Some Fine Points of Hybrid Natural Language Parsing

Peter Adolphs[♣], Stephan Oepen[♠◇], Ulrich Callmeier[♡], Berthold Crysmann[△], Dan Flickinger[◇♣], Bernd Kiefer[♣]

[♣]Deutsches Forschungszentrum für Künstliche Intelligenz (Germany),
[♠]Universitetet i Oslo, Department of Informatics (Norway)
[◇]Stanford University, Center for the Study of Language and Information, (USA)
[♡]acrolinx GmbH (Germany)
[△]Universität Bonn (Germany)
peter.adolphs@dfki.de, oe@ifi.uio.no, uc@acrolinx.com,
crysmann@ifk.uni-bonn.de, danf@csli.stanford.edu, kiefer@dfki.de

Abstract

Large-scale grammar-based parsing systems nowadays increasingly rely on independently developed, more specialized components for pre-processing their input. However, different tools make conflicting assumptions about very basic properties such as tokenization. To make linguistic annotation gathered in pre-processing available to 'deep' parsing, a hybrid NLP system needs to establish a coherent mapping between the two universes. Our basic assumption is that tokens are best described by attribute–value matrices (AVMs) that may be arbitrarily complex. We propose a powerful resource-sensitive rewrite formalism, 'chart mapping', that allows us to mediate between the token descriptions delivered by shallow pre-processing components and the input expected by the grammar. We furthermore propose a novel way of unknown word treatment where all generic lexical entries are instantiated that are licensed by a particular token AVM. Again, chart mapping is used to give the grammar writer full control as to which items (e.g. native vs. generic lexical items) enter syntactic parsing. We discuss several further uses of the original idea and report on early experiences with the new machinery.

1. Background—Motivation

Grammar-based parsing in frameworks like CCG (Clark & Curran, 2004), LFG (Riezler et al., 2002), and HPSG (Malouf & van Noord, 2004; Oepen, Flickinger, Toutanova, & Manning, 2004; Miyao, Ninomiya, & Tsujii, 2005) has matured to a point that allows 'deep' linguistic analysis of large collections of running text. At the same time, such systems increasingly rely on 'shallow' pre-processing of input—for example to complement lexical gaps based on part-of-speech (PoS) taggers and named entity (NE) recognizers (Crysmann et al., 2002, inter alios), or to reduce lexical and structural ambiguity by filtering against analyses suggested by a tagger or statistical parser (Prins & van Noord, 2001, Frank, Becker, Crysmann, Kiefer, & Schäfer, 2003, Dalrymple, 2006, inter alios). In a nutshell, the main benefits of shallow NLP tools lie in their broad coverage, robustness, and portability across domains; the main attraction in deep, grammar-based parsing, on the other hand, is the increased precision provided by the inclusion of fine-grained linguistic distinctions and semantics in such systems.

In our own work on building hybrid parsing systems from independently developed components (Callmeier, Eisele, Schäfer, & Siegel, 2004; Waldron, Copestake, Schäfer, & Kiefer, 2006),[1] we have repeatedly stumbled over a challenge that *prima facie* may seem trivial: different tools make conflicting assumptions about very basic properties, even at the level of tokenization, i.e. the breaking up of input text into basic building blocks for subsequent analysis. Although differences in the assumptions made by tokenizers have been acknowledged and discussed before (Grefenstette & Tapanainen, 1994; Habert et al., 1998), to our knowledge no satisfactory solution for the problem of integrating existing, independently developed shallow pre-processing components with diverging linguistic assumptions into deep parsing has been found so far.

Shallow components for English tend to be influenced heavily by the linguistic decisions made in the Penn Treebank (PTB; Marcus, Santorini, & Marcinkiewicz, 1993). The PTB treats most punctuation marks as separate tokens and breaks up contracted verb forms, e.g. the string *Don't you!* is tokenized as the four-element sequence $\langle do, n't, you, ! \rangle$. Linguistically, however, the implied analogy to a non-contracted form is, put mildly, mis-leading (seeing that **Do not you!* is ungrammatical), and it leads to the stipulation of pseudo-lexemes and false paradigms, as for example in breaking up *won't* as $\langle wo, n't \rangle$. Following Flickinger (2000), inter alios, and his broad-coverage English Resource Grammar (ERG; couched in the HPSG framework), at least some deep computational grammars reject the PTB tokenization approach. Besides disagreement about the limited class of contracted negations, these grammar writers have found that punctuation is best approached in an analysis akin to affixation, i.e. commas, parentheses, quote marks, et al. are attached as 'prefixes' or 'suffixes' (in a technical sense) directly to the word forms to which they are juxtaposed in standard orthography. While we cannot motivate this point of view at any level of detail here, observe that grammatical constructions like appositions or non-restrictive relative clauses require parallel bracketing with commas on either side. However, where multiple such constructions might in theory call for multiple punctuation marks—following a final apposition in a relative clause, say—conceptual punctuation clusters are nevertheless realized as a single comma only; further-

[1] See 'http://www.delph-in.net/' for background on existing shallow and deep NLP tools and linguistic resources (for many languages) in the HPSG framework.

