Arbeitspapiere des SFB 340
Bericht Nr. 135, Juli 1999

TRACTABILITY AND STRUCTURAL CLOSURES IN ATTRIBUTE LOGIC TYPE SIGNATURES

GERALD PENN

ABSTRACT.

CONTENTS

1. INTRODUCTION

The logic of typed feature structures (LTFS, Carpenter 1992) and, in particular, its implementation in the Attribute Logic Engine (Carpenter and Penn 1996), have been widely used as a means of formalising and developing grammars of natural languages that support computationally efficient parsing and SLD resolution, notably grammars within the framework of Head-driven Phrase Structure Grammar (HPSG, Pollard and Sag 1987; Pollard and Sag 1994). These grammars are formulated using a vocabulary provided by a finite partially ordered set of types and a set of features that must be specified for each grammar, and feature structures in these grammars must respect certain constraints that are also specified. These include *appropriateness conditions*, which specify all and only the features that may take values in feature structures of a given type, as

ISSN 0947-6954/99

well as more general implicational constraints of the form $\tau \rightarrow \phi$, where τ is a type, and ϕ is an expression from LTFS's description language. In LTFS and ALE, these four components, a partial order of types, a set of features, appropriateness declarations and type-antecedent constraints can be taken as the *signature* of a grammar, relative to which descriptions can be interpreted.

Carpenter 1992 and ALE also make several assumptions about the structure and interpretation of this partial order of types and about appropriateness, some for the sake of generality, others for the sake of efficiency or simplicity. Appropriateness is generally accepted as a good thing, from the standpoints of both efficiency and representational accuracy, and while many have advocated the need for implicational constraints that are even more general, type-antecedent constraints at the very least are also accepted as being necessary and convenient. Not all of these assumptions are universally observed upon by formal linguists or grammar developers, however.

This paper addresses the three most contentious assumptions that Carpenter 1992 and ALE make, and how to deal with their absence in a tractable manner. They are:

1. **Meet-semi-latticehood:** every partial order of types must be a meet semi-lattice. This means that every consistent pair of types has a least upper bound.
2. **Unique feature introduction:** for every feature, F, there is a unique most general type for which F is appropriate.
3. **Subtype covering not assumed:** there can be feature structures of a non-maximally-specific type that are not typable as any of its maximally specific subtypes. When subtype covering is not assumed, feature structures themselves can be partially ordered and taken to represent partial information states about some set of objects. When subtype covering is assumed, feature structures are unordered and totally informative, and can be taken to represent the objects themselves. The latter interpretation is subscribed to by Pollard and Sag 1994, for example.

All three of these conditions have been claimed elsewhere to be either intractable or impossible to restore in grammar signatures where they do not already exist. It will be shown here that meet-semi-latticehood and unique feature introduction can be restored to a signature in low-degree polynomial time, and that, while type inferencing when subtype covering is assumed is intractable in the worst case, a very elegant constraint logic programming solution combined with a special compilation

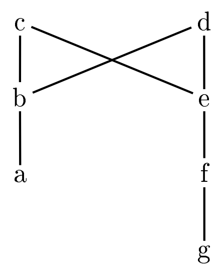

FIGURE 1. An example of a partial order that is not a meet semi-lattice.

method exists that can restore tractability in many practical contexts. The NP-completeness proof of non-disjunctive type inferencing with subtype covering (Carpenter and King 1995) will also be repeated here, as it was never published.

2. Meet-semi-latticehood

In LTFS and ALE, partial orders of types are assumed to be meet semi-lattices:

DEFINITION 1. *A partial order,* $\langle P, \sqsubseteq \rangle$, *is a* meet semi-lattice *iff for any* $x, y \in P$, $x \sqcap y\downarrow$.

$\sqcap$ is the binary greatest lower bound, or *meet* operation, and is the dual of the join operation, $\sqcup$, which corresponds to unification, or least upper bounds. Figure 1 is not a meet semi-lattice because c and d do not have a meet, nor do a and g, for example.

The assumption that every pair of types has a meet is equivalent to the assumption that every pair of consistent types, i.e., types with an upper bound, has a join. It is theoretically convenient when discussing the unification of feature structures to assume that the unification of two consistent types always exists. It can also be more efficient to make this assumption as, in some representations of feature structures and types, it avoids a source of non-determinism (selection among minimal but not least upper bounds) during search.

