Recognition can be Harder than Parsing

Bernard Lang*

INRIA – Rocquencourt

BP 105, 78153 Le Chesnay Cedex, France

e-mail: bernard.lang@inria.fr

The work presented here attempts to bring out some fundamental concepts that underlie some known parsing algorithms, usually called chart or dynamic programming parsers, in the hope of guiding the design of similar algorithms for other formalisms that could be considered for describing the "surface" syntax of languages. The key idea is that chart parsing is essentially equivalent to a simple construction of the intersection of the language (represented by its grammar) with a regular set containing only the input sentence to be parsed (represented by a finite state machine). The resulting grammar for that intersection is precisely what is usually called a shared forest: it represent all parses of a syntactically ambiguous sentence. Since most techniques for processing ill-formed input can be modeled by considering a non-singleton regular set of input sentences, we can expect to generalize these ill-formed input processing techniques to all parsers describable with our approach.

Key words: parsing, chart, dynamic programming, context-free, tree adjoining, parse tree, parse forest, sharing, ambiguity, ill-formed sentences.

1 Introduction

In formal language theory, grammars (not necessarily context-free) are often perceived as the means to define languages as sets of well formed *strings* or *sentences* on some alphabet. This view is clearly too simplistic for actual uses of grammars, whether for computer or for natural language processing. In practice, the well formed strings of a language must be the projection of some syntactic structure, often a tree structure, to which "meaning" can be associated.

The work presented here attempts to bring out some fundamental concepts that underlie some known parsing algorithms, usually called chart or dynamic programming parsers, in the hope of guiding the design of similar algorithms for other formalisms that could be considered for the description of the "surface" syntax of languages. This work also sheds light on sharing techniques used to represent ambiguity, and on its relations with the parsing of ill-formed input.

*This work has been partially supported by the Eureka Software Factory project.

Given a grammar $\mathcal{G}$, a *recognition* algorithm is one that determines for any given string whether it is in the language $\mathcal{L}(\mathcal{G})$ defined by that grammar. A *parsing* algorithm is one that, for any given string, can reconstruct the syntactic structure, called a *parse*, of which the string is a projection. Though this can be confused with retrieving the structure of the string generation process in the case of *context-free (CF)* formalisms (and several others), there are also situations where the generation structure is distinct from the syntactic structure that is projected as the given string, as is the case for *tree-adjoining grammars (TAG)* (Vijay-Shanker & Joshi, 1985). When the two structures are distinct, parsing could be defined as the retrieval of either; however the generation structure will not be considered in this paper.[1]

The fact that a string is a projection raises the issue of *ambiguity* when the same string may be the projection of several distinct syntactic structures. A parsing algorithm can then produce any of the possible structures underlying the string, or if it is *complete* it will produce all of them (and for example leave it to later processing to make a choice based on considerations not accounted for in the grammatical formalism[2]).

Unfortunately it is often the case that, at this purely syntactic level, sentences are highly ambiguous and the number of parses can be exponential in the size of the parsed sentence. Then, enumerating all possibles parses may be inconvenient for two reasons:

- the cost in time and space is proportional to the number of parses, while recognition or producing a single parse may be more efficient (e.g. polynomial in the size of the parsed sentence).
- further processing has to be done separately on each of the parses, while it is almost always the case that the corresponding structures have common subparts that could be shared, and then processed only once.

Hence, it is natural to produce as the result of a parsing algorithm a single structure that represents all possible parses with sharing of common subparts. In the case of CF languages, this was introduced with some of the early dynamic programming parsing algorithm (e.g. Earley's (1968)) which, in cubic time and space, could produce as result a shared structure, called *shared forest*, from which any specific parse can be extracted in time linear with the size of the extracted parse tree.[3]

The object of this paper is to discuss the scope and limitations of this approach, and to examine the suitability of several syntactic formalisms on the criterion of their ability to handle it.

2 Parsing as intersection

In the CF case, a shared forest $\mathcal{F}$ may be seen as a grammar generating only the parsed sentence, with precisely the same parse trees as the original grammar $\mathcal{G}$ of the

[1] Vijay-Shankar and Weir (1993) have developed for TAG a variant of the work presented here, based on the derivation structure.

[2] Here we concern ourselves with the "backbone" syntactic formalism, which is in practice complemented by other mechanisms such as feature structures.

