Arbeitspapiere des SFB 340
Bericht Nr. 141, July 1999

A QUASI-RING CONSTRUCTION FOR COMPILING ATTRIBUTED TYPE SIGNATURES

GERALD PENN

ABSTRACT. This paper identifies a misconception about the mathematical structure underlying subsumption matrices and their transitive closure, the fundamental compilation technique for efficient processing with partially ordered sets for the past ten years. It corrects it in a way that can be extended to the compilation of type hierarchies augmented with feature declarations and appropriateness conditions. These have been argued to be intended by linguists (King 1989), and to be valuable for processing efficiently with typed feature structures (Carpenter 1992; Carpenter and Penn 1996).

CONTENTS

1. INTRODUCTION

Typed feature structures (Carpenter 1992) and feature terms (Smolka 1988) have proven to be flexible data structures for representing knowledge in linguistic domains, both because of their use of named attributes

ISSN 0947-6954/99

for arguments, which allow for terse descriptions of potentially large structures, and because they incorporate inclusional polymorphism into their type systems, which allows for a logical control over non-deterministic search through under-specification. The most important, and most common operation performed on typed feature structures is unification, the consistent combination of information from two often at least partially under-specified structures. That operation consists of the consistent respective combination of matching feature values, a recursive operation in the general case, and the consistent combination of types.

Efficiently unifying two types, which corresponds to finding the join of two types in a partially ordered set of types, is thus of central importance for ensuring the efficient processing of typed feature structures in general. Very broadly speaking, three ways have been proposed to compute encodings of types in order to conduct unification efficiently: table lookup, term encodings, and bit-vector encodings. Term encodings attempt to reduce unification in the object domain into unification in some other domain, such as unification of Prolog terms (Mellish 1988; Mellish 1992), or of compact encodings of Prolog terms (Fall 1996). Bit-vector encodings typically attempt to reduce unification to one or more bit-wise operations, such as AND or OR.

Type unification is usually defined implicitly by defining the immediate subsumption relation over a set of types; so all three ways involve computing a transitive closure at compile-time over that relation to find the implied type unification function. The present study provides a uniform way of looking at this closure as well as the other kinds of compilation necessary to encode the more general class of type signatures used for logic programming over typed feature structures as presented in Carpenter (1992). In particular, all of the compilation required for these type signatures can be reduced to matrix arithmetic.

The next section reviews the reduction of compiling type unification to the transitive closure of boolean matrices. Section 3 discusses the mathematical structure that must underly elements of these matrices for the proposed method to work, arguing that the boolean semi-ring is the proper one, thus correcting an earlier misconception that sub-cubic matrix multiplication algorithms on the boolean ring are applicable. Section 4 shows how to construct a quasi-ring that can support matrix multiplication from any finite meet semi-lattice (Definition 6). Section 5 then reduces all of the compilation steps necessary for efficient processing relative to an attributed type signature to matrix operations based on this construction.

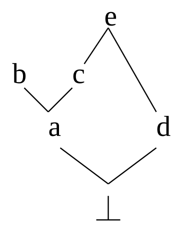

FIGURE 1. An example type hierarchy.

2. SUBSUMPTION MATRICES AND TRANSITIVE CLOSURE

The use of boolean matrix multiplication to compute transitive closures of graphs extends as far back as Prosser (1959); and was improved on by Warshall's famous transitive closure algorithm (Warshall 1962).

The application of reflexive-transitive closure to lattice representation theory extends back to the seminal paper by Aït-Kaci et al. (1989), which proposed the baseline bit-vector encoding of partially ordered types by which all others are now measured, namely one that uses n bits per code, where n is the number of types in the partial order.

DEFINITION 1. *Given a partially ordered set,* $\langle P, \sqsubseteq \rangle$*, and a total ordering of* P*'s elements,* $p_1, p_2, \ldots, p_{|P|}$*, the* subsumption matrix*,* S*, of* P *is a* $|P| \times |P|$ *boolean matrix, where* $S_{i,j} = 1$ *iff* $p_i \sqsubseteq p_j$*.*

Aït-Kaci's et al. (1989) observation was that we can use the ith row of S to encode the type p_i, with unification corresponding to bit-wise AND. An example partial order of types is given in Figure 1. Here, $\perp$ is the most general type, and more specific subtypes are situated above their more general supertypes. Its subsumption matrix is given in Figure 2. The AND of the rows for a and d yields the row for e, for example.

We can build a base subsumption matrix, H, in the same way, by using the base, or immediate, subsumption relation rather than true subsumption. In practice, this is what is specified in type hierarchy declarations, with reflexive and transitive closure being implicit. The question then becomes how to obtain S from H. The base subsumption matrix for Figure 1 is given in Figure 3.

