UNIFYING DISJUNCTIVE FEATURE STRUCTURES

LENA STRÖMBÄCK
Department of Computer and Information Science
Linköping University
S-58183 Linköping, Sweden
Telephone +46 13282676
email: lestr@ida.liu.se

Abstract

This paper describes an algorithm for unifying disjunctive feature structures. Unlike previous algorithms, except Eisele & Dörre (1990), this algorithm is as fast as an algorithm without disjunction when disjunctions do not participate in the unification, it is also as fast as an algorithm handling only local disjunctions when there are only local disjunctions, and expensive only in the case of unifying full disjunction. The description is given in the framework of graph unification algorithms which makes it easy to implement as an extension of such an algorithm.

1 Introduction

Disjunction is an important extension to feature structure languages since it increases the compactness of the descriptions. The main problem with including disjunction in the structures is that the unification operation becomes NP-complete. Therefore there have been many proposals on how to unify disjunctive feature structures, the most important being Karttunen's (1984) unification with constraints, Kasper's (1987) unification by successive approximation, Eisele & Dörre's (1988) value unification and lately Eisele & Dörre's (1990a, b) unification with named disjunctions. Since Kasper's and Eisele & Dörre's algorithms seem to be more general and efficient than Karttunen's algorithm I will restrict my discussion to them.

In Kasper's algorithm the structures to be unified are divided into two parts, one that does not contain any disjunctions and one that is a conjunction of all disjunctions in the structure. The idea is to unify the non-disjunctive parts first and then unify the result with the disjunctions, thus trying to exclude as many alternatives as possible. The last step is to compare all disjunctions with each other, making it possible to discard further alternatives. It is this comparison that is expensive. The algorithm is always expensive for disjunctions, regardless of whether they contain path equivalences or not and independent of whether they are affected by the unification or not. This is due to the representation, where all disjunctions are moved to the top level of the structure, which means that larger parts of the structures are moved into the disjunctions and must be compared by the algorithm. Carter (1990) has made a development of this algorithm which improves the efficiency when used together with bottom-up parsing.

Eisele & Dörre's (1988) approach is based on the fact that unification of path equivalences should return not only a local value, but also a global value that affects some other part of the structure. Their solution is to compute the local value and save the global value as a global result. The global results will be unified with the result of the first unification. This new unification can also generate a new global disjunction so that the unification with global results will be repeated until no new global result is generated. This solution generates at least one, but often more than one, extra unification for each path equivalence. Thus, the algorithm is always expensive for path equivalences, regardless of whether they are contained inside disjunctions or not.

The approach taken by Eisele & Dörre (1990) is similar to the approach taken in this paper. They use 'named disjunction' (Kaplan & Maxwell 1989) and one of their central ideas i.e. to use a disjunction as the value of a variable to decide when the value is dependent on the choice in some disjunction is similar to the way of unifying variables in the present paper. However, they use feature terms for representing the structures and their algorithm is described by a set of rewrite rules for feature terms. This makes the algorithm different from algorithms described for graph unification.

What is special with the algorithm in the present paper is that it is

1. As efficient as an algorithm not handling disjunction when the participating structures do not contain any disjunctions.
2. As efficient as an algorithm allowing only local disjunctions when the participating structures only contain such disjunction.
3. Expensive only when non-local disjunction is involved.

The description is given in a way that makes the algorithm easy to implement as an extension of a graph unification algorithm.

2 The Formulas

Feature structures are represented by formulas. The syntax of the formulas, especially the way of constructing complex graphs, is chosen so as to get a close relation to feature structures. This also makes it easy to construct a unification procedure similar to

graph unification and give the formulas a semantics based on graph models. For disjunction a generalization of Kaplan & Maxwell's (1989) 'named disjunction' is used. Their idea is to give the disjunctions names so that it is possible to restrict the choices in them. Kaplan and Maxwell use only binary disjunctions, and if the left alternative in one disjunction is chosen the left alternative in all disjunctions with the same name has to be chosen. In this paper I do not restrict the algorithm to binary disjunctions. Instead of giving the disjunction a name I give each alternative a name. Alternatives with the same name are then connected so that if one of them is chosen we also have to choose all the others.

