Axiomatizing Dependency Parsing Using Set Constraints

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

Abstract

We propose a new formulation of dependency grammar and develop a corresponding axiomatization of syntactic well-formedness with a natural reading as a concurrent constraint program. We demonstrate the expressivity and effectiveness of set constraints, and describe a treatment of ambiguity with wide applicability. Further, we provide a constraint programming account of dependent disjunctions that is both simple and efficient and additionally provides the benefits of constructive disjunctions. Our approach was implemented in Oz and yields parsers with very good performance for our currently middle scale grammars. Constraint propagation can be observed to be remarkably effective in pruning the search space.

1 Introduction

Modern linguistic theories such as HPSG (Pollard and Sag, 1987) and LFG (Kaplan and Bresnan, 1982) are primarily concerned with the formulation of general structural principles that determine syntactically well-formed entities, typically represented as hierarchical, possibly typed, feature structures. While feature structures are appealing for their perspicuity and are easily supported by languages with unification, they have the disadvantage that the resulting grammatical formalisms are hard to parse for both theoretical and practical reasons, even becoming undecidable in the worst case (Kaplan and Bresnan, 1982). These difficulties are magnified in languages like German where free word order and discontinuous constituents complicate the formal account and handicap parsing techniques relying on surface order.

While much research was devoted to the compilation of grammatical frameworks based on typed feature structures and on the compact representation and efficient processing of disjunctive feature structures, recent advances in constraint technology have received rather less attention in the computational linguistics (CL) community than they deserve, and their relevance has gone largely unnoticed. Concurrent constraint programming provides today far greater expressivity and efficiency than is commonly assumed.

In particular, set constraints are emerging as an especially elegant and computationally effective tool for applications in CL. In an earlier paper (Duchier and Gardent, 1999), we described how they could serve to efficiently solve dominance constraints and find minimal models of tree descriptions. We propose here to show their application to parsing.

In this paper, we are going to demonstrate how, with the help of state-of-the-art constraint programming, an elegant and concise axiomatic specification of syntactic well-formedness becomes naturally an efficient program. In so doing, we develop a treatment of ambiguity with fairly general applicability. In particular, we provide a constraint-based account of dependent disjunctions (Maxwell and Kaplan, 1989; Dörre and Eisele, 1990; Gerdemann, 1991; Griffith, 1990) that naturally supports constructive disjunction semantics (lifting of common information).

Our approach stands in sharp contrast to most extant parsing techniques. We abandon the generative view; we no longer build partial parses by combination of smaller ones, as is the case in e.g. chart parsing. Rather, we give a global well-formedness condition and proceed to enumerate its models.

We choose dependency grammar (DG) as our framework and axiomatize the notion of a syntactically well-formed dependency tree in a manner that is particularly well-suited to constraint-based processing: we propose to regard dependency parsing as a finite configuration problem that can be formulated as a constraint satisfaction problem (CSP).

This view takes advantage of the fact that for a sentence of length n, there are finitely many possible trees involving just n nodes. Out of this large number, we must select those that are grammatical. We do not attempt this through explorative generation, but rather via model elimination. What constraint programming affords us is effective model elimination through constraint propagation.

Maruyama (1990) was the first to propose a complete treatment of dependency grammar as a CSP and described parsing as a process of incremental disambiguation. Harper (Harper et al., 1999) continues this line of research and has proposed several algorithmic improvements within the MUSE CSP framework (Helzerman and Harper, 1993). Menzel (Menzel, 1998; Heinecke et al., 1998; Menzel and Schröder, 1998) advocates the use of soft "graded" constraints for robustness e.g. in parsing spoken language. His proposal turns parsing into a more expensive optimization problem, but adapts gracefully to constraint violations.

Our presentation has the advantage over Maruyama's that it follows modern linguistic practice: the grammar is specified by a lexicon and a collection of principles. Further, we illustrate the expressiveness of set constraints and selection constraints, and demonstrate how they can provide compact and elegant encodings of various forms of ambiguity such as lexical or attachment ambiguity. Finally, our axiomatization of dependency parsing has the property that, modulo details of syntax, it can also be regarded as a program in a concurrent constraint programming language such as Oz (Smolka, 1995). Several parsers were implemented in Oz as described and provide excellent performance, without any sort of optimization.