Two refinements are given in the same paper to produce more compact encodings. One allocates bits only for maximal types and unary branching types. The other allocates separate group codes for subsets that are "modular" in the way that base subsumption connects them to the rest of the network, which reduces the overall size of codes, although it makes

	$\perp$	a	b	c	d	e
$\perp$	1	1	1	1	1	1
a	0	1	1	1	0	1
b	0	0	1	0	0	0
c	0	0	0	1	0	1
d	0	0	0	0	1	1
e	0	0	0	0	0	1

FIGURE 2. The subsumption matrix of Figure 1.

	$\perp$	a	b	c	d	e
$\perp$	0	1	0	0	1	0
a	0	0	1	1	0	0
b	0	0	0	0	0	0
c	0	0	0	0	0	1
d	0	0	0	0	0	1
e	0	0	0	0	0	0

FIGURE 3. The base subsumption matrix of Figure 1.

the actual unification operation more complicated than bit-wise AND. Ganguly et al. (1994) provide another good one that places the burden of extra bits on types that inherit from multiple supertypes, to achieve a comparable improvement without the modularity restriction. These and others are summarised in Fall (1996). In every case, however, these refinements are simply allocating bits more sparingly along the way to deriving a reflexively-transitively closed matrix like S from H.

For the naive encoding, one way to achieve that derivation is a reflexive-transitive closure, by directly filling in the diagonal of H with 1's (reflexive) and multiplying the result by itself until it reaches the fixed point, S (transitive). This fixed point is obviously reached after no more than $|P|$ iterations. By re-using the results of previous multiplications, one can attain it in $\lceil log|P|\rceil$ iterations. The first refinement presented in Aït-Kaci et al. (1989) also uses matrix multiplication at one step to compute the reflexive-transitive closure of the symmetric closure of H. Aït-Kaci et al. (1989) claim that those matrix multiplication steps should be conducted in the boolean ring of $|P| \times |P|$ bit-matrices, and that they can be performed using an efficient sub-cubic algorithm such as Strassen's algorithm (Strassen 1969).

3. Rings, Quasi-Rings and Semi-Rings

There are actually two closely related boolean algebras with two operations each (roughly speaking, candidates for ring-hood). They are $B_{XOR} = \langle\{0,1\}, XOR, AND, 0, 1\rangle$, where $\oplus$ corresponds to XOR, and $B_{OR} = \langle\{0,1\}, OR, AND, 0, 1\rangle$, where $\oplus$ corresponds to OR. In order to see how these differ in practice, we need to define some basic structures:

Definition 2. *A* monoid *is a structure $\langle P, \cdot, e\rangle$ such that:*

- *P is a set closed under $\cdot$,*
- *$\cdot$ is an associative binary operator on P, and*
- *$e \in P$ is an identity for $\cdot$.*

Definition 3. *A* quasi-ring *is a structure $\langle P, \oplus, \otimes, \bar{0}, \bar{1}\rangle$, such that:*

- *$\langle P, \oplus, \bar{0}\rangle$ is a monoid,*
- *$\langle P, \otimes, \bar{1}\rangle$ is a monoid,*
- *$\bar{0}$ is an annihilator of $\otimes$: $a \otimes \bar{0} = \bar{0}$ for all $a \in P$,*
- *$\oplus$ is commutative, and*
- *$\otimes$ distributes over $\oplus$.*

Definition 4. *A* ring *is a quasi-ring with an additive inverse, i.e., for all $a \in P$, there exists $b \in P$ such that $a \oplus b = b \oplus a = \bar{0}$.*

If P is a quasi-ring, then multiplication of matrices is well-defined and has certain nice properties, such as associativity and the existence of an identity (in fact, it is also a quasi-ring).

Definition 5. *Given a quasi-ring, $Q = \langle P, \oplus, \otimes, \bar{0}, \bar{1}\rangle$, an $m \times n$ matrix, A, over Q, and an $n \times p$ matrix, B, over Q, then $A \cdot B$* (matrix multiplication) *is the $m \times p$ matrix, C, over Q such that:*

$$c_{i,j} = \bigoplus_{k=1}^{n} a_{i,k} \otimes b_{k,j}.$$

B_{XOR} and B_{OR} are both boolean quasi-rings.

B_{XOR} is also a boolean ring; but B_{OR} is not. B_{OR} is a closed boolean *semi-ring*, however, which, among other properties, means that OR is idempotent, i.e., that $1 \oplus 1 = 1$. This is vital for ensuring that matrix multiplication can compute a transitive closure, since transitively closed subsumption should not be “turned off” by immediate subsumption chains on more than one subtyping branch. So we need idempotence in the underlying boolean quasi-ring. XOR is not idempotent; so the boolean ring is not the correct structure to use.