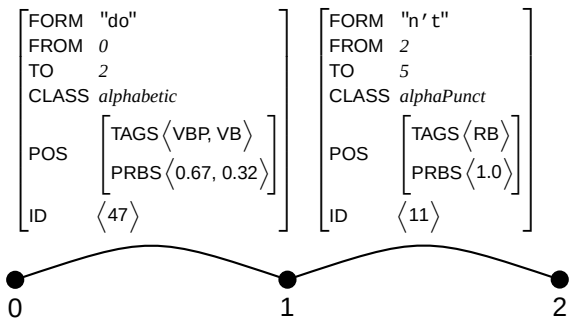

Figure 1: Sample input token chart for the string *don't*.

more a non-comma sentence-final punctuation mark can satisfy all construction-specific 'pending' punctuation requirements. (For discussion of the phenomenon, see (Nunberg, 1990) and (Briscoe, 1996).) This phenomenon is straightforwardly accounted for in an affixation approach to punctuation, using a hierarchy of punctuation marks and their candidate functions.[2]

Where a grammar-based parser is to benefit from shallow pre-processing, discrepancies in basic tokenization are problematic. Lacking a notion of affixation, typical taggers stand to gain from viewing punctuation marks as separate tokens, while at the same time a deep parser depends on its input being tokenized according to the assumptions made in its grammar. To make linguistic annotation gathered in preprocessing available to deep parsing, a hybrid NLP system needs to establish a coherent mapping between the two universes. Finally, existing grammars like the ERG often include their own 'lightweight' NE module, building on regular expressions to match various forms of numbers, email and web addresses, and the like. These devices tend to be carefully synchronized to subtle grammatical distinctions: the ordinal in *We'll meet the 3rd*, for example, is ambiguous between a temporal adverbial, a day of the month, and an elliptical NP object; *We'll meet the 42nd*, on the other hand, only admits the latter reading. Therefore, it is vital to accurately preserve such functionality when embedding a deep grammar in a hybrid parsing system.

2. A Candidate Solution

Assume an existing pre-processing pipeline, including sentence segmentation, PoS tagging, and (some) NE recognition. In the following, we will restrict ourselves to the downstream analysis of one utterance at a time. Upon completion of pre-processing, the original utterance string has been tokenized, and each token annotated with candidate PoS information. Additionally, individual tokens or multi-token sequences may be flagged as candidate named entities (of varying type and internal structure). Obviously, at this point already, there can be ambiguity—say in conflicting PoS tags for a word form like *sleeps*—and unless pre-processing were able to rule out competing hypotheses with great confidence, it can be beneficial to pass at least part of this ambiguity into the deep parser, leaving it to the grammar and ultimately statistical parse selection to further disambiguate. Thus, we assume that the input interface to the deep parser is a lattice of structured objects, i.e. a chart of attribute–value matrices (AVMs), where chart vertices correspond to shallow token boundaries. Figure 1 presents a small sample lattice, with some selected properties shown on each token. Different pre-processing pipelines may have proprietary token properties and naming schemes, but we assume that each token provides its raw surface form, stand-off pointers into the original input string, and a unique identifier.[3]

To mediate between original shallow tokens and a tokenization compatible with the deep grammar, and also to select which pieces of annotation to pass into the parser (or maybe rename and repackage values as needed), we propose a powerful mapping formalism. Token rewrite rules take the general form:

$$[\,\text{CONTEXT}:\,]\,\text{INPUT} \rightarrow \text{OUTPUT}$$

The CONTEXT, INPUT, and OUTPUT components are each (possibly empty) sequences of AVMs. Much like in chart parsing, in the basic case, a rule fires when all INPUT elements are successfully unified to a contiguous sequence of existing chart edges; in this case, OUTPUT is copied into the chart, at the span ranging from the start vertex of the first INPUT element to the end vertex of its last element.[4]

Unlike in chart parsing, however, the rewrite process is resource-sensitive, in the sense that each rule application consumes all chart edges that were used to 'prove' its INPUT component. When (re-)combining the token sequence *do* + *n't* into a single token, for example, the effect is both the addition of a new chart entry (a synthesized token *don't*, as required by the grammar) *and* the deletion from the chart of the unwanted shallow token edges. The CONTEXT component, on the other hand, can be used to condition the applicability of rules on successful unification against the current chart—much like INPUT—but without actually consuming edges used in unification against its elements. Thus, CONTEXT rewriting rules can be used to add alternative hypotheses into the chart, and obviously CONTEXT and INPUT can be freely combined.

Having both the left- and right-hand side of rules operate on the same token chart enables 'feeding and bleeding' among rules. To give grammarians full control over the rewrite process, rules are applied strictly sequentially, in the order

[2] Also, not splitting off punctuation marks into separate tokens can avoid spurious ambiguity, for example when separating non-directed quote marks or dashes from the preceding or following token to which they were actually attached.