Just because it would be convenient for unification to be well-defined, however, does not mean it would be convenient to think of any empirical domain's concepts as a meet semi-lattice, or that it would be convenient to add all of the types necessary to a would-be type hierarchy to ensure meet-semi-latticehood. The question then naturally arises as to whether

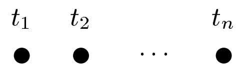

FIGURE 2. The discrete partially ordered set on n elements.

it would be possible, given any finite partial order, to add some extra elements (types, in this case) to make it a meet semi-lattice, and if so, how many extra elements it would take, which also provides a lower bound on the time complexity of the completion.

It is, in fact, possible to embed any finite partial order into a lattice that preserves existing meets and joins by adding extra elements. The resulting construction is the finite restriction of the Dedekind-MacNeille completion (Davey and Priestley 1990, p. 41).

DEFINITION 2. *Given a partially ordered set, P, the Dedekind-MacNeille completion of P, $\langle DM(P), \subseteq \rangle$, is given by:*

$$DM(P) = \{A \subseteq P | A^{ul} = A\}$$

This route is also considered in the context of taxonomical knowledge representation in both Aït-Kaći, Boyer, Lincoln, and Nasr 1989 and Fall 1996. Both view meet lattice completions as a practical step towards providing a semantics for arbitrary partial orders, but as an impractical preliminary step to performing computations over a partial order. Aït-Kaći, Boyer, Lincoln, and Nasr 1989, p. 3 claim that "as the size of the poset grows, there must be an exponential number of pairwise GLB's [for us, LUBs or joins] to be specified." This is a rather odd statement to make, because as a partially ordered set grows, it could very well be growing joins. What Aït-Kaći, Boyer, Lincoln, and Nasr 1989 probably meant to say is that there must be an exponential number of completion types as a function of the number of maximally specific types if the completion is to provide a unique representation to every subset of types in the original partially ordered set. That is indeed the case for Figure 2, the discrete partially ordered set of n elements, for which the power-set completion would have $O(2^n)$ elements. The relevant problem for closure under joins, however, is to find the *minimal* join-preserving completion lattice. For Figure 2, the minimal completion is shown in Figure 3. This is also the Dedekind-MacNeille completion of Figure 2. The only circumstance under which a power-set completion would be necessary is with

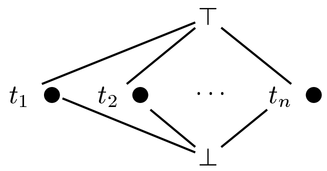

FIGURE 3. The minimal completion of Figure 2.

an open-world interpretation of subsumption,[1] which neither Aït-Kaći, Boyer, Lincoln, and Nasr 1989 nor this work assume, and in which the unification of any two types succeeds non-trivially anyway, i.e., produces a type other than inconsistency, represented by the type, $\top$ (for Aït-Kaći, Boyer, Lincoln, and Nasr 1989, $\perp$).

Fall 1996, p. 18 provides an algorithm for finding the minimal completion, but says that it may add an exponential number of elements to the original set. The algorithm, which tacitly assumes the prior existence of a most general type, is quite simple:

1. Find two elements, t_1, t_2 with minimal upper bounds, $u_1 \ldots u_k$, such that their $t_1 \sqcup t_2$ is undefined, i.e., $k > 1$. If no such pair exists, then stop.
2. Add an element, v, such that:
   - for all $1 \leq i \leq k$, $v \sqsubseteq u_i$, and
   - for all elements t, $t \sqsubseteq v$ iff for all $1 \leq i \leq k$, $t \sqsubseteq u_i$.
3. Go to (1).

PROPOSITION 1. *For any finite partially ordered set, P, the completion algorithm terminates on P.*

Proof: Consider the function:

$$S(P) = \sum_{p \in P} \sigma^2(p).$$

S(P) is always a non-negative integer and monotonically decreases with every pass through the loop. □ The sum of the supertype branching factors (without squaring) decreases for those pairs for which $k > 2$ and remains the same when $k = 2$. Because $S(P)$ always decreases, it can be

[1]The phrases "closed-world" and "open-world" are used by Carpenter, Gerdemann and King in various places, e.g., Carpenter 1992; Gerdemann and King 1994; Gerdemann 1995 to refer to what is here called subtype covering. In keeping with its use in the theory of programming languages, however, the closed-world assumption here is taken to refer to the assumption that types for which no subtypes are specified truly have no subtypes.

