Taming Complexity: Constraint-Based Dependency Parsing

Denys Duchier
Programming Systems Lab
University of the Saarland, Saarbrücken
`duchier@ps.uni-sb.de`

Abstract

Much of linguistic theorizing nowadays is concerned with the formulation of general structural principles that determine syntactically well-formed entities. Yet, devising effective parsing mechanisms for such axiomatic frameworks remains a stumbling block plagued by combinatorial explosion. We propose to take advantage of the axiomatic nature of such frameworks to obtain constraint-based formulations. In this fashion, we are able to achieve effective model elimination through constraint propagation, thus drastically reducing the need for search. We demonstrate this idea with a formal account of a dependency grammar for german. The mathematical equations that we present have a direct interpretation as constraints. Our approach was implemented in the concurrent constraint programming language Oz and proved to be surprisingly efficient.

Keywords: dependency parsing, formal grammars, concurrent constraint programming, inference, set constraints

1 Introduction

While overtly about parsing, this article more generally advocates and demonstrates a radical approach to problems with challenging combinatorial complexity. Traditionally, this complexity and the resulting combinatorial explosion have been addressed by attempts to devise better search strategies. We believe that this is curing the symptom rather than the disease. Thus our message takes the form of a truism too often overlooked:

> *The only effective way to deal with search is to do as little of it as possible.*

Having said that, what can we do about it? Search is a process of making choices, thus to reduce search we need to drastically decrease the number of choices that must be explicitly considered. This can be achieved through *inference*. Let inference automatically decide necessary choices and eliminate from consideration inconsistent ones.

For this to be possible, choices must be exposed to the inference mechanisms. For many parsing frameworks, such exposure is problematic, often impossible. Chart parsing is a typical offender: the underlying idea is to try all possible combination of adjacent edges. Each combination is a choice; it is generative; and has practically no influence on any other choice.

A radical approach, and the one which we demonstrate in the remainder of the article, is to make all choices explicit and exposed to the inference mechanisms. We abandon the generative view. We no longer build larger partial parses by combination of smaller ones. Rather, we give a global well-formedness condition that characterizes an admissible parse tree of the input sentence and proceed to explicitate its models. In our approach, every choice influences the entire partial model.

If this is beginning to sound familiar, it should. Much of linguistic theorizing nowadays is concerned with the formulation of general structural principles that determine syntactically well-formed entities. Yet, devising effective parsing mechanisms for such axiomatic frameworks remains a stumbling block. This is especially the case for languages with free word order where surface order is no longer a practical guide for search. As a consequence, the field has maintained a schizophrenic disposition. Blackburn [Bla95] described this duality as the "static" and "dynamic" perspectives:

> *... by a 'dynamic' perspective is meant a view that favours syntactic explanations couched in terms of the construction, manipulation and generation of structures. A 'static' perspective ... is one emphacising theorising couched in terms of general structural principles governing syntactically well-formed entities.*

While linguistic theorising has become largely of the "static" persuasion, parsing has remained, by and large, "dynamic." We suggest that the very axiomatic nature of "static" frameworks is the route to salvation. From it we can obtain sufficiently strong inference to beat the combinatorial explosion of search. We propose to replace the *generate and test* flavor of standard approaches with the constraint programming model of *propagate and distribute*, thus achieving effective model elimination through a powerful inference mechanism. To make this possible, we need constraints and they can best be obtained from an axiomatic formulation.

That is what we demonstrate in this article. We develop a “static” axiomatization of “admissible” dependency structures. We formulate this mathematical account in such a way that it is especially well-suited for model elimination through constraint propagation. We demonstrate how certain notions, such as the *yield*, can be given an axiomatic treatment that permits their natural involvement in constraint propagation. Our formulation also illustrates the expressivity, elegance and economy to be gained with the use of set constraints.

Further, our formulation has two very desirable properties: First, it has a direct computational reading. Modulo minor details of syntax, it can be regarded as a program in a concurrent constraint programming language such as Oz. Second, the resulting program is remarkably effective. Inference is so strong that typically the search tree enumerates all and only the valid readings without any failure.