We assume four basic sets A, F, X and Σ of atoms, feature attributes, variables and disjunction switches respectively. These sets contain symbols denoted by strings. They are all assumed to be enumerable and pairwise disjoint. From these basic sets we define the set S of feature structures. S contains the following structures:

- T : no information
- $\bot$: failure
- a for all $a \in A$: atoms
- x for all $x \in X$: variables
- $[f_1{:}s_1, ..., f_n{:}s_n]$ for any $f_i \in F$, $s_i \in S$, $n > 0$ such that $f_i \neq f_j$ for $i \neq j$: complex feature structure
- $\{\sigma_1{:}s_1, ..., \sigma_n{:}s_n\}$ for any $\sigma_i \in \Sigma$, $s_i \in S$, $n \geq 0$ such that $\sigma_j \neq \sigma_j$ for $i \neq j$: disjunction

A *formula* is defined to be a pair $\langle s, v \rangle$ where s is a feature structure and $v{:}X \rightarrow S$ a *valuation function* that assigns structures to variables. We demand that the formulas are acyclic.

An example of a formula is given in figure 1. Variables are denoted by using the symbol # and a number. The same formula is also given in matrix format which will be used to make the examples easier to read.

$$\langle [a{:}\ [e{:}\#1\ ]\ , b{:}3, c{:}\#1\ ],\ \{\ \langle \#1,\ [d{:}4\ ] \rangle \}\ \rangle$$

$$\begin{bmatrix} a{:} & \begin{bmatrix} e{:} & \#1{=}\begin{bmatrix} d{:} & 4 \end{bmatrix} \end{bmatrix} \\ b{:} & 3 \\ c{:} & \#1 \end{bmatrix}$$

Figure 1

We can observe that according to this definition formulas are not unambiguously determined. The same formula can for example be expressed with different variables. There is also nothing said about the value of the valuation function v for variables not occurring in the formulas.

3 Semantics

The semantics given for these formulas is similar to the one given by Kasper & Rounds (1986) for their logic of feature structures. This logic is modified in the same way as in Reape (1991) to allow for the use of variables instead of equational constraints as used by Kasper and Rounds. As Kasper and Rounds I will use a graph model for the formulas where each formula is satisfied by a set of graphs. I will use δ to denote the transition function between nodes in the graph. We also need to define a valuation to describe the semantics of variables. Given a graph a valuation is a function $V{:}X \rightarrow N$. By this function every variable is assigned a node in the graph as its value.

Satisfaction is defined by the following rules. The model $M = \langle G, V, L \rangle$ where G is a graph, V a valuation and L a subset of the switches occurring in the formula.satisfies a formula at node i iff it fulfils any of these cases. I will use the notion *sat(i)* if node i in the graph satisfies a formula.

- M *sat(i)* $\langle \mathrm{T}, v \rangle$ for all v
- M *sat(i)* $\langle \bot, v \rangle$ for no v
- M *sat(i)* $\langle a, v \rangle$ iff node i in G is the leaf $a \in A$
- M *sat(i)* $\langle x, v \rangle$ iff $V(x){=}i$ and M *sat(i)* $\langle v(x), v \rangle$
- M *sat(i)* $\langle [f_1{:}s_1, ..., f_n{:}s_n], v \rangle$ iff for all $k = 1 \ldots n$ $\delta(i f_k){=}j_k$ and M *sat*(j_k) $\langle s_k, v \rangle$
- M *sat(i)* $\langle \{\sigma_1{:}s_1, ..., \sigma_n{:}s_n\}, v \rangle$ iff precisely one of $\sigma_1 \ldots \sigma_n$ is in L and M *sat(i)* $\langle s_k, v \rangle$ for k such that $\sigma_k \in L$

These rules correspond to the usual satisfaction definitions for feature structures. The subset of switches L forces us to choose exactly one alternative in each disjunction and the model should satisfy this alternative.