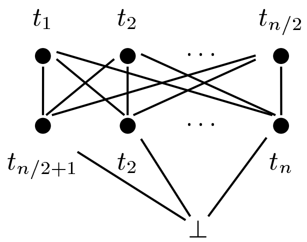

FIGURE 4. A $\perp$-lifted complete bipartite graph on n vertices.

used to count the number of elements added before it reaches its minimum value of 0, which is attained only with the type hierarchy consisting simply of $\perp$.

PROPOSITION 2. *The completion algorithm is correct on finite partially ordered sets, P, i.e., upon termination, it has produced $DM(P)$.*

Proof: Let $V(P)$ be the partially ordered set produced by the algorithm. Clearly, $P \subseteq V(P)$. It suffices to show that (1) $V(P)$ is a complete lattice (with $\top$ added), and (2) for all $v \in V(P)$, there exist subsets $A, B \subseteq P$ such that $v = \bigvee_{V(P)} A = \bigwedge_{V(P)} B$.[2]

Suppose there are $v, w \in V(P)$ such that $v \sqcap w\uparrow$. There is a least element, so v and w have more than one maximal lower bound, l_1, l_2 and others. But then $\{l_1, l_2\}$ is upper-bounded and $l_1 \sqcup l_2\uparrow$, so the algorithm should not have terminated. Suppose instead that $v \sqcup w\uparrow$. Again, the algorithm should not have terminated. So $V(P)$ with $\top$ added is a complete lattice.

Given $v \in V(P)$, if $v \in P$, then choose $A_v = B_v = \{v\}$. Otherwise, the algorithm added v because of a bounded set $\{t_1, t_2\}$, with minimal upper bounds, $u_1, \ldots u_k$, which did not have a least upper bound, i.e., $k > 1$. In this case, choose $A_v = A_{t_1} \cup A_{t_2}$ and $B_v = \bigcup_{1 \leq i \leq k} B_{u_i}$. In either case, clearly $v = \bigvee_{V(P)} A_v = \bigwedge_{V(P)} B_v$ for all $v \in V(P)$. $\square$

PROPOSITION 3. *If P has $n+1$ elements, then the completion algorithm adds at most $O(n^3)$ new elements.*

Proof: Consider the partially ordered set, B, given by a $\perp$-lifted complete bipartite graph on n vertices, where n is even, as shown in Figure 4. Each of the $n/2$ elements in the top row has a supertype branching factor of

[2]These are sometimes called the *join density* and *meet density*, respectively, of P in $V(P)$ (Davey and Priestley 1990, p. 42).

$O(n)$, so $S(B) = O(n^3)$, and thus over all P, the worst case of S's value is $\Omega(n^3)$.

The complete graph on n nodes, which cannot represent a partially ordered set but still has the maximum number of edges, has the same $O(n)$ bound on its vertex degrees, so this is a tight bound on the value of S: for any P, $S(P) = \Theta(n^3)$. S in turn is an upper bound on the number of new elements added. $\square$

$S(P)$, however, is not a tight bound on the number of new elements added to P. By choosing a tighter approximation than $S(P)$, it may be possible to provide an upper bound of lesser degree.

In practice, partially ordered sets are specified by their immediate subsumption relation, which must then be reflexively and transitively closed to obtain the true partial order. As new elements are added to the partially ordered set, they also must be transitively closed, which can be accomplished incrementally at the cost of $O(n^2)$ per new element (a more accurate upper bound would consider the branching factor instead). So the practical time cost of meet semi-lattice completion is more like $O(n^5)$ in the worst case. Transitive closure is considered more closely in Penn 1999b. In any case, the number of completion elements is by no means exponential.

