Abstract

We present an analysis of multiple question fronting in a restricted variant of the HPSG formalism (DELPH-IN) where unification is the only natively defined operation. Analysing multiple fronting in this formalism is challenging, because it requires carefully handling list appends, something that HPSG analyses of question fronting heavily rely on. Our analysis uses the append list type to address this challenge. We focus the testing of our analysis on Russian, although we also integrate it into the Grammar Matrix customization system where it serves as a basis for cross-linguistic modeling. In this context, we discuss the relationship of our analysis to lexical threading and conclude that, while lexical threading has its advantages, modeling multiple extraction cross-linguistically is easier without the lexical threading assumption.

1   Introduction

We present an analysis of multiple constituent question fronting in HPSG. We take prototypical constituent (aka *wh-*) questions to be a conventional and direct way of asking for information (Idiatov, 2007, p.6):

(1)   Who arrived? [eng]

(2)   Who saw what? [eng]

(3)   Who do you think arrived? [eng]

(4)   Kto   chto
      who.NOM what.ACC
      videl?
      see.PST.3SG
      'Who saw what?' [rus][1]

Constituent questions are a case of *long distance dependency* constructions (LDD) meaning that the question phrase can appear outside of the boundary of the clause to which it belongs (3).

Languages differ with respect to how many question phrases can front. Famously in Slavic languages, all question phrases may be fronted (4). Our goal is to systematically account for the data, presented in §2, where question words, one or more, may appear at the left edge of the clause. We do so by developing a Russian grammar fragment and integrating it into a cross-linguistic grammar engineering framework, the Grammar Matrix (Bender et al., 2002, 2010).

Our work is couched within the DELPH-IN joint reference formalism (JRF; Copestake, 2002a, p.227), a restrictive variant of HPSG developed to balance expressivity with computational efficiency. It does not allow *relational constraints* which stipulate the value of one feature to be some function of the value of one or more others (other than strict identity). Examples of relational constraints in other variants of HPSG include e.g. shuffle operators (Reape, 1994, p.271). The formalism furthermore requires that the number and order of daughters of each phrase

---

[1]Unless stated otherwise, the examples are constructed by the first author whose native language is Russian. The second author, a native speaker of English, vetted the English examples.

structure rule be fixed in the definition of the rule, precluding systems that separate immediate dominance from linear precedence (e.g. Engelkamp et al., 1992; Kathol, 1995). Thus, our analysis builds on Pollard & Sag 1994 and Ginzburg & Sag 2000 and exists in parallel with linearization-based accounts of multiple fronting such as Penn 1998.

The analysis is part of a larger project (Zamaraeva, forth) the goal of which is to add cross-linguistic support for constituent questions to the Grammar Matrix system (Bender et al., 2002, 2010). The system includes a questionnaire that elicits typological and lexical information about a language from a linguist-user and a back-end logic that customizes the Matrix core grammar types according to the elicited specifications. In addition to facilitating the development of grammars for practical applications, the system also can be used in linguistic hypothesis testing (Bierwisch 1963, p.163, Müller 1999, p.439, Bender et al. 2008; Fokkens 2014; Müller 2015). The resulting grammar fragments are suitable for both parsing and generation and map between surface strings and Minimal Recursion Semantics (MRS; Copestake et al., 2005) representations paired with HPSG feature structures. Integrating an analysis of question fronting into the system extends (i) the platform itself, so that other phenomena (such as relative clauses) can be modeled on top of and in interaction with our analysis; and (ii) the range of hypotheses which can be rigorously tested with the system, such as various combinations of single/multiple and optional/obligatory fronting. We present an analysis of multiple question fronting which represents a hypothesis that data such as Russian can be accounted for with multiple application of the *filler-gap* rule, without natively defined relational constraints.

One analysis that the Grammar Matrix has historically relied upon is *lexical threading*, a concept adapted from Bouma et al. 2001. Lexical threading posits that the length of and the order of elements on the SLASH list (a representation of the gaps in the sentence) is determined at the level of the lexical entry. This allows for an elegant analysis of the English *easy*-adjectives, in particular (Flickinger, 2000). However, we find that overall it complicates the analysis of multiple extraction in languages with flexible word order, particularly in interaction with other phenomena such as coordination. In this paper, we offer two alternatives for an analysis of multiple question fronting: with and without lexical threading. Current work has led to abandoning lexical threading assumptions in the Matrix in favor of a more readily cross-linguistic analysis without it.

The details on the DELPH-IN framework which will be helpful to understand our analysis constitute §3. In particular, we dedicate ample space to the *append-list* type, since, at the time of writing, the existing exposition of append lists is dense. Related work is summarized in §4. Two alternative analyses to account for the data in §2 are presented in §5. We explain how we tested the analyses in §6 and conclude with some thoughts on future work in §7.

2   Data: Multiple question fronting in Russian

Russian exhibits multiple question fronting (5), including in LDD constructions (6), although LDD *wh*-questions may be infrequent.[2]   Multiple adjunct fronting appears either impossible (7) or rare, (8) being the only example we have found so far in the Russian National Corpus. Finally, fronting appears optional (9) and adjuncts can appear in any position with respect to the arguments (5), (10)–(11).

(5)   *Kogda kto   kogo*   videl?
      when   who.NOM who.ACC see.PST
      'When did which person saw which other person?' [rus]

(6)   *Kogda kto   kogo*   ty   tochno znaesh (chto) videl?
      when   who.NOM who.ACC 2SG for.sure know   (that) see.PST
      'When do you know for sure who saw whom?' ('What are the sets of times and persons such that one person saw another at a certain time, such that you know this set of facts for sure?') [rus]

(7)   ??*Kogda gde*   my   kupili   eti   knigi?
      when   where 1PL.NOM buy.PAST.1PL this.PL.ACC book.PL.ACC
      Intended: 'When [and] where did we buy these books?'

(8)   DP. ru vypustilo   infografiku   obo vseh   kvartirah
      DP. ru publish.PAST infographic.ACC of   all.PREP apartment.PREP
      Dostoevskogo   (*gde   kogda* zhil,   *gde   chto* napisal)
      Dostoyevsky.GEN (where when live.PAST, where what write.PAST)
      'DP.ru published an infographic about all Dostoevsky's apartments (where he lived when, where he wrote what) [rus] (Oborin, 1987, RNC)

(9)   Ty *gde*   rabotaesh?
      2SG where work.2SG
      'Where do you work?' [rus][3]

(10) *Kto   kogo   kogda* videl?
      who.NOM who.ACC when see.PST
      'Who saw whom when?' [rus]

(11) *Kto   kogda kogo*   videl?
      who.NOM when who.ACC see.PST
      'Who saw whom when?' [rus]

---

[2]Some literature contends that they are not possible (Stepanov & Stateva, 2006) but the first author has observed herself producing such constructions, and we have found examples on the web, such as below:

(i)   I   kto   ty   dumaesh   budet   tretjim?
      And who.NOM 2SG.NOM think.2SG.PRES be.3SG.FUT third.INSTR
      'And who do you think will be the third [in the group]?' [rus] (Galikhin, 2017, loc.246)

[3]This very common Russian sentence was pointed out to the first author by John F. Bailyn in personal communication.