4 Unification

In this section I will define a set of rewrite rules for computing the unification of two formulas. I will start by introducing the operator $\wedge$ into our formulas. The syntax and semantics is given by the following rules:

- M *sat(i)* $fs_1 \wedge fs_2$ iff fs_1 and fs_2 are formulas and M *sat(i)* fs_1 and M *sat(i)* fs_2
- M *sat(i)* $\langle s_1 \wedge s_2, v \rangle$ iff M *sat(i)* $\langle s_1, v \rangle$ and M *sat(i)* $\langle s_2, v \rangle$

The operator $\wedge$ can be viewed as the unification operator. By the definition we can see that it is interpreted as a conjunction or intersection of the two participating formulas, which is the normal interpretation of unification. The task of unifying two formulas is then the task of rewriting two formulas containing $\wedge$ into a formula not containing $\wedge$. Here we can note that since a formula is not unambiguously determined the unified formula is not unique. Actually there is a set of formulas that all have the

same model as the unification of the formulas. The aim here is to compute one of these formulas as a representative for this set, and thus a representative for the unification of fs_1 and fs_2. The rewrite rules given below correspond to the unification algorithm for formulas not containing disjunction.

1. $\langle s_1, v_1\rangle \wedge \langle s_2, v_2\rangle \equiv \langle s_1 \wedge s_2, v\rangle$ if v_1 and v_2 are disjoint and $v(x)=v_1(x)$ for all x in v_1, $v(x)=v_2(x)$ for all x in v_2.
2. $\langle s_1 \wedge s_2, v\rangle \equiv \langle s_2 \wedge s_1, v\rangle$
3. $\langle \mathrm{T} \wedge s, v\rangle \equiv \langle s, v\rangle$
4. $\langle a \wedge a, v\rangle \equiv \langle a, v\rangle$ where $a \in A$
5. $\langle a \wedge b, v\rangle \equiv \langle \bot, v\rangle$ where $a \neq b$ and $a,b \in A$
6. $\langle a \wedge [f_1{:}s_1 \ldots f_n{:}s_n], v\rangle \equiv \langle \bot, v\rangle$ where $a \in A$
7. $\langle \bot \wedge s, v\rangle \equiv \langle \bot, v\rangle$
8. $\langle x \wedge s, v\rangle \equiv \langle x, v_2\rangle$
   where $\langle v(x) \wedge s, v\rangle \equiv \langle s_1, v_1\rangle$ and $x \in X$, $v_2(x)=s_1$ and $v_2=v_1$ for all other variables
9. $\langle [f_{11}{:}s_{11} \ldots f_{1n}{:}s_{1n}] \wedge [f_{21}{:}s_{21} \ldots f_{2m}{:}s_{2m}], v\rangle \equiv \langle s, v_p\rangle$
   where s is the complex feature structure containing:
   $f_{1j}{:}s_{1j}$ for any j such that $f_{1j} \neq f_{2k}$ for all k
   $f_{2j}{:}s_{2j}$ for any j such that $f_{2j} \neq f_{1k}$ for all k
   $f_{1j}{:}s_{3i}$ for any j,k such that $f_{1j}=f_{2k}$ where $\langle s_{1j} \wedge s_{2k}, v_{(i-1)}\rangle \equiv \langle s_{3i}, v_i\rangle$
   and i describes some enumeration of the resulting formulas $v_0=v$ and $\langle s_{3p}, v_p\rangle$ is the last of the formulas.

The first rule is a kind of entry rule and can be interpreted as saying that it is possible to unify two formulas if the variables occurring within them are disjoint. The second rule says that unification is commutative, and are used to avoid duplicating the other rules. The next rule says that T unifies with everything. Rules four to six says that an atom only unifies with itself and becomes failure when unified with some other atom or a complex structure. The seventh rule says that unifying failure always yields failure. The eighth rule deals with unification of variables. Here we have to start with unifying the value of the variable with the other structure. This unification gives a new pair of feature structure and valuation function as result where the new valuation function contains the changes of variables that have been made during this unification. The result of the unification of a variable is the pair of the variable and the new valuation function where the value of the variable is replaced with the unified one. Rule nine deals with the unification of two complex feature structures and says that the result is the structure obtained by unifying the values of the common attributes of the two structures and then adding all attributes that occurs in either of the structures to the result.