3. Unique Feature Introduction

LTFS and ALE also assume that appropriateness guarantees the existence of a unique introducer for every feature:

Definition 3. *Given a type hierarchy, $\langle T, \sqsubseteq \rangle$, and a finite set of features, Feat, an* appropriateness specification *is a partial function, $Approp : Feat \times T \longrightarrow T$ such that, for every $\textsc{f} \in Feat$:*

- ***(Feature Introduction)*** *there is a type $Intro(\textsc{f}) \in T$ such that:*
  - *$Approp(\textsc{f}, Intro(\textsc{f}))\downarrow$, and*
  - *for every $t \in T$, if $Approp(\textsc{f}, t)\downarrow$, then $Intro(\textsc{f}) \sqsubseteq t$, and*
- ***(Upward Closure / Right Monotonicity)*** *if $Approp(\textsc{f}, s)\downarrow$ and $s \sqsubseteq t$, then $Approp(\textsc{f}, t)\downarrow$ and $Approp(\textsc{f}, s) \sqsubseteq Approp(\textsc{f}, t)$.*

Feature introduction has been argued not to be appropriate for certain empirical domains either. This has even been alleged for HPSG (King and Goetz 1993), although the argumentation there was based on a counter-intuitive partial order defined over parametrically-typed lists and (Pollard and Sag 1994) does otherwise observe it. The debate, however, has focussed on whether to modify some other aspect of type inferencing in order to compensate for the lack of feature introduction, presumably under the assumption that feature introduction was difficult or impossible to restore automatically to grammar signatures that did not have it.

Just as with the condition of meet-semi-latticehood, however, it is possible to take a would-be signature without feature introduction and restore this condition through the addition of extra unique introducing types for certain appropriate features. The following algorithm achieves this:

1. Given candidate signature, S, find a feature, F, for which there is no unique introducing type. Let K be the set of minimal types to which F is appropriate, where $|K| > 1$. If there is no such feature, then stop.
2. Add a new type, v, to S, to which F is appropriate, such that:
   - for all $k \in K$, $v \sqsubseteq k$,
   - for all types, t in S, $t \sqsubseteq v$ iff for all $k \in K$, $t \sqsubseteq k$, and
   - $\mathit{Approp}(\text{F}, v) = \mathit{Approp}(\text{F}, k_1) \sqcap \mathit{Approp}(\text{F}, k_2) \sqcap \ldots \sqcap \mathit{Approp}(\text{F}, k_{|K|})$, the generalization of the value restrictions on F of the elements of K.
3. Go to (1).

In practice, the same signature completion type can be used for different features, provided that their minimal introducers are the same set, K. This clearly produces a partially ordered set with a unique introducing type for every feature. It may disturb meet-semi-latticehood, however, which means that the result must undergo meet semi-lattice completion, as described in Section 2. If generalisation has already been computed, the signature completion algorithm runs in $O(fn)$, where f is the number of features, and n is the number of types.

4. Subtype Covering

In HPSG, following King 1989, it is generally assumed that non-maximally-specific types are simply a convenient shorthand for talking about sets of maximally specific types, sometimes called *species*, over which the principles of a grammar are stated. In a view where feature structures are not partially ordered but represent objects in an empirical model themselves, every feature structure must bear one of these species. In particular, each non-maximally-specific type in a description is equivalent to the disjunction of he maximally specific subtypes that it subsumes.

There are some good reasons not to build this assumption, called "subtype covering," into LTFS or its implementations. Firstly, it is not an appropriate assumption to make for some empirical domains. Even in

HPSG, the denotations of parametrically-typed lists are more naturally interpreted without it. Secondly, not to make the assumption is more general: where it is appropriate, extra type-antecedent constraints can be added to the grammar signature of the form:

$$n \Rightarrow m_1 \vee \cdots \vee m_{k_i}$$

for each non-maximally-specific type, n, and its i maximal subtypes, $m_1, \ldots, m_{k_i}$. These constraints become crucial in certain cases where the possible permutations of appropriate feature values at a type are not covered by the permutations of those features on its maximally specific subtypes. This is the case for the type, `verb`, in the following signature:

```
bot sub [verb,bool].
  bool sub [+,-].
  verb sub [aux_verb,main_verb]
       intro [aux:bool,inv:bool].
    aux_verb sub [aux:+,inv:+].
    main_verb sub [aux:-].
```

The combination, AUX:$+ \wedge$ INV:$-$, is not attested by any of `verb`'s subtypes. While there are arguably better ways to represent this information, the extra type-antecedent constraint:

$$\text{verb} \Rightarrow \text{aux\_verb} \vee \text{main\_verb}$$

is necessary in order to decide satisfiability correctly under the assumption of subtype covering. We will call these types *deranged types*. Types that are not deranged are called *normal types*.