In addition, Strassen's algorithm can only compute matrix multiplication over true rings — not over all quasi-rings or closed semi-rings — because it requires the existence of an additive inverse; so Strassen's algorithm will not work with B_{OR}.

We could embed matrices over the boolean quasi-ring, B_{OR}, into a proper ring, and find some way of restoring the result in the original structure. Such an embedding can be found in the integers, for example, as shown in Figure 4. This would allow us to use Strassen's algorithm.

$$\begin{tikzcd} Z \arrow[r] & Z \arrow[d, "\begin{cases} 0 \mapsto 0, \\ x \mapsto 1 \; o.w. \end{cases}"] \\ B_{OR} \arrow[u, "\iota"] & B_{OR} \end{tikzcd}$$

FIGURE 4. An embedding of B_{OR} into Z for ring multiplication.

Presumably, this is what systems that purport to use either the naive encoding or its first refinement in Aït-Kaci et al. (1989), e.g., the TDL system (Krieger and Schäfer 1994), have actually been doing.

Even then, Strassen's algorithm belongs to a class of sub-cubic matrix multiplication algorithms that are only well-suited to multiplying very large, very dense matrices. While subsumption matrices can be very large, they are never very dense. Because they encode a partial order (or some approximation of its transitive closure), for every non-diagonal 1, corresponding to $p \sqsubseteq q$, for $p \neq q$, there is a non-diagonal 0, corresponding to $q \not\sqsubseteq p$. As a result, no more than $(|P|^2 + |P|)/2$ positions are non-zero. So even if we use the above embedding to rescue boolean ring-hood, Strassen's algorithm is still the wrong algorithm for the job. Either Warshall's classical algorithm must be used,[1] or specialised spare matrix multiplication algorithms could be applied, most of which only require

[1] Warshall (1962) claims his algorithm increases "slightly faster" than quadratically; but it is known to be tightly bounded at cubic. Although there is no discussion of the choice of algebra, it also uses B_{OR}. Warshall's algorithm and Floyd's extension of it to the all-pairs-shortest-path problem are both instances of a general dynamic programming algorithm on closed semi-rings (Aho, Hopcroft, and Ullman 1974). Aït-Kaci et al. (1989) also present a non-standard algorithm for transitive closure by way of introducing their first refinement that (apparently unwittingly) uses B_{OR} rather than the boolean ring. To the present author's knowledge, the fastest known general multiplication algorithm for matrices over quasi-rings is still $O(n^3)$.

By contrast, nearly every algorithm in the class of sub-cubic algorithms to which Strassen's algorithm belongs requires an underlying ring. The only exception of which the present author is aware is Shamir's randomised boolean matrix multiplication algorithm (Cormen, Leiserson, and Rivest 1990), which can conduct multiplications an underlying quasi-ring, since the rows and columns of H and S can be sorted so that they are upper-triangular sparse matrices. In particular, it can be shown (Aho, Hopcroft, and Ullman 1974) that within any closed semi-ring, the transitive closure of an upper-triangular matrix is:

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}^* = \begin{pmatrix} A^* & A^*BC^* \\ 0 & C^* \end{pmatrix}$$

for any square sub-matrix, A. Using the right-hand side to compute the transitive closure cuts the number of required arithmetic operations by 75%, although it has the same asymptotic complexity.

The first refinement for computing "compact" bit-vector codes in Aït-Kaci et al. (1989) is effectively a sparse, or at least sparser, matrix encoding of S suitable for the component-wise multiplication (AND) of its rows. In particular, Aït-Kaci's et al. (1989) observation in their first refinement can be restated as: every column corresponding to a meet-reducible type, $t = u \sqcap v$, where $u \neq t$ and $v \neq t$, can be reconstructed by component-wise multiplying the columns corresponding to u and v. Given rows i and j, $s_{i,t} \otimes s_{j,t}$ is thus $(s_{i,u} \otimes s_{i,v}) \otimes (s_{j,u} \otimes s_{j,v})$. If the underlying quasi-ring is commutative, as is the case for B_{OR}, this equals $(s_{i,u} \otimes s_{j,u}) \otimes (s_{i,v} \otimes s_{j,v})$, the product of the columns for u and v in the result. So an encoding that must preserve only component-wise multiplication of rows of S can dispense with the columns corresponding to meet-reducible types altogether. What remain are the meet-irreducible types, which are exactly the maximal types and unary-branching types in the partial orders that Aït-Kaci et al. (1989) consider.