Figure 2 gives an example that illustrates what modifications that must be made to the rewrite rules to be able to handle unification of disjunction. Unifying a disjunction is basically unifying each of its alternatives. But the example also shows what must

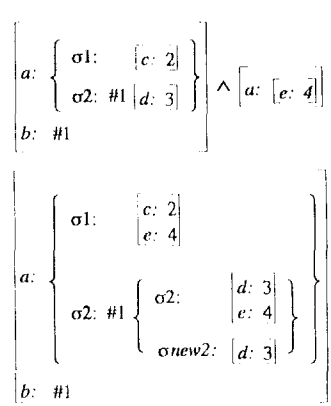

Figure 2

happen if a variable occurs within the disjunction. The value of the variable is global since it can affect parts of the structure outside the disjunction. Therefore this value must be dependent on what alternative that is chosen in the disjunction. This is done by representing the value of the variable as a new disjunction where we only choose the unified value if the alternative σ_2 is chosen. To express this in the rewrite rule we index all rules by the list of switches that are traversed in the formula. This is expressed by replacing the $\equiv$ with $\equiv^X$ in all rules where X is a list of the switches passed to reach this point of the unification. We also need to split rule 8 into two rules depending on if any disjunctions have been passed to reach the variable. The new rules are given below and we assume that the switches occurring in each formula are unique.

8.a $\langle x \wedge s, v\rangle \equiv^{\langle\rangle} \langle x, v_1 \cup \langle x, s_1\rangle\rangle$
   where $\langle v(x) \wedge s, v\rangle \equiv^{\langle\rangle} \langle s_1, v_1\rangle$ $x \in X$, $v_2(x)=s_1$ and $v_2=v_1$ for all other variables

8.b $\langle x \wedge s, v\rangle \equiv^{\langle\sigma 1 \ldots \sigma n\rangle} \langle x, v_2\rangle$
   where $\langle v(x) \wedge s, v\rangle \equiv^{\langle\sigma 1 \ldots \sigma n\rangle} \langle s_1, v_1\rangle$, $x \in X$,
   $v_2(x)=\{\sigma_1{:}\{\sigma_2{:}\{\ldots\{\sigma_n{:}s_1\ \sigma_{newn}{:}v(x)\ldots\}$
   $\sigma_{new2}{:}v(x)\}\ \sigma_{new1}{:}v(x)\}$, $v_2=v_1$ for all other variables and σ_{newi} is a switch name not used before.

10. $\langle \{\sigma_1{:}s_{11} \ldots \sigma_n{:}s_{1n}\} \wedge s, v\rangle \equiv^X \langle \{\sigma_1{:}s_{21} \ldots \sigma_n{:}s_{2n}\}, v_n\rangle$
    where $\langle s_{1i} \wedge s, v_{(i-1)}\rangle \equiv^{\langle\sigma i|X\rangle} \langle s_{2i}, v_i\rangle$ and $v_0=v$

In Strömbäck (1991, 1992) these rewrite rules are proved to compute the unification of two formulas.

5 Discussion

The syntax and semantics of the formulas are very similar to what is given in Reape (1991 pp 35) which is a development of the semantics given in Kasper & Rounds (1986) that allows the use of vari-

ables to express equational constraints. The difference is that I use formulas of the form $[f_1{:}s_1 .. f_n{:}s_n]$ instead of an ordinary conjunction and that we use named disjunction. This restricts the syntax of the formulas somewhat and makes them closer to ordinary feature structures. The restricted syntax is also the reason why we need to include a valuation function in the formulas.

It is easy to represent the formulas as ordinary directed acyclic graphs where variables are represented as references to the same substructure in the graphs. If we think of the formulas as graphs it is also easy to compare the rewrite rules 1-9 above with an ordinary graph unification algorithm. Doing this we can conclude that each of the rewrite rules three to nine corresponds to a case in the unification algorithm. The only difference is that when variables are represented as reentrant subgraphs we never have to look-up the variable to find its value. The main advantage with defining unification by a set of rewrite rules is that the procedure can be proved to be correct.

