In Search of Epistemic Primitives in the English Resource Grammar

Gerald Penn and Kenneth Hoetmer

Department of Computer Science
University of Toronto

Proceedings of the HPSG03 Conference

Michigan State University, East Lansing

Stefan Müller (Editor)

2003

CSLI Publications

http://cslipublications.stanford.edu/HPSG/4/

pp. 318–337

Abstract

This paper seeks to improve HPSG engineering through the design of more terse, readable and intuitive type signatures. It argues against the exclusive use of IS-A networks and, with reference to the English Resource Grammar, demonstrates that a collection of higher-order datatypes are already acutely in demand in contemporary HPSG design. Some default specification conventions to assist in maximizing the utility of higher-order type constructors are also discussed.

1 Introduction

Types are good to have around. Not only do they assist in compile-time error detection and efficient run-time code generation, but they have the ability to reflect the grammar designer's perspective or intuitions about constructs within the grammar, simply by their presence in the source code as names/labels. They also make grammars more modular. In particular, to take the classical view on this topic from the theory of programming languages, types are what mediate communication between modules. Within the logic of typed feature structures, types can also serve as an alternative to structure sharing in complex descriptions, which can often be difficult to conceptualize or debug. This essentially enforces a kind of modularity on descriptions.

In HPSG, types are related by subtyping, otherwise known as the IS-A relation, and this relation is interpreted as subset inclusion. Many of the early attempts at developing knowledge representations in the 1960s posited perfectly reasonable relations among their concepts when viewed in isolation, but they were unsuccessful in the long term because there were no systematic principles at work across those different attempts — principles that anyone else could adhere to and by which they could understand how to reuse and modify those resources. This point was demonstrated quite convincingly by Brachman with his work on the KL-ONE system [Brachman, 1977]. This work ultimately led to a large number of conceptual reasoning systems that were able to automate certain forms of inference by exploiting the semantic properties of a small number of primitives used for organizing knowledge. Foremost among those primitives was IS-A, which has also since formed the backbone of class relationships in many object-oriented programming languages with subtyping [Ait-Kaci, 1984]. It was from this trend that HPSG took its initial inspiration in employing types with inheritance [Pollard, personal communication]. In HPSG, this same partial order defines how types inherit features.

In the intervening 20 or so years, however, there have been a number of further developments in the type systems of both description logics and the theory of programming languages that have largely passed grammar development in HPSG by — although there has been no shortage of more theoretical work on the connections among formal grammar, type theory and category theory. There has been a recent trend in HPSG towards using types (rather than features) wherever possible to encode distinctions among information states in signatures. The reasoning given has generally been consistent with the benefits mentioned above, e.g., greater efficiency without loss of elegance [Flickinger, 2000], but the down-side of this trend, that simple types can mediate only simple communication, has not received much attention or redress. HPSG's almost exclusive use of IS-A is a very simple type system indeed. The only "method," again to appeal to programming languages terminology, is unification, or the least upper bound operation.[1] In the case of the English Resource Grammar (ERG), this least upper bound is taken relative to a signature with between 2,000 and 10,000 types, depending on how one counts, and this is anything but modular to work with.

The present research programme began with an attempt to determine whether simple HPSG-style typing, while it may not be modular, has performed adequately in its other role of capturing and accentuating the intuitions of the ERG's designers. Although we were not the designers, our extensive study of the ERG type signature has forced us to conclude that is has not. In what follows, we seek to contribute the missing grammar-development-oriented perspective on the potential for using a richer set of typing constructors in HPSGs, in part by enumerating a collection of higher-order datatypes that are provably "in demand." This proof takes the form of references to (in places, simplified) examples from the ERG signature,[2] in addition to a discussion of conventions that will assist in maximizing their utility.

Specifically, we observe the informal but routine use of the following higher-order constructors among the types of the ERG:

1. parametric products,
2. optionality,
3. Smyth powerdomains,
4. purity / strictness,
5. finite domains.

We discuss several default specification conventions (not to be confused with default unification) as well as a further generalization of the proposals made by Erbach [1994] and Penn [1998] for embedding these constructions into larger type hierarchies.

There are probably other higher-order constructors worth using — we do not intend this to be a closed class. None of the constructors enumerated above, moreover, should come as a surprise. Parametric types have been used in Pollard and

[1] Breaking with the ERG literature's convention of writing more specific types below their more general supertypes, we will follow Carpenter's [1992] convention of inverting the type hierarchy, but still calling the more specific types 'subtypes.'

[2] In particular, we refer to a near-ALE-compatible port of an October, 1999 version of the ERG generated from the CSLI test suite using scripts written for this purpose by Ann Copestake. We are indebted to her for making the grammar, test suite and scripts available to us.

Sag [1994] and earlier for reasoning about lists. Finite domains were available in Erbach's ProFIT system [Erbach, 1995] and the Smyth powerdomain construction has been identified as highly relevant to signature representations of feature neutrality and coordination [Levy and Pollard, 2002]. To our knowledge, however, the closest any grammar development environment (GDE) has come to realizing these is ProFIT, and even then only as finite domains and a limited form of parametric typing without the conventions necessary (in our view) to encourage their use on a large scale. In addition, our proposal for default specification bears some similarity to Koenig and Jurafsky's [1994] proposal of "on-line type construction," and to the treatment of intersection types in the TDL system [Krieger and Schaefer, 1994].