4.1. **Non-Disjunctive Type Inference under Subtype Covering is NP-Complete.** Third, when subtype covering is implicitly assumed, determining whether a non-disjunctive description is satisfiable under appropriateness conditions is an NP-complete problem, whereas this is known to be polynomial time without it (and without type-antecedent constraints, of course). This was originally proven by Carpenter and King 1995, and its proof (with minor corrections) is repeated here because it was never published. Consider the translation of a 3SAT formula into a description relative to the following type and appropriateness specification:

```
bot sub [bool,formula].
  bool sub [true,false].
  formula sub [propsymbol,conj,disj,neg,trueform,falseform].
    propsymbol sub [truepropsym,falsepropsym].
    conj sub [trueconj,falseconj1,falseconj2].
```

```
        intro [conj1:formula,conj2:formula].
      trueconj intro [conj1:trueform,conj2:trueform].
      falseconj1 intro [conj1:falseform].
      falseconj2 intro [conj2:falseform].
    disj sub [truedisj,falsedisj]
        intro [disj1:formula,disj2:formula].
      truedisj sub [truedisj1,truedisj2].
        truedisj1 intro [disj1:trueform].
        truedisj2 intro [disj2:trueform].
      falsedisj intro [disj1:falseform,disj2:falseform].
    neg sub [trueneg,falseneg]
        intro [neg:propsymbol].
      trueneg intro [neg:falsepropsym].
      falseneg intro [neg:truepropsym].
  trueform sub [truepropsym,trueconj,truedisj,trueneg].
  falseform sub [falsepropsym,falseconj1,falseconj2,
                 falsedisj,falseneg].
```

The resulting description is always non-disjunctive, since logical disjunction is encoded in subtyping. Asking whether a formula is satisfiable then reduces to asking whether this description conjoined with `trueform` is satisfiable. Every type is normal except for `truedisj`, for which the combination, DISJ1:falseform $\wedge$ DISJ2:falseform, is not attested in either of its subtypes. Enforcing subtype covering on this one deranged type is the sole source of complexity for this problem.

4.2. **Practical Enforcement of Subtype Covering.** Instead of enforcing subtype covering along with type inferencing, an alternative is to suspend constraints on feature structures that encode subtype covering restrictions, and conduct type inferencing in their absence. This restores tractability at the cost of rendering type inferencing sound but not complete. This can be implemented very transparently in systems like ALE that are built on top of another logic programming language with support for constraint logic programming such as SICStus Prolog. In the worst case, an answer to a query to the grammar signature may contain variables with constraints attached to them that must be exhaustively searched over in order to determine their satisfiability, and this is still intractable in the worst case. The advantage of suspending subtype covering constraints is that other principles of grammar and proof procedures such as SLD resolution, parsing or generation can add deterministic information that may result in an early failure or a deterministic set of constraints that can then be applied immediately and efficiently.

Given a diagnosis of which types in a signature are deranged (discussed in the next section), suspended subtype covering constraints can be implemented for the SICStus Prolog implementation of ALE by adding relational attachments to ALE's type-antecedent universal constraints that will suspend a goal on candidate feature structures with deranged types such as `verb` or `truedisj`. The suspended goal unblocks whenever the deranged type or the type of one of its appropriate features' values is updated to a more specific subtype, and checks the types of the appropriate features' values. Of particular use is the SICStus Constraint Handling Rules (CHR, Frühwirth and Abdennadher 1997) library, which has the ability not only to suspend, but to suspend until a particular variable is instantiated or even bound to another variable. This is the powerful kind of mechanism required to check these constraints efficiently, i.e., only when necessary. Re-entrancies in a Prolog term encoding of feature structures, such as the one ALE uses (Penn 1999a), may only show up as the binding of two uninstantiated variables, and re-entrancies are often an important case where these constraints need to be checked.