[3] For finitely ambiguous grammars, this is equivalent to being time linear in the size of the parsed sentence.

language, up to a renaming of the non-terminals. This understanding, introduced in (Lang, 1974; Billot & Lang, 1989) was extended to other formalisms by Vijay-Shanker and Weir (1990b). As a corollary, all parses can be trivially enumerated from the shared forest in exactly the same way one generates all possible sentences of a language in the formalism under consideration.

It was also shown in (Lang, 1991) that, more generally, it is possible to build a grammar $\mathcal{F}$ that generates with the same parse-trees (up to non-terminal renaming) as a CF grammar $\mathcal{G}$, the sentences produced by a finite state device (a word lattice, for example) that are in the language of $\mathcal{G}$. This extension of parsing from a single sentence to a regular set of sentences can be a basis for a systematic approach to several ill-formed input processing techniques, since they are often based on finite state devices (Teitelbaum, 1973; Lang, 1989). The resulting grammar is then a shared forest for a set of possible sentences, rather than for a single sentence.

These results are also a clue to a strong relationship between parsing and intersection with regular sets. The shared forest $\mathcal{F}$ built by a $\mathcal{G}$ parser for the acceptable sentences in a regular set $\mathcal{R}$ is precisely a grammar $\mathcal{F}$ for the intersection of $\mathcal{R}$ and the CF language $\mathcal{L}(\mathcal{G})$.

Turning the problem around, let us consider a regular language $\mathcal{L}(\mathcal{A})$ defined by a finite state device $\mathcal{A}$ and a CF language $\mathcal{L}(\mathcal{G})$ generated by a CF grammar $\mathcal{G}$. There is a well known construction, due to Bar-Hillel, Perles and Shamir (1961) — referred to as BPS in this paper — that produces a CF grammar for the intersection of these two languages, with precisely the same parse trees as $\mathcal{G}$ (though this was not established by BPS). Hence the grammar may be considered as a shared forest for the $\mathcal{G}$-acceptable sentences in $\mathcal{L}(\mathcal{A})$. The time complexity of the construction and the size of the resulting grammar are both $\mathcal{O}(n^{p+1})$, where n is the number of states of $\mathcal{A}$ and p is the length of the longest rule right-hand-side in $\mathcal{G}$. Hence they are both cubic in the number of states of $\mathcal{A}$, when the grammar $\mathcal{G}$ is in so-called *2-form* or *binary form*, i.e., the right-hand-side of each rule is of length 2 at most.

When parsing a single sentence of length n, one can consider this sentence as a linear finite state device $\mathcal{A}$ with $n+1$ states that generates only that sentence. Then the same construction gives a shared parse forest for the sentence, with a complexity that is cubic in the length n of the sentence.

The various dynamic programming CF parsing algorithms that have been published since in the literature (Cocke & Schwartz, 1970; Younger, 1967; Kasami, 1965; Earley, 1968; Lang, 1974; Tomita, 1987; Billot & Lang, 1989; Schabes, 1991) may be seen as "optimizations" of this original construction. The gain in performance comes from the fact that the raw BPS construction produces a grammar that contains a large number of useless rules and non-terminals and thus needs simplification, while the more recent constructions attempt to directly produce a grammar (i.e., shared forest) which is as clean as possible of useless components. Note however that these algorithms may give a reduced grammar that is more complex than the reduced grammar produced with the BPS construction, which is almost minimal[4] in its number of non-terminals. Hence

[4]This notion of minimality of reduced grammars can be precisely defined in terms of grammar morphisms. Actually the grammar resulting from the intersection construction mimics the compu-

our use of the quotation symbols around the word optimization above. For example the well known CKY construction avoids producing non-terminals and rules that are *non-generating*, i.e., that do not derive on any terminal string. Most other constructions have top-down filters that eliminate some *inaccessible* non-terminals and rules, i.e., which would not be reachable from the initial symbol of the shared forest.

The constraint of using grammars in binary form to get the cubic complexity applies to all these algorithms, whether explicitly (Cocke & Schwartz, 1970; Younger, 1967; Kasami, 1965) or implicitly (Earley, 1968; Lang, 1974; Schabes, 1991).