6 Detection of failure and improvements

The problem with the rewrite rules is that they sometimes produces formulas which have no model. Such formulas must be detected in order to know when the unification fails. As long as the formulas only contain local disjunction this is not a problem and it is easy to change the rewrite rules in order to propagate a failure to the top level in the formula. The ninth rule is, for example, changed to return $\langle \perp, v_p \rangle$ whenever any of the values of the attributes in the resulting formula is fail.

When nonlocal disjunction is included we must find some of keeping track of which choices of switches in the disjunctions that represent a failure. This can be done by building a tree where the paths represents possible choices of switches and the leaf nodes in the tree contains a value that is *false* if this choice represents a subset of switches for which the formula has no model and *true* otherwise. Figure 3 shows an example of a formula and its corresponding choice tree. To reach the leaf b in the tree the switches σ1, σ3, and σn have been chosen and σ2, σ4, and σ3 have not. So σ3 is both chosen and not chosen and the value of this leaf must be *false*. Continuing this reasoning for the other paths in the tree we could see that the leafs b, e, and f must have the value false and the other leafs must have the value true. If some value of an alternative is ⊥ the corresponding leafs in the choice tree must be *false*. If we, for example assume that the value of σ4 is fail we must assign *false* to the leafs *c*, *f*, and *g*.

Choice trees can be built ones for each formula and merged during the unification of formulas. A better solution is to only build the choice trees when they are needed, i.e. when a disjunction alternative

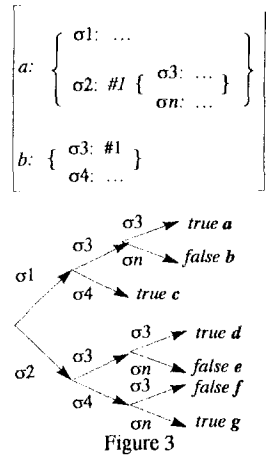

Figure 3

where the disjunction shares some switch name with another disjunction fails. If this is done we only have to do the expensive work when really needed which is when we have failure in a non-local disjunction and achieves a better performance of the algorithm for all other cases.

Strömbäck (1991, 1992) discusses how the choice tree is best used. The papers also discuss how the choice tree can be used to remove failed alternatives from a formula without destroying the interpretation of the formula. The main idea here is to see what switches that must be chosen to reach each disjunction alternative in the formula. For this set of switches we find all leafs in the choice tree that can be reached if these switches are chosen. If all these leafs are false the alternative should be removed. For example, if we assume that the value of σ4 in figure 3 is fail and that we have assigned *false* to the corresponding leafs in the choice tree, we can also see that there is no way of reaching a leaf with the value *true* if we have to choose σn. In this case we can as well remove both σ4 and σn from the feature structure.

The two papers mentioned above also discuss various improvements that can be made in order to get a more efficient algorithm. Most important here is that we can build only parts of the choice tree and that the notion of switches for a disjunction can be extended to allow sets of switches in order to avoid creating too many new disjunctions.

7 Implementation

The algorithm has been implemented in Xerox Common Lisp and is running on the Sun Sparcstations.

8 Complexity

To analyze the complexity of this algorithm I will look at three cases. If we assume that there are no disjunctions in the formulas the procedure can be implemented almost linearly. If we have local disjunction in the formulas, i.e. disjunctions which do not contain variables and which not are connected by switch names, the total complexity becomes exponential on the maximum depth of disjunctions occurring within each other. For the third case we have to add the complexity for the removal strategies when alternatives have failed. The complexity for this procedure is also exponential in the size of a^a, where a is the total number of alternatives occurring in the formulas. For a more complete discussion of the complexity see Strömbäck (1991, 1992)

When considering complexity one must remember that the second case will only be performed when there are disjunctions in the formula and when these disjunctions are actually affected by the unification. Disjunctions in some subpart of the formula not affected by the unification never affect the complexity. It is also reasonable to assume that in most cases when a disjunction really participates in the unification, some of its alternatives will be removed due to failure. The same thing holds for the last case; it will only be performed when some global alternative has failed. This means that this procedure can at most be performed once for each ordinary alternative in the initial formulas.

