Expanding feature-based constraint grammars: Experience with a large-scale HPSG grammar for English

Liviu Ciortuz
University of York, Heslington, York, YO10 5DD, UK.
E-mail: ciortuz@cs.york.ac.uk.

Abstract

ABC Light is a compiler that implements Light, a simple CLP(OSF) language characterised by the fact that the control level above the OSF-theory unification [Aït-Kaci *et al.*, 1993][Aït-Kaci *et al.*, 1997] of feature structures (OSF-terms) is achieving head-corner chart-based parsing [Kay, 1989][Sikkel, 1997][Shieber *et al.*, 1995]. The system was tested and tuned on LinGO, [Copestake *et al.*, 1999] the large-scale HPSG [Pollard and Sag, 1994] grammar for English developed at CSLI, University of Stanford. In the Light framework, the LinGO grammar is viewed as a Light logic grammar.

The aim of this paper is to give the reader the main insights on how we proceeded to prepare the LinGO grammar for the parsing with the ABC Light system, by obtaining an equivalent order- and type-consistent form, via (off-line/partial) expansion, and determine automatically the grammar's "canonical" appropriateness constraints, not explicitly stated by the grammar writer, but used further on [Ciortuz, 2000] to optimise the grammar's compiled code.

Significant progress has been achieved during the last couple of years in the area of efficient processing with feature-based grammars. Concerning parsing with large-scale HPSG grammars [Pollard and Sag, 1994], notably the LinGO grammar [Copestake *et al.*, 1999] for English developed at CSLI, University of Stanford, the [Oepen *et al.*, 2000] work presents some of the most advanced results.[1] In the meantime, while still under development, our ABC Light compiler, designed to do parsing with feature constraint-based grammars, provided on LinGO parsing results which are competitive with the best results reported in [Oepen *et al.*, 2000].

Compilation in ABC Light is done via an abstract machine called Light AM, which substantially expands the AM designed for OSF-unification [Aït-Kaci and Di Cosmo, 1993]; the code produced by Light AM is further translated down into C [Ciortuz, 2000].

Other compiler systems dealing with HPSG-like grammars are: AMALIA [Wintner, 1997][Wintner and Francez, 1999], LiLFeS [Miyao *et al.*, 2000], (both based, like [Brown and Manandhar, 2000], on somehow related AMs, all derived from WAM [Warren, 1983]), and ALE [Carpenter and Penn, 1992] (based on compilation of typed feature structures into Prolog terms). Up to our knowledge, the only compilers on which LinGO was tested until now are LiLFeS and ABC Light. Among the different interpreters dealing with LinGO — LKB [Copestake, 1999], TDL/PAGE [Krieger and Schäfer, 1994] and PET [Callmeier, 2000], the last one imported the expansion conception first implemented in ABC Light.

While the other compilers relate very much to the typed feature structure theory [Carpenter, 1992], the ABC Light system was elaborated with the OSF logic constraint theory in mind. The two theories were elaborated somehow in parallel in the early 90's; the first one was influential in computational linguistics, the other one in constraint logic programming. One can view the work on Light as continuing an interesting and challenging link between the two domains.

The notion of *expansion* was used in [Krieger and Schäfer, 1995] to designate the equivalent transformation of an OSF-term (or: feature structure (FS)) into a well-typed feature structure, w.r.t. a given OSF-theory/set of OSF-terms. Expansion corresponds to Carpenter's notion of *type inference*. The difference is that Carpenter's approach [Carpenter, 1992] starts from the notion of appropriateness, while there is (rightly) no need to use this notion in the [Krieger and Schäfer, 1995] approach. [2]

We claim that we proceed one step further than [Krieger and Schäfer, 1995] when

- we define the notion of type-consistency which in conjunction with the notion of order-consistency [Aït-Kaci *et al.*, 1997] designates a larger class of OSF-theories than well-typed systems of feature structures (FSs):
- our notion of expansion does not require that all sub-structures in a FS must be subsumed by the

[1] The LinGO grammar is already used by a number of commercial companies in USA and Germany.

[2] Goetz defines the notion of *extension* — close to expansion — to further built up the notion of *unextension* in a framework allowing for the use of disjunction [Götz, 1994].

corresponding type; we limit this request only to non-atomic nodes;[3]

- this notion has already been proved to be beneficial for LinGO-like grammars [Callmeier, 2000], since it lead to a both significant reduction of the expanded size of the grammar and the parsing time (due to reduction of copying or other structure manipulation operations during unification).

While expansion in [Krieger and Schäfer, 1995] is designed to work in a unitary (however parameterised, and memoization-based) manner for both the pre-parsing and parsing time, expansion in ABC Light is a combined mechanism, made of the following complementary components:

• off-line expansion — presented in this paper — which is lazy (like in TDL/PAGE), allowing only terminal-recursive feature structures (such a FS is one in which if the root sort appears also elsewhere, then it is only on non-atomic nodes);

• on-line expansion — to be called "by need" in OSF-theory unification (w.r.t. order- and type-consistent OSF-theories), presented in [Ciortuz, 2000] — which allows the use of recursive feature structures and is eager, leading (in our opinion) to a fast detection of unification failure.[4]

Moreover, off-line expansion (or simply, for now on in this paper: expansion) in ABC Light is a two phase operation: order-consistent expansion and type-consistent expansion. An interleaving of the two phases is possible (like in TDL/PAGE, see [Krieger and Schäfer, 1995]), but our new approach allows to

- detect interesting main-line similarities between the two phases;
- see that the unification to used in the first phase is simple OSF-unification — therefore it speeds up the expansion process —, while in the second it is OSF-theory unification (which corresponds to typed FS unification);
- design a new kind of quasi-destructive unification which is highly effective in this set up: directed OSF-unification;[5]
- have a simple way to automatically infer appropriateness constraints which are implicitly encoded in the (LinGO-like) input grammar.

After introducing the formal background of the ABC Light system in the section 1, we will elaborate first on the relationship between the semi-lattice condition (to be) imposed on sorts and respectively types in a LingGO-like grammar.[6] The objective of the sections 3 and 4 will be to transform the input set of feature structures $\{\Psi(s)\}_{s\in\mathcal{S}}$ so that it become order-consistent, and respectively order- and type-consistent. The inference of appropriateness constraints is done in connection with order-consistent expansion.