The payoff, ultimately, will naturally include more readable and transparent grammar signatures, but also the potential to automate certain portions of the grammar development process, to increase the inferential capacity of GDEs, and thus to assist developers in understanding the grammars they build. With a few superficial exceptions, that capacity is currently limited to automatically computing the unification algebra implied by the signature. Feature structure unification is a by-product of the primitives IS-A and HAS-A (feature appropriateness), and this limitation is due to the conventional restriction of using only these two primitives in signature development.

2 The case against IS-A

As external observers examining the ERG signature after its completion, our primary sources of evidence that IS-A is not sufficient are the naming conventions applied to types and the regular or near-regular correspondences which are apparent relative to the IS-A relationships posited between those types. These sources are corroborated by discussions in the linguistics literature (as early as Pollard and Sag [1994]) of the intended significance of various types and alternative formulations. To this extent, IS-A networks have adequately conveyed to us the intentions behind the types employed, but at a cost, both in terms of the time required, and in terms of our inability to automatically deduce many of these regularities.

As a result of this study, we can cite three specific shortcomings evident in the exclusive use of IS-A in the ERG, as enumerated in the subsections below.

2.1 Lack of a uniform semantics

Problems with semantic uniformity should be readily apparent to those who have attempted to construct object models in programming languages using subsumption hierarchies. The problem centers around the difficulty of expressing relationships other than inclusion. Object-oriented programming languages differ in the remedies they provide, such as user-defined methods, *ad hoc* overloading or inheritance-based polymorphism in C++, and interface implementation in Java.

In an orthodox view of both typing and HPSG, the only remedies provided in the context of grammar development exist outside the type system itself, such as feature values with appropriateness and description-level structure sharing. A less orthodox view, both linguistically and relative to the role of typing in programming languages, suggests that types and description-level functions or relations are in fact equivalent (an instance of the so-called *Curry-Howard isomorphism*), and thus that Prolog-style relations can also mediate communication between modules, namely through their arguments. Such relations, as operationally distinct constructs, are not productively used in the ERG, and in HPSG its mention generally evokes the expectation of very costly run-time proof searches.[3] Against the backdrop of such a prejudice, higher-order typing constructors are, to our knowledge, the only available formal alternative. The use of relations will not be explored further here, but it is important to note the availability and relatedness of this option.

In the HPSG linguistics literature, on the other hand, one instead often finds a resort to informal typographical conventions that also exist outside the type system. As a very influential example on the ERG, we may consider Sag's [1997] treatment of relative clauses (Figure 1). This paper analyzes relative clauses along two separate dimensions: clausality and headedness. In other words, every subtype of *phrase* must make some claim regarding whether or not it is a clause and whether or not it has a head. The capitalization and framing of CLAUSALITY implicitly indicates that this is not a kind of phrase but a dimension of phrasal classification.

The problem with such a convention is that within the formal type system itself, there is still no multi-dimensionality. The link from *phrase* to CLAUSALITY, for example, simply looks like any other IS-A link. In addition, if CLAUSALITY and HEADEDNESS are indeed different dimensions, they should not have common subtypes such as *wh-subj-rel-cl*. That this particular join is not an ordinary upper bound but in fact a subtype of phrase that reifies a particular choice of CLAUSALITY and HEADEDNESS is not indicated with even a typographical convention.

2.2 Erosion of dimensionality

The ERG, to its credit, has eliminated the types CLAUSALITY and HEADEDNESS, but has retained the essential problem with the above analysis. In addition, these types have been replaced by types called *clause* and *headed-phrase*, which are two among the many immediate subtypes of *phrasal*, a subtype of *phrase* (Figure 2). In doing so, it is simply less apparent that phrases can be analyzed along these two independent dimensions.

Parametric typing, the use of functions that map products of types to types, circumvents this problem by allowing us to explicitly identify each of the top row of intersection types by the combination of properties it represents, e.g., *phrase(wh-*

[3] In HPSG's type system, these searches actually have a parallel in the requisite task of maximal sort resolution. This is NP-complete [Penn, 2001], and as a result, many grammars, including the ERG, have been developed with an alternative view of subtyping in mind in which this resolution is never performed.

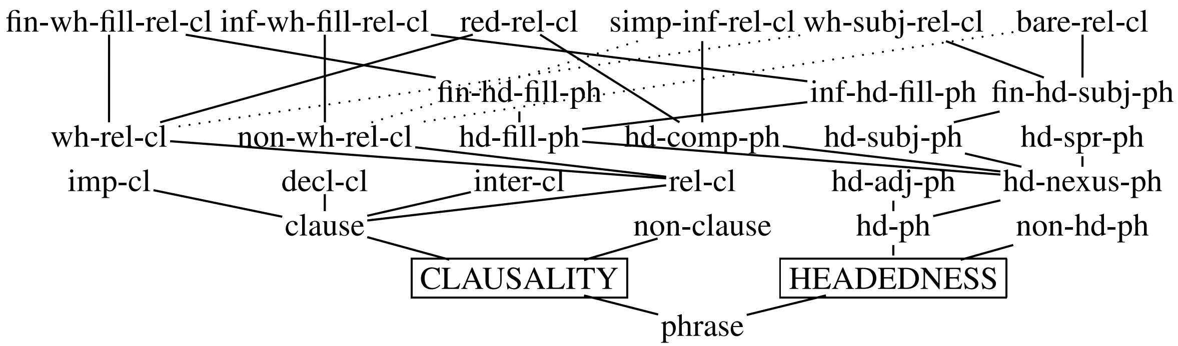


Figure 1: Dimensions of Classification of Relative Clauses.