Finally we should note that the BPS construction builds a shared parse forest $\mathcal{F}$, which is a grammar. However it does not tell directly whether the string that was "parsed" is actually recognized as being in the language $\mathcal{L}(\mathcal{G})$. As noted by Aho (1968, Theorem 4.2), this can be determined by checking whether the language $\mathcal{L}(\mathcal{F})$ is empty or not, which can be done in time linear with the size of $\mathcal{F}$.

3 Generalization to Tree Adjoining Grammars

3.1 Abstract description of the technique

The obvious next step is to use this insight to design complete parsers for other families of formalisms. Given a family Φ of grammatical formalisms, we attempt to derive the construction of dynamic programming parsers for Φ from a constructive proof that the family Φ is closed under intersection with regular sets. More precisely, given a Φ-grammar $\mathcal{G}$ and a finite state automaton $\mathcal{A}$, if we can construct from them a new Φ-grammar $\mathcal{F}$ for the intersection $\mathcal{L}(\mathcal{G}) \cap \mathcal{L}(\mathcal{A})$, we define this new grammar $\mathcal{F}$ to be the shared forest for all parses of the sentences in the intersection. A parser is then an algorithm that can build a Φ-grammar $\mathcal{F}$, given any $\mathcal{G}$ and $\mathcal{A}$. *To achieve good complexity of the construction, we may have to find, for grammars in the Φ family, a normal form similar to the binary form of CF grammars.*

As above, for parsing a single sentence, we consider a finite state automaton generating only that sentence. Furthermore this approach generalizes to word lattice parsing, and to many ill-formed sentence processing techniques based on finite state models.

To illustrate this approach, we now apply it to the case of Tree Adjoining Grammars. However, we shall not use the traditional definition of TAG as given for example in (Vijay-Shanker & Joshi, 1985). This definition has deliberately restricted the formalism so as to more closely reflect linguistic phenomena. However since we are interested in formal algorithmic issues, we shall use here a more general definition (very close to modified Head Grammars), to which we can more easily associate a "binary form".

tation of the finite state automaton (FSA): even if the FSA is minimal, it may happen that two of its states are distinct only because they discriminate between two classes of sentence prefixes, which actually behave identically, i.e., accept the same suffixes (Hopcroft & Ullman, 1969, theorem 3.1), within the CF language considered. The intersection construction preserves this discrimination in the shared forest grammar, where it is no longer useful. Hence this grammar is not minimal. Some chart parsers may introduce other such ultimately useless discriminations, when attempting early detection of failing non-deterministic computation paths (Billot & Lang, 1989).

3.2 TAG parsing as intersection

A *Tree Adjoining Grammar (TAG)* determines a set of syntax trees. It is defined as a 5-tuple $\mathcal{G} = (V, C, T, \mathrm{S}, \mathcal{P})$ where, V, C and T are sets of respectively *variable*, *constant* and *terminal* symbols[5], S is an initial variable symbol, labeling the only node of the *initial tree*, $\mathcal{P}$ is a set of adjoining *rules* (or *productions*) of the form $\mathrm{X} \longrightarrow \tau$, where $\mathrm{X} \in V$ is the *left-hand-side (lhs)*, and the *right-hand-side (rhs)* τ is a tree labeled with symbols from $V \cup C \cup T$. Variable and constant symbols are also called *non-terminal* symbols.

The rhs τ of a rule may have a distinguished leaf node called the *foot*. The label of a foot or a non-leaf node is a non-terminal. *A foot is marked on graphical tree representations by an asterisk next to its label.* Leaf nodes are labeled with variables, terminals, or the empty string ϵ. A production $\mathrm{X} \longrightarrow \tau$ can be applied to any X-labeled node of a tree, by adjoining τ at that node, if τ has a foot. If τ does not have a foot, it can only be substituted to a leaf node labeled with the lhs X of the production.

The adjunction of τ at a X-labeled node N is performed as follows: *excise* the subtree θ' of θ occurring in N, replace it with a copy of the tree τ, and graft the excised subtree θ' to the foot of this copy of τ, without changing the foot label (i.e., the label of the node where adjunction takes place is lost).

A tree is generated by the TAG $\mathcal{G}$ iff it can be obtained from the initial tree by a succession of applications of adjoining rules. A tree is said to be *variable* if it contains variables, and to be *constant* otherwise. The string language of the TAG $\mathcal{G}$ is the set of frontiers (which are terminal strings) of the constant trees generated by $\mathcal{G}$.