3 Background

This section briefly reviews the general approach to LDD in HPSG (§3.1); explains the specifics of the version of HPSG which we use here, paying special attention to *list* types which are used for non-local features (§3.2); and concludes with some characteristics of the Grammar Matrix system which are relevant to the presentation of our work (§3.3).

3.1 Non-local features and question fronting in HPSG

In GPSG (Gazdar, 1981) and subsequently HPSG (Pollard & Sag, 1994; Ginzburg & Sag, 2000), the analysis of long distance dependencies (LDD) relies on set-valued non-local features SLASH, QUE, and REL. The SLASH feature is used to account for constituents which do not appear in their usual place, and distinct features REL and QUE serve the separate analyses of relative clauses and constituent questions, respectively. For a fronted constituent question, the headed filler-gap rule licenses a phrase with two daughters, a head daughter with a nonempty SLASH value, and a "filler" daughter that has a nonempty QUE value and matches an element of that SLASH value in its LOCAL feature values. The nonempty SLASH value is ultimately licensed by an extraction rule.

Bouma et al. (2001) suggested an influential idea of SLASH amalgamation at the level of the lexical entry, a mechanism which here we call *lexical threading*. A lexical entry combines the NON-LOCAL features of its arguments; thus, a verb's SLASH is the union of the verb's subject's and complements' SLASH sets.[4] At the lexical level, the arguments' SLASH sets are underspecified, but they are specified once the arguments have been realized (either as a constituent or as a gap). The SLASH set is propagated via the head, without the need to stipulate any additional constraints at the level of phrase structure rules. What this means in context of extraction is that the extraction rules do not combine or extend SLASH sets but merely specify that a particular set is nonempty (for example, as discussed in §5.1, the subject extraction rule (26) constrains the SUBJ's SLASH list to be nonempty, by using the *gap* type (25)).

3.2 DELPH-IN Joint Reference Formalism

DELPH-IN (DEep Linguistic Processing with HPSG INitiative)[5] is an international consortium of researchers who are interested in engineering grammars using HPSG. Furthermore, the DELPH-IN Joint Reference Formalism (JRF; Copestake, 2002a, p.227) is a version of HPSG restricted to rely on only unification as a native operation, without relational constraints such as list reordering or counting. The

---

[4]Bouma et al. (2001) actually use a single DEPS feature instead of SUBJ and COMPS, and furthermore DEPS includes adjuncts, but the decision to use DEPS is separate from the decision to use lexical threading, and we will not discuss DEPS further here.

[5]http://www.delph-in.net

design of the DELPH-IN JRF aims to balance linguistic considerations with engineering ones. On the one hand, it should be possible to implement broad-coverage precision grammars, and on the other hand, it should be possible to effectively use such a grammar in practical applications. (For further discussion, see: Bender & Emerson, 2020, §3.2.)

For the purposes of the non-local features SLASH, QUE, and REL, the most important characteristic of the DELPH-IN JRF is the need to use lists instead of sets. While set-valued features are often used in HPSG, unification of sets is not guaranteed to produce a unique result (Pollard & Moshier, 1990; Moshier & Pollard, 1994). So that unification always produces a unique result, the DELPH-IN JRF does not allow set-valued features, which means that features like SLASH must be list-valued rather than set-valued. A list fixes the order of its elements, and combining two lists (*appending* them) must similarly fix the order.

3.2.1 Lists

Lists can be implemented in the DELPH-IN JRF as follows. The type *list* has two subtypes *nonempty-list* and *empty-list*. The *nonempty-list* type has two features, as shown in (12), where FIRST holds the first element of the list (which can be of any type, hence the most general type *top*), and REST holds the rest of the list. This allows a list to be specified recursively, following the REST feature multiple (0 or more) times. A fully specified list consists of *nonempty-list* multiple times, eventually terminating in an *empty-list*, illustrated in (13) for the list $\langle a,b \rangle$.

$$
(12) \quad
\begin{bmatrix}
\textit{nonempty-list} \\
\text{FIRST} \quad \textit{top} \\
\text{REST} \quad \textit{list}
\end{bmatrix}
\qquad
(13) \quad
\begin{bmatrix}
\textit{nonempty-list} \\
\text{FIRST} \quad a \\
\text{REST} \quad
\begin{bmatrix}
\textit{nonempty-list} \\
\text{FIRST} \quad b \\
\text{REST} \quad \textit{empty-list}
\end{bmatrix}
\end{bmatrix}
$$

3.2.2 Difference lists

As mentioned above, the only native operation in the DELPH-IN JRF is unification. However, in order to manipulate the SLASH feature, we would like to be able to append lists. Because a fully specified list terminates with an *empty-list*, the list cannot be extended further. One solution, which DELPH-IN grammars and the Grammar Matrix in particular have relied on so far, is to use *difference lists* (for an exposition, see: Copestake, 2002b, §4.3).[67] The basic idea is that, rather than working with fully specified lists, we can work with underspecified lists which are

---

[6]The concept of difference lists dates back to the early history logic programming (Geske & Goltz, 2007)

[7]Another solution is to use so-called *junk slots* (Aït-Kaci, 1984) (for a summary, see: Götz & Meurers, 1996). However, junk slots require disjunctive type definitions and fully sort-resolved feature structures, which are not part of the DELPH-IN JRF.

easier to append — in particular, lists which end with an underspecified *list*, rather than a *nonempty-list*. We will refer to such a list as a *open* list, in contrast to a fully specified *closed* list.

The *diff-list* type wraps an open list to make list appends convenient. It has two features, as shown in (14), where the value of LIST is intended to be an open list, and the value of LAST is intended to be the open end of that list. The definition in (14) doesn't enforce the fact that LAST should point to the end of the list in LIST, but for a difference list to be useful, this needs to be true. An example is given in (15), for a difference list $\langle ! a,b ! \rangle$. Note that $\boxed{1}$ is of type *list*.

$$
(14)\quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} & \textit{list} \\
\text{LAST} & \textit{list}
\end{bmatrix}
\qquad
(15)\quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} &
\begin{bmatrix}
\textit{nonempty-list} \\
\text{FIRST} & a \\
\text{REST} &
\begin{bmatrix}
\textit{nonempty-list} \\
\text{FIRST} & b \\
\text{REST} & \boxed{1}\textit{list}
\end{bmatrix}
\end{bmatrix} \\
\text{LAST} & \boxed{1}
\end{bmatrix}
$$

By keeping track of the notional end of the list, using the LAST feature, it is possible to append lists, as shown in (16), where the first *diff-list* is the append of the following two.

$$
(16)\quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} & \boxed{1} \\
\text{LAST} & \boxed{3}
\end{bmatrix}
\quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} & \boxed{1} \\
\text{LAST} & \boxed{2}
\end{bmatrix}
\quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} & \boxed{2} \\
\text{LAST} & \boxed{3}
\end{bmatrix}
$$

Difference lists make it possible to append lists, but there is an important downside, because the notional list is not the same as the value of the LIST feature. Notionally, the contents of a difference list start at LIST, and end at LAST (15). However, once a difference list has been appended to, the value of LAST is the next list, and so the LIST actually contains not only the notional list, but also all lists appended to it.