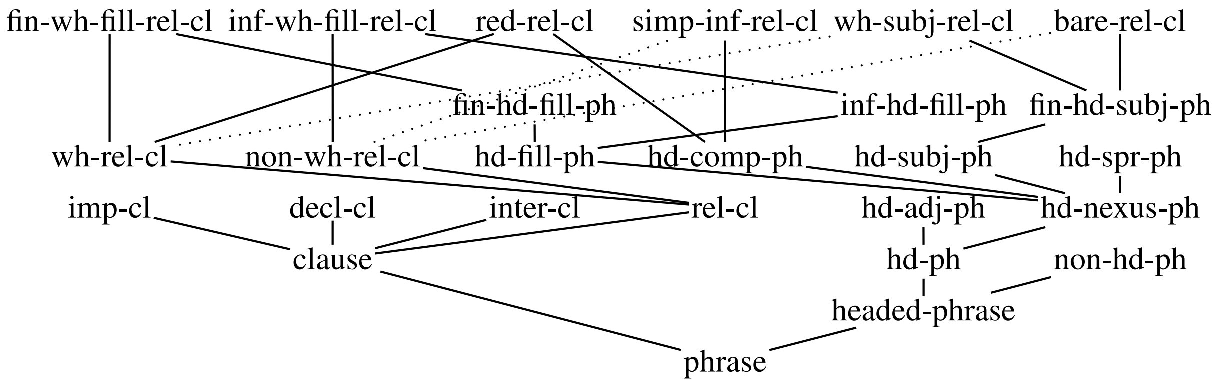


Figure 2: Erosion of Dimensionality in the Relative Clause analysis.

rel-cl, hd-subj-ph) rather than *wh-subj-rel-cl*, and define the type signature without having to explicitly enumerate all of the possible combinations. The parameters of parametric types cannot be "structure-shared" because they are only types, not feature structures, so the potentially non-modular effects of structure-sharing are still absent.

2.3 Inconsistent naming conventions

Most HPSG linguists probably realize what a *wh-subj-rel-cl* is, and the name itself does suggest that this phrasal type is a *rel-cl* and a *hd-subj-ph* (although headedness itself is not indicated), but there are other cases in the ERG where the naming conventions are far less transparent. For example:

- Order is sometimes used rather than an additional compound name. The difference between a *head-adj-ph* and a *adj-head-ph*, for example, is that the former is both *head-initial* and a *head-mod-phrase-simple*, while the latter is *head-final* and a *head-mod-phrase-simple*.

- Some would-be parameters actually appear in their negated forms, such as the subtypes, *nonque*, *nonrel* and *nonslash*, of *word*. Presumably, this choice of polarity serves to reduce the number of intersection types that would otherwise need to have been explicitly defined.

- The type *non1sg* does not actually refer to all non-first-singular person-number combinations, but only to those that are also non-third-singular. To know this, we must observe that *non1sg* is actually a subtype of *non3g* in the ERG. The name presupposes an acquaintance with English verbal inflectional patterns.

- Several different kinds of connectives are employed in names, and, because these names are simply strings, it is not always clear what their scope is. We thought we understood *1or3pl+2per+1per+non1sg*, for example, until we saw that it is a subtype of *1sg*+2per+1per+non1sg*.

- Other connectives are simply not clear in their intended meaning. *basic-cp-prop+ques-verb*, for example, has only one supertype (*verb-synsem*). This is not the same + that denotes intersection elsewhere.

With parametric types, intersection types are implicitly created, and the names of the parametric types themselves serve to better identify their decomposition and purpose. Notice that *pernum*, the base person-number combination, could just as well be *index(person,number)*, *noun(person,number)* or *verb(person,number)*, to indicate what is intended. As for *head-adj-ph* and *adj-head-ph*, there are by our count at least five independent dimensions on which phrases are being classified:

1. initial vs. final,

2. binary vs. unary,

3. headed vs. non-headed,

4. intersective vs. scopal, and

5. 'h' vs. 'n' (we have not determined what these letters stand for).

These are in addition, although not unrelated, to the more familiar distinctions among complement phrases, subject phrases, etc. of HPSG. It took us a day to determine that these were the parameters, but we can now say where an *n-adj-redrel-ph* stands with respect to all of them. Can you?

3 Higher-order constructors for the ERG

3.1 Parametric/Product types

We have already seen a few instances where parametric types seem to be called for. For the most part, we follow Penn [2000] in the formal details of extending type signatures to parametric type signatures. Formally, parametric types are functions that provide access or a means of reference to a set of types (their image) by means of argument types called *parameters* (their domain). In HPSG, the best known example is the unary parametric type, $list$. $list(\alpha)$ labels feature-structure-encoded lists in which each member is of type α.