2 An Example of Constraint Propagation

Before diving into the outline our formal framework, let us illustrate on a simple example how drastically inference achieved through constraint propagation may reduce the search space. In particular, it will become clear how negative information performs radical model elimination. Consider the sentence:

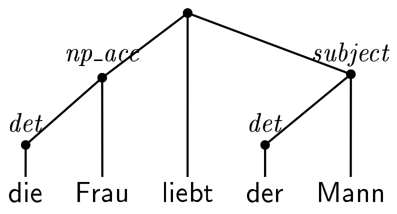
die Frau liebt der Mann

It is clear to the reader that *der Mann* must be the subject and *die Frau* the accusative object. Here is how constraint propagation arrives at this conclusion.

1. *liebt* must be the root of the sentence since it is the only finite verb
2. *der* cannot be determiner for *Frau* since it comes after it
   (a) therefore *Mann* is the only possible head for *der*
   (b) as a consequence, *die* cannot also be determiner of *Mann*
   (c) thus, only *Frau* can be head of *die*
3. due to the agreement constraint, *der Mann* must be nominative
   (a) therefore *Mann* cannot be the accusative object of *liebt*
   (b) the only role left for *Mann* is to be subject of *liebt*

(c) therefore, the only role left for *Frau* is to be object of *liebt*

The complete dependency structure for the sentence has been decided by inference, without any need for search. This was achieved through model elimination: after constraint propagation, only one single model was left.



3 How To Make This Happen

Our goal is to develop an approach in which inference may operate as illustrated above. This is achieved by computing explicitly with an underspecified representation of the dependency tree. The tree consists of a collection of "nodes," one per word in the sentence to be parsed, that are to be connected by edges of immediate dominance. For each node, the value of its features and the choice of its daughters are underspecified. Thus, at the beginning of the parse, although we have in our hands all the parts that need to be assembled together, there is much about them that is not fully known, in particular how they are to be connected.

A classical approach would now attempt to build a parse tree by incrementally connecting increasingly more nodes together, as if they were Lego pieces. Our approach is more like carving, we start from a large amorphic whole and let the shape emerge through the elimination of what cannot be part of it. We apply up front all our axiomatic principles to the underspecified representation, thus constraining the latter to admit only grammatical instances. The job of inference is then to refine the underspecified values in this representation by eliminating inconsistent choices and determining necessary ones.

Of course, inference cannot always do the whole job and we may need search. However, the intent of search is no longer to build the tree, rather it is to explicitate the underspecified pieces of its representation. Every decision permits further inference.

Our formalization is primarily formulated in terms of sets. Considerable expressiveness accrues from the set theoretical setting, and we obtain an axiomatization that is both elegant and economical. Further, it demonstrates the advantages of recent developments in constraint programming: constraints over finite sets have very efficient implementations [MM97, Ger95]. Much like a logic variable is the underspecified representation of a term, and a finite domain variable that of an integer, a finite set variable is the underspecified representation of a set. All our choices will reside in the underspecification of set variables. In this fashion, we achieve full exposure of choices to the inference mechanisms.

In [DG99], we already demonstrated the theoretical elegance and computational effectiveness of set constraints for solving dominance descriptions. In the following, we shall do the same for dependency parsing. We believe, however, that the scope of our methodology extends to other frameworks such as HPSG, and especially to Dtree grammars.

4 Preliminaries

Consider the sentence *"das Buch hat mir Peter versprochen zu lesen."*[1] It has two syntactic analyses, one whose dependency tree is displayed in Figure 1 and in which *Peter* is the subject, and one in which *Buch* is the subject. The efficiency of our approach may be appreciated with the observation that it derives just these two readings and using only one choice point.

The example illustrates the sort of non-projective analysis with fronting, scrambling and extraposition that is typical of german sentences. Figure 2 presents the same analysis in the form of a collection of attribute value matrices (AVMs), where conventionally the boxed integers stand for coreference indices.