[3] In order to increase interoperability with existing preprocessing tools, it is desirable to use established stand-off annotation formats like SMAF (Waldron et al., 2006), an XML-based exchange format inspired by MAF (Clément & Villemonte de La Clergerie, 2005), for data exchange and a software architecture such as the *Heart of Gold* (Schäfer, 2006), among many others, for synthesizing the annotation gained by the various tools. We do not address this problem with our approach.

[4] In Sections 4. and 5., we argue that this mode of operation is actually overly restrictive, i.e. only one of the possible variants in 'positioning' constraints among rule elements. We might want to match non-contiguous INPUT as well as to specify other OUTPUT locations. For now, we also do not address the question of how the position of CONTEXT items is specified.

specified by the grammar.[5]

Although operating on a single chart, thus resembling unification-based parsing, the non-monotonicity of rewrite rules consuming edges from the chart provides an elegant means of suppressing unwanted input elements (incompatible with the deep grammar) from downstream processing. In Section 4. below, we will suggest another motivation for non-monotonic rule applications.

3. Some Examples

Conceptually, all three components of rewrite rules are treated as a single feature structure, such that re-entrancies can be effected across components; thus, unifying INPUT elements to existing chart entries will typically further specify the OUTPUT AVMs, and provides a means of 'copying' information from the left-hand side of the rule to its right-hand side. We furthermore assume that standard feature structure unification is augmented with regular expression operations over strings. Consider the following example:[6]

$$\left[\begin{array}{l}\text{FORM } /\hat{}(.+)\$/ \\ \text{TO } \boxed{1}\end{array}\right], \left[\begin{array}{l}\text{FORM "n't"} \\ \text{FROM } \boxed{1}\end{array}\right] \rightarrow \left[\text{FORM } /\backslash 1 n't/\right]$$

This rule reverts the pre-processing 'damage' to contracted negations. The RE group operators '`(`' and '`)`' on the INPUT side establish a binding for the actual FORM value of the first element; on the OUTPUT side, the group match is then inserted into the synthesized FORM string by means of a RE back reference (indicated here by the RE backslash operator '\').

Another example rule demonstrates lightweight NE recognition, controlled by the deep grammar:

$$\left[\text{FORM } /\hat{}([0\text{-}2]?[0\text{-}9]:[0\text{-}5][0\text{-}9])\$/\right] \rightarrow \left[\begin{array}{l}\text{FORM } /\backslash 1/ \\ \text{CLASS } \textit{clockTime}\end{array}\right]$$

This rule recognizes a sequence of one or two digits (the hour), followed by a colon, followed by another two digits (minutes) as a token of class *clockTime*.[7] Note that this rule does not alter the surface FORM value, but rather marks the NE class as the value of the CLASS attribute (or another property of choice, defined by the grammar). Section 4. below demonstrates how lexical lookup and unknown word handling can take advantage of this information. For the purpose of input normalization, subsequent rewrite rules can inspect the full AVM, including the token CLASS, and filter accordingly—for example when detecting 'sandwiched' punctuation marks.

As a final example, we present a rule with more than one element on its right-hand side, a simplified version of a robustness measure in parsing noisily punctuated text. In email, for example, punctuation marks (and specifically colons) are often not properly separated from adjacent tokens. A rewrite rule like the following normalizes non-standard colons:

$$\left[\begin{array}{l}\text{FORM } /\hat{}(.+:)([a\text{-}zA\text{-}Z0\text{-}9].*)\$/ \\ \text{CLASS } \textit{alphaPunct}\end{array}\right] \rightarrow \left[\text{FORM } /\backslash 1/\right], \left[\text{FORM } /\backslash 2/\right]$$

In this case, one INPUT element is split into two, converting the 'sandwiched' colon in this specific configuration (unless part of a previously matched NE that can legitimately contain colons) into a token boundary.[8]

These few examples illustrate the kind of mapping required between the shallow and deep token universes. Embedding rewrite rules with the grammar allows the grammarian to make explicit their assumptions about the input interface to deep parsing. Furthermore, it allows re-use of the description language and engineering tools used with the ERG and similar grammars already, and it provides full access to the grammar-internal type hierarchy during token rewriting. We envision that some token-level rewrite rules will be generally applicable, while others may be tied to a specific pre-processing pipeline. Therefore a configuration mechanism on rules should be provided to the grammarian, so as to define clusters of active rules in various setups.