Definition 1. *A parametric (type) hierarchy is a finite bounded-complete partial order (BCPO), $\langle P, \sqsubseteq_P \rangle$, plus an arity function, $arity : P \longrightarrow \mathsf{Nat} \cup \{0\}$, and a partial argument assignment function, $a_P : P \times P \times \mathsf{Nat} \rightarrow \mathsf{Nat} \cup \{0\}$, in which:*

- *P consists of (simple and) parametric types, and includes the most general type, $\perp$, which is simple, i.e., $arity(\perp) = 0$,*
- *For $p, q \in P$, $a_P(p, q, i)$, written $a_p^q(i)$, is defined iff $p \sqsubseteq_P q$ and $1 \leq i \leq arity(p)$,*
- *$0 \leq a_p^q(i) \leq arity(q)$, when it exists, and*
- *if $a_p^q(i) \neq 0$ and $a_p^q(i) = a_p^q(j)$, then $i = j$.*

Every parametric type hierarchy, P, is equivalent to a possibly infinite non-parametric IS-A network, $I(P)$:

Definition 2. *Given parametric type hierarchy, $\langle P, \sqsubseteq_P, arity, a \rangle$, the induced (type) hierarchy, $\langle I(P), \sqsubseteq_I \rangle$, is defined such that:*

- *$I(P) = \bigcup_{n<\omega} I_n$, where the sequence $\{I_n\}_{n<\omega}$ is defined such that:*
  - *$I_0 = \{p \mid p \in P, arity(p) = 0\}$,*

– $I_{n+1} = I_n \cup \{p(t_1,\ldots,t_{arity(p)}) \mid p \in P, t_i \in I_n, 1 \le i \le arity(p)\}$, *and*

- $p(t_1,\ldots,t_{arity(p)}) \sqsubseteq_I q(u_1,\ldots,u_{arity(q)})$ *iff* $p \sqsubseteq_P q$, *and, for all* $1 \le i \le arity(p)$, *either* $a_p^q(i) = 0$ *or* $t_i \sqsubseteq_I u_{a_p^q(i)}$.

Subtyping in $I(P)$ is given by subtyping according to P, and subtyping in every dimension according to $I(P)$.

A parametric type signature consists of a parametric type hierarchy together with a feature appropriateness specification:

$$Approp_P : Feat_P \times P \longrightarrow (I(P) \longrightarrow I(P)),$$

in which the value restrictions can make reference to the parameters of the type that bears their features. Penn [2000] also defines the structural restrictions on parametric type hierarchies and appropriateness specifications, called *semi-coherence*, *persistence* and *parametric determination*, that ensure that the equivalent non-parametric signature is a BCPO.

In practice, parametric type signatures can be defined using an adjacency representation of a cover relation and type variables that take scope over these and the value restrictions of any attached appropriateness specifications. For example, in ALE-like notation, parametric lists can be defined by:

```
list(X) sub [e_list(X),ne_list(X)].
ne_list(X) intro [hd:X, tl:list(X)].
```

Here the type variable `X` ranges over all possible types, including other lists. As alluded to in Penn [2000], however, it is possible to employ *parameter restrictions* to force the equivalent non-parametric BCPO to be finite. In the case of the parametric *index* type referred to above, we can restrict its parameters to the sensible portions of the type hierarchy that deal with *person*, *number*, and *gender*:

```
index(P:person,N:number,G:gender) sub [ref(P,N,G)].
index(3rd,sing,neut) sub [there,it].
```

Here, each parameter restriction declares a *filter*, or upward closed set of types, from which the corresponding parameter must be chosen. The definitions from Penn [2000] are not compatible with the second line of the *index* example above, but can be extended, once parameter restrictions are in place, to allow maximal types in the image of a parametric type to be used on the left-hand-side of a subtyping declaration. Again, the only trick is to define the structural conditions in the original parametric type signature that preserve bounded-completeness in the equivalent non-parametric signature, if that is desired.

In the example above, this extension is necessary because there is only one kind of *index(P,N,G)* that requires further speciation, and *there* and *it* cannot be viewed as subtypes of other combinations of *person*, *number* and *gender*, such as

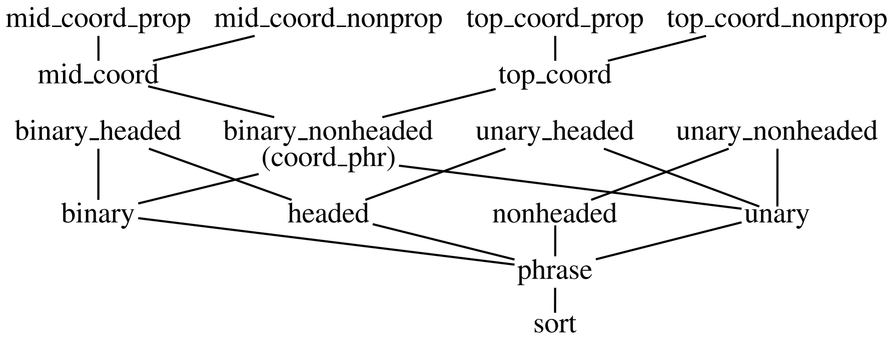


Figure 3: Extension of a filter of parametric types.