In the following, we are going to turn our attention entirely to the study of such collections of AVMs and to this question: *"When can we say that a set $\mathcal{W}$ of AVMs forms a grammatically admissible dependency structure?"* We are going to spell out precisely the conditions under which the judgment of admissibility holds.

In this article, for reasons of space, we focus on the expression of principles of immediate dominance and say very little on the subject of word order. However, our framework is especially well suited for the separate ex-

[1] Joachim Niehren proposes the following sentence, which exhibits the same structure, but sounds more convincing to the german ear: *"Genau diese Flasche Wein hat mir mein Kommissionär versprochen auf der Auktion zu ersteigern"*

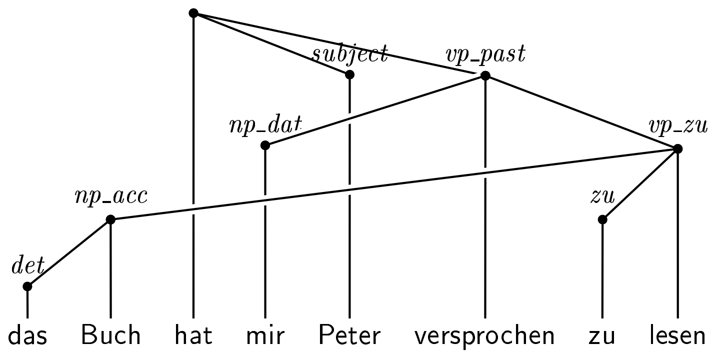


Figure 1: Dependency Tree With Crossing Edges

$$\boxed{1}\begin{bmatrix} \text{string} & : & \texttt{das} \\ \text{index} & : & \texttt{1} \\ \text{cat} & : & \texttt{det} \\ \text{agr} & : & \boxed{2a} \\ \text{comp} & : & \begin{bmatrix} \; \end{bmatrix} \end{bmatrix} \qquad \boxed{5}\begin{bmatrix} \text{string} & : & \texttt{Peter} \\ \text{index} & : & \texttt{5} \\ \text{cat} & : & \texttt{n} \\ \text{agr} & : & \boxed{5a}\ \langle\texttt{masc sing 3 nom}\rangle \\ \text{comp} & : & \begin{bmatrix} \; \end{bmatrix} \end{bmatrix}$$

$$\boxed{2}\begin{bmatrix} \text{string} & : & \texttt{Buch} \\ \text{index} & : & \texttt{2} \\ \text{cat} & : & \texttt{n} \\ \text{agr} & : & \boxed{2a}\ \langle\texttt{neut sing 3 acc}\rangle \\ \text{comp} & : & \begin{bmatrix} \text{det} & : & \boxed{1} \end{bmatrix} \end{bmatrix} \qquad \boxed{6}\begin{bmatrix} \text{string} & : & \texttt{versprochen} \\ \text{index} & : & \texttt{6} \\ \text{cat} & : & \texttt{vpast} \\ \text{comp} & : & \begin{bmatrix} \text{np\_dat} & : & \boxed{5} \\ \text{vp\_inf} & : & \boxed{8} \end{bmatrix} \end{bmatrix}$$

$$\boxed{3}\begin{bmatrix} \text{string} & : & \texttt{hat} \\ \text{index} & : & \texttt{3} \\ \text{cat} & : & \texttt{vfin} \\ \text{agr} & : & \boxed{5a} \\ \text{comp} & : & \begin{bmatrix} \text{subject} & : & \boxed{5} \\ \text{vp\_past} & : & \boxed{6} \end{bmatrix} \end{bmatrix} \qquad \boxed{7}\begin{bmatrix} \text{string} & : & \texttt{zu} \\ \text{index} & : & \texttt{7} \\ \text{cat} & : & \texttt{part} \\ \text{comp} & : & \begin{bmatrix} \; \end{bmatrix} \end{bmatrix}$$