Because of this, there is an important but awkward division of labour between a difference list and the value of its LIST. The elements of the notional list are to be found in the value of LIST, but the length of the notional list is implicitly defined by the value of LAST. Because the length is only implicitly defined, it is not directly accessible, which means it is even difficult to check if the notional list is empty or nonempty. For this reason, Flickinger (2000) constrained SLASH lists to be of length of at most 1 (which is sufficient for almost all of English), and this constraint was inherited by the Grammar Matrix. However, to accommodate multiple question fronting, this constraint needs to be taken out. While it is possible to analyse multiple long-distance dependencies using difference lists

(Crysmann, 2015), working with difference lists is error-prone, and so we present an alternative that makes it easier to implement a grammar and maintain it.

3.2.3 Append lists

Emerson (2017, 2019) proposed *append lists*[8] as an alternative to difference lists, where there is no discrepancy between the notional list and the value of LIST. This makes working with append lists relatively straightforward.[9]

The *append-list* type wraps a list, using the LIST feature. This can be treated exactly as a normal list, and in particular it can be a closed list (unlike the *diff-list* type, where the list must be open to allow appends). The *append-list* type also has an APPEND feature, as shown in (17), which can be used to specify that this append list is the result of appending some other append lists.

$$
(17) \quad
\begin{bmatrix}
\textit{append-list} \\
\text{LIST} \quad \boxed{0}\textit{list} \\
\text{APPEND} \quad
\begin{bmatrix}
\textit{list-of-append-lists} \\
\text{APPEND-RESULT} \quad \boxed{0}
\end{bmatrix}
\end{bmatrix}
$$

Append lists are easy to use when writing a grammar, with an example shown in (18), where the first append list is the result of appending the second and third append lists. The first append list's LIST value is $\langle a,b,c \rangle$, with these elements being token-identical to the elements in the second and third lists. In comparison to (16), a grammarian does not need to worry about linking up the end of one list with the start of the next.

$$
(18) \quad
\begin{bmatrix}
\textit{append-list} \\
\text{APPEND} \quad \langle \boxed{1}, \boxed{2} \rangle
\end{bmatrix}
\boxed{1}
\begin{bmatrix}
\textit{append-list} \\
\text{LIST} \quad \langle a, b \rangle
\end{bmatrix}
\boxed{2}
\begin{bmatrix}
\textit{append-list} \\
\text{LIST} \quad \langle c \rangle
\end{bmatrix}
$$

The following two sections can be safely skipped by a reader uninterested in the technical details of how append lists are implemented.

3.2.3.1 The *list-of-append-lists* type

Closed lists cannot be directly appended, so the *list-of-append-lists* type first creates an open list from each closed list, and then appends the open lists in the same way as with difference lists. Just as the *list* type has two subtypes, we have the subtypes *nonempty-list-of-append-lists* and *empty-list-of-append-lists*, with constraints as defined in (19)–(20). Each list in a *list-of-append-lists* is unified with the type *list-with-diff-list*, which creates an open list (a *diff-list*) containing the

---

same elements. The LAST of each diff-list is identified with the result of appending the remaining lists (compare (19) against the re-entrancy $\boxed{2}$ in (16)). This continues recursively until the end of the list (*empty-list-of-append-lists*), where the result is simply an empty list.

$$
(19)\quad
\begin{bmatrix}
\textit{nonempty-list-of-append-lists} \\[4pt]
\text{FIRST|LIST} \quad
\begin{bmatrix}
\textit{list-with-diff-list} \\[4pt]
\text{DIFF-LIST} \quad
\begin{bmatrix}
\textit{diff-list} \\
\text{LIST} \quad \boxed{1} \\
\text{LAST} \quad \boxed{2}
\end{bmatrix}
\end{bmatrix} \\[20pt]
\text{REST} \quad
\begin{bmatrix}
\textit{list-of-append-lists} \\
\text{APPEND-RESULT} \quad \boxed{2}
\end{bmatrix} \\[12pt]
\text{APPEND-RESULT} \quad \boxed{1}
\end{bmatrix}
$$

$$
(20)\quad
\begin{bmatrix}
\textit{empty-list-of-append-lists} \\
\text{APPEND-RESULT} \quad \textit{empty-list}
\end{bmatrix}
$$

3.2.3.2 The *list-with-diff-list* type

Finally, the *list-with-diff-list* type is a subtype of *list*, which creates a *diff-list* containing the same elements. The re-entrancy $\boxed{1}$ ensures the new list contains the same elements, $\boxed{2}$ ensures the new list is linked up correctly, $\boxed{3}$ propagates the new open end of the list, and $\boxed{4}$ creates the new open end of the list. This is shown graphically in Figure 1.

$$
(21)\quad
\begin{bmatrix}
\textit{nonempty-list-with-diff-list} \\
\text{FIRST} \quad\quad \boxed{1} \\[6pt]
\text{REST} \quad
\begin{bmatrix}
\textit{list-with-diff-list} \\[4pt]
\text{DIFF-LIST} \quad
\begin{bmatrix}
\text{LIST} \quad \boxed{2} \\
\text{LAST} \quad \boxed{3}
\end{bmatrix}
\end{bmatrix} \\[20pt]
\text{DIFF-LIST} \quad
\begin{bmatrix}
\text{LIST} \quad
\begin{bmatrix}
\text{FIRST} \quad \boxed{1} \\
\text{REST} \quad \boxed{2}
\end{bmatrix} \\[12pt]
\text{LAST} \quad \boxed{3}
\end{bmatrix}
\end{bmatrix}
$$

$$
(22)\quad
\begin{bmatrix}
\textit{empty-list-with-diff-list} \\[4pt]
\text{DIFF-LIST} \quad
\begin{bmatrix}
\text{LIST} \quad \boxed{4} \\
\text{LAST} \quad \boxed{4}
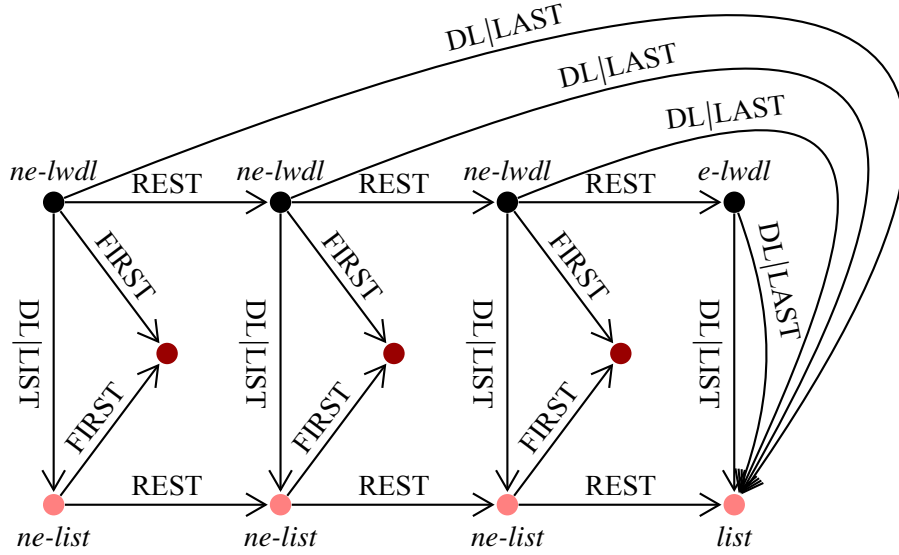\end{bmatrix}
\end{bmatrix}
$$

Figure 1: Example of creating an open list from a closed list. The closed list is at the top (black nodes), the open list is at the bottom (pink nodes), and the elements of the list are in the middle (red nodes). In type names, *e* and *ne* stand for *empty* and *nonempty*, and *lwdl* stands for *list-with-diff-list*; DL stands for DIFF-LIST.