index(2nd,plural,masc). In the case of the ERG, we can see this at work within the classification of English phrase types (as simplified in Figure 3). *phrase* is classified along the dimensions of arity and headedness, but only *binary_nonheaded* requires further speciation along the dimensions of *mid* vs. *top*, and *prop*. This can be declared as follows:

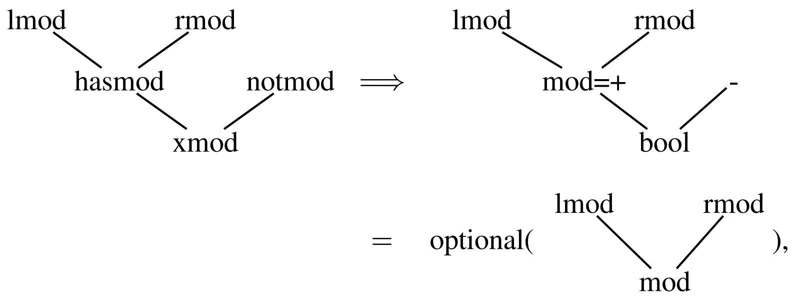
```
% boolean dimension
bool sub [+, -].

% arity dimension
arity sub [binary, unary].

% "semantic height" dimension
semheight sub [mid,top].

% phrase is classified according to arity and headedness
sort sub [phrase(arity:arity,head:bool)].
coord_phr syn phrase(arity:binary,head:-).

% add extra dimensions where necessary
coord_phr adds (sem:semheight,prop:bool).
```

Notice that each dimension or parameter can bear a name, such as `head`, to permit greater reuse of more general filters such as *bool*. Also note that new parameters can simply be added to an existing product where necessary with `adds/2` without introducing a new parametric type, and that type synonyms like *coord_phr* can be defined for greater readability.

Parametric typing is a very expressive device, especially because parameter variables can take scope over appropriateness specifications. The other constructors presented below, in fact, can be viewed as parametric types for which the correspondence to a non-parametric IS-A network is given by something other than a product.

3.2 Optionality

Several dimensions can be thought of as optional. When they are not present, extra types are used in the ERG to assert this. For example, there is a type *no_head*, and a *no_cl_mode*, and although they do not occur as types on their own, the suffixes, *_no_affix_word*, *_no_quant*, and *_notopkey* are attached to many type names. In the case of *luk*, a supertype of *bool*, it is called *na* (alongside the usual $+$ and $-$). In the case of *xmod*, absence is signified by *notmod*, and there is even a positive counterpart called *hasmod* (not to be confused with *has_aux*, which refers to the English auxiliary verb, "has"). All of these represent a special kind of linear sum with the standard *bool* type filter. Decomposing *xmod*'s filter as follows:



we can view this as an application of the higher-order constructor *optional*, which glues its argument (actually the filter rooted at its argument) to a copy of the *bool* filter. As with parametric types, the *bool* filter still exists in the induced IS-A network, so the following naming convention can be used to refer to the members of the type hierarchy that this constructor induces:

$$\begin{array}{rcl} \textit{xmod} & \mapsto & \texttt{mod?} \\ \textit{notmod} & \mapsto & \sim\!\texttt{mod} \\ \textit{hasmod} & \mapsto & \texttt{mod} \\ \textit{lmod} & \mapsto & \texttt{lmod} \\ \textit{rmod} & \mapsto & \texttt{rmod}. \end{array}$$

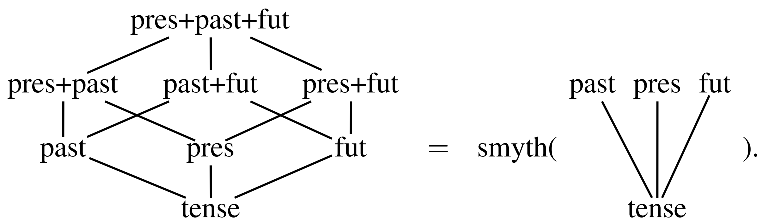
3.3 Smyth powerdomains

The ERG also defines some types as conjunctions or disjunctions of other types. These types have received a great deal of attention in the literature on coordination in languages with overt case, because they seem to be necessary to capture various generalizations about the coordination of unlike cases (disjunctive), and they establish a symmetry to treatments of feature neutrality in parasitic gap constructions (conjunctive).

We agree with the arguments presented in Levy and Pollard [2002] that these conjunctive and disjunctive types are drawn from the Smyth powerdomain closure of an underlying partial order of basic types (such as cases and their disjunctions). As will be seen below (Section 4), this is not the same as believing that the full

Smyth powerdomain is warranted or even correct in every language, only that some subset of it is. The ERG itself uses only various subsets depending on the basic partial order involved. Where we depart from the ERG is in believing that the (subset of the) Smyth closure must be specified in terms of its basic IS-A links. The Smyth construction can be specified explicitly with a *smyth* constructor that expresses this more straightforwardly.