4. An Extensible Quasi-Ring Construction

Now that we know that we need a quasi-ring with idempotence, we can consider which kinds of quasi-rings would be most convenient. The boolean quasi-ring, B_{OR}, suffices for processing with simple type hierarchies. We began this study with thinking about how to process with typed feature structures, however. Feature logics can improve the run-time efficiency of unification over their terms by imposing further restrictions on which features can occur with a given type, or restrictions on which subsets of types can bear a particular feature. This may, in turn, entail other restrictions on the type hierarchy itself. The restrictions that govern the

for matrices over B_{OR} using matrices over B_{XOR} with a probability of at least $1-1/n^k$ for any constant $k > 0$ in $O(n^{lg7} lgn)$ time. The price, of course, is the small chance of error. The fastest known "sure-fire" multiplication algorithm for matrices over proper rings is $O(n^{2.376})$ (Coppersmith and Winograd 1990).

combination of features with types are called *appropriateness conditions* and the type hierarchy plus these conditions is called a *type signature*. For the sake of example, we will consider for the rest of this paper the feature logic of Carpenter (1992), a widely used and very restrictive feature logic. Other logics can avail themselves of the techniques discussed here or additional ones as far as they are applicable to their own restrictions. We will also consider only the correlates of the naive encoding of Aït-Kaci et al. (1989) — similar improvements for compactness and speed can be made on these as well.

For Carpenter (1992), type hierarchies are finite meet semi-lattices. These are partial orders in which, for all pairs $a, b \in P$, their meet, $a \sqcap b$, also exists in P. In the finite case, this is equivalent to saying that the join, or unification, $a \sqcup b$, exists for every pair of consistent types, $a, b \in P$, and that P has a least, or most general, element, here called $\bot$. Requiring type hierarchies to be finite meet semi-lattices effectively eliminates a potential source of disjunction inherent to unification in general partial orders. Bit-vector encodings capture disjunctions of types for free (as the OR of the disjuncts), but in more restricted feature logics such as Carpenter's (1992), that disjunction may also delay the enforcement of appropriateness conditions, which exist to prune ill-formed structures. The reason for this is that individual disjuncts may have different appropriateness conditions — different appropriate features, for example. Instead of simply disjoining the types, the appropriateness conditions common to all of the disjuncts should be factored out, effectively creating a new meet in the type hierarchy.

We can think of 0 and 1 in B_{OR} as constituting a very small type hierarchy, as shown in Figure 5. If $\top$ corresponds to 1, and $\bot$, to 0, then

$$\begin{array}{c}\top\\ |\\ \bot\end{array}$$

FIGURE 5. The boolean type hierarchy.

unification in this hierarchy corresponds to boolean OR. We can also write this as in Figure 6, in which the trivial type hierarchy, consisting of just $\bot$, has been bottom-lifted to add a new bottom, $\underline{\bot}$. Because bottom-lifting preserves meet-semi-latticehood, we can trivially extend $\sqcup$ to any $P \cup \{\underline{\bot}\}$ where P is a finite meet semi-lattice. Now, we need something to correspond to AND:

$$a \mathbin{\underline{\sqcup}} b = \begin{cases} \underline{\bot} & \text{if } a = \underline{\bot} \text{ or } b = \underline{\bot} \\ a \sqcup b & \text{otherwise} \end{cases}$$

DEFINITION 6. *Let* $\langle P, \sqsubseteq \rangle$ *be a type hierarchy. Then* $Q(P) = \langle P \cup \{\underline{\bot}\}, \sqcup, \underline{\sqcup}, \underline{\bot}, \bot \rangle$, *is the* quasi-ring induced by P.

Notice that we can define this for all P, not just the trivial type hierarchy, because in all type hierarchies in Carpenter (1992), $\sqcup$, and therefore its extension to $P \cup \{\underline{\bot}\}$ and to $\underline{\sqcup}$, are total functions, and there is a least element. As can easily be verified:

PROPOSITION 1. *For all type hierarchies P with a greatest element, $Q(P)$ is a quasi-ring.*

The existence of a greatest element ensures that $\sqcup$ and $\underline{\sqcup}$ are closed in $P \cup \{\underline{\bot}\}$. Without loss of generality, we can assume that the greatest element, $\top$, does not explicitly appear in P, and that it does not occur anywhere else in the signature, e.g., in appropriateness conditions. $\top$ can be smashed onto any such P, and is typically implemented as type unification failure in the original signature. Figure 7 shows the type hierarchy in Figure 1 $\underline{\bot}$-lifted and $\top$-smashed to form its quasi-ring.