3.3 Grammar Matrix

The Grammar Matrix (Bender et al., 2002, 2010) is a DELPH-IN-based grammar customization system. This means that the user fills out a web-based questionnaire with typological, lexical, and morphological information, and, based on the particular combination of such choices, the system applies a programmed customization logic to the right set of 'core' types[10] and outputs an implemented grammar fragment. Loaded into a parser such as the LKB system (Copestake, 2002b) or ACE (Crysmann & Packard, 2012), the grammars automatically map sentences to syntactic HPSG and semantic MRS structures.

The analysis presented here is part of the constituent questions library for the Grammar Matrix (Zamaraeva, forth). As such, we build on the existing analyses — on the lexical and phrasal types implemented in the Grammar Matrix, including those for word order, modification, argument extraction and filler-gap construction, reimplementing them with append lists instead of difference lists. As detailed in §5, most of the novelty we present here is in the space of adjunct extraction, along with the lexical threading-free version of the whole system which relies on append lists instead of difference lists.

Lexical threading was implemented in the Grammar Matrix like in the ERG

---

[10]Not to be confused with "core vs. periphery" as in Chomsky 1995. The Matrix core types were originally distilled from the English Resource Grammar (Flickinger, 2000), as part of Bender et al. 2002.

(Flickinger, 2000). Most lexical entries inherit from an appropriate supertype, depending on the length of the ARG-ST. For example, the Russian verb *videl* from (4) would be a subtype of *basic-two-arg-lex-item* lexical threading supertype (23).

(23)
$$
\begin{bmatrix}
\textit{basic-two-arg-lex-item} \\[4pt]
\text{ARG-ST} \quad \left\langle \begin{bmatrix} \text{NON-LOCAL} & \begin{bmatrix} \text{SLASH} & \boxed{1} \\ \text{REL} & \boxed{2} \\ \text{QUE} & \boxed{3} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{NON-LOCAL} & \begin{bmatrix} \text{SLASH} & \boxed{4} \\ \text{REL} & \boxed{5} \\ \text{QUE} & \boxed{6} \end{bmatrix} \end{bmatrix} \right\rangle \\[20pt]
\text{SYNSEM}|\text{NON-LOCAL} \quad \begin{bmatrix} \text{SLASH}|\text{APPEND} & \langle \boxed{1}, \boxed{4} \rangle \\ \text{REL}|\text{APPEND} & \langle \boxed{2}, \boxed{5} \rangle \\ \text{QUE}|\text{APPEND} & \langle \boxed{3}, \boxed{6} \rangle \end{bmatrix}
\end{bmatrix}
$$

The Grammar Matrix has a *regression testing* system associated with it so that any change to the core type hierarchy or to the customization logic is ensured to not have broken any of the previous analyses (Bender et al., 2007). Pairings of language specifications and test suites are stored along with the *gold* semantic representations in the MRS formalism. Each specification–test suite pair represents some language, real or artificial. At the time of writing this paper, there are 499 languages in the regression testing system, 56 of them natural languages. The size of the test suite ranges from 1 to 6165 sentences, the average being 34. The Matrix provided us with testing grounds for our analysis, as explained in §6.

To summarize, our analysis (§5) is situated within a framework which both dictates a number of design decisions (e.g. treating non-local features as lists) and provides us with means for testing our analysis of question fronting cross-linguistically and in interaction with other phenomena.

4   Related Work

As explained above, our analysis exists in parallel to analyses of Slavic languages which use non-DELPH-IN variants of HPSG, like Penn 1998, Przepiórkowski 1998, and Chaves & Paperno 2007, and so cannot be informed by them directly (e.g. we do not have at our disposal a natively defined *shuffle*-operator). In terms of data, we agree with Przepiórkowski (1998), *inter alia*, that any apparent restrictions on the order of extraction should probably not be explained solely on syntactic grounds (and as such we leave them out of scope in §2).

Several grammars of Slavic languages written in the DELPH-IN JRF exist (Avgustinova & Zhang, 2009; Osenova, 2010; Fokkens & Avgustinova, 2013) but none of them cover multiple questions. Osenova 2010 does include an account of single questions as well as relative clauses. Being a Matrix-based grammar, Osenova 2010 also ends up relaxing the non-local constraints inherited from the ERG so as to allow *wh*-words in non-fronted positions, as do we in §5.

Sag et al. (2003, p.452) describe multiple extraction as part of an analysis of English topicalization. To our knowledge, multiple extraction as suggested by Sag et al. (2003) was implemented in the DELPH-IN JRF once before, by

Crysmann (2015) for resumptive pronouns in Hausa. We implement Sag et al.'s (2003) analysis for interrogatives and make it available for automated Grammar Matrix-based implementation after testing it for cross-linguistic applicability.

Append lists are a relatively new concept, and our work is one of the first examples of how they can be used in DELPH-IN grammars. They were first used by Aguila-Multner & Crysmann (2018) for gender resolution in French.

5   Analysis

We would ultimately like to analyse the data in §2 including the flexible order of extracted elements (5), (10)–(11) using recursive application of one filler-gap rule. Append lists, as presented in §3.2.3, allow us to manipulate the SLASH, QUE, and REL features for this purpose.

We offer two alternative analyses, one with the lexical threading assumption and one without. Each option has its advantages and disadvantages, and while for the purposes of the Grammar Matrix we favor the second option, the first option could also serve as a basis for future work.

Lexical threading makes possible an elegant analysis of *easy*-adjectives (Sag et al. 2003, p.439, Flickinger 2000), which would otherwise require additional phrasal rules; the analysis of morphological marking of questions is also easier.[11] However, the combination of *append-list* and lexical threading makes the analysis of VP coordination more problematic.[12]

On the other hand, without lexical threading, we give a simple account of multiple extraction of arguments and adjuncts in the context of flexible word order and have no issues with coordination while also gaining in parsing speed, as multiple adjunct extraction rules are costly for the parser performance.[13]

Under both analyses, at the level of the filler-gap phrase, what is required is simply restating as (24) the version suggested by Sag et al. 2003, (p.448), except in terms of append lists (so, the value of SLASH has a feature LIST).

---

[11] Assuming lexical threading, one can simply state that a lexical rule applies to e.g. a QUE-nonempty verb, to distinguish lexical rules which participate in an interrogative paradigm. This is not possible without lexical threading because one needs to explicitly state non-local constraints on the verb's various arguments. See also: Zamaraeva (forth).