$$\boxed{4}\begin{bmatrix} \text{string} & : & \texttt{mir} \\ \text{index} & : & \texttt{4} \\ \text{cat} & : & \texttt{pro} \\ \text{agr} & : & \langle\texttt{sing 3 dat}\rangle \\ \text{comp} & : & \begin{bmatrix} \; \end{bmatrix} \end{bmatrix} \qquad \boxed{8}\begin{bmatrix} \text{string} & : & \texttt{lesen} \\ \text{index} & : & \texttt{8} \\ \text{cat} & : & \texttt{vinf} \\ \text{comp} & : & \begin{bmatrix} \text{zu} & : & \boxed{7} \\ \text{np\_acc} & : & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Figure 2: AVM Representation of Dependency Tree

pression of principles of linear precedence. In particular, our formulation in terms of sets allows great expressivity. One example is ($LP_{\mathtt{n}}$), on page 13, expressing a restriction on prenominal word order domains.

Our formal setting assumes the existence of a finite set $\mathcal{C}$ of categories such as `n` for noun, `det` for determiner, or `vfin` for finite verb, of the finite set $\mathcal{A}$ of all agreement tuples such as $\langle$`masc sing 3 nom`$\rangle$, of a finite set $\mathcal{R}$ of complement role types such as `subject` or `np_dat` for dative noun phrase, of a finite set $\mathcal{M}$ of modifier role types such a `adj` for adjectives, disjoint from $\mathcal{R}$.

4.1 Lexicon

We assume the existence of a lexicon whose purpose is to map a string, representing the full form of a word, to a set of lexical entries. Each entry e specifies a set of alternative categories $\mathsf{cats}(e)$, a set of alternative agreement tuples $\mathsf{agrs}(e)$, a set of required complements $\lfloor\mathsf{roles}\rfloor(e)$ and a set of permissible complements $\lceil\mathsf{roles}\rceil(e)$.[2]

$$\lfloor\mathsf{roles}\rfloor(e) \subseteq \lceil\mathsf{roles}\rceil(e)$$

An optional complement role is in $\lceil\mathsf{roles}\rceil(e)$ but not in $\lfloor\mathsf{roles}\rfloor(e)$. The set $\mathcal{E}$ of lexical entries is then defined as consisting of those AVMs with signature:

$$\begin{bmatrix} \mathsf{cat} & : & 2^{\mathcal{C}} \\ \mathsf{agr} & : & 2^{\mathcal{A}} \\ \lceil\mathsf{roles}\rceil & : & 2^{\mathcal{R}} \\ \lfloor\mathsf{roles}\rfloor & : & 2^{\mathcal{R}} \end{bmatrix}$$

and the lexicon will be regarded as a function $\mathsf{Lex} : \mathcal{S} \to 2^{\mathcal{E}}$ from strings to sets of lexical entries.

4.2 Lexical Nodes

We consider now a set $\mathcal{W}$ of AVMs that we call *lexical nodes*. Each lexical node is intended to correspond to a word in the sentence of which $\mathcal{W}$ is the dependency analysis. Our first condition is that $\mathcal{W}$ must be totally ordered. The total order represents the linearization of the words in the sentence. In Figure 2 this total order is encoded using the integer valued feature index.

Each lexical node w represents a word and is associated with the set of lexical entries obtained for this full form from the lexicon:

$$\mathsf{Entries}(w) = \mathsf{Lex}(\mathsf{string}(w))$$

[2]The notation $\lceil\ \rceil$ conventionally indicates an upper bound and $\lfloor\ \rfloor$ a lower bound.

A lexical node w must *realize* one of its lexical entries. We call it the selected entry for w and denote it by $\mathsf{entry}(w)$.

$$\mathsf{entry}(w) \in \mathsf{Entries}(w)$$

In Section 5 we explicitate the conditions under which w is said to realize its selected entry $\mathsf{entry}(w)$.