The benefit of using $Q(P)$ is that it allows us to generalise to other computations on signatures that require matrices with types in them rather than just 0s and 1s. The subsumption matrix of P can still be constructed using $\underline{\bot}$ and $\bot$ in place of 0 and 1, respectively. The next section presents all of the major compile-time operations that must be performed for processing with the feature logic of Carpenter (1992), such as in ALE (Carpenter and Penn 1996), along with how they can be performed using matrices and vectors over this quasi-ring construction.

$$\begin{array}{c} \bot \\ | \\ \underline{\bot} \end{array}$$

FIGURE 6. The trivial type hierarchy lifted to produce the boolean hierarchy.

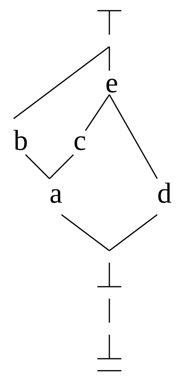

FIGURE 7. The quasi-ring constructed from Figure 1.

5. TYPE AND APPROPRIATENESS RESTRICTIONS IN CARPENTER (1992)

5.1. **Subtyping Cycles.** The first two checks we need to make still use only $\underline{\perp}$ and $\perp$, i.e., the former 0 and 1. Unification ($\sqcup$) over all of $Q(P)$ is not necessarily well-defined until we can guarantee that P is a meet semi-lattice.

We first need a way of checking that a type hierarchy we are given is legitimate. We can assume that we are given a candidate type hierarchy in the form of its base subsumption matrix, H. We then reflexively and transitively close H as described above, to obtain its subsumption matrix, S. The first check we need to make is that P is a partial order, by checking for anti-symmetry.

PROPOSITION 2. *If there exist* $1 \leq i, j \leq |P|$, $i \neq j$ *such that* $S_{i,j} = S_{j,i} = \perp$, *then* P *is not a partial order.*

An easier way to check this is to carry out the construction of S by embedding H as a matrix $\bar{H}$ in Z, as in Figure 4, and building its transitive closure, $\bar{S}$. The embedding method described above can still be used because H and S still contain only $\underline{\perp}$ and $\perp$.

PROPOSITION 3. *If for any* $1 \leq i \leq |P|$, $\bar{S}_{i,i} > 1$, *then* P *is not a partial order.*

A diagonal entry of more than 1 indicates that reflexive closure could have been achieved at that entry without explicitly setting it to 1 by transitively closing over a pre-order with a symmetry.

Once we have S, it is easy to express which types are consistent:

DEFINITION 7. *The* join matrix *of* P *is* $J = S \cdot S^T$*, where* S^T *is the transpose of* S*.*

PROPOSITION 4. t_i *and* t_j *are consistent iff* $J_{i,j} = \underline{\perp}$*.*

Note that J is always symmetric, i.e., $J_{i,j} = J_{j,i}$ for all i,j.

5.2. **Meet Semi-latticehood.** We can check for meet-semi-latticehood by exploiting the alternative definition of meet semi-lattices in the finite case and using the rows of S as codes of types in P, in the manner of Aït-Kaci et al. (1989):

PROPOSITION 5. *Let* $\vec{\underline{\sqcup}}$ *be the component-wise application of* $\underline{\sqcup}$ *to two vectors of elements from* $Q(P)$*. A partial order* P *is a finite meet semi-lattice iff for all* $i \leq i,j \leq |P|$*, if* $J_{i,j} = \underline{\perp}$ *then there exists a (unique)* $1 \leq k \leq |P|$ *such that* $S_i \vec{\underline{\sqcup}} S_j = S_k$*.*

If two types are consistent, then their codes intersect to produce the code of another type. If P is not a finite meet semi-lattice, then the unification of some codes will be the disjunction of two or more other codes, which will not be found as a single row in S.

Along the way to verifying this, we can build the unification table for P:

DEFINITION 8. *The* unification table *of* P*,* U*, is the join matrix of* P*, with each* $\underline{\perp}$ *replaced by* $\top$*, and each* $\perp$ *replaced by* t_k*, where* k *is as stated in Proposition 5.*

PROPOSITION 6. *For all* $1 \leq i,j \leq |P|$*,* $U_{i,j} = t_i \sqcup t_j$*.*

That S_k in Proposition 5 corresponds to the unification of S_i and S_j follows from the correctness of Aït-Kaci's et al. (1989) method, proven in that paper.

5.3. **Feature Introduction.** In Carpenter (1992), appropriateness conditions take the form of a restriction on which types can bear a particular feature, and which types that feature's value can have.