1 OSF notions

Let $\mathcal{S}$ be a set of symbols called *sorts*, $\mathcal{F}$ a set of *features*, and $\prec$ a computable partial order relation on $\mathcal{S}$. We assume that $<\mathcal{S},\prec>$ is a lower semi-lattice, meaning that, for any $s,s'\in\mathcal{S}$ there is a unique greatest lower bound glb(s,s') in $\mathcal{S}$. This glb is denoted $s\wedge s'$.

Note that the above definition for glbs of sorts is extended naturally to the notion of glbs of feature structures w.r.t. subsumption (denoted $\sqsubseteq$) defined over $\mathcal{S}\cup\mathcal{F}$, and exactly this later notion of glb is of FSs is taken as (or: coincides with) the definition of FS unification.

Notations: $root(\psi)$ and $\psi.f$ denote the sort of the root node in the term ψ, and respectively the value of the feature f at the root level in ψ. The reflexive and transitive closure of $\prec$ will be denoted as $\preceq$. The *logical form* associated to an OSF-term $\psi\equiv s[f_1\rightarrow\psi_1,...,f_n\rightarrow\psi_n]$ is $Form(\psi,X)\equiv\exists X_1...\exists X_n((X.f_1\doteq Form(\psi_1,X_1)\wedge...\wedge X.f_n\doteq Form(\psi_n,X_n))\leftarrow X:s)$, where $\leftarrow$ denotes logical implication, and $X,X_1,...,X_n$ belong to a countable infinite set $\mathcal{V}$ of variables, disjunct from $\mathcal{S}\cup\mathcal{F}$.

An OSF-*theory* is a set of OSF-terms $\{\Psi(s)\}_{s\in\mathcal{S}}$ such that $root(\Psi(s))=s$, and for any $s,t\in\mathcal{S}$, $\Psi(s)$ and $\Psi(t)$ have no common variables. The term $\Psi(s)$ will be called the s-sorted type, or simply the s type of the given OSF-theory. A *model* of the theory $\{\Psi(s)\}_{s\in\mathcal{S}}$ is a logical interpretation in which every $Form(\Psi(s),X)$ is valid. We mention that we are dealing here only with finite OSF-theories. i.e., whose set of sorts is finite.

The notion of OSF-term unification is naturally generalised to *OSF-theory unification:* ψ_1 and ψ_2 unify w.r.t. the theory $\{\Psi(s)\}_{s\in\mathcal{S}}$ if there is ψ such that $\psi\sqsubseteq\psi_1,\psi\sqsubseteq\psi_2$, and $\{\Psi(s)\}_{s\in\mathcal{S}}$ entails ψ, i.e., $Form(\psi,X)$ is valid in any model of the given theory.

Now we can formalise the link towards well-typed feature structures:

Following the definition given in [Aït-Kaci *et al.*, 1993], an OSF-theory $\{\Psi(s)\}_{s\in\mathcal{S}}$ is *order-consistent* if $\Psi(s)\sqsubseteq\Psi(t)$ for any $s\preceq t$. An OSF-theory is *type-consistent* if for any non-atomic subterm ψ of a $\Psi(t)$, if the root sort of ψ is s, then $\psi\sqsubseteq\Psi(s)$. A term is said to be non-atomic (or: framed) if it contains at least one feature.

Example 1 *Let us consider*

$\psi_1 = a[$ FEAT1 $\;\; b\,]$,

$\psi_2 = a[$ FEAT1 $\;\; c[$ FEAT2 $\;\;$ bool $]\,]$,

[3] For instance, if a[F   cons] is type-consistent, its well-typed correspondent would be
a[F   cons[FIRST   top, REST   list]].

[4] An attempt to implement a lazy version of the on-line expansion mechanism in ABC Light is currently on its way.

[5] For other quasi-destructive unifiers used in feature-based parsing see [Wroblewski, 1987],[Tomabechi, 1991] and [Malouf *et al.*, 2000].

[6] Thanks to Dan Flickinger, who made me start to think on this issue in December 1998, when I presented him type hierarchy completion in CHIC/ago, the development prototype for ABC Light.

a sort signature in which $b \wedge c = d$, *and the symbol* $+$ *is a subsort of* bool. *We consider the OSF-theory made (uniquely) of*

$$\Psi(d) = d[\ \mathsf{FEAT2} \quad +\ ].$$

The glb of ψ_1 *and* ψ_2 *is*

$$\psi_3 = a[\ \mathsf{FEAT1} \quad d[\ \mathsf{FEAT2} \quad \text{bool}\ ]\ ],$$

while the $\{\Psi(d)\}$ *OSF-theory relative glb is*

$$\psi_4 = a[\ \mathsf{FEAT1} \quad d[\ \mathsf{FEAT2} \quad +\ ]\ ].$$

A *well-typed* OSF-theory is an order-consistent theory in which the following conditions are satisfied for any $s, t \in \mathcal{S}$:

i. if $f \in$ *Arity(s)* $\wedge f \in$ *Arity(t)*, then $\exists u \in \mathcal{S}$, such that $s \preceq u, t \preceq u$ and $f \in Arity(u)$;

ii. for every subterm ψ in $\Psi(t)$, such that $root(\psi) = s$, if a feature f is defined for ψ, then $f \in Arity(s)$, and $\psi.f \sqsubseteq \Psi(root(s.f))$,

where *Arity(s)* is the set of features defined at the root level in the term $\Psi(s)$. An OSF-term ψ satisfying the condition *ii* from above is (said) *well-typed* w.r.t. the OSF theory $\{\Psi(s)\}_{s \in \mathcal{S}}$.

Notes:

1. The condition i implies that for every $f \in \mathcal{F}$ there is at most one sort s such that f is defined for s but undefined for all its supersorts. This sort will be denoted *Intro*(f), and will be called the *appropriate domain* on the feature f. Also, *root*$(\Psi(s).f)$, if defined, will be denoted *Approp*(f, s), and will be called the *appropriate value* on the feature f for the sort s. *Approp*$(f$, *Intro*$(f))$ is the maximal appropriate value for f. [7] The appropriate domain and values for all features $f \in \mathcal{F}$ define the "canonical" *appropriateness constraints* for a well-typed OSF-theory.