A TAG is in binary form if all of its tree rewrite rules have one of the following forms, where $\mathrm{a} \in T \cup \{\epsilon\}$, $\mathrm{X,Y,Z} \in (V \cup C)$, and $\mathrm{W} \in C$, i.e., nodes labeled W are not rewritable:

```
a)  X ⟶ Y*              c)  X ⟶ W              e)  X ⟶ a
                                / \
                              Y*   Z

b)  X ⟶ Y               d)  X ⟶ W
        |                       / \
        Z*                     Y   Z*
```

It is easy to show that for any TAG, there is a binary TAG that generates the same language, with the same underlying tree structures for each sentence up to some trivial recoding of the trees, such as exclusive use of binary or unary nodes exactly as is the case for binary CF grammars. Note that the c) and d) rules do not allow any adjunction

[5]The use of rewrite rules for adjunction gives the control equivalent of *selective* adjoining, while the use of variable and constant non-terminals, together with the constraint of ending up with only constant ones gives the equivalent of *null* or *obligating* adjoining (Vijay-Shanker & Joshi, 1985).

on their rhs root (because W$\in C$), but this can always be encoded with the use of an intermediate b) rule. Note also that e) rules can only apply to leaf nodes.

We consider a TAG $\mathcal{G} = (V, C, T, \mathrm{S}, \mathcal{P})$ in binary form, and a finite state automaton $\mathcal{A}= (\Sigma, Q, \delta, \mathrm{q}_0, Q_f)$ where Q is the set of states, q_0 is the initial state, Q_f the set of final states, and we assume that every transition in δ reads exactly one input symbol in Σ. We can build a (binary) TAG for the intersection $\mathcal{L}(\mathcal{G}) \cap \mathcal{L}(\mathcal{A})$ with the TAG analog of the BPS construction. This construction is very similar to that found in (Vijay-Shanker & Joshi, 1985) for both the TAG parser and the proof of closure under intersection with regular sets. However we use explicitly the binary form which is kept implicit in the parser of (Vijay-Shanker & Joshi, 1985) (though the exclusive use of binary nodes is explicit).

From $\mathcal{G}$ and $\mathcal{A}$ we construct a new TAG $\mathcal{G}' = (V', C, T, \mathrm{S}', \mathcal{P}')$, where S' is a new variable symbol and $V' = (V \times Q^4) \cup \{\mathrm{S}'\}$.

We consider successively each type of binary rule above:

a) For the production $\mathrm{X} \longrightarrow \mathrm{Y}^*$, we consider 2 cases depending on the symbol Y.

Y$\in C$: we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s} \rangle \longrightarrow \mathrm{Y}^*$ in $\mathcal{P}$' for every 4-tuple of states p,q,r,s$\in Q$.

Y$\in V$: we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s} \rangle \longrightarrow \langle \mathrm{Y}, \mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s} \rangle^*$ in $\mathcal{P}$' for every 4-tuple of states p,q,r,s$\in Q$.

b) For the production $\mathrm{X} \longrightarrow \begin{array}{c} \mathrm{Y} \\ | \\ \mathrm{Z}^* \end{array}$ we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{p}'', \mathrm{q}'', \mathrm{q} \rangle \longrightarrow \begin{array}{c} \langle \mathrm{Y}, \mathrm{p}, \mathrm{p}', \mathrm{q}', \mathrm{q} \rangle \\ | \\ \langle \mathrm{Z}, \mathrm{p}', \mathrm{p}'', \mathrm{q}'', \mathrm{q}' \rangle^* \end{array}$

in $\mathcal{P}$' for every 6-tuple of states $\mathrm{p}, \mathrm{p}', \mathrm{p}'', \mathrm{q}, \mathrm{q}', \mathrm{q}'' \in Q$.

c) For the production $\mathrm{X} \longrightarrow \begin{array}{ccc} & \mathrm{W} & \\ / & & \backslash \\ \mathrm{Y}^* & & \mathrm{Z} \end{array}$ we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{p}', \mathrm{q}', \mathrm{q} \rangle \longrightarrow \begin{array}{ccc} & \mathrm{W} & \\ / & & \backslash \\ \langle \mathrm{Y}, \mathrm{p}, \mathrm{p}', \mathrm{q}', \mathrm{r} \rangle^* & & \langle \mathrm{Z}, \mathrm{r}, \mathrm{q}, \mathrm{q}, \mathrm{q} \rangle \end{array}$

in $\mathcal{P}$' for every 5-tuple of states $\mathrm{p}, \mathrm{p}', \mathrm{q}, \mathrm{q}', \mathrm{r} \in Q$.