DEFINITION 9. *A* type signature *is a quadruple,* $\langle P, \sqsubseteq, F, A \rangle$*, where* $\langle P, \sqsubseteq \rangle$ *is a finite meet semi-lattice,* F *is a set of features, and* $A : F \times P \rightharpoonup P$ *is a partial function such that:*

- *(Feature Introduction)*
  for every $f \in F$*, there is a most general type* $Intro(f) \in P$ *such that* $A(f, Intro(f))\downarrow$*, and*

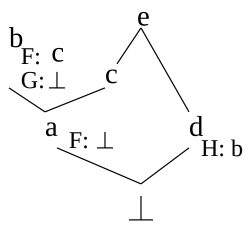

FIGURE 8. An example type signature.

- *(Upward Closure / Right Monotonicity)*
  if $A(f,t)\downarrow$ *and* $t \sqsubseteq u$, *then* $A(f,u)\downarrow$ *and* $A(f,t) \sqsubseteq A(f,u)$.

Feature Introduction eliminates a source of disjunction in inferring types from feature names in descriptions. By having a unique introducing type, it is possible to apply appropriateness in the other direction immediately to infer some of the other features that must also exist, along with the types of their values.

In practice, signature declarations specify appropriateness only by declaring (1) where a feature is introduced, along with the type its value must have and (2) where a feature takes on a value whose type cannot be inferred to be the least type that satisfies Upward Closure and/or Right Monotonicity given its value on supertypes. The type to which a feature's value is constrained is called a *value restriction.* Figure 8, for example, is Figure 1 with appropriateness declarations added. F is appropriate to a, for example, with value restriction, $\perp$. Because F is appropriate to a, it is also appropriate to b, c and e, although b refines the value restriction to c. b has two appropriate features because it also introduces G. e has two appropriate features by Upward Closure because H was introduced at d.

In order to enforce this view of appropriateness conditions on P, we can build a matrix over $Q(P)$ for these declarations:

DEFINITION 10. *Given a set of types,* P, *a set of features,* F, *and a set of appropriateness declarations* $D \subseteq F \times P \times P$, *where* D *is a partial function, the* value declaration matrix *for* D *over* F *and* P *is a* $|P| \times |F|$ *matrix,* V, *over* $Q(P)$, *in which* $V_{i,j} = u$, *if there exists a* $u \in P$ *such that* $(f_j, t_i, u) \in D$, *and* $V_{i,j} = \perp$ *if there is no such* u.

The uniqueness of u, when it exists, is guaranteed by the fact that D is a partial function. The value declaration matrix for Figure 8 is shown in Figure 9. The entry for type d, feature H is b because H is declared as appropriate to d with its value restricted to b.

	F	G	H
$\perp$	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$
a	$\perp$	$\underline{\perp}$	$\underline{\perp}$
b	c	$\perp$	$\underline{\perp}$
c	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$
d	$\underline{\perp}$	$\underline{\perp}$	b
e	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$

FIGURE 9. The value declaration matrix of Figure 8.

	F	G	H
$\perp$	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$
a	$\perp$	$\underline{\perp}$	$\underline{\perp}$
b	c	$\perp$	$\underline{\perp}$
c	$\perp$	$\underline{\perp}$	$\underline{\perp}$
d	$\underline{\perp}$	$\underline{\perp}$	b
e	$\perp$	$\underline{\perp}$	b

FIGURE 10. The value restriction matrix of Figure 8.

Notice that $\underline{\perp}$ is being used here as a place-holder for pairs of type, t and feature, F, for which F is not appropriate to t. We use $\underline{\perp}$ rather than $\top$ so that feature introduction (with a value restriction of $\perp$ or greater) still respects Right Monotonicity. This trick has applications outside type signature compilation to any task that requires a uniform view of type-feature pairs or accessible feature paths while still respecting appropriateness conditions with respect to unification, since all features are effectively appropriate to all types, albeit sometimes with $\underline{\perp}$ as the value.

DEFINITION 11. *The* value restriction matrix *of P is $R = S^T \cdot V$.*

Pre-multiplying V by the transpose of S closes the appropriateness declarations under subsumption. $V_{i,j}$ is thus something other than $\underline{\perp}$ iff feature j is appropriate to type i. The value restriction matrix of Figure 8 is shown in Figure 10. Notice that the entry for type e, feature H is also b because e inherits H from d. Using the value restriction matrix, we can then express the condition on unique feature introduction:

	F	G	H
$\perp$	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$
a	$\perp$	$\underline{\perp}$	$\underline{\perp}$
b	$\underline{\perp}$	$\perp$	$\underline{\perp}$
c	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$
d	$\underline{\perp}$	$\underline{\perp}$	b
e	$\underline{\perp}$	$\underline{\perp}$	$\underline{\perp}$

FIGURE 11. The introduction matrix of Figure 8.