2. As a well-typed OSF-theory is (by definition) order-consistent, it implies that *Arity(s)* $\supseteq$ *Arity(t)*, and *Approp(f, s)* $\preceq$ *Approp(f, t)* for every $s \preceq t$;

3. A stronger version for the condition *ii* would be: the feature f is defined (at the root level) for ψ iff $f \in Arity(s)$, and $\psi.f \sqsubseteq \Psi(s.f)$. In the latter case, the theory is said to be *totally well-typed.*

For well-typed OSF theories $\{\Psi(s)\}_{s \in \mathcal{S}}$, the notion of OSF-unification extends naturally to *well-typed* OSF-*unification*. The well-typed glb of two feature structures ψ_1 and ψ_2 is the most general (w.r.t. $\sqsubseteq$) well-typed feature structure subsumed by both ψ_1 and ψ_2. The well-typed glb of two feature structures is subsumed by the glb of those feature structures.

Obviously, the well-typed OSF-theories are a particular class of order- and type-consistent OSF-theories. On this class, well-typed unification coincides with OSF-theory unification (up to atomic nodes' type unfolding).

[7] Our current implementation of Light uses a weaker version for the condition i: if $f \in$ *Arity(s)* $\wedge f \in$ *Arity(t)*, and $f \notin$ *Arity*$(s \wedge t)$, then *AppropDom*$(f) = \text{lub}(s, t)$, and *AppropVal*$(f) = \text{lub}(root(s.f), root(t.f))$, provided that the lub (least upper bound) of the two sorts exists.

Note: The second main difference between the class of order- and type-consistent OSF-theories on one side, and that of well-typed OSF-theories on the other side is related to appropriate features:[8] well-typed theories do not allow a subterm ψ of root sort s to use features not defined at the root level in the corresponding type $\Psi(s)$. For instance, if

$$\psi_5 = a[\ \mathsf{FEAT1} \quad d[\ \mathsf{FEAT2} \quad +, \mathsf{FEAT3} \quad bool\ ]\ ].$$

then the OSF-theory glb of ψ_2 and ψ_5 will be defined (and equal to ψ_5), while their well-typed glb relative to the same theory does not exist, simply because ψ_5 is not well-typed w.r.t. $\Psi(d)$, due to the non-appropriate feature FEAT3.

Therefore, Light will allow the grammar writer more freedom. The source of this freedom resides in the *openness* of OSF-terms.

2 On the semi-lattice condition: sorts vs types in ABC Light

Let us make (a comparison between) the following *important points:*

1. $< \mathcal{S}, \prec >$, the sort hierarchy associated to an OSF-theory/Light grammar must be a lower semi-lattice (LSL), i.e. for every $s, t \in \mathcal{S}$ there is a unique glb of s and t in $\mathcal{S}$, that we denoted (as usually) $s \wedge t$. The existence of a single glb for s and t in $\mathcal{S}$ means that $s \wedge t \prec s$, $s \wedge t \prec t$, and if there is $u \in \mathcal{S}$ such that $u \prec s$, $u \prec t$, then $u \preceq s \wedge t$. A lower semi-lattice contains a "clash"/bottom sort $\perp$ such that $\perp \preceq s$ for every $s \in \perp$.

In practice, the sort hierarchy is specified by the programmer/grammarian only as a partially-ordered set (*poset*). Efficient embedding of a poset $< \mathcal{S}, \prec >$ into a minimal LSL that extends it is presented in [Aït-Kaci *et al.*, 1989].

As already noticed in the section 1, the above definition for glbs of sorts is extended naturally to the notion of glbs of feature structures w.r.t. subsumption, and exactly this later notion of glb of FSs is taken as the definition of FS unification. (See [Copestake, 2000], definition 4.)

2. If $\mathcal{G} = \{\Psi(s)\}_{s \in \mathcal{S}}$ is an OSF-theory over the signature $\Sigma = (< \mathcal{S}, \prec >, \mathcal{F})$, $\mathcal{FS}$ is the set of all feature structures definable over Σ, and $\sqsubseteq$ is the FS subsumption relation, then the type lattice associated to the expanded form of $\mathcal{G}$ must be a certain, conveniently-chosen sub-lattice of the lattice $< \mathcal{FS}, \sqsubseteq >$.

In ABC Light, we drop out the (somehow natural) requirement that in this lattice the glb of two types must be exactly the unification result for those types. We will show that taking the expanded type (sub)lattice as the one induced on $\mathcal{FS}$ by the sort lattice $< \mathcal{S}, \prec >$ is a right choice (in a closed-word assumption), and it leads to a significant improvement in parsing efficiency.

[8] The first difference concerns the subsumption condition — limited to non-atomic substructures ψ: if $root(\psi) = s$, then $\psi \sqsubseteq \Psi(s)$.

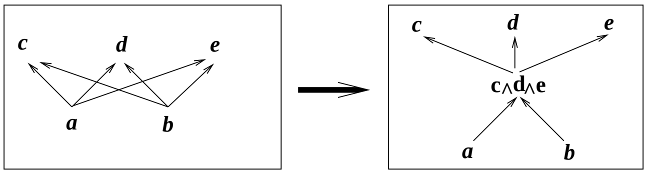

Figure 1: Lattice embedding of a sort hierarchy.

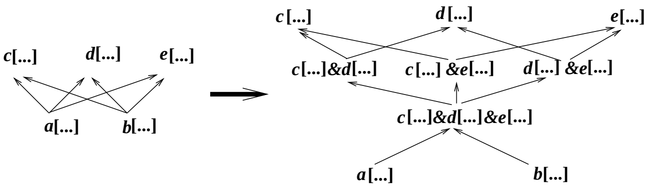

Figure 2: Lattice embedding of a FS hierarchy such that $glb_{\sqsubseteq} \equiv$ *FS unif.*

Let us build our case and argue for the proposed solution in ABC Light by the means of a simple example:

When comparing the LSL embeddings of a sort hierarchy and respectively a type hierarchy exemplified in Figure 1 and Figure 2 — where the notation [...] suggests a feature structure's frame and & designates well-typed FS unification — one can see that in the embedding result of a type hierarchy $< \{\Psi(s)\}_{s \in \mathcal{S}}, \sqsubseteq >$, the (is-a) partial order relation is more elaborated than the one in the embedding result for the sort hierarchy $< \mathcal{S}, \prec >$.[9]

As Figure 3 shows for our example, in ABC Light we synthesise new (glb) types only when required by the transformation of the sort hierarchy into a LSL. That is: if $s \wedge t$ is added to $\mathcal{S}$ during its embedding into an LSL, then its associated type $\Psi(s \wedge t)$ will be the result of unifying all types corresponding to its (immediate) ascendents:

$$\Psi(s \wedge t) = \&_{s \wedge t \, \prec \, u} \Psi(u).$$

This is obviously different than if we would define $\Psi(s \wedge t) = \Psi(s) \ \& \ \Psi(t)$. In our setup, $\Psi(s \wedge t) \sqsubseteq \Psi(s) \ \& \ \Psi(t)$.[10]

Working in the above presented framework preserves the correctness of the parsing (as deduction) with the resulting grammar, if we assume a kind of *closed-world assumption* like in Logic Programming: in order to get the "ground" set of parses/solutions for a certain input sentence, a synthesised glb sort present in the parsing result will be instantiated to the disjunction of its maximal non-synthesised subsorts in the original sort hierarchy. For our example, $\Psi(c \wedge d \wedge e) = c[...] \ \& \ d[...] \ \& \ e[...]$ will be replaced by the disjunction $a[...] \vee b[...] = \Psi(a) \vee \Psi(b)$.