d) For the production $\mathrm{X} \longrightarrow \begin{array}{ccc} & \mathrm{W} & \\ / & & \backslash \\ \mathrm{Y} & & \mathrm{Z}^* \end{array}$ we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{p}', \mathrm{q}', \mathrm{q} \rangle \longrightarrow \begin{array}{ccc} & \mathrm{W} & \\ / & & \backslash \\ \langle \mathrm{Y}, \mathrm{p}, \mathrm{r}, \mathrm{r}, \mathrm{r} \rangle & & \langle \mathrm{Z}, \mathrm{r}, \mathrm{p}', \mathrm{q}', \mathrm{q} \rangle^* \end{array}$

in $\mathcal{P}$' for every 5-tuple of states $\mathrm{p}, \mathrm{p}', \mathrm{q}, \mathrm{q}', \mathrm{r} \in Q$.

e) For the production $\mathrm{X} \longrightarrow \mathrm{a}$, we consider 2 cases depending on the symbol a.

a$\in T$: we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{q}, \mathrm{q}, \mathrm{q} \rangle \longrightarrow \mathrm{a}$ in $\mathcal{P}$' for every pair of states p,q$\in Q$ such that q$\in\delta$(p,Y).

a=ϵ: we create a rule $\langle \mathrm{X}, \mathrm{p}, \mathrm{p}, \mathrm{p}, \mathrm{p} \rangle \longrightarrow \epsilon$ in $\mathcal{P}$' for every state p$\in Q$.

Finally we add a rule $\mathrm{S}' \longrightarrow \langle \mathrm{S}, \mathrm{q}_0, \mathrm{q}_f, \mathrm{q}_f, \mathrm{q}_f \rangle$ in $\mathcal{P}$' for every state $\mathrm{q}_f \in Q_f$.

This construction has clearly an $\mathcal{O}(n^6)$ time and space complexity, where n is the number of states of the finite automaton $\mathcal{A}$. Hence, it can be used for parsing TAGs with $\mathcal{O}(n^6)$ complexity, and produce a shared forest which is a TAG generating only the parsed sentence. Using Aho's theorem, since emptiness is linearly decidable for TAGs, we can recognize the sentences of a TAG language in time $\mathcal{O}(n^6)$. Finally, all ill-formed input processing techniques based on finite state models (i.e., most of them) can be applied to TAGs.

Of course, as in the CF case, better constructions can be devised, analogous to CKY or Earley's, to achieve more efficient parsing and cleaner shared forests.

3.3 The role of normal forms

One more point worth discussing is that, in the CF case, the binary form is required — explicitly or implicitly — to achieve the low complexity result. Hence we believe it is better to do so explicitly so as to control better the shape of the shared forest and the amount of sharing in that forest, cf. the discussion in (Billot & Lang, 1989).

The binary form for TAGs, like the CFG binary form, is chosen to achieve good complexity bounds. In both cases, TAG and CFG, it is possible to transform any grammar $\mathcal{G}$ into an equivalent binary form grammar $\widehat{\mathcal{G}}$ without modifying the shape of parse trees in any essential way, and without changing the ambiguity of sentences. The parses of a sentence according to $\mathcal{G}$ can be trivially and linearly computed from the parses according to $\widehat{\mathcal{G}}$. The role of the binary form is merely to allow the decomposition of the parsing computation into smaller steps, so as to attain a better sharing of the computation steps, and of the parse structure components.

Vijay-Shankar and Weir (1993) propose to achieve the same result for a TAG by encoding the necessary information in the derivation grammar, which is put in binary form, without changing the original TAG. The price to be paid is of course that the derivation grammar has to be included explicitly in the formalism, in addition to the TAG. Both techniques are essentially equivalent computationally, and our binary form requirement is not a more significant issue. Furthermore, it seems that they should produce the same binary derivation trees or forests, which, for further linguistic processing, are probably better representations of the parses than the derived trees.