Consider the sentence[1] below which illustrates the sort of non-projective analysis with fronting, scrambling and extraposition that is typical of German sentences.

das	Buch	hat	mir	Peter	versprochen	zu	lesen	
the	*book*	*has*	*me*(dat)	*Peter*	*promised*	*to*	*read*	(1)

Since both "das Buch" and "Peter" can be indifferently assigned nominative or accusative case, either one may be subject of "hat" while the other is accusative object of "lesen." There are thus two readings.

(Fig 1) demonstrates the effectiveness of constraint propagation: the two readings are enumerated in approximately 190ms using a single choice point.[2] A graphical representation of the preferred reading is shown in the upper window, while the search tree is displayed in the lower window.

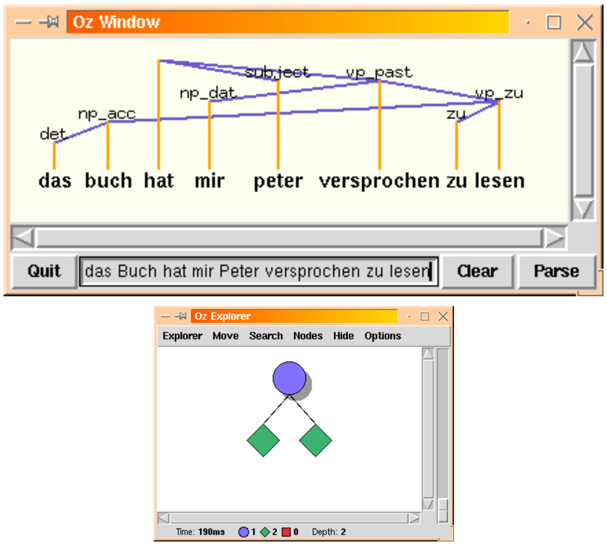


Figure 1: Parser Demo

In our approach, constraint propagation alone constructs the dependency tree. Search is only needed to explore alternatives that cannot be ruled out by constraint propagation. In practice, we observe that search is required to enumerate readings, but very rarely leads to failure where e.g. backtracking is required.

Section 2 introduces the formal framework; Section 3 presents the novel constraint programming ideas such as set and selection constraints;

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

[2] We expect better performance when e.g. the selection constraint is given low-level builtin support. It is currently implemented in Oz itself.

and in Section 4 we axiomatize a constraint model for syntactic well-formedness in DG and turn the problem into a CSP.

2 Formal Framework

We now present our notion of a dependency grammar and of dependency trees. The formulation below ignores issues of word order as they are beyond the scope of this article. However, the full treatment includes ideas derived from Reape's word order domains (Reape, 1994) as well as from the theory of topological fields in German sentences.

2.1 Dependency Grammar

In our formal setting, a dependency grammar G is a 7-tuple

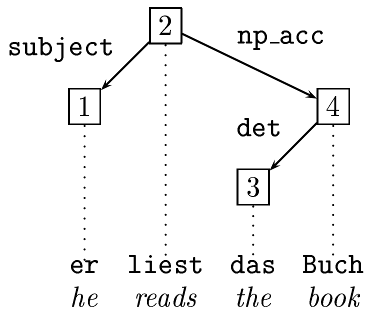
$$\langle \mathit{Words}, \mathit{Cats}, \mathit{Agrs}, \mathit{Comps}, \mathit{Mods}, \mathit{Lexicon}, \mathit{Rules}\rangle$$

where *Words* is a finite set of strings notating the fully inflected forms of words, *Cats* is a finite set of categories such as `n` for noun, `det` for determiner, or `vfin` for finite verb, *Agrs* is a finite set of agreement tuples such as ⟨`masc sing 3 nom`⟩, *Comps* is a finite set of complement role types such as `subject` or `np_dat` for dative noun phrase, *Mods* is a finite set of modifier role types, such as `adj` for adjectives, disjoint from *Comps*. We write $\mathit{Roles} = \mathit{Comps} \uplus \mathit{Mods}$ for the set of all role types; they will serve to label the edges of a dependency tree. *Lexicon* is a finite set of lexical entries (see below), and *Rules* is a family of binary predicates, indexed by role labels, expressing local grammatical principles: for each $\rho \in \mathit{Roles}$, there is $\Gamma_\rho \in \mathit{Rules}$ such that $\Gamma_\rho(w_1, w_2)$ characterizes the grammatical admissibility of an edge labeled ρ from mother w_1 to daughter w_2.