Simplifying the ERG's tense filter somewhat, for example, we can fit this constructor to it:



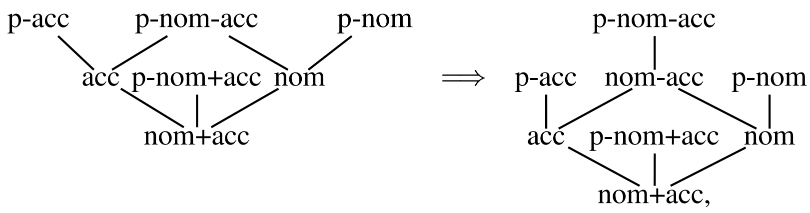
Many other *sort* subtypes in the ERG signature, including, but not limited to, *case*, *gender*, *pernum*, and *mood*, have filters containing disjunctions and conjunctions, suggesting a Smyth powerdomain.

3.4 Purity / Strictness

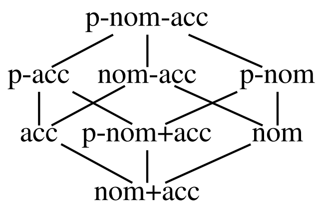
In the ERG, many types also have a "strict" variant declared as a subtype, e.g., *strict_pernum* as a subtype of *pernum*, *strict_tense*, a subtype of *tense*, etc. Strict variants isolate those subtypes with a more classical or narrowly defined sense within a larger classification. Levine et al. [2001] calls this aspect of types "purity" rather than strictness, and extends it to apply to conjunctive types to account for instances of case neutralization. Daniels [2002] proposes to extend it further to disjunctive types to account for certain coordination data. The following table illustrates the notational variation between these approaches on the one hand and the ERG on the other:

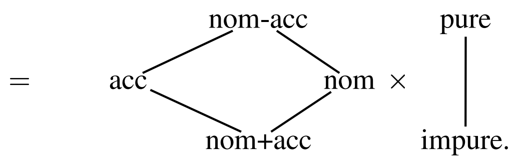
Aspect	Daniels	ERG
purity	'p-' prefix	unmarked or *strict_* prefix
impurity	unmarked	'-*' suffix
conjunctive	'-' connective	'+' connective (non-minimal)
		'and' connective (minimal)
disjunctive	'+' connective	'or' connective

Strict extensions of type filters in the ERG do differ somewhat in their structure from that of purity in Daniels [2002] (notably, pure types are never subtypes of other pure types), but as Daniels's [2002] proposal is more systematic in its application of the extension, we shall consider it further in this section rather than the ERG. There is a near one-to-one correspondence between pure and impure variants of types, which can be analyzed into a product between a simpler hierarchy and a pure-impure filter. Regularizing one of Daniels's [2002] examples by adding a new type to distinguish between impure and pure *nom-acc*:[4]

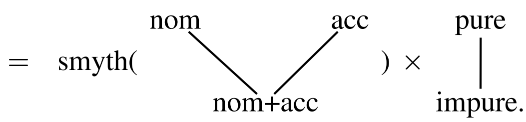


we see that this is contained within:

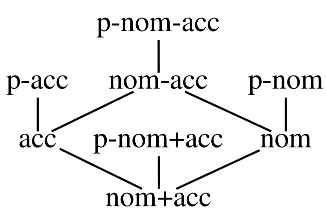


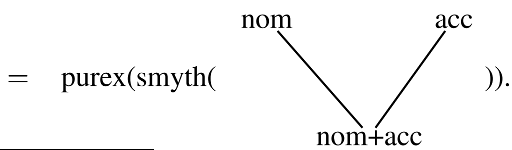


The left-hand-side of this product, however, is simply the Smyth powerdomain of a classic case distinction:



The *purex* constructor builds the necessary portion of this product, in which the IS-A links between pure types are missing:





---

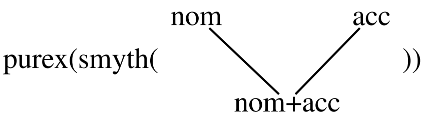
[4] This does not change the meaning of the construction because *p-nom-acc* is the sole maximal extension of *nom-acc*.

Notice the following symmetry: *optional* is a sum with the discretely ordered $+$ and $-$, whereas *purex* is formed from a product with the totally ordered *impure* and *pure*.

In the ERG, strict variants appear as part of many type declarations, including *tense*, *aspect*, *gender*, *pernum*, and *luk*.

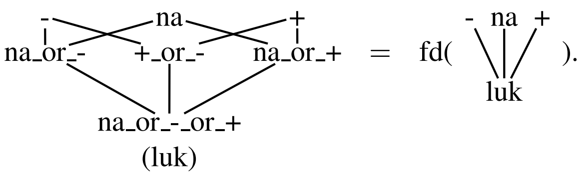
3.5 Finite domains

The ERG also employs finite domains, or powersets of finite sets by enumerating all disjunctive combinations of a discretely ordered set of basic elements. The case example above contains a simple instance of this:



$$= \text{purex(smyth(fd(\{nom,acc\})))}$$

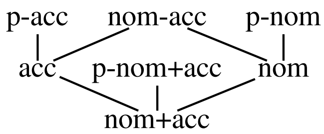
Another example is the ERG's system of extended boolean types, rooted at *luk*. Systematizing the ERG naming conventions used here and simplifying the filter somewhat, we can see:

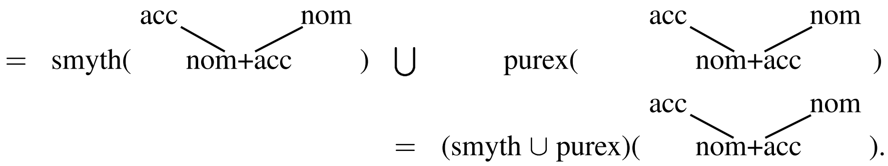


Portions of the *phrase* filter also have finite-domain-like structure.