4. Lexical Instantiation and Selection

Once token processing is complete, the deep parser looks up lexical entries (LEs) from its lexicon according to surface forms. To benefit from shallow pre-processing, existing grammars provide an inventory of underspecified 'generic' LEs, for example a simple noun (mass or count) and an optionally transitive verb. These are typically activated as a fall-back device, i.e. in case the hand-built, 'native' lexicon provides no LE for an input token. For this purpose, existing grammars include mappings from, say, a specific inventory of PoS tags to identifiers of generic LEs. However, a common type of lexical gaps in deep grammars is not the complete absence of any information for a token spelling, but rather *partial* lexical coverage—for example providing a noun LE for *bus* but omitting its verb reading. PoS taggers are likely to tag the verb form correctly in contexts like *We'll bus to Paris.* Where the grammar lacks a native verbal LE, it is desirable to trigger unknown word processing (despite an existing native LE, the noun, which would lead to parse failure); and even where the grammar provided both entries, it can be benefical to block the noun

[5] Abstractly, our formalism is similar in nature to the approach used in (semantic) transfer machine translation, see for example Oepen et al. (2004).

[6] In the example rules, we restrict the information shown to what is immediately relevant. For example, unless indicated otherwise, we assume that all rules determine FROM (the stand-off start pointer), TO (the end pointer), ID, and other values appropriately: where multiple input tokens are synthesized, for example, the OUTPUT would bear the FROM of the first, and TO value of the last left-hand side element. Keeping exact track of original shallow tokens, we further assume that all rules determine the OUTPUT ID value(s) as the union of ID values on all left-hand side elements.

[7] Note that our sample regular expression does not try to circumscribe exactly the valid number ranges for hours, i.e. ruling out numbers greater than 24; nor does it foresee the idiosyncratic English 'am' and 'pm' time modifiers. More elaborate expressions or, equivalently, a family of *clockTime* rules can be constructed to overcome these deficiencies.

[8] With multiple right-hand side elements in rewrite rules, the determination of appropriate surface FROM and TO values presents an interesting challenge, but for space limitations we cannot discuss this aspect. Likewise, such rules create new internal chart vertices, adjacent only to the OUTPUT elements themselves; we assume a generalized chart, a token lattice, where vertex indices are abstract entities instead of just integers.

reading prior to full parsing, eliminating unnecessary lexical ambiguity. Existing unknown word handling—the fallback strategy—lacks the flexibility to combine the two sets of related native and generic LEs, respectively, as needed, i.e. invoking either intersection, union, or set difference.

We propose a novel approach to lexical look-up, instantiation, and selection. For all input tokens contained in the chart once token-level normalization is complete (i.e. rewriting has reached a fix-point), both 'native' and 'generic' lexical retrieval are invoked in parallel. A grammar simply declares its set of available generic entries and, in place of existing idiosyncratic grammar-external mappings, directly encodes constraints on their use (with respect to information associated to input tokens) as part of the AVM of each entry. When instantiating a generic entry e for an input token i, the full AVM of i is unified into e under a pre-defined path, say TOKEN. Assuming PTB PoS tags, a generic noun entry can then be specified as $\left[\text{TOKEN} \mid \text{TAG } \textit{NN}\right]$,[9] and a more specialized LE for clock time named entities as $\left[\text{TOKEN} \mid \text{CLASS } \textit{clockTime}\right]$.

Unification of token information into native and generic LEs serves two purposes: (a) only generic entries compatible with the actual properties of a specific input token will succeed, such that no unwarranted LEs are hypothesized; and (b) all token information is made available to the lexical entry (and hence subsequent processing): AVM re-entrancies allow the grammar to project into the syntax whatever properties are deemed appropriate. Furthermore, grammars are free to deploy the type hierarchy (the core tool for linguistic generalization in HPSG) to reflect hierarchical relations among input constrains—for example where a PoS tagger can provide both designated singular and plural noun tags, as well as an underspecified tag in cases it cannot resolve ('NN1', 'NN2', and 'NN', respectively, in the so-called CLAWS tag set). In this latter example, the plural noun generic LE could be further constrained in terms of surface properties, e.g. $\left[\text{TOKEN} \mid \text{FORM } /\text{s\$}/\right]$.

Finally, to selectively block parallel lexical entries, say where both a native and generic LE of comparable category exist, we extend our rewrite machinery to rules where (a) left-hand side elements can be taken from the same chart cell and (b) the right-hand side can be empty. For example, the following rule, applied to the chart after lexical instantiation but prior to parsing, blocks generic noun LEs whenever they fall into the same cell as a native nominal entry:

$$\begin{bmatrix}\textit{native\_le}\\ \text{SYNSEM } \textit{nominal}\end{bmatrix} \doteq \begin{bmatrix}\textit{generic\_le}\\ \text{SYNSEM } \textit{nominal}\end{bmatrix} \rightarrow$$

Note that this rule utilizes a CONTEXT condition, so as to only consume one of its left-hand side elements, and that we use the symbol '$\doteq$' to indicate that both elements occupy the same chart cell.[10]

[9] At this point, we assume that token normalization has multiplied out the original POS sequence of tags (compare to Figure 1). As part of the same process, low-probability or low-confidence PoS assignments may have been suppressed.