A lexical entry is an attribute value matrix (AVM) with signature:

$$\begin{bmatrix} \mathsf{string} & : & \mathit{Words} \\ \mathsf{cat} & : & \mathit{Cats} \\ \mathsf{agr} & : & \mathit{Agrs} \\ \mathsf{roles} & : & 2^{\mathit{Comps}} \end{bmatrix}$$

We write attribute access in functional notation. If e is a lexical entry: $\mathsf{string}(e)$ is the full form of the corresponding word, $\mathsf{cat}(e)$ is the category, $\mathsf{agr}(e)$ the agreement, and $\mathsf{roles}(e)$ the valency expressed as a set of complement roles.



$$\boxed{1}\begin{bmatrix} \mathsf{string} & : & \texttt{er} \\ \mathsf{cat} & : & \texttt{pro} \\ \mathsf{agr} & : & \langle\texttt{masc sing 3 nom}\rangle \\ \mathsf{roles} & : & \{\} \end{bmatrix}$$

$$\boxed{2}\begin{bmatrix} \mathsf{string} & : & \texttt{liest} \\ \mathsf{cat} & : & \texttt{vfin} \\ \mathsf{agr} & : & \langle\texttt{masc sing 3 nom}\rangle \\ \mathsf{roles} & : & \{\texttt{subject},\texttt{np\_acc}\} \end{bmatrix}$$

$$\boxed{3}\begin{bmatrix} \mathsf{string} & : & \texttt{das} \\ \mathsf{cat} & : & \texttt{det} \\ \mathsf{agr} & : & \langle\texttt{neut sing 3 acc}\rangle \\ \mathsf{roles} & : & \{\} \end{bmatrix}$$

$$\boxed{4}\begin{bmatrix} \mathsf{string} & : & \texttt{Buch} \\ \mathsf{cat} & : & \texttt{n} \\ \mathsf{agr} & : & \langle\texttt{neut sing 3 acc}\rangle \\ \mathsf{roles} & : & \{\texttt{det}\} \end{bmatrix}$$

Figure 2: Example Dependency Tree

2.2 Dependency Trees

We assume an infinite set *Nodes* of nodes and define a labeled directed edge to be an element of $\mathit{Nodes} \times \mathit{Nodes} \times \mathit{Roles}$. Thus, given a set $\mathcal{V} \subseteq \mathit{Nodes}$ of nodes and a set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathit{Roles}$ of labeled edges between these node, $\langle \mathcal{V}, \mathcal{E}\rangle$ is a directed graph, in the classical sense, with labeled edges. We will restrict our attention to finite graphs that are also trees.

Every node of a dependency tree contributes a word to the sentence; the position where that word is contributed will be represented by a mapping index from nodes to integers. Further, every node must be assigned syntactic features (e.g. case, agreement ...), which will be realized by a mapping entry from nodes to lexical entries.

A dependency tree T is then defined as a 4-tuple:

$$\langle \mathcal{V}, \mathcal{E}, \mathsf{index}, \mathsf{entry}\rangle$$

where $\mathcal{V}$ is a finite set $\{w_1, \ldots, w_n\}$ of nodes, $\mathcal{E}$

is a finite set of labeled edges $w_i \xrightarrow{\rho} w_j$ between elements of $\mathcal{V}$, and $\langle \mathcal{V}, \mathcal{E} \rangle$ is a tree as defined above.

index is a bijection from $\mathcal{V}$ to $[1..n]$ assigning to each node a linear position in the corresponding sentence, and $\mathsf{entry} : \mathcal{V} \mapsto \mathit{Lexicon}$ assigns a lexical entry to each node in $\mathcal{V}$.

Example. Figure 2 contains a graphical depiction of a dependency tree. The upper part shows nodes represented as boxes connected by labeled directed edges. The integer appearing in each box is the position which index assigns to the node. For ease of reading, the linearization of the sentence is also provided and a dotted line indicates for each node the corresponding word in the sentence. The lower part of the figure displays for each node the lexical entry which entry assigns to it.

Figure 3 shows the dependency tree for example sentence (1). In places where the value assignment is fully unconstrained (i.e. all assignments are acceptable), we have used the standard "don't care" notation '_'.