[9] The bottom sort $\perp$ is not shown in Fig. 1, 2, and 3.

[10] In our example, $\Psi(c \wedge d) = \Psi(c \wedge d \wedge e) = \Psi(c) \ \& \ \Psi(d) \ \& \ \Psi(e) \sqsubseteq \Psi(c) \ \& \ \Psi(d)$.

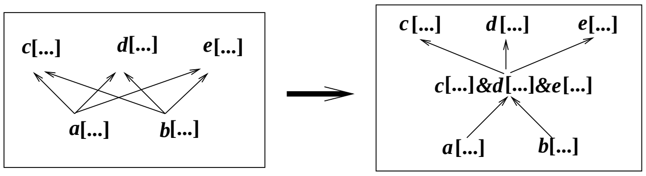

Figure 3: Lattice embedding of a FS hierarchy in ABC Light .

In practice, this "unpacking" of the solution set is not done in ABC Light, just because having the solution set in that compact form is simply more convenient (note that unpacking is exponential).

Note that our type hierarchy embedding is an important source of efficiency for both pre-processing (expansion) and parsing with LinGO-like grammars: significantly less synthesised type glbs are computed (1/4 in the illustrated simple example).[11]

3 Inference of canonical appropriateness constraints and order-consistent expansion

The following simple schema is used in the ABC Light system to transform an input OSF-theory into an order-consistent one and to infer its "canonical" appropriateness constraints, i.e., the maximal appropriate domain and value for every feature.

For every feature f, the first sort s that has the feature f defined at the root level in the s-sorted type $\Psi(s)$ in the input OSF-theory/grammar $\mathcal{G}$ is the *appropriate domain* of f, and is denoted $\mathit{Intro}(f)$. Here the "first" sort is taken in the sense of top-down depth-first traversal of the sort hierarchy $< \mathcal{S}, \prec >$.[12]

The propagation of appropriate domain constraints is done in ABC Light by taking every non-atomic subterm ψ in $\mathcal{G}$ and replacing its root sort $\mathit{root}(\psi)$ with

$$\bigwedge\nolimits_{f \text{ defined for } \psi} \mathit{Intro}(f) \qquad (*)$$

Note that conjunction/glb in $(*)$ may not always succeed. In such cases, either the grammarian corrects the grammar, or he/she lets the (ABC Light) system itself

[11] This is why the LKB system (which, like TDL, used the general approach to embed the type hierarchy into a LSL), imported (via PET) the type hierarchy embedding we proposed here.

[12] ABC Light does more: if several sorts $s_1, s_2, ..., s_n$ $(n \geq 2)$ introduce the same feature f (i.e., f is defined at root level in $\Psi(s_1), \Psi(s_2), ..., \Psi(s_n)$, but for any supersort/ancestor of $s_i, i = 1, 2, ..., n$, the feature f is not defined at root level), then the system computes the minimal common ancestor for $s_1, s_2, ..., s_n$. ABC Light takes it as $\mathit{Intro}(f)$, and then defines $\mathit{Approp}(\mathit{Intro}(f))$ as the minimal supersort for $\Psi(s_1.f), \Psi(s_2.f), ..., \Psi(s_n.f)$. As ABC Light considers a reserved sort *top* as supersort of all sorts in $\mathcal{S}$, the minimal supersort of n sorts always exists.

soften the appropriate domain for the feature that caused the sort unification clash. In this case, at the end, the inference process must be restarted.

The next step (before getting the appropriate value for features) is to get the order-consistent form of $\mathcal{G}$. Let $s_1, s_2, ..., s_n$ be the sorts in $\mathcal{S}$, sorted reverse-topologically according to $\prec$, i.e., if $s_i \prec s_{j_1}, ..., s_{j_k}$, then $i > j_1, ..., i > j_k$. Note that s_1 can be assumed the most general sort (*top*) in $\mathcal{G}$. For $i = 1, 2, ..., n$, if $\{s_{j_1}, ..., s_{j_k}\}$ is the set of the minimal supersorts of s_i in $(\mathcal{S}, \prec)$, and $\Psi(s_1), ..., \Psi(s_n)$ are the types associated respectively to $s_1, ..., s_n$ in $\mathcal{G}$, then $\Psi(s_i)$ is replaced with $\Psi(s_i) \wedge \Psi(s_{j_1}) \wedge \Psi(s_{j_2}) \wedge ... \Psi(s_{j_k})$.[13] So,

$$\Psi(s_i) := \Psi(s_i) \wedge \Psi(s_{j_1}) \wedge ... \wedge \Psi(s_{j_k}) \qquad (**)$$

If the is-a relation $\prec$ is acyclic — and this fact is checked out by the ABC Light system — then the process of getting the input theory order-consistent terminates in finite time for any finite input OSF theory.

Note that:
1. From the procedural point of view, in the $(**)$ relation, the conjunction $\wedge$ can be replaced by simple OSF-term unification, i.e., not OSF-theory unification.
2. If unification succeeds always in $(**)$, then the newly resulting OSF-theory is order-consistent and verifies the well-typedness condition i introduced in the section 1. (The condition ii will be accomplished later by the theory type-consistent expansion, to be presented in the next section.)

Finally, after order-consistent expansion, for every $f \in \mathcal{F}$ and $s \in \mathcal{S}$, the *appropriate value* of f for s is (by definition) the root sort of the sub-term $\Psi(s).f$. The *maximal appropriate value* for f is $root(Intro(f))$.