4.3 Complement sets

Consider $\rho \in \mathcal{R}$, a complement role type such as `subject`. It is traditionally regarded as a partial function: $\rho(w)$ is defined iff w has a complement of type ρ. An important idea of our approach is to view ρ as a total function: it denotes the set of all complements of w of type ρ. Thus, instead of being undefined $\rho(w)$ may simply denote the empty set.

5 Local Conditions

For each lexical node w, the lexicon supplied a collection of alternative lexical entries. Precisely one must be selected and realized by w. This stipulation induces strong local conditions on w and we explicitate them below.

5.1 Local conditions on complement sets

Let ρ be a complement role type such as `subject` or `np_dat` for dative object. $\rho(w)$ is the set of lexical nodes that are complement of w of type ρ. $\rho(w)$ contains at most one element:

$$\rho(w) \subseteq \mathcal{W} \quad \wedge \quad |\rho(w)| \leq 1$$

If $\rho(w)$ is non-empty, the entry selected for w must be one that permits a complement of type ρ. We write $\mathsf{Entries}(w)|_{\rho}^{\lceil\mathsf{roles}\rceil}$ for the set of lexical entries of w that permit a complement of type ρ:

$$\mathsf{Entries}(w)|_{\rho}^{\lceil\mathsf{roles}\rceil} = \{\ e \in \mathsf{Entries}(w) \ \mid\ \rho \in \lceil\mathsf{roles}\rceil(e)\ \}$$

The condition just stated can be expressed as follows:

$$|\rho(w)| = 1 \quad \Rightarrow \quad \mathsf{entry}(w) \in \mathsf{Entries}(w)|_{\rho}^{\lceil\mathsf{roles}\rceil}$$

Conversely, if the selected entry is one that requires a complement of type ρ, then $\rho(w)$ must be non-empty. We write $\mathsf{Entries}(w)|_{\rho}^{\lfloor\mathsf{roles}\rfloor}$ for the set of lexical entries of w that require a complement of type ρ:

$$\mathsf{Entries}(w)|_{\rho}^{\lfloor\mathsf{roles}\rfloor} = \{\ e \in \mathsf{Entries}(w) \ \mid\ \rho \in \lfloor\mathsf{roles}\rfloor(e)\ \}$$

the converse condition becomes:

$$|\rho(w)| = 1 \quad \Leftarrow \quad \mathsf{entry}(w) \in \mathsf{Entries}(w)|_{\rho}^{\lfloor \mathsf{roles} \rfloor}$$

5.2 Local conditions on category

If c is the category of w then the entry selected for w must be one that permits c. We write $\mathsf{Entries}(w)|_c^{\mathsf{cats}}$ for the set of entries permitting c:

$$\mathsf{Entries}(w)|_c^{\mathsf{cats}} = \{\ e \in \mathsf{Entries}(w) \ \mid\ c \in \mathsf{cats}(e)\ \}$$

the condition is then:

$$\forall c \in \mathcal{C} \qquad c = \mathsf{cat}(w) \quad \Rightarrow \quad \mathsf{entry}(w) \in \mathsf{Entries}(w)|_c^{\mathsf{cats}}$$

Conversely, if e is the entry selected for w then the category of w must be one of those permitted by e:

$$\forall e \in \mathsf{Entries}(w) \qquad e = \mathsf{entry}(w) \quad \Rightarrow \quad \mathsf{cat}(w) \in \mathsf{cats}(e)$$

5.3 Local conditions on agreement

If a is the agreement tuple of w then the entry selected for w must be one that permits a. We write $\mathsf{Entries}(w)|_a^{\mathsf{agrs}}$ for the set of entries permitting a:

$$\mathsf{Entries}(w)|_a^{\mathsf{agrs}} = \{\ e \in \mathsf{Entries}(w) \ \mid\ a \in \mathsf{agrs}(e)\ \}$$

the condition is then:

$$\forall a \in \mathcal{A} \qquad a = \mathsf{agr}(w) \quad \Rightarrow \quad \mathsf{entry}(w) \in \mathsf{Entries}(w)|_a^{\mathsf{agrs}}$$

Conversely, if e is the entry selected for w then the category of w must be one of those permitted by e:

$$\forall e \in \mathsf{Entries}(w) \qquad e = \mathsf{entry}(w) \quad \Rightarrow \quad \mathsf{agr}(w) \in \mathsf{agrs}(e)$$

5.4 Other local conditions

There are other types of local conditions. For example category specific conditions, one example of which is (LP_{n}), on page 13, expressing a restriction on prenominal word order domains. For lack of space, we shall say no more about them.