[12] With lexical threading, both the "input" and "output" of the append operation are accessible in the feature structure, e.g. by looking at a VP's SLASH and SUBJ|...SLASH. However, if adjuncts are not included in lexical threading, then there can be any number of append operations between the SLASH and SUBJ|...SLASH. Even if two coordinated VPs have compatible values for SLASH|LIST and SUBJ|...SLASH|LIST, they may have incompatible values for SLASH and SUBJ|...SLASH, which means that more care is required in writing coordination rules. This illustrates that, unlike with "true" relational constraints, unifying a feature structure with one of the types introduced in §3.2.3 permanently modifies that structure.

[13] On the Russian test described in §6 and with the LKB parser run on a MacBook Pro 2015 laptop with 16GB memory and 3.1GHz Intel Core i7 processor, Analysis 1 speed is 1.47 seconds per sentence on average; Analysis 2 speed is 0.39 seconds.

(24)
$$\begin{bmatrix} \textit{filler-gap-phrase} \\ \text{SLASH} \quad \boxed{1} \\ \text{ARGS} \quad \left\langle \boxed{2}, \begin{bmatrix} \text{SLASH|LIST} \begin{bmatrix} \text{FIRST} & \boxed{2} \\ \text{REST} & \boxed{1} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

The two analyses diverge at the level of extraction rules.

5.1 Analysis 1: With lexical threading

At the level of the argument extraction rules, assuming we use lexical threading, we can use the existing Grammar Matrix phrasal types carried over from the English Resource Grammar (Flickinger, 2000; Bender et al., 2002), except we use *append-lists* instead of *diff-lists* and we remove all constraints on the length of the NON-LOCAL lists. Note how (26)–(27) indeed do not even mention the NON-LOCAL features because the SLASH value will be handled by the lexical threading mechanism (23) and the type *gap* (25).

(25)
$$\begin{bmatrix} \textit{gap} \\ \text{SYNSEM} \begin{bmatrix} \text{LOCAL} & \boxed{1} \\ \text{NON-LOCAL|SLASH|LIST} & \left\langle \boxed{1} \right\rangle \end{bmatrix} \end{bmatrix}$$

(26)
$$\begin{bmatrix} \textit{extracted-subj-phrase} \\ \text{SYNSEM} \quad \begin{bmatrix} \text{LOCAL|CAT|VAL|SUBJ} \; \left\langle \right\rangle \end{bmatrix} \\ \text{HEAD-DTR|SYNSEM} \quad \begin{bmatrix} \text{LOCAL|CAT|VAL|SUBJ} \; \left\langle \textit{gap} \right\rangle \end{bmatrix} \end{bmatrix}$$

(27)
$$\begin{bmatrix} \textit{extracted-comp-phrase} \\ \text{SYNSEM} \quad \begin{bmatrix} \text{LOCAL|CAT|VAL|COMPS} \; \boxed{1} \end{bmatrix} \\ \text{HEAD-DTR|SYNSEM} \quad \begin{bmatrix} \text{LOCAL|CAT|VAL|COMPS} \begin{bmatrix} \text{FIRST} & \textit{gap} \\ \text{REST} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The main novelty we present here pertains to adjunct extraction in the space of multiple question fronting. We introduce a small hierarchy as shown in Fig. 2 of adjunct extraction rules which allows extracting exactly one adjunct either before or after the arguments, to account for (5) and (10).
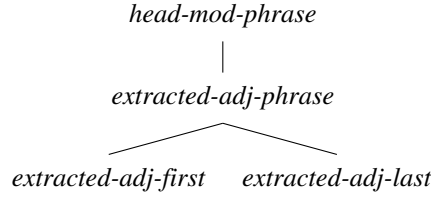
Figure 2: The adjunct extraction rules hierarchy

$$(28) \begin{bmatrix} \textit{extracted-adj-last} \\ \text{SS} \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT|HEAD} & \boxed{1} \\ \text{CONT|HOOK} & \boxed{2} \end{bmatrix} \\ \text{NLOC|SLASH|APP} \left\langle \boxed{3}, \begin{bmatrix} \text{LIST} \left\langle \begin{bmatrix} \text{CAT|HEAD|MOD} \left\langle \begin{bmatrix} \text{LOC} \begin{bmatrix} \textit{intersective-mod} \\ \text{CAT|HEAD} & \boxed{1} \\ \text{CONT|HOOK} & \boxed{2} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \\ \text{MODIFIED} \quad \textit{hasmod} \end{bmatrix} \\ \text{HDR|SS} \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT|HEAD} & \boxed{1} \\ \text{CONT|HOOK} & \boxed{2} \end{bmatrix} \\ \text{NLOC|SLASH} & \boxed{3} \\ \text{MODIFIED} & \textit{notmod} \end{bmatrix} \end{bmatrix}$$

Example (28)[14] shows the rule that is used to extract an adjunct after any arguments (accounting for (5)). Following Flickinger (2000), we block multiple adjunct extraction (7) by the MODIFIED feature and a hierarchy of mutually exclusive types (e.g. *hasmod* vs. *notmod*) appropriate for it.[15] Because SLASH is of type *append-list*, there are no issues with placing the extracted adjunct at the specified position on the mother's SLASH. This works under the lexical threading analysis where the order of the arguments put on the SLASH list is determined at the level of the lexical entry, even before any arguments were actually extracted, but any extracted adjuncts have to be inserted at some specific position (see §3.1).

5.2 Analysis 2: No lexical threading

Without lexical threading, additional phrase structure rules may be required for the English *easy*-adjectives, and modeling interrogative morphology is less straightforward. However, the analysis of extraction becomes simpler, as we may use (28) as the sole adjunct extraction rule. In fact, all extraction rules: subject, complement, and adjunct, append the *gap* element to the existing SLASH list of the head daughter. Without lexical threading, the SLASH list can actually be constructed

---

[14]Abbreviations: SS: SYNSEM, LOC: LOCAL; NLOC: NON-LOCAL; APP: APPEND; HDR: HEAD-DTR

[15]Removing the MODIFIED constraint would allow both (7) and (8), and then the limit on how many adjuncts can be extracted would have to be put either formally or through constraints on the parsing algorithm (the latter could in principle be seen as a way of modeling processing constraints).

Figure 3a (left tree):

```
                    S_filler-gap
                  [SLASH ⟨⟩]
                 /           \
            ③ ADV          S_filler-gap
              |           [SLASH ⟨③⟩]
           kogda         /          \
          'when'     ② NP         S_filler-gap
                       |        [SLASH ⟨②, ③⟩]
                      kto       /          \
                     'who'   ① NP        S_adj-extr
                               |      [SLASH ⟨①, ②, ③⟩]
                             kogo            |
                            'whom'        S_subj-extr
                                        [SLASH ⟨①, ②⟩]
                                             |
                                         VP_comp-extr
                                        [SLASH ⟨①⟩]
                                             |
                                             V
                                         [SLASH ⟨⟩]
                                           videl
                                           'saw'
```

Figure 3b (right tree):

```
                    S_filler-gap
                  [SLASH ⟨⟩]
                 /           \
            ③ NP           S_filler-gap
              |           [SLASH ⟨③⟩]
            kto           /          \
           'who'      ② ADV        S_filler-gap
                        |       [SLASH ⟨②, ③⟩]
                      kogda      /          \
                      'when'  ① NP        S_subj-extr
                                |      [SLASH ⟨①, ②, ③⟩]
                              kogo            |
                             'whom'       S_adj-extr
                                        [SLASH ⟨①, ②⟩]
                                             |
                                         VP_comp-extr
                                        [SLASH ⟨①⟩]
                                             |
                                             V
                                         [SLASH ⟨⟩]
                                           videl
                                           'saw'
```

(a) The extracted adjunct is in front of the extracted arguments, as in example (5).