Comparing this to the other proposed alternatives we can see that Kasper's (1987) algorithm has a better worst case complexity ($2^{d/2}$). On the other hand this complexity holds for all disjunctions in the structure regardless of whether they are affected by the unification or not. The algorithm by Eisele & Dörre (1988) has a similar worst case complexity. The disadvantage here is that this algorithm is expensive even if the structures do not contain any disjunctions at all. The third algorithm (Eisele & Dörre 1990a, b) will also be NP-complete in the worst case and will probably have a similar performance compared to the algorithm described in this paper.

9 Conclusion

This paper describes an algorithm for unifying disjunctions which calls for as little computation as possible for each case. Disjunctions only affect the complexity when they directly participate and are affected by the unification, which is the only case when we expand to disjunctive normal form. The most expensive work is done only when there is a failure in a disjunction which affects some other part of the structure. The only algorithm that shows similar complexity is the algorithm proposed by Eisele & Dörre (1990). However the description given by Eisele and Dörre is harder to relate and implement as a graph unification algorithm. This paper shows that it is possible to use similar ideas together with graph unification. The description given here is fairly easy to implement as an extension of a graph unification algorithm.

Acknowledgements

This work is part of the project Dynamic Language Understanding supported by the Swedish Council for Research in the Humanities and the Swedish Board for Industrial and Technical Development. I would also like to thank Lars Ahrenberg and Tore Langholm for valuable comments on this work.

References

Carter, David (1990). Efficient Disjunctive Unification for Bottom-Up Parsing. *Proc. 13th International Conference on Computational Linguistics*, vol. 3, pp 70-75.

Eisele, Andreas and Jochen Dörre (1988). Unification of Disjunctive Feature Descriptions. *Proc. 26th Annual Meeting of the Association for Computational Linguistics*, pp 286-294.

Eisele, Andreas and Jochen Dörre (1990a). Disjunctive Unification. IWBS Report 124, IWBS, IBM Deutschland, W. Germany, May 1990.

Eisele, Andreas and Jochen Dörre (1990b). Feature Logic with Disjunctive Unification. *Proc. 13th International Conference on Computational Linguistics*, vol. 2, pp 100-105.

Karttunen, Lauri (1984). Features and Values. *10th International Conference on Computational Linguistics/22nd Annual Meeting of the Association for Computational Linguistics*, Stanford, California, pp 28-33.

Karttunen, Lauri (1986). D-PATR: A Development Environment for Unification Based Grammars. *Proc. 11th International Conference on Computational Linguistics*, Bonn, Federal Republic of Germany, pp 74-80.

Kaplan, Ronald M. and John T. Maxwell III (1989). An Overview of Disjunctive Constraint Satisfaction. *Proc. International Workshop on Parsing Technologies*, Pittsburgh, Pennsylvania, pp 18-27.

Kasper, Robert T. (1987). A Unification Method for Disjunctive Feature Descriptions. *25th Annual Meeting of the Association for Computational Linguistics*, pp 235-242.

Reape, Mike (1991). An Introduction to the Semantics of Unification-Based Grammar Formalisms. Deliverable R3.2.A DYANA - ESPRIT Basic Research Action BR 3175.

Rounds, William C. and Robert Kasper (1986). A Complete Logical Calculus for Record Structures Representing Linguistic Information. *Proc. Symposium on Logic in Computer Science*, Cambridge Massachusetts, pp 39 - 43

Strömbäck, Lena (1991). Unifying Disjunctive Feature Structures. Technical Report LiTH-IDA-R-91-34, Department of Computer and Information Science, Linköping University, Linköping, Sweden.

Strömbäck, Lena (1992). *Studies in Extended Unification Formalisms for Linguistic Description*. Licentiate thesis. Department of Computer and Information Science, Linköping University, Linköping, Sweden.