6 Static Axiomatization of Yields

The notion of *yield* of a lexical node, i.e. the set of lexical nodes reachable through the transitive closure of immediate dominance edges (complements and modifiers), is essential for the expression of grammatical principles. In a generative framework, the yield of a node cannot be calculated until its full dependency tree has been constructed. In this section, we exhibit a static axiomatization of yields that fully exposes the underspecification of a yield to the inference mechanisms.

We write $\mathsf{daughters}(w)$ for the set of immediate complements and modifiers of w:

$$\mathsf{daughters}(w) = \bigcup_{\rho \in \mathcal{R} \cup \mathcal{M}} \rho(w)$$

We are going to compute the yield of w in terms of contributions made to it by all lexical nodes. We write $\mathsf{contrib}(w, w')$ for the contribution of w' to the yield of w. If w' is a daughter of w then it contributes its own yield else nothing:

$$\begin{array}{ll} & \mathsf{contrib}(w, w') \subseteq \mathsf{yield}(w') \\ \wedge & w' \in \mathsf{daughters}(w) \equiv \mathsf{contrib}(w, w') = \mathsf{yield}(w') \\ \wedge & w' \notin \mathsf{daughters}(w) \equiv \mathsf{contrib}(w, w') = \emptyset \end{array}$$

The *strict* yield of w is defined as the set of all lexical nodes that are strictly below w in the dependency tree, and is given by the equation:

$$\mathsf{yield!}(w) = \bigcup_{w' \in \mathcal{W}} \mathsf{contrib}(w, w')$$

The *full* yield of w is defined as the set of all lexical nodes in the dependency tree rooted at w and is obtained by adding w to its own strict yield:

$$\mathsf{yield}(w) = \{w\} \cup \mathsf{yield!}(w)$$

In order to enforce *treeness* and disallow circular dependencies, we simply require that w must not appear in its own strict yield:

$$w \notin \mathsf{yield!}(w)$$

7 Global Conditions

A global condition involves more than just one lexical node. For example, the *head/daugther condition* holds between every two nodes. The *Sentence partitioning condition*, on the other hand, simultaneously involves all nodes.

Head/Daughter conditions. For any two lexical nodes $w, w' \in \mathcal{W}$ and any daughter role $\rho \in \mathcal{R} \cup \mathcal{M}$, either w' is a daughter of w of type ρ or it is not. If it is, it must additionally satisfy the role specific constraint $\mathsf{roleC}(\rho, w, w')$:

$$w' \in \rho(w) \quad \Rightarrow \quad \mathsf{roleC}(\rho, w, w')$$

Among other things, this implication has the effect that $w' \notin \rho(w)$ can be inferred as soon as it is discovered that $\mathsf{roleC}(\rho, w, w')$ is inconsistent, thereby eliminating all models in which $w' \in \rho(w)$.

The role specific constraint is further developed in Section 8.