4 Limitations of the approach

This approach extends nicely to such formalisms as Modified Head Grammars, Linear Indexed Grammars (Vijay-Shanker & Weir, 1990b, 1990a), and more generally to Linear Context-Free Rewriting Systems, though with higher polynomial complexity.[6] It is therefore natural to ask whether the same approach remains meaningful for more powerful formalisms, which one could think of using as syntactic backbone of a natural language formalism.

We shall consider two examples. The first one is the type 0 languages of the Chomsky hierarchy. Given a type 0 grammar $\mathcal{G}$ and a finite automaton $\mathcal{A}$, it is quite simple

[6] The complexity of parsing LCFRS is always polynomial in the length of the input, but the exponent can take any value depending on the chosen grammar. This result is actually related to the polynomial complexity of Datalog programs (Immerman, 1982).

to build a new type 0 grammar $\mathcal{F}$ for the intersection of their languages. Unfortunately, this new grammar $\mathcal{F}$ hardly qualifies as a shared-forest since it gives very little information about the specific structure of the parsed sentence. The real truth is that type 0 grammars themselves do not associate in any simple way a syntactic structure to the sentences in their language. Hence they are not adequate as syntactic backbone, and though it is possible in theory to use our approach for type 0 grammars, it is in practice meaningless.[7]

The case of *Indexed Grammars (IG)*, as defined by Aho (1968), is more interesting. IGs do associate a syntactic structure with the sentences of their language. Furthermore, they are closed under intersection with regular set, and the corresponding construction has cubic complexity in the number of states of the finite automaton defining the regular set, as for CF grammars. Hence parsing can be done in cubic time and produce a shared forest with cubic size. The catch is that testing whether the language of an IG is empty has exponential complexity in the size of the grammar: the algorithm given by Aho (1968) is exponential, and Rounds (1973) has established that recognition of IG languages is NP-complete. Thus, while parsing is cubic (according to our notion of parsing as producing a shared forest), recognition is exponential.

In practice this means that, though it is quite feasible to build a meaningful shared forest for IGs, the corresponding structure is too complex to be of practical use. This implies not only that parsing (in the traditional sense that includes recognition) is a costly operation for IGs, but also that there seem to be no obvious convenient structure to represent in shared form the set of IG-parses of an ambiguous sentence.

5 Conclusion

The above analysis shows that the idea of representing the shared parse forest of an ambiguous sentence as a Φ-grammar (where Φ is the type of formalism considered) is applicable to a variety of known formalisms. It also links dynamic programming parsing and intersection with regular sets, thus giving strong hints to the design of dynamic programming parser, notably w.r.t. complexity and sharing control, even though optimizing these parsers remains a substantial task. It also shows that ill-formed input processing techniques, based mostly on finite-state models, are generalizable in the same way.

For example, confusion matrices and hidden markov models used in speech processing are finite state devices. Similarly word lattices may be interpreted as acyclic finite state automata generating a set of possible sentences. Many lexical error corrections, such as inversion, replacement, omission or addition of words, can be modeled by finite

[7]In a previous version of this paper, the author had taken as an example the *Context-Sensitive (CS)* languages, i.e., the type 1 languages of the Chomsky hierarchy. Erik Aarts and K. Vijay-Shanker pointed out to him that CS grammars do assign a structure to sentences which corresponds to the CF backbone of the CS grammar, at least for one definition of CS grammars (Hopcroft & Ullman, 1969, page 13). It is indeed possible to build parse forests for CS languages, but it raises the same emptiness testing problem as the indexed grammars. Emptiness of CS languages is undecidable in general (Hopcroft & Ullman, 1969); however in the present case it may be tested in exponential time by a linear bounded automaton with a store size not exceeding the length of the parsed sentence.

state tranformations of the input sentence, resulting in a regular set of sentences to be parsed. Applying the intersection construction to this regular set results in a shared forest for precisely those of the possible input sentences that are syntactically correct. In this framework, the processing of ill-formed input is not essentially different from that of ambiguous sentences.

However, shared forests must not only be constructed easily enough, which is often the case, but they must also be structurally meaningful. Furthermore they must be usable with sufficient ease, preferably in linear time, for example to generate individual trees.