[10] This notational convention for specifying positional constraints is provisionary. In our current implementation, it is possible to state that two items are in the same chart cell, that they are adjacent, or that one item is somewhere to the left or right of another item. Likewise, output items can be placed in the same chart cell instead of inserting them as a consecutive sequence.

5. Chart Dependencies

Another mechanism that could be treated with chart mapping rules is the so-called *chart dependencies* filter, which was first described by Kiefer & Krieger (1998) and Kiefer, Krieger, Carroll, & Malouf (1999). It is based on the observation that some lexical items depend on the presence of others in the chart, and that these dependencies are of a non-local nature.

Particle verbs in German are an example of this kind of dependency. In the sentence *She hielt ihn davon ab.* ('she kept him from doing this.'), *hielt* is the past form of *halten* ('to keep'), but the actual verb here is *abhalten* and the particle *ab* has been split off. The entry of the verbal form without the particle can only contribute to an analysis of the sentence if a corresponding entry for the particle can be found somewhere in the chart. Such verbs are very common and thus, lots of useless entries end up in the chart and slow down parsing massively if they are not filtered out. To continue the example, the verb *halten* has more than thirty corresponding particle verbs, e.g. *anhalten* ('to pause'), *aufhalten* ('to delay'), *zurückhalten* ('to detain'), and so forth.

At the moment, the information about what is required or provided by a lexical item can be found under paths in the AVM that are specified by the grammar. At the end of lexical processing, those paths are looked up for every item and all provided information as well as the items with requirements are stored. Then, each item with requirements that are not satisfied is removed from the chart. A requirement is fulfilled if the information that has been found is compatible (unifiable) with the required information.

This can now be implemented with the machinery described before, using chart mapping rules similar to the lexical selection approach discussed in Section 4. above. As the left-hand side elements of our rewrite rules need not be adjacent, non-local dependencies among chart entries can be encoded in a straightforward manner. A filtering rule for verb – particle lexical entries, for example, could suppress instantiated lexical entries for *abhalten* in case there is no *ab* particle entry elsewhere in the chart. However, note that a rule like this would need to condition on a negative constraint, the *absence* of a compatible particle from the chart. Nevertheless, it would be possible to implement this dependency using our current formalism, by stipulating an additional feature, say assuming that lexical items that depend on other chart entries are marked $\left[\text{COMPLETE } -\right]$. A chart mapping rule could then 'toggle' this feature value if and only if the external dependency can be satisfied (e.g. by the *ab* particle in our running example). Finally, the syntactic component of the grammar could then block remaining 'incomplete' lexical entries from further processing. Albeit feasible in principle, 'toggling' a feature like COMPLETE in chart mapping rules (which are instantiated using unification, i.e. monotonically) would imply copying everything *but* the specific feature from the input to the output side.[11]

[11] A similar challenge often arises in category-changing lexical rules in unification-based grammars, where it is common to arrange the feature geometry of linguistic signs so as to reflect

An alternate mechanism that we currently investigate is extending the rewrite formalism with an additional component, say FILTER (essentially a negated CONTEXT), which would allow negative conditioning of rules. When present, a FILTER component would need to be checked *after* other left-hand side elements, and successful unification of FILTER against the chart would effectively block application of the rule in question.

Another problematic aspect of German particle verbs is that they not only constitute discontinuous lexical items, but rather discontinuous predicates, that is, the argument structure of a particle verb is jointly determined by the verb and the particle, often in a non-compositional way. For example *halten* 'to hold' is a transitive verb taking an accusative NP complement, but *vorhalten* 'to reproach' takes a dative NP and a propositional that-clause complement. By contrast, *fahren* 'to drive' is an intransitive, but so is *vorfahren* 'to drive up'. As illustrated by these contrasts, argument extension is neither directly predictable on the basis of the verb, nor on the basis of the particle. Thus, the argument structure of the discontinous predicate must be specified as a whole on either of its parts. Since particles combine with verbs quite productively, it is prohibitive to specify the argument structure of the complex as a lexical property of the particle, because this would lead to a proliferation of particle entries in the lexicon, and ultimately, the chart. Instead, in the German HPSG grammar GG (Müller & Kasper, 2000, Crysmann, 2003, Crysmann, 2007), the specific argument structure is associated with the verb, and passed down the tree to the clause-final particle. As a result, the argument structure is locally underspecified during bottom-up parsing, which constitutes another source of inefficiency. On the basis of our new chart mapping technology, however, this second issue with particle verbs can easily be resolved by means of unifying the verb's argument structure onto the lexical entry of the particle. As a result, the particle's argument structure will now be fully specified before the actual parsing phase, thereby greatly reducing the search space in parsing proper. Although we have not yet evaluated the potential efficiency benefits, we expect them to be considerable, given our observations that discontinuous predicates in German are one of the main culprits for suboptimal runtime performance.