DEFINITION 12. *$\delta : P \cup \{\underline{\perp}\} \to \{\underline{\perp}, \perp\}$ is the* characteristic function *for P, such that:*

$$\delta(t) = \begin{cases} \perp & \text{if } t \in P, \\ \underline{\perp} & \text{if } t = \underline{\perp} \end{cases}$$

PROPOSITION 7. *R satisfies the feature introduction restriction iff for all i, there exists a j, such that $\vec{\delta}(R_i^T) = S_j$.*

δ projects the elements of $Q(P)$ back onto the trivial quasi-ring, according to whether they belong to P. Feature introduction is satisfied if, after component-wise projection, every column of R is the same as some row of S. Rows of S encode types as the upward closed sets that they subsume. The columns of R have non-$\underline{\perp}$ values for the types to which a feature, F, is appropriate; and we know that that set is upward-closed, having left-multiplied by S^T. If that set is one of the rows of S, then that row corresponds to a minimal type, which is $Intro$(F).

Along the way, we can also compute those introducing types:

DEFINITION 13. *The* introduction matrix*, I, of P is a $|P| \times |F|$ matrix in which $I_{j,i} = V_{j,i}$ when j is the j specified for i in Proposition 7, and $\underline{\perp}$ elsewhere.*

The introduction matrix for Figure 8 is given in Figure 11. The entry for type b, feature F is $\underline{\perp}$ because, although b places a non-inferrable value restriction on F, it does not introduce F.

5.4. **Value Restriction Consistency.** Because of Right Monotonicity, join-reducible types can not only multiply inherit features, but also inherit value restrictions on the same feature from two or more different branches; and these must be consistent. Figure 12 shows an example of this. Right Monotonicity from b and c requires F to be appropriate to d with a value of both f and g. In Figure 12, this is consistent — the value of F at d

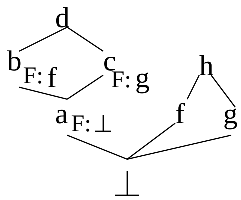

FIGURE 12. A type signature with consistent value restrictions.

	F
$\bot$	$\underline{\bot}$
a	$\bot$
b	f
c	g
d	$\underline{\bot}$
f	$\underline{\bot}$
g	$\underline{\bot}$
h	$\underline{\bot}$

FIGURE 13. The value declaration matrix of Figure 12.

must be of type h. Without h, it would not be consistent. We can use value restriction matrices to enforce this consistency check as well.

PROPOSITION 8. *The value restrictions of P are consistent iff there is no i,j for which $R_{i,j} = \top$.*

$\top$ corresponds to inconsistency in the original signature.

The value declaration matrix for Figure 12 is given in Figure 13. The value restriction matrix of Figure 12 is given in Figure 14. Without h, the entry for d would have been $\top$.

5.5. **Appropriateness Cycles.** All types in Carpenter (1992) must have a finite most general satisfier, a finite least informative feature structure of that type that respects appropriateness conditions. This means that appropriateness conditions may not conspire so as to require a feature structure of type t to have a proper substructure of type either t or a

	F
$\perp$	$\underline{\perp}$
a	$\perp$
b	f
c	g
d	h
f	$\underline{\perp}$
g	$\underline{\perp}$
h	$\underline{\perp}$

FIGURE 14. The value restriction matrix of Figure 12.

subtype of t.[2] This kind of *appropriateness cycle* can very naturally be articulated using R.

DEFINITION 14. *The* convolution matrix *of R, C, is a $|P| \times |P|$ matrix over $Q(P)$ such that $C_{i,j} = \perp$ if there exists a k such that $R_{i,k} = t_j$, and $C_{i,j} = \underline{\perp}$ otherwise.*

PROPOSITION 9. *P has an appropriateness cycle iff there exists an i such that $C^*_{i,i} = \perp$, where C^* is the (non-reflexive) transitive closure of C.*

$C_{i,j} = \perp$ means that type t_j is accessible as a substructure by some feature from structures of type t_i. By transitively closing C, we extend that accessibility to finite paths of features, and so can detect whether t_i is accessible from t_i. Because we convoluted C from R, which was upward closed by left-multiplication with S^T, we detect accessibility to subtypes of t_i as well. The convolution of Figure 14 is given in Figure 15 (0 and 1 are used for readability). The entry for row b, column f is $\perp$, because f is accessible from b along the feature F. In this case, the transitive closure of C is the same as C itself, because all types that occur as value restrictions of features are atomic, i.e., they have no features of their own.

In practice, this transitive closure can be computed directly from R, without explicitly constructing C.