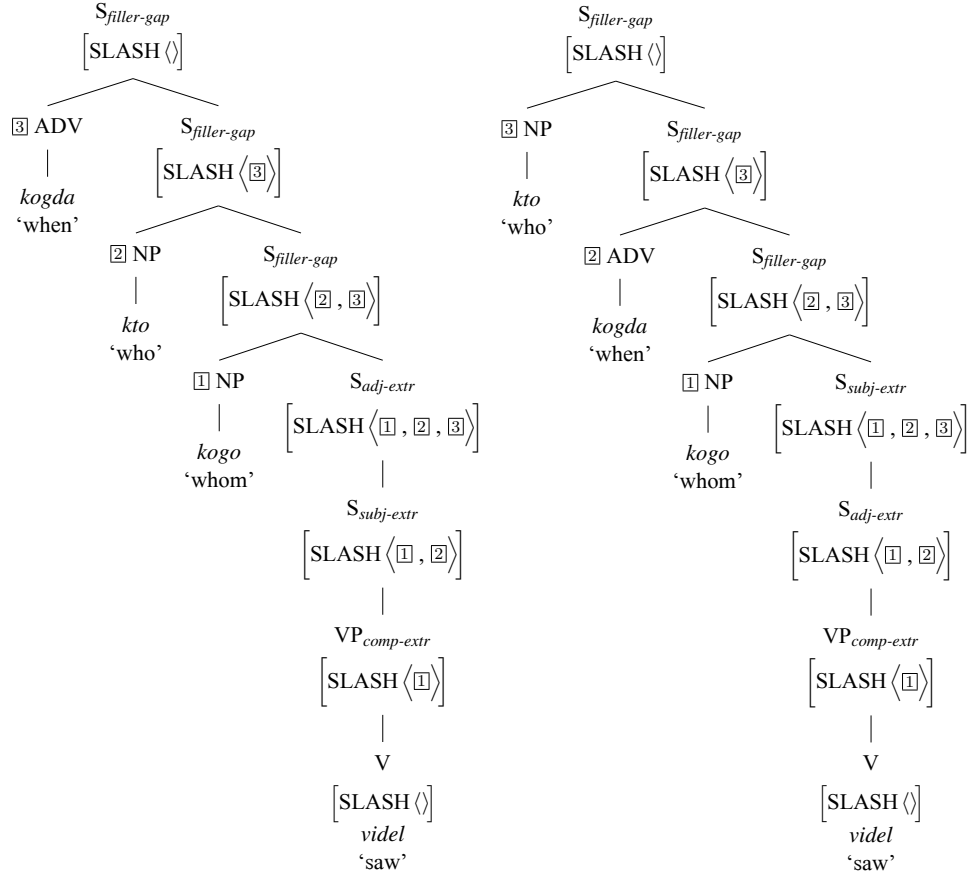(b) The extracted adjunct is between the extracted arguments, as in example (11).

Figure 3: Analysis 2, for extracted adjuncts and arguments in Russian.

based on the order of the application of extraction rules. In particular, compare Fig. 3a to Fig. 3b, for sentences (5) and (11). It is remarkably easy under Analysis 2 to have an extracted adjunct intervene between the two extracted arguments. With lexical threading, we would need to complicate Analysis 1 to license (11) with additional extraction rules to insert an element into the middle of the list.

Under Analysis 2, we can no longer reuse the Grammar Matrix core as it was originally designed (but we expect the revisions to facilitate future Matrix development). We replace lexical supertypes like (23) by (29):

$$(29) \quad \begin{bmatrix} \textit{non-local-none-lex-item} \\ \\ \text{SYNSEM|NON-LOCAL} \begin{bmatrix} \text{SLASH|LIST} & \langle\rangle \\ \text{REL|LIST} & \langle\rangle \\ \text{QUE|LIST} & \langle\rangle \end{bmatrix} \end{bmatrix}$$

The supertype (29) states that all NON-LOCAL features are empty and is used for most lexical entries. The phrasal types such as head-subject and head-complement no longer rely on lexical threading and need to explicitly append the NON-LOCAL features of the daughters. We posit the supertypes (30)–(31) and have most phrasal rules inherit from one of them. Of course, extraction rules and the filler-gap rule do not inherit from these types; instead they either append an item to the existing SLASH list (28), (32)–(33) or subtract an item from it (24).

(30)
$$
\begin{bmatrix}
\textit{binary-non-local-phrase} \\[4pt]
\text{ARGS} \quad \left\langle
\begin{bmatrix} \text{SS|NLOC} \begin{bmatrix} \text{SLASH} & \boxed{1} \\ \text{REL} & \boxed{2} \\ \text{QUE} & \boxed{3} \end{bmatrix} \end{bmatrix},
\begin{bmatrix} \text{SS|NLOC} \begin{bmatrix} \text{SLASH} & \boxed{4} \\ \text{REL} & \boxed{5} \\ \text{QUE} & \boxed{6} \end{bmatrix} \end{bmatrix}
\right\rangle \\[20pt]
\text{SS|NLOC} \begin{bmatrix} \text{SLASH|APPEND} & \langle \boxed{1},\boxed{4} \rangle \\ \text{REL|APPEND} & \langle \boxed{2},\boxed{5} \rangle \\ \text{QUE|APPEND} & \langle \boxed{3},\boxed{6} \rangle \end{bmatrix}
\end{bmatrix}
$$

(31)
$$
\begin{bmatrix}
\textit{unary-non-local-phrase} \\[4pt]
\text{DTR|SYNSEM|NON-LOCAL} \begin{bmatrix} \text{SLASH} & \boxed{1} \\ \text{REL} & \boxed{2} \\ \text{QUE} & \boxed{3} \end{bmatrix} \\[16pt]
\text{SYNSEM|NON-LOCAL} \begin{bmatrix} \text{SLASH} & \boxed{1} \\ \text{REL} & \boxed{2} \\ \text{QUE} & \boxed{3} \end{bmatrix}
\end{bmatrix}
$$

(32)
$$
\begin{bmatrix}
\textit{extracted-subj-phrase} \\[4pt]
\text{SYNSEM} \begin{bmatrix} \text{LOCAL|CAT|VAL|SUBJ} & \langle\,\rangle \\ \text{NON-LOCAL|SLASH|APPEND} & \langle \boxed{0}, [\text{LIST}\,\langle\boxed{1}\rangle] \rangle \end{bmatrix} \\[20pt]
\text{HEAD-DTR|SYNSEM} \begin{bmatrix} \text{LOCAL|CAT|VAL|SUBJ} & \left\langle \begin{bmatrix} \textit{gap} \\ \text{LOCAL} & \boxed{1} \end{bmatrix} \right\rangle \\ \text{NON-LOCAL|SLASH} & \boxed{0} \end{bmatrix}
\end{bmatrix}
$$