These criteria show that "mildly context-sensitive formalisms" such as TAG, LCFRS, etc. are quite adequate as backbone syntactic formalisms w.r.t. several issues of linguistic processing. They also show that too much power, as in the case of indexed grammars, quickly leads to intractability of these issues because of the complexity of shared forests.

Thus it seems that, considering their ability to handle discontinuous constituents, mildly context-sensitive formalisms provide the adequate level of syntactic complexity needed for a linguistic backbone formalism. However this ability entails some added complexity in parsing and size of the shared forest, and it should probably be used as sparingly as possible.

Acknowledgements

The author wishes to thank Jean-Marc Steyaert for timely reminding him of the intersection construction for CF languages and regular sets.

Reference

Aho, A. (1968). Indexed grammars — an extension to context-free grammars. *Journal of the ACM*, *15*(4), 647–671.

Bar-Hillel, Y. (1964). *Language and Information — Selected Essays on their Theory and Application.* Series in Logic. Addison-Wesley.

Bar-Hillel, Y., Perles, M., & Shamir, E. (1961). On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunicationsforschung*, *14*, 143–172. Reprinted in (Bar-Hillel, 1964, pp. 116-150).

Billot, S., & Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27^{th} Annual Meeting of the Association for Computational Linguistics*, pp. 143–151 Vancouver, British Columbia.

Cocke, J., & Schwartz, J. T. (1970). *Programming Languages and Their Compilers.* Courant Institute of Mathematical Sciences, New York University, New York.

Earley, J. (1968). *An Efficient Context-Free Parsing Algorithm.* Ph.d. thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Hopcroft, J. E., & Ullman, J. D. (1969). *Formal Languages and their Relation to Automata*. Series in Computer Science and Information Processing. Addison-Wesley Publishing Company.

Immerman, N. (1982). Relational queries computable in polynomial time. In *Proceedings of the 14th ACM/SIGACT Symposium*, pp. 147–152.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Report, University of Hawaii. Also AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford (Massachusetts), and 1966, University of Illinois Coordinated Science Lab. Report, No. R-257.

Lang, B. (1974). Deterministic techniques for efficient non-deterministic parsers. In Loeckx, J. (Ed.), *Proceedings of the 2^{nd} Colloquium on Automata, Languages and Programming*, pp. 255–269 Saarbrücken. Springer Lecture Notes in Computer Science 14. Also: Rapport de Recherche 72, IRIA-Laboria, Rocquencourt (France).

Lang, B. (1989). A generative view of ill-formed input processing. In *Proceedings of the ATR Symposium on Basic Research for Telephone Interpretation (ASTI)* Kyoto (Japan).

Lang, B. (1991). Towards a uniform formal framework for parsing. In Tomita, M. (Ed.), *Current Issues in Parsing Technology*, pp. 153–171. Kluwer Academic Publisher.

Rounds, W. C. (1973). Complexity of recognition in intermediate-level languages. In *Proceedings of the 14^{th} Annual Symposium on Switching & Automata Theory*, pp. 145–158. IEEE Computer Society.

Schabes, Y. (1991). Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *Proceedings of the 29^{th} Annual Meeting of the Association for Computational Linguistics* Berkeley, California.

Teitelbaum, R. (1973). Context-free error analysis by evaluation of algebraic power series. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pp. 196–199 Austin (Texas).

Tomita, M. (1987). An efficient augmented-context-free parsing algorithm. *Computational Linguistics*, *13*(1-2), 31–46.

Vijay-Shanker, K., & Joshi, A. K. (1985). Some computational properties of tree adjoining grammars. In *Proceedings of the 23^{rd} Annual Meeting of the Association for Computational Linguistics*, pp. 145–152 Chicago (Illinois).

Vijay-Shanker, K., & Weir, D. J. (1990a). Parsing constrained grammar formalisms. *Computational Linguistics*. to appear.

Vijay-Shanker, K., & Weir, D. J. (1990b). Polynomial time parsing of combinatory categorial grammars. In *Proceedings of the 28^{th} Annual Meeting of the Association for Computational Linguistics*, pp. 1–8 Pittsburgh (Pennsylvania).

Vijay-Shanker, K., & Weir, D. J. (1993). The use of shared forests in tree adjoining grammar parsing. In *Proceedings of the 6^{th} Conference of the European Chapter of the Association for Computational Linguistics*, pp. 384–393 Utrecht, The Netherlands.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, *10*(2), 189–208.