6. Current State of Play

We have started implementing this approach on top of the DELPH-IN tool chain, as part of the parser in the PET system (Callmeier, 2000). While originally designed as an experimentation platform for developing and comparing techniques in unification based grammar processing, within the context of the DELPH-IN collaboration the PET parser has taken the role of the high-efficiency engine for batch processing and deployment of DELPH-IN grammars. The PET parser is a bottom-up chart parser with support for ambiguity packing and parse ranking. In the context of different applications, over time a whole range of lattice-based input methods have been implemented for the PET parser, each with its individual advantages and limitations. The implementation of the approach we describe here is intended to generalize and consolidate the existing input methods at some point.

Whereas the underlying assumption in traditional chart parsing (Kay, 1986) is that the input is a linearly ordered sequence of tokens, chart-mapping rules may output elements that are not aligned with the original edges in the input chart. Therefore we use a generalized chart, that is a directed acyclic graph whose vertices are abstract objects rather than indexed token boundary positions (commonly represented as consecutive integers, numbering inter-token positions).

Chart-mapping rules are applied in two phases during parsing: (a) the *token mapping* and (b) the *lexical filtering* phase. Token mapping takes place directly after reading in the (usually already tokenized) input. This phase is used to adjust and augment the input so that it fits the assumptions made by the grammar, including lightweight NE recognition as sketched above. When token mapping is finished, lexical entries are instantiated for each token in the chart. To this end, the surface form of each token is analyzed by the (integrated) morphology component—resulting in pairs, each comprising a candidate lexical stems and a chain of hypothesized orthographemic rules, relating the actual form to the stem. Each stem is then looked up in the lexicon, and its corresponding lexical entries, if any, are copied into the chart as new lexical items. Independently, all compatible generic lexical entries defined in the grammar are also instantiated for each token. The parser then enters a lexical parsing phase, where the lexical items in the chart are turned into items that are suited for syntactic parsing. At this point, lexical items might turn out to be incompatible with orthographemic rules postulated earlier during morphological analysis. In order to filter out unwanted lexical items, e.g. generic items where native items are available in the same chart cell and have survived lexical parsing (see Section 4. above), lexical chart mapping is performed. Finally, the parser enters the actual syntactic parsing phase.

The core functionality of the proposed chart-mapping machinery has by now been implemented, and first practical experiences are already gained (see below). One of the conclusions so far is that the means to specify the positional constraints on the matched arguments and the output items of a rule are too restrictive. We are therefore looking a more general constraint language on precedence relations between chart items that also allows us to take advantage of type inheritance for a better factorization of rule types.

Prior to development of this token chart approach, the ERG employed a finite-state preprocessor for which the grammar defined a collection of some 228 token-manipulation rules to normalize text. These rules fall broadly into two sets: one for regularizing punctuation and spacing, and the other for dealing with numeric and alphanumeric entities such as integers, decimals, fractions, ratios, dates, times, ranges, phone numbers, measure phrases, temperatures, addresses, and product name identifiers. To make use of the new token chart approach, those 228 substitution, deletion, and insertion rules were manually converted to 225 roughly equivalent rules expressed in terms of input, output, and context

generalizations over common clusters of information that remain constant, constrasting with the linguistic properties commonly affected by lexical alternation.

token lists. The two sets of rules were checked for equivalent results by employing each of the two preprocessing engines on several standard treebanked test suites (some 3000 sentences in total), and verifying that the resulting syntactic analyses were identical to the analysis recorded in the treebank. There were a handful of uninteresting differences due to slight differences in how the two engines treat a few special characters which are also operators within regular expressions, but the token chart engine was otherwise equivalent in what it presented to the parser on these data sets where all of the vocabulary was within the scope of the manually constructed lexicon.

In addition, a small set of additional token chart rules were defined to constrain the interaction between a part-of-speech tagger and the unknown-word handling mechanism used with the ERG in parsing open-domain text. These rules favor manually-defined lexical entries over those proposed on the basis of PoS tags, and also filter out some unwanted generic entries in cases where the tagger assigns multiple likely tags. An example of the latter involves unknown prenominal modifiers, which the tagger will often label with tags for both noun and adjective, leading to unwanted syntactic ambiguity if a generic lexical entry is introduced for each of the two. The token chart rule here filters out one of the two, reducing both parsing cost and ambiguity of the resulting parse forest.

Figure 2 presents preliminary results for comparing end-to-end parsing performance using the ERG, contrasting the original system configuration with our augmented setup. Although conversion of existing (external) preprocessing rules and fine-tuning of the lexical instantiation and chart dependencies is not yet complete, already we see an improvement in both the number of sentences that succeed in parsing and the average processing time per input. These figures reflect a sample of some 15,000 sentences drawn from technical manuals (of diverse products and manufacturers), a domain for which the ERG has only been adapted recently. Therefore, and due to the substantial diversity of these texts, the proportion of unknown words and named entities is comparatively high in this corpus.