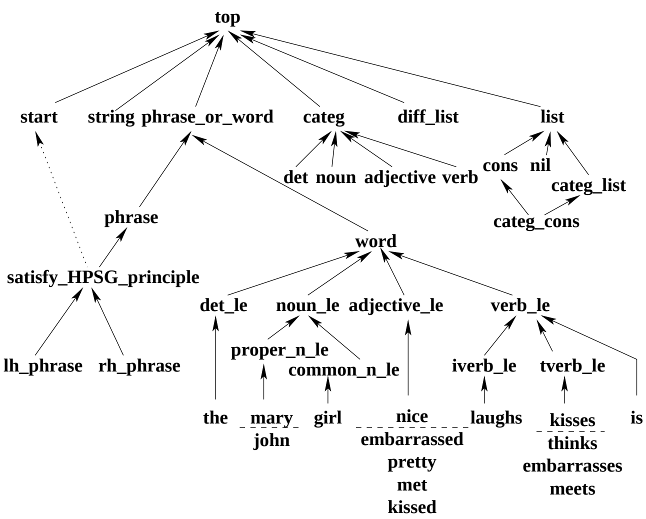

Figure 4: The sort signature for the grammar in Figure 5.

[13] t is a minimal supersort (i.e., "parent") of s in $(\mathcal{S}, \prec)$ if $s \prec t$ and there is no $u \in \mathcal{S}$ such that $s \prec u \prec t$.

Example 2 *Let us consider a grammar presented in [Shieber, 1992]. We adapted it to the OSF/Light format as shown in Figure 5.*[14] *The sort hierarchy corresponding to this grammar is shown in Figure 4.*[15] *The sorts* string, start, list *and the list subsorts* cons *and* nil *are reserved ABC Light sorts, and so is the difference list sort,* diff_list. *Strings are surrounded by quotes and tag/variable names are preceded/introduced by #. The notation <! !> in the grammar's code is a syntax sugar for difference lists, just as < > is used for lists. Also, ! is a constructor for difference lists, playing a similar role to that played by | for* cons *lists.*[16]

The appropriateness constraint inference task for this grammar finishes with the maximal appropriateness constraints shown in Figure 6. As described earlier in this subsection, in the beginning only appropriate domain constraints are computed, hence building up the second column of the table in Figure 6. Then, these appropriate domain constraints are propagated into the type feature structures. For instance, when the syntactic sugar corresponding to the feature path constructor (.) is eliminated, the [HEAD.PHON #1!#2] *constraint in the* lh_phrase *structure becomes* [HEAD *top*[PHON #1!#2]. *After propagating the appropriate domain constraint for the feature* PHON, *we get* [HEAD phrase_or_word[PHON #1!#2]], *because the appropriate domain for* PHON *is* phrase_or_word.

After the appropriate domain propagation task is finished for the whole grammar, it is time for inheritance-based constraint propagation. For instance, the feature structures for satisfy_HPSG_principles *and* girl *become respectively:*

```
satisfy_HPSG_principles
[ PHON   diff_list,
  CAT    #1:categ,
  SUBCAT #2:categ_list,
  HEAD   #4:phrase_or_word
            [ CAT #1,
              SUBCAT #3|#2 ],
  COMP   #5:phrase_or_word
            [ CAT #3,
              SUBCAT nil ],
  ARGS   <#4, #5> ]
      girl
      [ PHON   <!"girl"!>,
        CAT    noun,
        SUBCAT <det> ].
```

[14] Note in Figure 5 that ABC Light distinguishes two special classes of types: rules and lexical entries.

[15] Just for convenience, sorts situated under a dashed line are derived (i.e., have the same parents) as the sort found immediately above that line. For instance, pretty is derived from adjective_le, exactly like nice.

[16] Formally, $<!a_1, a_2, ..., a_n! >$ and #1!#2 stand respectively for

diff_list[FIRST_LIST $a_1|a_2|...|a_n|$#1,
REST_LIST #1]

diff_list[FIRST_LIST #1,
REST_LIST #2].

```
types:

start[ SUBCAT nil ]
cons
[ FIRST top,
  REST  list ]
diff_list
[ FIRST_LIST list,
  REST_LIST  list ]
categ_cons
[ FIRST categ,
  REST  categ_list ]
phrase_or_word
[ PHON   diff_list,
  CAT    categ,
  SUBCAT categ_list ]
phrase
[ HEAD #1:phrase_or_word,
  COMP #2:phrase_or_word,
  ARGS <#1, #2> ]
satisfy_HPSG_principle
[ CAT    #1,
  SUBCAT #2,
  HEAD   top
         [ CAT #1,
           SUBCAT #3|#2 ],
  COMP   top
         [ CAT #3,
           SUBCAT nil ] ]
det_le
[ CAT det,
  SUBCAT nil ]
noun_le
[ CAT noun ]
proper_noun_le
[ SUBCAT nil ]
common_noun_le
[ SUBCAT <det> ]
adjective_le
[ CAT    adjective,
  SUBCAT nil ]
intrans_verb
[ CAT    verb,
  SUBCAT <noun> ]
trans_verb
[ CAT    verb,
  SUBCAT <noun, noun> ]
```

```
program: // rules

lh_phrase
[ PHON      #1!#3,
  HEAD.PHON #1!#2,
  COMP.PHON #2!#3 ]
rh_phrase
[ PHON      #1!#3,
  HEAD.PHON #2!#3,
  COMP.PHON #1!#2 ]

query: // lexical entries

the[ PHON <!"the"!> ]
girl[ PHON <!"girl"!> ]
john[ PHON <!"john"!> ]
mary[ PHON <!"mary"!> ]
nice[ PHON <!"nice"!> ]
embarrassed[ PHON <!"embarrassed"!> ]
pretty[ PHON <!"pretty"!> ]
met[ PHON <!"met"!> ]
kissed[ PHON <!"kissed"!> ]
is[ PHON   <!"is"!>,
    CAT    verb,
    SUBCAT <adjective, noun> ]
laughs[ PHON <!"laughs"!> ]
kisses[ PHON <!"kisses"!> ]
thinks[ PHON <!"thinks"!> ]
meets[ PHON <!"meets"!> ]
embarrasses[ PHON <!"embarrasses"!> ]
```

Figure 5: A Light sample grammar (adapted from [Shieber, 1992]).

feature	approp. domain	approp. value
FIRST	cons	top
REST	cons	list
FIRST_LIST	diff_list	list
REST_LIST	diff_list	list
PHON	phrase_or_word	diff_list
CAT	phrase_or_word	categ
SUBCAT	phrase_or_word	categ_list
HEAD	phrase	phrase_or_word
COMP	phrase	phrase_or_word
COMP	phrase	list

Figure 6: The canonical maximal appropriateness constraints inferred for the grammar in Figure 5.