Head condition. We write $\mathsf{head}(w)$ for the set of lexical nodes that are heads of w. This set contains precisely one element, except in the case of the root node, for which it is empty. We write ROOT for the distinguished lexical node that is the root of the sentence.

$$\mathsf{head}(w) \subseteq \mathcal{W} \quad \wedge \quad |\mathsf{head}(w)| \leq 1$$

$$\mathsf{head}(w) = \emptyset \quad \equiv \quad w = \textsc{root}$$

w is head of w' precisely when w' is a daughter of w:

$$w \in \mathsf{head}(w') \quad \equiv \quad w' \in \mathsf{daughters}(w)$$

Sentence partitioning condition. Each lexical node is the daughter of precisely one head, except for the root which has no head. Thus the sets $\rho(w)$ together with the root form a partition of the sentence:

$$\mathcal{W} = \{\textsc{root}\} \uplus \biguplus_{\substack{\rho \in \mathcal{R} \\ w \in \mathcal{W}}} \rho(w)$$

Root condition. The root must be one of the lexical nodes in the sentence:

$$\textsc{root} \in \mathcal{W}$$

All words must be accounted for; i.e. the yield of the root must contain all lexical nodes:

$$\mathsf{yield}(\textsc{root}) = \mathcal{W}$$

Further, the root must be the main, hence finite, verb:

$$\mathsf{cat}(\textsc{root}) = \mathtt{vfin}$$

The conditions given above capture fully the requirements to be met by the root, but do not expose choices sufficiently to permit effective inference. We know that the root must be picked in $\mathcal{W}$, but the remaining conditions can only be checked after the choice has been made. The *generate and test* trap rears its ugly head again. Instead, we can *unfold* the root conditions for each element of $\mathcal{W}$. In other words: for each $w \in \mathcal{W}$, it is either the root, or it is not:

$$\begin{array}{ll} & w = \textsc{root} \wedge |\mathsf{head}(w)| = 0 \wedge \mathsf{yield}(w) = \mathcal{W} \wedge \mathsf{cat}(w) = \mathtt{vfin} \\ \vee & w \neq \textsc{root} \wedge \mathsf{head}(w)| \neq 0 \end{array}$$

8 Role Specific Constraint

The role specific constraint $\mathsf{roleC}(\rho, w, w')$ may be expressed modularly as a conjunction of clauses:

$$\mathsf{roleC}(\rho, w, w') \quad \equiv \quad C_{\mathtt{det}} \wedge C_{\mathtt{subject}} \wedge C_{\mathtt{adj}} \wedge \ldots$$

We explicitate below some of these clauses. The examples given are intended to be illustrative rather than normative. We extend no claim of linguistic adequacy.

Determiner. The determiner of a noun must agree with its head and occur left-most in the yield of the noun.

$$\begin{array}{lcll} \rho = \mathtt{det} & \Rightarrow & & \mathsf{cat}(w') = \mathtt{det} \\ & & \wedge & \mathsf{agr}(w) = \mathsf{agr}(w') \\ & & \wedge & w' = \mathsf{min}(\mathsf{yield}(w)) \end{array} \qquad (C_{\mathtt{det}})$$

Subject. The subject of a finite verb must be either a noun or a pronoun, it must agree with the verb in person and number, and must have nominative case. We write NOM for the set of agreement tuples with nominative case and pose NP $= \{\mathtt{n}, \mathtt{pro}\}$.

$$\begin{array}{lcll} \rho = \mathtt{subject} & \Rightarrow & & \mathsf{cat}(w') \in \textsc{np} \\ & & \wedge & \mathsf{agr}(w') = \mathsf{agr}(w) \\ & & \wedge & \mathsf{agr}(w') \in \textsc{nom} \end{array} \qquad (C_{\mathtt{subject}})$$

Adjective. An adjective may modify a noun and must agree with it:

$$\rho = \texttt{adj} \quad \Rightarrow \quad \begin{array}{rl} & \mathsf{cat}(w) = \texttt{n} \\ \wedge & \mathsf{cat}(w') = \texttt{adj} \\ \wedge & \mathsf{agr}(w) = \mathsf{agr}(w') \end{array} \qquad (C_{\texttt{adj}})$$

Adjectives must be placed between the determiner (if any) and the noun. Furthermore, and, for simplicity of presentation, ignoring the possibility of PPs, nothing else is allowed to land between the determiner and the noun:

$$\mathsf{cat}(w) = \texttt{n} \quad \Rightarrow \quad \begin{array}{rl} & \mathsf{det}(w) \prec \mathsf{adj}(w) \prec \{w\} \\ \wedge & \mathsf{convex}(\mathsf{det}(w) \cup \mathsf{adj}(w) \cup \{w\}) \end{array} \qquad (LP_{\texttt{n}})$$

In other words: determiners, adjectives and noun occur in this sequence and form a convex set (i.e. without holes, thus forbidding insertions).