(33)
$$
\begin{bmatrix}
\textit{extracted-comp-phrase} \\[4pt]
\text{SYNSEM} \begin{bmatrix} \text{LOCAL|CAT|VAL|COMPS} & \boxed{0} \\ \text{NON-LOCAL|SLASH|APPEND} & \langle \boxed{1}, [\text{LIST}\,\langle\boxed{2}\rangle] \rangle \end{bmatrix} \\[20pt]
\text{HEAD-DTR|SYNSEM} \begin{bmatrix} \text{LOCAL|CAT|VAL|COMPS} & \begin{bmatrix} \text{FIRST} & \begin{bmatrix} \textit{gap} \\ \text{LOCAL} & \boxed{2} \end{bmatrix} \\ \text{REST} & \boxed{0} \end{bmatrix} \\ \text{NON-LOCAL|SLASH} & \boxed{1} \end{bmatrix}
\end{bmatrix}
$$

6 Testing

Analysis 2 is integrated in the Grammar Matrix and as such is in principle tested by all of the regression tests that are currently there.[16] This means in particular that any new types (28) or changes to any old types (24) must not result in any undesirable changes with respect to all the previous analyses, and the system still produces correctly behaving grammars for all previously analysed languages. There are currently 499 tests in the Grammar Matrix (including the Russian test discussed here).[17] Some of them rely on analyses actively involving NON-LOCAL features, particularly the 44 information structure typology tests added by Song (2014). We ensure that integrating our analysis of question fronting into the system does not negatively affect any of the existing analyses; all of the tests pass. Other tests do not always target NON-LOCAL features, in which case they "only" test that the new analyses presented here do not interfere with other analyses in unexpected ways — a crucial methodological point, in our view.

The constituent questions Matrix library (Zamaraeva, forth) adds 26 test suites, 5 of them for natural languages.[18] The results for three of them are shown in Table 1.[19] Russian has multiple fronting while English has strictly single fronting, (so for English, the length of SLASH in the filler-gap rule is restricted in the customization stage). Japanese is an *in situ* language and in that case we test that our extraction and filler-gap rules do not conflict with the *in situ* analysis.

Language	Family	Gram./ungram.	cov%	overgen%	avg. ambig	*wh*-strategy
Russian	Indo-European	186/87	78.5	6.9	1.76	Multiple fronting
English	Indo-European	27/23	100	0	1.11	Single fronting
Japanese	Japonic	7/3	100	0	1.14	*In situ*

Table 1: Results for languages the analyses for which rely on SLASH

The Russian test suite includes not only various patterns of constituent and polar questions, including embedded questions and long distance dependencies, but also simple and complex propositions. The lack of coverage is primarily due

---

[16]Analysis 1 was also tested and deemed less preferable for the Grammar Matrix, though it could be preferable for e.g. a separate grammar of English or a language relying on complex morphology for interrogative marking, such as Yukaghir (Maslova, 2003, p.152). Under Analysis 1, the results are the same for English and Japanese as in Table 1, but for Russian the coverage is smaller (78.0%) because of the VP coordination issue mentioned in §5 and because sentences like (11) are not covered; the average number of analyses per sentence is larger (2.03) due to the multiple adjunct extraction rules which may apply spuriously, and the overgeneration is the same.

[17]The code with the complete analysis, all the test suites, and the testing software can be downloaded from: https://github.com/delph-in/matrix/releases/tag/HPSG2020-Zamaraeva-Emerson.

[18]More tests for natural languages will be added in the final evaluation stage of Zamaraeva forth.

[19]Table legend. *Gram./ungram.:* The number of grammatical and ungrammatical items in the test suite; *cov%:* The percentage of correctly parsed grammatical sentences; *overgen%:* The percentage of admitted ungrammatical sentences; *avg. ambig.:* Average number of trees per sentence (ambiguity can be both meaningful and spurious).

to interacting phenomena discussed in detail in Zamaraeva forth. The fact that we analysed question fronting as truly optional, allowing *wh-* words in declarative rules such as adjunct-head to accommodate *wh*-words in positions like in (9), accounts for most of the spurious ambiguity and for some of the overgeneration, including sentences like (7). This points us in the direction of reanalysing optional fronting in terms of information structure in the future, as discussed briefly below.

7   Future work

Easy modeling of flexible order in multiple extraction allows us to extend the existing analyses of information structure (Song, 2014) to model optional question fronting (9). Instead of allowing QUE-nonempty elements (*wh*-words) in declarative rules such as subject-head or adjunct-head, we can extract non-*wh*-arguments and then utilize topicalization-type filler-gap rules in the same derivation with the *wh*-question phrase.[20] This way background information (the personal pronoun in (9)) or contrastive topic or focus may appear in the front of the *wh-word* but without spurious derivations which arise from allowing QUE-nonempty elements in declarative rules and require additional features to avoid them. Such an analysis will require multiple additional filler-gap constructions, head initial and head final, to account for the plethora of possible information structures but we do not expect it to complicate the *wh*-fronting analysis which we presented here.

8   Conclusion

We showed that multiple extraction and fronting can be straightforwardly implemented in the DELPH-IN version of HPSG, including with flexible order, allowing us to account for Russian *wh*-questions. Unification remains the only natively defined operation under this analysis, and the new type *append-list* allows for easier grammar writing (compared to the previous practice of using difference lists). We test the analysis cross-linguistically using the Grammar Matrix framework and conclude that while it is possible to implement multiple extraction and fronting under the assumption that lexical entries amalgamate their arguments' non-local features (lexical threading), for the purposes of the multilingual Grammar Matrix project, we prefer an analysis which rejects lexical threading in favor of more flexibility in constructing SLASH lists. Information structure in clauses which appear to exhibit optional question fronting is one area where future work could focus.

---

[20]In the Minimalist tradition, it has long been suggested that examples like (9) be analysed as multiple movement (Bailyn, 2005).

Acknowledgments

References

Aguila-Multner, Gabriel & Berthold Crysmann. 2018. Feature resolution by lists: The case of French coordination. In Annie Foret, Greg Kobele & Sylvain Pogodalla (eds.), *Formal Grammar. 23rd International Conference* Lecture Notes in Computer Science, 1–15. Heidelberg: Springer.

Aït-Kaci, Hassan. 1984. *A lattice-theoretic approach to computation based on a calculus of partially-ordered type structures (property inheritance, semantic nets, graph unification)*: University of Pennsylvania dissertation.

Avgustinova, Tania & Yi Zhang. 2009. Exploiting the Russian National Corpus in the development of a Russian Resource Grammar. In *Proceedings of the workshop on adaptation of language resources and technology to new domains*, 1–11. Association for Computational Linguistics.

Bailyn, John Frederick. 2005. Against the scrambling anti-movement movement. In *Formal approaches to Slavic linguistics*, Citeseer.