Well-Formedness. A dependency tree is grammatically admissible iff conditions (2), (3) and (4) below are satisfied. First, any complement required by a node's valency must be realized precisely once:

$$\begin{array}{l}\forall w_i \in \mathcal{V}, \forall \rho \in \mathsf{roles}(\mathsf{entry}(w_i)) \\ \qquad \exists! w_j \in \mathcal{V}, w_i \xrightarrow{\rho} w_j \in \mathcal{E}\end{array} \tag{2}$$

Second, if there is an edge emanating from w_i, then it must be labeled either by a complement type in w_i's valency or by a modifier type:

$$\begin{array}{l}\forall w_i \xrightarrow{\rho} w_j \in \mathcal{E} \\ \qquad \rho \in \mathsf{roles}(\mathsf{entry}(w_i)) \cup \mathit{Mods}\end{array} \tag{3}$$

Third, whenever there is an edge $w_i \xrightarrow{\rho} w_j$, then the grammatical condition $\Gamma_\rho(w_i, w_j)$ for $\Gamma_\rho \in \mathit{Rules}$ must be satisfied in T:

$$\forall w_i \xrightarrow{\rho} w_j \in \mathcal{E} \quad T \models \Gamma_\rho(w_i, w_j) \tag{4}$$

2.3 Improving Lexical Economy

Typically, the same full form of a word may correspond to several distinct agreements. In the interest of a more compact representation, we wish to collapse together entries that differ only in their agreement. We replace attribute agr by agrs whose values are now sets of agreement tuples. Although of less frequent applicability, we do the same for categories and replace cat by cats.

Optional complements are another source of considerable redundancy in the lexicon. Instead of modeling the valency with just one set $\mathsf{roles}(e)$ of complement types, we propose to use instead a lower bound $\lfloor\mathsf{roles}\rfloor(e)$ and an upper bound $\lceil\mathsf{roles}\rceil(e)$, such that $\lfloor\mathsf{roles}\rfloor(e) \subseteq \lceil\mathsf{roles}\rceil(e)$. $\lfloor\mathsf{roles}\rfloor(e)$ represents the required roles and $\lceil\mathsf{roles}\rceil(e)$ the permissible roles. Optional roles are simply $\lceil\mathsf{roles}\rceil(e) \setminus \lfloor\mathsf{roles}\rfloor(e)$.

We call the new, more compact representation a *lexicon* entry to distinguish it from a *lexical* entry which remains as defined earlier. A lexicon entry is said to generate lexical entries. The lexical entries generated by (5) below are of the form (6) and correspond to all solutions of constraint (7).

$$\begin{bmatrix} \mathsf{string} & : & S \\ \mathsf{cats} & : & C \\ \mathsf{agrs} & : & A \\ \lfloor\mathsf{roles}\rfloor & : & R_{\mathrm{lo}} \\ \lceil\mathsf{roles}\rceil & : & R_{\mathrm{hi}} \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} \mathsf{string} & : & S \\ \mathsf{cat} & : & c \\ \mathsf{agr} & : & a \\ \mathsf{roles} & : & r \end{bmatrix} \tag{6}$$

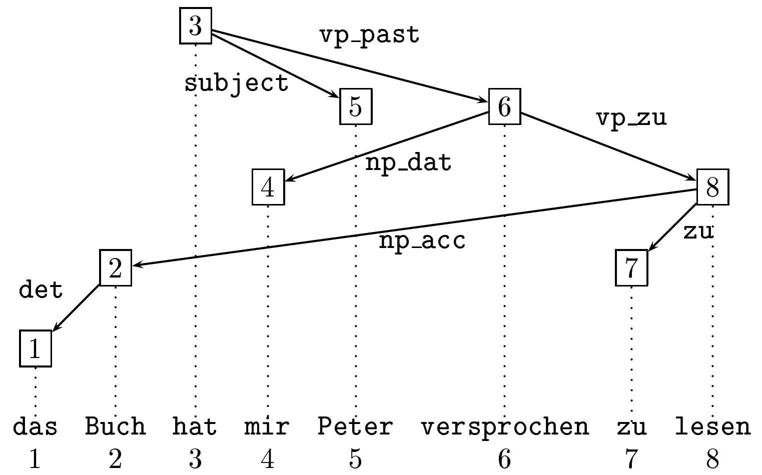
$$c \in C \;\wedge\; a \in A \;\wedge\; R_{\mathrm{lo}} \subseteq r \subseteq R_{\mathrm{hi}} \tag{7}$$

This simple formulation illustrates how constraints can be used to produce compact representations of certain forms of lexical ambiguity.

3 Constraint Programming

The foundations of concurrent constraint programming (CCP) are well documented e.g. in (Saraswat et al., 1991; Smolka, 1995) and have been implemented in the programming language Oz (Mozart Consortium, 1998).