9 Conclusion

In this article, we contributed a formalization of grammatical admissibility in the framework of dependency grammar. The set theoretic formulation is both economical and arguably elegant. Also, it has a direct computational reading as a *concurrent constraint* program. This program, thanks to powerful inference support through constraint propagation, is also extremely effective.

Further, we also demonstrated how to produce "static" axiomatizations of notions (such as *yield*) that have traditionally received only "dynamic" treatment, thus permitting their natural involvement in constraint propagation.

Although our presentation focussed on the formalization of principles of immediate dominance, our framework is especially well-suited for the expression of principles of linear precedence. Again, this benefit accrues naturally from the use of set constraints. In particular, the rest of our treatment (not presented here) includes an axiomatization of *topological fields*.

The message we most wish to convey is that one effective way to overcome combinatorial explosion is to take full advantage of axiomatic treatments to obtain powerful inference. The declarative ideals of a grammatical framework need not be compromised; on the contrary, they become a source of efficiency.

We also hope to promote awareness of recent developments in constraint programming. Set constraints, in particular, were shown in [DG99] and here again, to be both a source of new expressivity and of surprising efficiency.

References

[Bla95] Patrick Blackburn. Introduction: Static and dynamic aspects of syntactic structure. *Journal of Logic Language and Information*, 4:1–4, 1995.

[DG99] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In *Proceedings of the 3rd International Workshop on Computational Semantics (IWCS-3)*, Tilburg, 1999.

[Ger95] Carmen Gervet. *Set Intervals in Constraint-Logic Programming: Definition and Implementation of a Language.* PhD thesis, Université de France-Compté, September 1995. European Thesis.

[Hol90] Christian Holzbaur. *Specification of Constraint Based Inference Mechanisms through Extended Unification.* PhD thesis, Technisch-Naturwissenschaftliche Fakultät der Technischen Universität Wien, October 1990.

[LL96] Vincenzo Lombardo and Leonardo Lesmo. An earley-type dependency parser for dependency grammar. In *COLING 96*, pages 723–728, Kyoto, Japan, 1996.

[Mar90] Hiroshi Maruyama. Constraint dependency grammar. Research Report RT0044, IBM Research, Tokyo, March 1990.

[Men98] Wolfgang Menzel. Constraint satisfaction for robust parsing of spoken language. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(1):77–89, 1998.

[MHF83] Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In 21^{st} *Meeting of the Association for Computational Linguistics (ACL'83)*, pages 129–136. Association for Computational Linguistics, 1983.

[MM97] Tobias Müller and Martin Müller. Finite set constraints in Oz. In François Bry, Burkhard Freitag, and Dietmar Seipel, editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München, 17–19 September 1997.

[Moz98] The Mozart Programming System, 1998. Programming Systems Lab, Universität des Saarlandes: http://www.mozart-oz.org/.

[NB97] Peter Neuhaus and Norbert Bröker. The complexity of recognition of linguistically adequate dependency grammars. In *Proceeding of the 35th Annual Meeting of the ACL and the 8th Conference of the EACL*, pages 337–343, Madrid, 1997.

[RVS92] Jim Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Proceedings of the* 30^{th} *Meeting of the Association of Computational Linguistics.* Association for Computational Linguistics, 1992.

[Smo95] Gert Smolka. The Oz Programming Model. In *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343, 1995.

[SRP91] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 333–352, Orlando, Florida, January 21–23, 1991. ACM SIGACT-SIGPLAN, ACM Press. Preliminary report.

[VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517, December 1992.