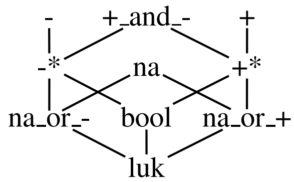
3.6 Unions of constructors

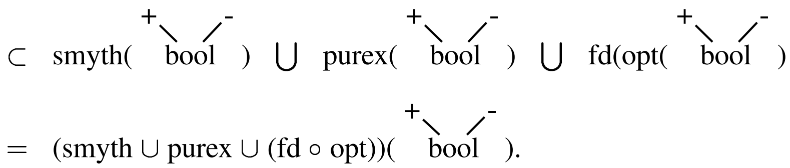
Some of the examples above are slightly modified from the type hierarchy fragments that actually occur in the ERG. As they actually appear, they can still be thought of as reflexes of higher-order constructors, but only by taking the union of several different ones. Union is the implicit operator that combines the different subtyping declarations in a signature, so this is nothing unusual. In the case of higher-order typing constructors in which the names of individual types are established by convention, however, some additional means is necessary for taking the union of non-disjoint sets of types in order to determine which types are being referred to by multiple names. In the ERG, the unions we have analyzed for which this is necessary all consist of higher-order constructors that apply to identical filters, so this is most easily achieved by thinking of union as a higher-order combination of these constructors. For example, in the case of the pure-impure cases as they appear in Daniels [2002]:





Taking *bool* to be *bool** (because it has subtypes, $+*$ and $-*$) and equivalent to *+_or_-*, and *luk* to be equivalent to *na_or_+_or_-*, we can approximate the decomposition of the *luk* filter as it appears in the ERG as follows:

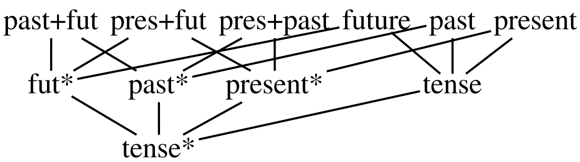




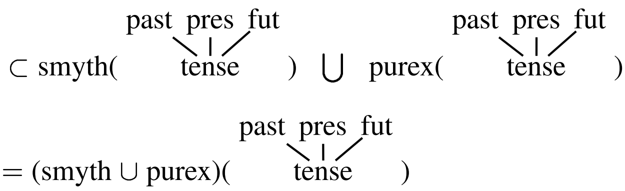
This decomposition is only approximate (hence the subset sign, $\subset$) because there is no pure extension of the *bool* type. In a GDE, only a basis of most general types would need to be provided as arguments, on the assumption that the argument sets are upward-closed:

```
luk type union([smyth,purex,fd(opt)],bool).
```

The tense hierarchy as it stands in the ERG:

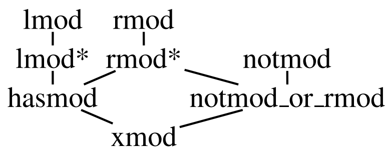


can similarly be approximated:



This, too, is an approximation because there is no type, *pres+past+fut*, in the ERG.

Finally, the xmod hierarchy:



can be approximated, taking *hasmod* to be *lmod_or_rmod* and *xmod* to be *notmod_or_lmod_or_rmod*:

$$\subset \mathrm{fd}(\mathrm{opt}(\overset{\text{lmod} \qquad \text{rmod}}{\text{mod}}) \bigcup \mathrm{purex}(\overset{\text{lmod} \qquad \text{rmod}}{\text{mod}}).$$

It is an approximation because there is no type, *notmod_or_lmod*, and there is no pure extension of *mod* or *hasmod*.

In the next section, we address the problem of working with these approximations in practice.

4 Default Specifications

Why did the ERG's designers not use parametric types or these other constructors in the first place? A major reason is that, in many cases, the least upper bounds they were attempting to achieve could only be approximated with them. To reconsider Figure 1, not every combination of CLAUSALITY and HEADEDNESS is licensed in English — the allowable combinations are explicitly and exhaustively enumerated in the intersection types given at the top of the figure, and this enumeration is a major component of this hierarchy's factual contribution. With parametric types, one defines the entire range of possible products, unless there is some other convention to tell us which combinations to select or exclude.

There are several reasons to prefer higher-order constructors with such a convention over simply using IS-A networks to enumerate the possibilities. First, we would argue that it is often a better indication of the developers' perspective on grammar design to use higher-order constructors to define a "smoother," more regular landscape of possibilities from which those admitted by the grammar can be selected. This is analogous to the benefit that accrues to constraint-based grammars by using signatures to create a more general canvas of possible typed feature structures from which principles of grammar select the ones licensed by the theory. Second, the higher-order declarations make the subtyping definitions more terse and structurally richer, which is then easier for others to navigate through. Third, semi-lattice completion types and other structurally necessary closure types can draw upon this more regular landscape to select their own names. The semi-lattice completion types in the ERG are currently named with "glbtype" plus a number. Fourth, it is possible in principle to use this larger range of types to define a set of possibilities from which a statistical method could select those that are appropriate to a particular corpus or other large domain with more reliability than human grammar designers are capable of.[5]

[5] We are indebted to Rob Malouf for this suggestion during the conference. He also reports that some intersection types that were excluded from the ERG have since been discovered within corpora.