Bender, Emily M., Scott Drellishak, Antske Fokkens, Laurie Poulson & Safiyyah Saleem. 2010. Grammar customization. *Research on Language & Computation* 8. 1–50.

Bender, Emily M. & Guy Emerson. 2020. Computational linguistics and grammar engineering. In Stefan Müller, Anne Abeillé, Robert D. Borsley & Jean-Pierre Koenig (eds.), *Head-Driven Phrase Structure Grammar: The handbook*, https://hpsg.hu-berlin.de/Projects/HPSG-handbook/PDFs/cl.pdf.

Bender, Emily M. Dan Flickinger & Stephan Oepen. 2002. The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk & Richard Sutcliffe (eds.), *Proceedings of the Workshop on grammar engineering and evaluation at the 19th International Conference on Computational Linguistics*, 8–14. Taipei, Taiwan.

Bender, Emily M. Dan Flickinger & Stephan Oepen. 2008. Grammar engineering for linguistic hypothesis testing. In *Proceedings of the texas linguistics society x conference: Computational linguistics for less-studied languages*, 16–36.

Bender, Emily M, Laurie Poulson, Scott Drellishak & Chris Evans. 2007. Validation and regression testing for a cross-linguistic grammar resource. In *Acl 2007 workshop on deep linguistic processing*, 136–143.

Bierwisch, Manfred. 1963. *Grammatik des deutschen verbs*. Akademie Verlag.

Bouma, Gosse, Robert Malouf & Ivan A. Sag. 2001. Satisfying constraints on extraction and adjunction. *Natural Language & Linguistic Theory* 19(1). 1–65.

Chaves, Rui P & Denis Paperno. 2007. On the Russian hybrid coordination construction. In *The proceedings of the 14th international conference on head-driven phrase structure grammar*, 46–64.

Chomsky, N. 1995. The Minimalist program. *Current studies in linguistics* 28.

Copestake, Ann. 2002a. Definitions of typed feature structures. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii & Hans Uszkoreit (eds.), *Collaborative language engineering*, 227–230. Stanford, CA: CSLI Publications.

Copestake, Ann. 2002b. *Implementing typed feature structure grammars*, vol. 110. CSLI publications Stanford.

Copestake, Ann, Dan Flickinger, Carl Pollard & Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on language and computation* 3(2-3). 281–332.

Crysmann, Berthold. 2015. Resumption and extraction in an implemented HPSG grammar of Hausa. In *Proceedings of the grammar engineering across frameworks (geaf) 2015 workshop*, 65–72.

Crysmann, Berthold & Woodley Packard. 2012. Towards efficient HPSG generation for German, a non-configurational language. In *Proceedings of the 24th International Conference on Computational Linguistics*, 695–710.

Emerson, Guy. 2017. (Diff)list appends in TDL. Presented at the 13th DELPH-IN summit, Oslo, Norway. http://www.delph-in.net/2017/append.pdf.

Emerson, Guy. 2019. Wrapper types: Relational constraints without relational constraints. Presented at the 15th DELPH-IN summit, Cambridge, UK. http://users.sussex.ac.uk/~johnca/summit-2019/wrapper-types.pdf.

Engelkamp, Judith, Gregor Erbach & Hans Uszkoreit. 1992. Handling linear precedence constraints by unification. In *Proceedings of the 30th annual meeting on association for computational linguistics*, 201–208. ACL.

Flickinger, Dan. 2000. On building a more effcient grammar by exploiting types. *Natural Language Engineering* 6(01). 15–28.

Fokkens, Antske Sibelle. 2014. *Enhancing empirical research for linguistically motivated precision grammars*: Department of Computational Linguistics, Universität des Saarlandes dissertation.

Fokkens, Antske Sibelle & Tania Avgustinova. 2013. SlaviCLIMB: Combining expertise for Slavic grammar development using a metagrammar. In *Workshop on high-level methodologies for grammar engineering*, 87–92.

Galikhin, Sergei. 2017. *Povarennaya kniga Mardnaila*. Moscow, Russia: LitRes.

Gazdar, Gerald. 1981. Unbounded dependencies and coordinate structure. In *The formal complexity of natural language*, 183–226. Springer.

Geske, Ulrich & Hans-Joachim Goltz. 2007. A guide for manual construction of difference-list procedures. In *Applications of declarative programming and knowledge management*, 1–20. Springer.

Ginzburg, Jonathan & Ivan A. Sag. 2000. *Interrogative investigations*. Stanford: CSLI publications.

Götz, Thilo & Walt Detmar Meurers. 1996. The importance of being lazy – Using lazy evaluation to process queries to HPSG grammars. In *Actes de la conférence*

traitement automatique de la langue naturelle (taln), .

Idiatov, Dmitry. 2007. *A typology of non-selective interrogative pronominals*: University of Antwerp dissertation.

Kathol, Andreas. 1995. *Linearization-based German syntax*: The Ohio State University dissertation.

Maslova, Elena. 2003. *A grammar of Kolyma Yukaghir*, vol. 27. Walter de Gruyter.

Moshier, M. Andrew & Carl J Pollard. 1994. The domain of set-valued feature structures. *Linguistics and Philosophy* 17(6). 607–631.

Müller, Stefan. 1999. Deutsche Syntax deklarativ. *Head-Driven Phrase Structure Grammar für das Deutsche* 394.

Müller, Stefan. 2015. The CoreGram project: Theoretical linguistics, theory development and verification. *Journal of Language Modelling* 3(1). 21–86.

Oborin, L. 1987. Timur i ego komanda, kvartiry Dostoyevskogo i antologiya polskoy poezii. Found via Russian National Corpus. https://gorky.media/context/timur-i-ego-komanda-kvartiry-dostoevskogo-i-antologiya-polskoj-detskoj-poezii/.

Osenova, Petya. 2010. Bulgarian Resource Grammar–Efficient and Realistic. Tech. rep. Stanford University, CSLI.

Penn, Gerald. 1998. Linearization and *wh*-extraction in HPSG: Evidence from Serbo-Croatian. In *In: Slavic in HPSG*, Citeseer.

Pollard, Carl & Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Pollard, Carl J. & M. Drew Moshier. 1990. Unifying partial descriptions of sets. In Philip P. Hanson (ed.), *Information, language, and cognition* (Vancouver Studies in Cognitive Science 1), 285–322. University of British Columbia Press.

Przepiórkowski, Adam. 1998. On complements and adjuncts in Polish. In *Slavic in HPSG*, .

Reape, Mike. 1994. *A formal theory of word order: A case study in West Germanic*: University of Edinburgh dissertation.

Sag, Ivan A., Thomas Wasow, Emily M. Bender & Ivan A. Sag. 2003. *Syntactic theory: A formal introduction*. CSLI.

Song, Sanghoun. 2014. *A grammar library for information structure*: University of Washington dissertation.

Stepanov, Arthur & Penka Stateva. 2006. Successive cyclicity as residual *wh*-scope marking. *Lingua* 116(12). 2107–2153.

Zamaraeva, Olga. forth. *A cross-linguistic analysis of constituent questions for the Grammar Matrix*: University of Washington dissertation.