We can create a constraint, `subtype_cover/2`, whose first argument is a deranged type. For each such type, we can add three kinds of constraint handling rules. The first retrieves the current type of a feature structure from its term encoding and compares it to a deranged type that it used to have.

```
subtype_cover(Type,FS) <=>
   true & (type_index(FS,FSType),FSType \== Type)
 | true.
```

In the case of our `verb` example, `verb` is deranged, so the following constraint handling rule is generated:

```
subtype_cover(verb,FS) <=>
   true & (type_index(FS,FSType),FSType \== verb)
 | true.
```

If a feature structure of type `verb` is refined to be a well-formed feature structure of type `aux_verb`, then it is no longer in danger (unification over the term encoding itself fails on appropriateness failures, i.e., non-empty denotations, in such a case). In fact, no maximally specific type can be deranged. If the type has changed, then if it is still deranged, it is the responsibility of the rules for the new deranged type to check it.

The second kind of rule compares a feature structure's appropriate feature values to a product of types that belong to one of the deranged type's "safe" subtypes (such as its maximally specific subtypes). If this product subsumes the types of the feature structure's current values, then it is out of danger. So for each such product, there is one rule of the form:

```
subtype_cover(Type,AppropFeatProduct) <=> true.
```

In the case of `verb`, the following rules are generated:

```
subtype_cover(verb,'$verb'(_,'$bool'(+,_),'$bool'(+,_),_))
 <=> true.
subtype_cover(verb,'$verb'(_,'$bool'(-,_),'$bool'(_,_),_))
 <=> true.
```

The first rule contains a term encoding of the feature structure:

$$\begin{bmatrix} \text{verb} & \\ \text{AUX} & + \\ \text{INV} & + \end{bmatrix}$$

Its feature values' types correspond to the appropriateness conditions for `aux_verb`. The second rule's feature values' types correspond to the appropriateness conditions for `main_verb`. In either case, if a feature structure of type `verb` has these, then it is guaranteed to have a maximally specific extension and thus be well-formed.

Finally, it could be that the feature structure already has no maximal extensions, or only one maximal extension. In the former case, we should fail, and in the latter case, we should extend the feature structure since that is the only possible extension it can have. The following rule uses a guard to count how many extensions are still consistent with the current feature structure: 0, 1 or more than 1. If there is more than 1, it fails, resuspending the constraint — we may still need to check later because it could lose consistency with one of those extensions. If it has 0 or 1, the guard succeeds and the body of the rule fails or extends the feature structure through unification as necessary.

```
subtype_cover(Type,FS)
               <=> true & stc_unify_test(Type,FS,N)
                   | ( N == 0 -> fail
                     ; % N == 1
                       stc_unify_product(Type,FS)).
```

4.3. **Detecting Deranged Types.** The detection of deranged types themselves is also a potential problem. This is something that needs to be detected at compile-time when subtype covering constraints are generated, and as small changes in a partial order of types can have drastic effects on other parts of the signature, incremental compilation of the grammar signature itself can be extremely difficult. This means that the detection of deranged types must something that can be done very quickly, as it will normally be done repeatedly during the development of a grammar.

A naive algorithm would be, for every type, to expand the product of its features' appropriate values into the set, A, of all possible maximally specific products, then to do the same for each of the type's maximally specific subtypes, forming sets B_i, and then to remove the products in the B_i from A. If any maximally specific products remain in $A \backslash (\cup_i B_i)$, then the type is deranged. If the maximum number of features appropriate to any type is a, and there are t types in the signature, then the cost of this is dominated by the cost of expanding the products, t^a, since in the worst case all features could have $\perp$ as their appropriate value.

A less naive algorithm would treat normal (non-deranged) subtypes as if they were maximally specific when doing the expansion. This works because the products of appropriate feature values of normal types are, by definition, covered by those of their own maximal subtypes. Maximally specific types, furthermore, are always normal and do not need to be checked. Atomic types (types with no appropriate features) are also trivially normal.