Finally, if the inheritance-based constraint propagation (or: order-consistent expansion) task is performed successfully for all types in the given grammar, the appropriate value constraints are collected, thus filling the third column in the table given in Figure 6.

Note: The dotted line in Figure 4 corresponds to a notable exception in inheritance-based constraint propagation (i.e., order-consistent expansion) in ABC Light: constraints associated with the reserved *start* symbol[17] are not inheritable. In our example, while lh_phrase and lh_phrase are derived from the start symbol, the corresponding types do not inherit from Ψ(start).[18] The constraints associated with this last type will be used at the parsing time to decide whether an obtained parse is/isn't acceptable: a derivation corresponding to an input string is acceptable only if its corresponding feature structure (lh_phrase or lh_phrase) can be unified with Ψ(start). [19] Also, Ψ(start) is excepted from the inference of appropriateness domain constraints (this is why in our example SUBCAT has the appropriate domain phrase_or_word, not start).

4 Type-consistent expansion

Type-consistent expansion is a strategy designed to put an order-consistent OSF-theory under an equivalent type-consistent theory. It does (iterative) *partial unfolding*, limited to checking of type constraints only on non-atomic subtypes. Therefore, ABC Light allows a limited form of recursion: in every type $\Psi(s)$ of the input theory/grammar $\mathcal{G}$, apart from the root node, the sort s may occur also (at most) on atomic/leaf nodes.This condition, however does not guarantee that recursiveness will not appear at non-leaf levels during expansion.

[17]It corresponds to root_strict in LinGO.

[18]It is like the dotted line would be deleted during order-consistent expansion but present/reintroduced for type-consistent expansion.

[19]Note that certain Light logic grammars allow for the static computation of this condition. This applies for LinGO, but not for the exemplified (Shieber) grammar.

Using the notation $\Psi_s(X) = Form(\Psi(s), X)$ to designate the logical formula associated to $\Psi(s)$ (see Section 1), we can give the formal definition for the type-consistent expansion of a FS by means of the logical unfold operation:

- Unfolding: $Unfold(X : s \wedge \varphi) \equiv \Psi_s(X) \wedge \varphi$;
- Partial unfolding: $PartUnfold(X : s \wedge X.f \doteq Y \wedge \varphi) \equiv \Psi_s(X) \wedge X.f \doteq Y \wedge \varphi$;
- Recursive partial unfolding:
  $PartUnfold^*(\Psi) \equiv \bigwedge_{n=0}^{\infty} PartUnfold^n(\Psi)$, where
  $PartUnfold^0(\Psi) = \Psi$, and
  $PartUnfold^{n+1}(\Psi) = PartUnfold(PartUnfold^n(\Psi))$.

Iterative partial unfolding:
Let $\Psi_0, \Psi_1, ..., \Psi_n, ...$ be a possibly infinite sequence of distinct OSF-terms such that:

i. $\Psi_0 = \Psi, \Psi_1 = PartUnfold(\Psi_0)$;

ii. for every $k > 1$, there is $1 < j < k$ such that
$\Psi_k = X : s' \wedge \varphi_k$, $\Psi_j = X : s' \wedge \varphi_j$, $\Psi_{j-1} = X : s \wedge \varphi_{j-1}$, with $s \neq s'$,
(meaning that the sort of X was affected by a previous unfolding step), and
$\Psi_{k+1} = \Psi_s(X) \wedge \varphi_k$.

The sorts s in the definitions of partial unfolding, respectively iterative partial unfolding of Ψ are called *critical sorts* for $t = root(\Psi)$.[20] Also — in correspondence with the case *ii.* in the definition of iterative partially unfolding — the sort s' is critical for s if s' enters (as the new root sort of a non-atomic subterm) into the iterative partially unfolded form of Ψ due to unification with Ψ_s.

If all critical sorts s, s' involved in the above definition are type-consistent and the sequence $\Psi_0, \Psi_1, ..., \Psi_n, ...$ is finite (meaning that no more unfolding step can be performed such that $\Psi_{k+1} \neq \Psi_k$), then Ψ_n is a type-consistent equivalent form of Ψ; we call it simply the expanded form of Ψ. If the sequence $\Psi_0, \Psi_1, ..., \Psi_n, ...$ is infinite, then the expansion of Ψ has to be stopped blocked due to type recursiveness.

Our strategy for doing type-consistent expansion of a whole input theory/grammar is in its main lines similar to the schema used in the precedent subsection to put an OSF theory under an order-consistent form.

Let $s_1, s_2, ..., s_n$ be the sorts in the signature $<\mathcal{S}, \prec>$ of the input grammar, with $\Psi(s_1), \Psi(s_2), ..., \Psi(s_n)$ the corresponding types. We define on $\mathcal{S}$ the partial order relation $\lhd$ in the following way: if ψ is a non-atomic subterm of $\Psi(s_i)$, then $root(\psi) \lhd s_i$. If the relation $\unlhd$, the reflexive and transitive closure of $\lhd$ is acyclic, let $t_1, t_2, ..., t_n$ be the sorts $s_1, s_2, ..., s_n$ topologically sorted

[20]Alternatively, we can say that the type $\Psi(t)$ depends critically on the types $\Psi(s)$. The intuitive meaning of the critical sort relation is the following: the sort s is critical for the sort t if $\Psi(s)$ has to be expanded before $\Psi(t)$, because the computation of the expanded form of $\Psi(t)$ has to use the expanded form of $\Psi(s)$.

w.r.t. $\trianglelefteq$. Then, type expansion for $\Psi(t_i), i = 1, 2, ..., n$ is achieved by replacing every non-atomic subterm ψ of $\Psi(t_i)$ with $\psi \wedge \Psi(root(\psi))$.[21] So,

$$\psi := \psi \wedge \Psi(root(\psi)). \qquad (***)$$

Again, we can make some remarks:
1. conjunction/unification in $(***)$ may fail; in this case the grammar writer has to revise the grammar;
2. if i. the root sort of ϕ, a non-atomic subterm of ψ becomes more specific when applying $(***)$, or ii. ϕ is an atomic subterm of ψ which gets unified with with a non-atomic subterm φ of $\Psi(root(\psi))$, then ϕ's consistency has to be checked, by further applying $(***)$ to ϕ, assuming that the expansion of the type corresponding to its new root sort has already been done;
3. if the $\trianglelefteq$ relation (which due to the above point ii is dynamically computed) proves to be acyclic, then the expansion of the input grammar $\mathcal{G}$ proceeds in finite time;
4. if the application of $(***)$ on the theory/grammar $\mathcal{G}$ as described above terminates, it ensures the satisfaction of the type-consistency condition for the expanded, newly obtained form of $\mathcal{G}$.
5. if the input form of $\mathcal{G}$ is such that for any subterm ψ in the grammar, if $root(\psi) = s$, then $Arity(\psi) \subseteq Arity(\Psi(s))$, then the well-typedness condition ii defined in Section 1 is fully satisfied by the expanded form of $\mathcal{G}$, therefore the it is totally well-typed;
6. one can see that if the above initial/static definition of the relation $\lhd$ defined on the input grammar $\mathcal{G}$ is extended such that if $s \trianglelefteq t$ and $s' \prec t$, then $s' \lhd t$, and $\trianglelefteq$ is acyclic, type-consistent expansion terminates in finite time on $\mathcal{G}$, and OSF-theory unification w.r.t. the expanded form of $\mathcal{G}$ is guaranteed to terminate.