5.6. **Join Preservation Condition.** Not all type signatures as construed by Carpenter (1992) are statically typable. This means that there are some signatures for which the result of unifying two well-typed feature

[2]Very often, the appropriateness conditions will *allow* a subtype of t to occur, but that is not the same thing. Non-empty lists, for example, have a tail appropriate to lists, of which non-empty lists are a subtype. That makes lists potentially recursive, but lists still have finite most general satisfiers.

	⊥	a	b	c	d	f	g	h
⊥	0	0	0	0	0	0	0	0
a	1	0	0	0	0	0	0	0
b	0	0	0	0	0	1	0	0
c	0	0	0	0	0	0	1	0
d	0	0	0	0	0	0	0	1
f	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0

FIGURE 15. The convolution of Figure 14.

structures must be reverified as being a well-typed structure. Although these signatures are technically valid ones, systems that process with the logic of Carpenter (1992) normally check for static typability because of the computational cost of this run-time verification. Signatures that are not statically typable are those that do not satisfy the following condition:

DEFINITION 15. *A signature,* $\langle P, \sqsubseteq, F, A\rangle$*, satisfies the* join preservation condition *iff for all consistent* $t, u \in P$ *and* $f \in F$*:*

$$A(f, t \sqcup u) = \begin{cases} A(f,t) \sqcup A(f,u) & \text{if } A(f,t)\downarrow \text{ and } A(f,u)\downarrow \\ A(f,t) & \text{if only } A(f,t)\downarrow \\ A(f,u) & \text{if only } A(f,u)\downarrow \\ \text{anything} & \text{otherwise} \end{cases}$$

PROPOSITION 10. $\langle P, \sqsubseteq, F, A\rangle$ *satisfies the join preservation condition iff for all* i, j *for which* $J_{i,j} = \perp$*, and* $U_{i,j} = t_k$*,* $R_i \vec{\sqcup} R_j \vec{\sqcup} I_k = R_k$*.*

Viewed in terms of R, join preservation is a linear dependence condition among consistent types — recall that in $Q(P)$, $\sqcup$ corresponds to the additive operator. In a join-preserving signature, joins cannot add new information to the system, apart from introducing new features.

6. CONCLUSION

A new quasi-ring construction has been presented over which all type and feature restrictions inherent to Carpenter (1992) can be computed and enforced. Along the way, the mathematical foundations of encodings involving matrix multiplication have been corrected. The induced quasi-ring construction has wider applicability for handling computations in logics for which it is not the case that all features necessarily occur on all types, as it provides a uniform set of feature paths up to any finitely bounded length, that can be used for transparent classification or indexing of feature structures with appropriateness, for example.

This new construction can also serve as the basis for efficiently precompiling type signature information. The techniques discussed in Section 5 have all been implemented in C using a naive representation of matrices. This code, when combined with a Prolog tokeniser and parser for ALE input declarations, has yielded a type compiler that is approximately 30 times faster than ALE 3.0's compiler on the HPSG signature distributed with ALE, which has 162 types and 37 features. Further experiments remain to be conducted to see whether performance improves with an optimised sparse matrix multiplier and how well the method scales up on larger signatures.

References

Aho, A., J. Hopcroft, and J. Ullman (1974). *The Design and Analysis of Computer Algorithms.* Addison-Wesley.

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

Coppersmith, D. and S. Winograd (1990). Matrix multiplication via arithmetic progression. *Journal of Symbolic Computation 9*(3), 251–280. Special Issue on Computational Algebraic Complexity.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest (1990). *Introduction to Algorithms.* MIT Press.

Fall, A. (1996). *Reasoning with Taxonomies.* Ph. D. thesis, Simon Fraser University.

Ganguly, D., C. Mohan, and S. Ranka (1994). A space-and-time-efficient coding algorithm for lattice computations. *IEEE Transactions on Knowledge and Data Engineering 6*(5), 819–829.

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

Krieger, H.-U. and U. Schäfer (1994, November). TDL — a type description language for HPSG part 1: Overview. Technical Report RR-94-37, DFKI.

Mellish, C. (1988). Implementing systemic classification by unification. *Computational Linguistics 14*(1), 40–51.

Mellish, C. (1992). Term-encodable description spaces. In D. Brough (Ed.), *Logic Programming: New Frontiers*, pp. 189–207. Kluwer.

Prosser, R. T. (1959). Applications of boolean matrices to the analysis of flow diagrams. In *Proceedings of the 16th Eastern Joint Computer Conference*, pp. 133–138.

Smolka, G. (1988). A feature logic with subsorts. Technical Report LILOG Report 33, IBM Germany, Stuttgart.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik 14*(3), 354–356.

Warshall, S. (1962, January). A theorem on boolean matrices. *Journal of the ACM 9*(1), 11–12.