There are several possible conventions that we can imagine using in combination with higher-order typing. All of them use a combination of three devices:

1. Explicit declarations that accompany the signature declaration (such as types to include or exclude),

2. *Generators*, seed sets of included types that are implicitly inferred from their presence in other constructs of the grammar (principles, phrase-structure rules, lexicon, etc.), and

3. Closure under certain structural operations in the signature. Possible operations include:

   (a) joins: if two types are included, so should their least upper bound be,
   (b) supertyping: if a type is included, so should all of the more general types that it extends,
   (c) subtyping: if a type is included, so should all of its more specific extensions,
   (d) appropriateness: if a type is included, so should all of the types that have appropriate features with that type as a value restriction,
   (e) value restriction: if a type is included, so should all of the value restrictions that its appropriate features bear.

Again, this is not intended as a closed class of possibilities. It may also be the case that different closures or conventions are used with different sets of types, according to which constructors were used to declare them, or according to where they appear in the grammar. For example, types that appear in a construct other than a lexical item or lexical rule may be closed under joins. No matter what the choice, the equivalent induced IS-A network can be calculated off-line, and thus at no run-time computational cost.

Which conventions are appropriate is naturally an empirical question, and given that only a single grammar has been the object of our study to date, it is one that remains to be answered. In the ERG, at least, what we observe is that closure under supertyping is generally appropriate for types found in lexical rules and the *phrase* filter, and in the case of pure/strict constructions, this is augmented with closure under joins. *strict_2per*, for example, never appears in the grammar apart from its declaration in the signature. But parsing the sentence, “you jump,” requires the existence of this type, as the least upper bound of *strict_non3sg*, the PN value of the lexical entry for “jump,” and *2per*, the PN value of the entry for “you.” We assume that filters would play a significant role not only in serving as the arguments of constructors, as in the previous section, but in defining the scope of these conventions.

5 Conclusion

This paper provided an argument for using higher-order type constructors within grammar development, drawn largely from examples in the ERG signature. Of the 1503 ERG types that we have manually inspected and classified so far, 894 have been semi-lattice completion types, 234 have been substitutes for parametric types, 60 have been auxiliary types to enforce strictness (such as those suffixed with '-*'), 34 have been disjunctive closures of other types present (such as could be achieved with finite domains), and 16 have been conjunctive (such as could be achieved with Smyth closure). That means that approximately 56.5% of the non-completion types could be replaced by a certainly much smaller collection of higher-order constructions with a default specification convention. An additional 195 were lexical semantic relations, over which other higher-order constructors may possibly exist. This remains a very tantalizing area of further exploration.

References

H. Ait-Kaci. *A lattice theoretic approach to computation based on a calculus of partially ordered type structures*. PhD thesis, University of Pennsylvania, 1984.

R. Brachman. What's in a concept: structural foundations for semantic networks. *International Journal of Man-Machine Studies*, 9(1):127–152, 1977.

B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, 1992.

M. W. Daniels. On a type-bases analysis of feature neutrality and the coordination of unlikes. In L. Hellan F. van Eynde and D. Beermann, editors, *Proceedings of the 8th International HPSG Conference*. CSLI Publications, 2002.

G. Erbach. Multi-dimensional inheritance. In *Proceedings of KONVENS-94*, 1994.

G. Erbach. ProFIT: Prolog with features, inheritance and templates. In *Proceedings of EACL-95*, 1995.

D. Flickinger. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28, 2000.

J.P. Koenig and D. Jufrasky. Type underspecification and on-line type construction in the lexicon. In *Proceedings of WCCFL-94*, 1994.

H.U. Krieger and U. Schaefer. TDL - a type description language for constraint-based grammars. In *Proceedings of COLING-94*, pages 893–899, 1994.

R. Levine, T. Hukari, and M. Calcagno. Parasitic gaps in English: Some overlooked cases and their theoretical implications. In P. Culicover and P. Postal, editors, *Parasitic Gaps*, pages 181–222. MIT Press, Cambridge, MA, 2001.

R. Levy and C. Pollard. Coordination and neutralization in HPSG. In L. Hellan F. van Eynde and D. Beermann, editors, *Proceedings of the 8th International HPSG Conference*. CSLI Publications, 2002.

G. Penn. Parametric types for typed attribute-value logic. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics and the 17th International conference on Computational Linguistics*, 1998.

G. Penn. *The Algebraic Structure of Attributed Type Signatures*. PhD thesis, Carnegie Mellon University, 2000.

G. Penn. Tractability and structural closures in attribute logic type signatures. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 410–417, 2001.

C. Pollard and I. A. Sag. *Head Driven Phrase Structure Grammar*. The University of Chicago Press, 1994.

I. A. Sag. English relative clause constructions. *Journal of Linguistics*, 33(2): 431–483, 1997.