Concerning the Remark 2 from above, the reader should note that at this point the relation $\lhd$ must be re(de)fined, to include $root(\phi) \lhd root(\psi)$, and we have to check again for the $\trianglelefteq$ acyclicity (cycles would correspond to type recursiveness).

Lazy, OSF-theory unification used in the type-consistent expansion phase is achieved in ABC Light by a function consistent_osf_unify which extends the function *osf-unify* in [Aït-Kaci and Di Cosmo, 1993]. The key issue in defining this extension is that the type constraints corresponding to critical sorts must be locally checked (by calling a function check_osf_unify_result) after the execution of *osf-unification*. The information about the places (inside the to-be-unified terms) on the heap where those type constraints must be checked is collected "on fly" via (an extended form of) the bind_refine function in [Aït-Kaci and Di Cosmo, 1993].

Example 3 *Consider the OSF-theory made of $\Psi(a) = \psi_1$ and $\Psi(d)$, like in Example 1, and let's say that we have to expand (a term containing) ψ_2. OSF–theory unification will first obtain ψ_3 out of from ψ_2. Since during unification the root sort of ψ_2.*FEAT1 *is refined to d, a new unification is launched: $\Psi(d)$ is unified with ψ_2.*FEAT1*, causing ψ_3 to be refined into the form of ψ_4.*

[21] Here, $\wedge$ must be understood as OSF-theory unification.

Finally, the ABC Light system softens the well-typing condition ii through *unfilling* [Gerdemann, 1995], a constraint relaxation technique for type systems satisfying appropriateness constraints. Its aim is to get a smaller grammar equivalent with the type-consistent/well-typed grammar provided. The idea behind unfilling is very simple: certain maximal appropriateness constraints may be eliminated. Technically: for every type $\Psi(s)$ in the grammar, if it contains a feature constraint $\psi.f \doteq \psi'$, (ψ is a subterm of $\Psi(s)$), then this constraint can be eliminated from $\Psi(s)$ if

1. ψ' is the atomic feature structure $Approp(f, Intro(f))$, non-corefered in $\Psi(s)$, and
2. if ψ is the type $\Psi(s)$ itself, then $s \neq Intro(f)$.

Example 4 *Let us consider again the grammar described in the Example 2. When the order-consistent form of this grammar is further expanded, type constraints are propagated at all non-atomic nodes of the feature structures in the grammar. In this way, for instance, the variable #3 in the description of* satisfy_HPSG_principles *becomes subject to the sort constraint #3:categ. The fully expanded* lh_phrase *type becomes like in Figure 7. Further on, when doing unfilling, the* HEAD *and* COMP *features will be eliminated from both* lh_phrase*'s head and complement substructures (#7, and respectively #8).*

```
lh_phrase
[ PHON   diff_list
         [ FIRST #1:list,
           REST  #3:list ],
  CAT    #4:categ,
  SUBCAT #5:categ_list,
  HEAD   #7:phrase_or_word
            [ PHON   diff_list
                     [ FIRST #1:list,
                       REST  #2:list ],
              CAT    #4:categ,
              SUBCAT categ_cons
                     [ FIRST #6:categ,
                       REST  #5:categ_list ],
              HEAD   phrase_or_word,
              COMP   phrase_or_word ],
  COMP   #8:phrase_or_word
            [ PHON   diff_list
                     [ FIRST #2:list,
                       REST  #3:list ],
              CAT    #6:categ,
              SUBCAT nil,
              HEAD   phrase_or_word,
              COMP   phrase_or_word ],
  ARGS   <#7, #8> ]
```

Figure 7: The fully expanded form of the type lh_phrase in Example 2.

Implementation hints:
Actually, the expander module in ABC Light uses a quasi-destructive version of the osf_unify function (respectively consistent_osf_unify), namely the *directed OSF-*

unifier. It affects only its second operand, leaving intact the first one, found lower on the heap.[22] If the second operand is the highest on the heap, then the unification result will be/remain in a compact area. This second property is exploited by different functions which are invoked during or immediately after expansion, because it enables us to work with simple, iterative versions instead of recursive ones.

Therefore, the type-consistent expansion task is achieved in a straightforward manner:

- critical sorts in the unexpanded $\Psi(t)$ are easily identifiable assuming that its representation on the heap is contiguous, i.e, using the cells [r, wH-1];
- the other critical sorts are collected "on fly" via bind_refine.

Details on the type-consistent and directed OSF-unifiers used in ABC Light can be found in [Ciortuz, 2001].

5 Conclusion

ABC Light is a compiler that we implemented at DFKI-Saarbrücken, Germany. It translates HPSG-like grammars into C via an abstract machine.

The objective of our previous paper about ABC Light [Ciortuz, 2000] was to show how the AM designed by Aït-Kaci and Di Cosmo to perform ψ-term unification (or, equivalently: empty OSF-theory unification) can be upgraded so to perform OSF-theory unification for a substantial class of OSF-theories, namely the order- and type-consistent theories.

The present paper shows how to expand a OSF-theory, i.e., how to put it (if possible) in an order- and type-consistent form suitable for parsing.

On-line type expansion presented in [Ciortuz, 2000] completes "by need" at the run time the off-line (order- and type-consistent) expansion presented in this paper. That is, when a leaf node of a term gets framed (i.e. it is associated at least one feature) during the unification process, or the sort sort of a non-atomic FS ψ gets more refined ($s \prec t$), then the unfolding operation ($***$) is applied. However, this time, the conjunction $\wedge$ in ($***$) is replaced by compiled unification, i.e., the system calls a compiled "program" function corresponding to $\Psi(root(\psi))$.