It is also possible to avoid doing a great deal of the remaining expansion, simply by counting the number of maximally specific products rather than enumerating them. The resulting algorithm is thus:

```
Topologically sort the types of the signature
  from maximally specific down to bot;

For each type, t, in topological order do
  /* special cases */
  if t is maximal or atomic then
     1) declare that t is normal;
     2) tabulate normals(t) = {t};
     3) tabulate maximal(t) = number of maximal subtypes of t;
     4) continue
  else do
    /* find minimal normal subtypes of t */
    N := {};
    for each immediate subtype, s, of t do
      N := union(N,normals(s));
```

```
    od;

    /* calculate the number of maximally specific products
       of feature values subsumed by t's appropriate features */
    d : = num_max_products(t);

    /* subtract the number of those products subsumed by the
       minimal normal subtypes of t */
    for each i from 1 to |N| do
      for each subset, S, of N of cardinality i do
        lub := LUB(S);
        if lub is top (does not exist) then continue
        else do
          l := num_max_products(lub);
          d := d + (-1)^i * l;
        od;
      od;
      if lub did not exist for any S of cardinality i
        then break;
    od;

    if d then
      1) declare that t is deranged;
      2) tabulate normals(t) = N;
      3) tabulate maximal(t) = number of maximal subtypes of t;
      4) continue
    else
      1) declare that t is normal;
      2) tabulate normals(t) = {t};
      3) tabulate maximal(t) = number of maximal subtypes of t;
      4) continue;
od;

num_max_products(t)
  d := 1;
  for each feature, f, appropriate to t do
    d := d * maximal(appropriate_value(f,t));
  return d;
```

`num_max_products` can also be tabulated. At each cardinality, we either subtract or add products to the running total in order to make up for products that were respectively added or subtracted more than once because they are subsumed by the feature values of more than one immediate subtype of `t`. A similar trick can be used to calculate `maximal`.

The complexity of this approach, in practice, is much better. There are, of course, an exponential number of subsets, S, but now the dominant term is $t2^b$, where b is the maximum subtype branching factor in the signature. In the LinGO grammar, a large-scale LTFS-based English grammar recently developed at Stanford University, for example, t is 8092, the average a is 4.8 overall, and 5.1 among non-atoms, but the average b is only 1.7 overall. Moving t to a strictly linear coefficient can result in a substantial savings.

5. Conclusion

It has been shown that meet semi-lattice closure can be computed in cubic time, after a separate closure for unique introduction, which runs in linear time in both the number of features and types. Subtype covering results in NP-complete non-disjunctive type inferencing, but the postponement of these constraints using constraint handling rules can often hide that complexity in the presence of other principles of grammar. There is also a fairly efficient algorithm for determining which types, the deranged types, require these special constraints.

References

Aït-Kaći, H., R. Boyer, P. Lincoln, and R. Nasr (1989). Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems 11*(1), 115–146.

Carpenter, B. (1992). *The Logic of Typed Feature Structures.* Cambridge.

Carpenter, B. and P. King (1995). The complexity of closed world reasoning in constraint-based grammar theories. In *Fourth Meeting on the Mathematics of Language*, University of Pennsylvania.

Carpenter, B. and G. Penn (1996). Compiling typed attribute-value logic grammars. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technologies*, pp. 145–168. Kluwer.

Davey, B. A. and H. A. Priestley (1990). *Introduction to Lattices and Order.* Cambridge University Press.

Fall, A. (1996). *Reasoning with Taxonomies.* Ph. D. thesis, Simon Fraser University. Repaginated as a single-spaced document with minor corrections, and available from **http://www.cs.sfu.ca/cs/people/GradStudents/fall/personal/pub/thesis.ps**.

Frühwirth, T. and S. Abdennadher (1997). *Constraint-Programmierung.* Springer Verlag.

Gerdemann, D. (1995). Open and closed world types in nlp. In J. Kilbury and R. Wiese (Eds.), *Integrative Ansätze in der Computerlinguistik: Proceedings der 5. Fachtagung der Sektion Computerlinguistik der DGfS*, pp. 25–30.

Gerdemann, D. and P. J. King (1994). The correct and efficient implementation of appropriateness specifications for typed feature structures. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94).*

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

King, P. and T. Goetz (1993). Eliminating the feature introduction condition by modifying type inference. Technical Report 31, Sonderforschungsbereich 340 (SFB 340), Tübingen.

Penn, G. (1999a). An optimised prolog encoding of typed feature structures. Technical Report 138, Sonderforschungsbereich 340, Tübingen.

Penn, G. (1999b). A quasi-ring construction for compiling attributed type signatures. Technical Report 141, Sonderforschungsbereich 340 (SFB 340), Tübingen.

Pollard, C. and I. Sag (1987). *Information-based Syntax and Semantics.* Number 13 in CSLI Lecture Notes. CSLI Publications.

Pollard, C. and I. Sag (1994). *Head-driven Phrase Structure Grammar.* Chicago.