the ill-formedness of the input. Of course, this part may become automatic using some semantic-based approach as well. In either case, the difference is that the hypothesis that are lost during a parse (the process for which are invisible to the user) can be considered at the end of the parse. We find this an important property if we consider the future application areas of the unification-based CFG parsing such as spoken language processing. The modification to the standard unification algorithm for adopting the proposed scheme should not be difficult. We currently have a version for quasi-destructive scheme. We hope to report the statistical data using the proposed scheme during the workshop.

References

[Emele, 1991] Emele, M. "Unification with Lazy Non-Redundant Copying". In *Proceedings of ACL-91*, 1991.

[Godden, 1990] Godden, K. "Lazy Unification" In *Proceedings of ACL-90*, 1990.

[Karttunen, 1986] Karttunen, L. "D-PATR: A Development Environment for Unification-Based Grammars". In *Proceedings of COLING-86*, 1986. (Also, Report CSLI-86-61 Stanford University).

[Kirtner and Lytinen, 1991] Kirtner, H. and S. Lytinen. "ULINK: A Semantics-Driven Approach to Understanding Ungrammatical Input".

[Kogure, 1990] Kogure, K. "Strategic Lazy Incremental Copy Graph Unification". In *Proceedings of COLING-90*, 1990.

[Morimoto, *et al*, 1990] Morimoto, T., H. IIda, A. Kurematsu, K. Shikano, and T. Aizawa. "Spoken Language Translation: Toward Realizing an Automatic Telephone Interpretation System". In *Proceedings of InfoJapan 1990*, 1990.

[Pereira, 1985] Pereira, F. "A Structure-Sharing Representation for Unification-Based Grammar Formalisms". In *Proceedings of ACL-85*, 1985.

[Saito and Tomita, 1988] Saito, H. and M. Tomita. "Parsing Noisy Sentences". In *Proceedings of COLING-88*, 1988.

[Tomabechi and Tomita, 1988] Tomabechi, H. and M. Tomita "The Integration of Unification-based Syntax/Semantics and Memory-based Pragmatics for Real-Time Understanding of Noisy Continuous Speech Input". In *Proceedings of AAAI88*, 1988.

[Tomabechi, 1991] Tomabechi, H. "Quasi-Destructive Graph Unification". In *Proceedings of ACL-91*, 1991.

[Tomabechi, 1992] Tomabechi, H. *MONA-LISA: Multimodal Ontological Neural Architecture for Linguistic Interaction and Scalable Adaptations*, Technical Report, ATR Interpreting Telephony Research Laboratories, 1992.

[Tomabechi, 1993] Tomabechi, H. *Efficient Unification for Natural Language*, Ph.D. Dissertation, Carnegie Mellon University, 1993.

[Tomita and Carbonell, 1987] Tomita, M. and J. Carbonell. "The Universal Parser Architecture for Knowledge-Based Machine Translation". In *Proceedings of IJCAI87*, 1987.

[Wroblewski, 1987] Wroblewski, D."Nondestructive Graph Unification". In *Proceedings of AAAI87*, 1987.