For the German Grammar (GG), we also implemented an unknown-word handling mechanism on the basis of PoS tags, reproducing the behaviour which was previously hard encoded in PET. The prior behaviour was not always satisfactory, though. Due to the syncretism inherent in German noun inflection, for instance, the set of possible morpho-syntactic properties for unknown nouns cannot be accurately determined. Hence the morpho-syntactic properties for unknown nouns had previously been left underspecified, accepting the potential additional ambiguity during syntactic parsing. Most unknown nouns in German texts, however, are built along productive word formation patterns. With the help of token chart rules, we could further refine the instantiated generic lexical entries for unknown nouns by mapping the morpho-syntactic properties returned by an external computational morphology (Petitpierre & Russell, 1994) to the corresponding agreement feature in the lexical item. The way in which these morphological mappings are defined illustrates another interesting application of in-cell mapping rules: in the GG, morphological readings are compactly represented as types, representing ambiguity classes by means of underspecification, for reasons of parsing efficiency. External morphologies often do not follow the same representation format, but instead list readings in disjunctive normal form. Our mapping rules therefore map clusters of disjunctive readings within the same cell to a single compact type-based representation (see the discussion above on the blocking of generic lexical entries by native ones). Another feature of the present chart mapping formalism that plays a crucial role in the definition of these mappings is the order-sensitivity of rule application: German nominal morphology recognises ten regular paradigms, involving different patterns of syncretism. Some of the readings stand in a subset relation, e.g. zero marking in Class 2 (*Computer* 'computer', Nom/Dat/Acc.Sg + Nom/Gen/Acc.Pl) is a proper superset of the readings for zero marking in Class 1 (*Tag* 'day', Nom/Dat/Acc.Sg) By ordering mapping rules for supersets before mapping rules for subsets, we elegantly map the sets of readings onto a unique type identifier, as defined in the grammar.

Because token chart rules are defined as typed feature structures within the grammar's own formalism, the grammar writer can use a hierarchy of rule types to capture shared properties among subsets of rules. These typed structures also proved to provide a significant benefit by enabling more focused error-checking during development of the rule set.

7. Discussion—Outlook

Albeit not rocket science, the issues we discuss present genuine road blocks in large-scale hybrid parsing today: the integration of shallow and deep NLP tools in existing hybrid pipelines to date cannot accomodate token-level divergencies and a flexible mapping between the two universes (Callmeier et al., 2004). We propose a comparatively powerful device, unification-based rewriting of AVM charts, to address this problem, fully integrated with the deep parsing grammar and its linguistic constraints. First practical experiences show that current pre-processing tools which were designed for a more narrowed set of problems can easily be replaced with the new machinery, while sustaining the performance of the parsing system, and allowing the grammar writers to address a greater range of pre-processing tasks with the same formalism as for the grammar.

Acknowledgements

This work reflects the collective experience of many over an extended period of R&D and joint application building with industrial partners. We are grateful for numerous in-depth discussions with our colleagues in the DELPHIN community (and beyond). We would like to gratefully acknowledge the following collaborators: Nuria Bertomeu, Ann Copestake, Remy Sanouillet, Ulrich Schäfer, and Benjamin Waldron.

Part of the work reflected here was funded by the ProFIT program of the German federal state of Berlin and the EFRE program of the EU (to the DFKI project Checkpoint), and by the University of Oslo (through its scientific partnership with CSLI).

Aggregate	old system		new system		delta
	coverage	time	coverage	time	time
	%	ϕ (s)	%	ϕ (s)	%
$20 <$ *i-length* ≤ 85	71.4	4.76	67.3	2.86	40.1
$10 <$ *i-length* ≤ 20	81.2	0.51	85.6	0.50	2.3
$5 <$ *i-length* ≤ 10	83.4	0.9	87.1	0.10	-17.4
$0 <$ *i-length* ≤ 5	84.7	0.02	87.8	0.04	-74.3
Total	**81.0**	**0.96**	**83.5**	**0.67**	**29.8**

(generated by [incr tsdb()] at 31-mar-2008 (10:18 h))

Figure 2: Preliminary contrastive evalution using the ERG. The improved control over the creation of generic lexical entries (based on pre-processing using a PoS tagger) yields a substantial efficiency gain, albeit (for the time being) at some loss of parses for excessively long inputs. At the same time, the relative speed-up over the original system increases in sentence length, i.e. the effect is largest where it matters most.

References

Briscoe, T. (1996). The syntax and semantics of punctuation. In *Proceedings of acl workshop on punctuation in cl*. Santa Cruz, CA.

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 99 – 108.

Callmeier, U., Eisele, A., Schäfer, U., & Siegel, M. (2004). The DeepThought core architecture framework. In *Proceedings of the 4th International Conference on Language Resources and Evaluation* (pp. 1205 – 1208). Lisbon, Portugal.

Clark, S., & Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics* (pp. 104 – 111). Barcelona, Spain.