The expansion of the LinGO grammar — 2.5MB in source code in Light format — took 33.8 seconds on a SUN Sparc station at 400MHz. Unfilling reduced the expanded form of LinGO grammar with a factor of 2.8, bringing its size down to 40.4MB. U. Callmeier reported that our partial/Light expansion strategy reducing the size of LinGO by a factor of 41%, compared to the totally expanded (well-typed) form computed in other systems like LKB and TDL.

[22] An indexation scheme minimises the number of copies that must be done when carrying information (i.e., substructures) from the first operand into the second.

In our last measurements, both ABC Light and PET — which in [Oepen *et al.*, 2000] was reported as the fastest system running LinGO — parsed the CSLI Stanford test suite on a SUN Sparc machine at 400MHz reporting the average speed of 0.04sec/sentence (ABC Light is several percents faster than PET). If the so called "quick-check" pre-unification filter is disabled, ABC Light goes significantly (40%) faster than PET. We currently work on an improved compiled form of the quick-check to be included in ABC Light.

References

[Aït-Kaci and Di Cosmo, 1993] H. Aït-Kaci and R. Di Cosmo. Compiling order-sorted feature term unification. Technical report, Digital Paris Research Laboratory, 1993. PRL Technical Note 7, downloadable from http://www.isg.sfu.ca/life/.

[Aït-Kaci *et al.*, 1989] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11, N0. 1:115–146, 1989.

[Aït-Kaci *et al.*, 1993] H. Aït-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. In Dalle Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pages 506–524, Vancouver, 1993. The MIT Press. Downloadable from http://www.isg.sfu.ca/life/.

[Aït-Kaci *et al.*, 1997] H. Aït-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. *Journal of Logic, Language and Information*, 30:99–124, 1997.

[Brown and Manandhar, 2000] J. C. Brown and S. Manandhar. Compilation versus abstract machines for fast parsing of typed feature strcuture grammars. *Future Generation Computer Systems*, 16 (2000):771–791, 2000.

[Callmeier, 2000] U. Callmeier. PET — a platform for experimentation with efficient hpsg processing techniques. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99–108, 2000.

[Carpenter and Penn, 1992] B. Carpenter and G. Penn. ALE: The Attribute Logic Engine. User's Guide. Technical report, Carnegie-Mellon University. Philosophy Department. Laboratory for Computational Linguistics, Pittsburgh, 1992.

[Carpenter, 1992] B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution.* Cambridge University Press, 1992.

[Ciortuz, 2000] L.-V. Ciortuz. Scaling up the abstract machine for unification of OSF-terms to do head-corner parsing with large-scale typed unification grammars. In *Proceedings of the ESSLLI 2000 Workshop on Linguistic Theory and Grammar Implementation*, pages 57–80, Birmingham, UK, August 14–18, 2000.

[Ciortuz, 2001] L.-V. Ciortuz. Compiling HPSG into C. Research report, Computer Science Department, University of York, UK, 2001. (In preparation).

[Copestake *et al.*, 1999] A. Copestake, D. Flickinger, and I. Sag. *A Grammar of English in HPSG: Design and Implementations*. Stanford: CSLI Publications, 1999.

[Copestake, 1999] A. Copestake. *The (new) LKB system*. CSLI, Stanford University, 1999.

[Copestake, 2000] A. Copestake. Definitions of typed feature structures. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG), 2000.

[Gerdemann, 1995] D. Gerdemann. Term encoding of typed feature structures. In *Proceedings of the 4th International Workshop on Parsing Technologies*, pages 89–97, Prague, Czech Republik, 1995.

[Götz, 1994] Thilo Götz. A normal form for typed feature structures. Magisterarbeit, Universität Tübingen, Tübingen, Germany, 1994.

[Kay, 1989] M. Kay. Head driven parsing. In *Proceedings of Workshop on Parsing Technologies*, Pittsburg, 1989.

[Krieger and Schäfer, 1994] H.-U. Krieger and U. Schäfer. TDL – A Type Description Language for HPSG. Research Report RR-94-37, German Research Center for Artificial Intelligence (DFKI), 1994.

[Krieger and Schäfer, 1995] H.-U. Krieger and U. Schäfer. Efficient parameterizable type expansion for typed feature formalisms. In *Proceedings of IJCAI'95*, 1995.

[Malouf *et al.*, 2000] R. Malouf, J. Carroll, and A. Copestake. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):29–46, 2000.

[Miyao *et al.*, 2000] Y. Miyao, T. Makino, K. Torisawa, and J. Tsujii. The LiLFeS abstract machine and its evaluation with the LinGO grammar. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):47–61, 2000.

[Oepen *et al.*, 2000] S. Oepen, D. Flickinger, H. Uszkoreit, and J. Tsujii. Introduction to the special issue on efficient processing with hpsg: Methods, systems, evaluation. *Journal of Natural Language Engineering*, 6 (1), 2000.

[Pollard and Sag, 1994] C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. Center for the Study of Language and Information, Stanford, 1994.

[Shieber *et al.*, 1995] S.M. Shieber, Y. Schabes, and F. Pereira. Principles and implementation of deductive parsing. *Jornal of Logic Programming*, pages 3–36, 1995.

[Shieber, 1992] S. Shieber. *Constraint-Based Grammar Formalisms – Parsing and Type Inference for Natural and Computer Languages*. MIT Press, 1992.

[Sikkel, 1997] N. Sikkel. *Parsing Schemata*. Springer Verlag, 1997.

[Tomabechi, 1991] H. Tomabechi. Quasi-destructive graph unification. In *Proceedings of the 29th meeting of the Association for Computational Linguistics*, pages 315–322, Berkeley, 1991.

[Warren, 1983] D.H.D. Warren. An abstract Prolog instruction set. Technical report, SRI International, Menlo Park, CA, 1983. Technical Note 309.

[Wintner and Francez, 1999] S. Wintner and N. Francez. Efficient implementation of unification-based grammars. *Journal of Language and Computation*, 1(1):53–92, 1999.

[Wintner, 1997] S. Wintner. *An Abstract Machine for Unification Grammars*. PhD thesis, Technion – Israel Institute of Technology, 32000 Haifa, Israel, 1997.

[Wroblewski, 1987] D. A. Wroblewski. Non-destructive graph unification. In Dalle Miller, editor, *Proceedings of the 6th national conference on artificial intelligence (AAI'87)*, pages 582–587, Seattle, 1987.