Clément, L., & Villemonte de La Clergerie, E. (2005). MAF: a morphosyntactic annotation framework. In *Proc. of the 2nd language & technology conference (lt'05)* (pp. 90–94). Poznan, Poland.

Crysmann, B. (2003). On the efficient implementation of German verb placement in HPSG. In *Proceedings of ranlp 2003* (pp. 112–116). Borovets, Bulgaria.

Crysmann, B. (2007). Local ambiguity packing and discontinuity in german. In T. Baldwin, M. Dras, J. Hockenmaier, T. H. King, & G. van Noord (Eds.), *Proceedings of the acl 2007 workshop on deep linguistic processing* (pp. 144–151). Prague, Czech Republic: Association for Computational Linguistics.

Crysmann, B., Frank, A., Kiefer, B., Müller, S., Neumann, G., Piskorski, J., Schäfer, U., Siegel, M., Uszkoreit, H., Xu, F., Becker, M., & Krieger, H.-U. (2002). An integrated architecture for shallow and deep processing. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics.* Philadelphia, PA.

Dalrymple, M. (2006). How much can part-of-speech tagging help parsing? *Natural Language Engineering*, *12*(4), 373 – 389.

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

Frank, A., Becker, M., Crysmann, B., Kiefer, B., & Schäfer, U. (2003). Integrated shallow and deep parsing. TopP meets HPSG. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics* (pp. 104 – 111). Sapporo, Japan.

Grefenstette, G., & Tapanainen, P. (1994). What is a word, what is a sentence? problems of tokenization. In *Proceedings of the 3rd international conference on computational lexicography and text research (complex '94)* (pp. 79–87). Budapest.

Habert, B., Adda, G., Adda-Decker, M., Marëuil, P. B. de, Ferrari, S., Ferret, O., Illouz, G., & Paroubek, P. (1998). Towards tokenization evaluation. In A. Rubio, N. Gallardo, R. Castro, & A. Tejada (Eds.), *Lrec:98* (Vol. I, pp. 427–431). Granada.

Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. In B. Grosz, K. Sparck Jones, & B. Webber (Eds.), *Readings in natural language processing* (pp. 35 – 70). San Francisco, CA: Morgan Kaufmann Publishers.

Kiefer, B., & Krieger, H.-U. (1998). *A bag of useful techniques for efficient and robust parsing* (Research Report # RR-98-04). Saarbrücken, Germany: German Research Center for Artificial Intelligence (DFKI).

Kiefer, B., Krieger, H.-U., Carroll, J., & Malouf, R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th annual meeting of the association for computational linguistics on computational linguistics* (pp. 473–480). Morristown, NJ, USA: Association for Computational Linguistics.

Malouf, R., & van Noord, G. (2004). Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP workshop Beyond Shallow Analysis.* Hainan, China.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English. The Penn Treebank. *Computational Linguistics*, *19*, 313 – 330.

Miyao, Y., Ninomiya, T., & Tsujii, J. (2005). Corpus-oriented grammar development for acquiring a Head-Driven Phrase Structure Grammar from the Penn Treebank. In K.-Y. Su, J. Tsujii, J.-H. Lee, & O. Y. Kwong (Eds.), *Natural language processing* (Vol. 3248, pp. 684 – 693). Hainan Island, China.

Müller, S., & Kasper, W. (2000). HPSG analysis of German. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 238 – 253). Berlin, Germany: Springer.

Nunberg, G. (1990). *The linguistics of punctuation*. Stanford, CA: CSLI Lecture Notes 18.

Oepen, S., Dyvik, H., Lønning, J. T., Velldal, E., Beermann, D., Carroll, J., Flickinger, D., Hellan, L., Johannessen, J. B., Meurer, P., Nordgård, T., & Rosén, V. (2004). Som å kapp-ete med trollet? Towards MRS-based Norwegian – English Machine Translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*. Baltimore, MD.

Oepen, S., Flickinger, D., Toutanova, K., & Manning, C. D. (2004). LinGO Redwoods. A rich and dynamic treebank for HPSG. *Journal of Research on Language and Computation*, *2*(4), 575 – 596.

Petitpierre, D., & Russell, G. (1994). *MMORPH — the Multext morphology program* (Multext deliverable # 2.3.1). ISSCO, University of Geneva.

Prins, R., & van Noord, G. (2001). Unsupervised POS-tagging improves parsing accuracy and parsing efficiency. In *Proceedings of the 7th International Workshop on Parsing Technologies* (pp. 154 – 165). Beijing, China.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell III, J. T., & Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*. Philadelphia, PA.

Schäfer, U. (2006). Middleware for creating and combining multi-dimensional nlp markup. In *Proceedings of the eacl-2006 workshop on multi-dimensional markup in natural language processing*. Trento, Italy.

Waldron, B., Copestake, A., Schäfer, U., & Kiefer, B. (2006). Preprocessing and tokenisation standards in DELPH-IN tools. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Genoa, Italy.