Modern constraint technology, as provided by CHIP (Dincbas et al., 1988), clp(FD) (Codognet and Diaz, 1996), ECLiPSe (Aggoun et al., 1995), ILOG Solver (ILOG, 1996), and Oz (Mozart Consortium, 1998) has proven quite successful at solving practical problems with high combinatorial complexity, such as scheduling and configuration, that were resisting traditional methods of operations research.



$$\boxed{1}\begin{bmatrix} \textsf{string} & : & \texttt{das} \\ \textsf{cat} & : & \texttt{det} \\ \textsf{agr} & : & \langle\texttt{neut sing 3 acc}\rangle \\ \textsf{roles} & : & \{\} \end{bmatrix} \quad \boxed{2}\begin{bmatrix} \textsf{string} & : & \texttt{Buch} \\ \textsf{cat} & : & \texttt{n} \\ \textsf{agr} & : & \langle\texttt{neut sing 3 acc}\rangle \\ \textsf{roles} & : & \{\texttt{det}\} \end{bmatrix}$$

$$\boxed{3}\begin{bmatrix} \textsf{string} & : & \texttt{hat} \\ \textsf{cat} & : & \texttt{vfin} \\ \textsf{agr} & : & \langle\texttt{masc sing 3 nom}\rangle \\ \textsf{roles} & : & \{\texttt{subject,vp\_past}\} \end{bmatrix} \quad \boxed{4}\begin{bmatrix} \textsf{string} & : & \texttt{mir} \\ \textsf{cat} & : & \texttt{pro} \\ \textsf{agr} & : & \langle\texttt{\_ sing 3 dat}\rangle \\ \textsf{roles} & : & \{\} \end{bmatrix}$$

$$\boxed{5}\begin{bmatrix} \textsf{string} & : & \texttt{Peter} \\ \textsf{cat} & : & \texttt{n} \\ \textsf{agr} & : & \langle\texttt{masc sing 3 nom}\rangle \\ \textsf{roles} & : & \{\} \end{bmatrix} \quad \boxed{6}\begin{bmatrix} \textsf{string} & : & \texttt{versprochen} \\ \textsf{cat} & : & \texttt{vpast} \\ \textsf{agr} & : & \texttt{\_} \\ \textsf{roles} & : & \{\texttt{np\_dat,vp\_zu}\} \end{bmatrix}$$

$$\boxed{7}\begin{bmatrix} \textsf{string} & : & \texttt{zu} \\ \textsf{cat} & : & \texttt{part} \\ \textsf{agr} & : & \texttt{\_} \\ \textsf{roles} & : & \{\} \end{bmatrix} \quad \boxed{8}\begin{bmatrix} \textsf{string} & : & \texttt{lesen} \\ \textsf{cat} & : & \texttt{vinf} \\ \textsf{agr} & : & \texttt{\_} \\ \textsf{roles} & : & \{\texttt{zu,np\_acc}\} \end{bmatrix}$$

Figure 3: Dependency tree of demo sentence

The key to success, presented here, is to reduce parsing to a problem that constraint programming (CP) is good at, namely configuration.

3.1 Basic Notions

A CSP consists of a finite collection of variables (x_i) taking values in finite domains (D_i) and a constraint ϕ on these variables. A solution is an assignment σ of values to these variables such that ϕ is satisfied. In CP, σ is computed incrementally as a sequence of improving approximations: $\sigma(x_i)$ is an approximation of the assignment to x_i and is typically specified either by listing the remaining values or by providing upper and lower bounds. Initially the approximation of x_i contains all of D_i. Solutions are derived by a search process consisting of alternating *propagation* and *distribution* steps until all variables are determined or a contradiction is derived. A variable x_i is said to be determined when its approximation $\sigma(x_i)$ has been reduced to a single value.

In CCP, σ is called the store and primitive constraints are implemented by concurrent agents that observe the store and correspondingly improve the current approximations by deriving new *basic constraints* according to their declarative semantics. Basic constraints are e.g. $I = 5$, $I \neq 5$, $5 \in S$ or $5 \notin S$.

Propagation. This is a process of deterministic inference. It removes from every approximation $\sigma(x_i)$ all values that can be inferred to be inconsistent with ϕ. In practice only a cheap local inference process is applied such as "arc consistency."

Distribution. When propagation has reached a fixed point but not all variables are determined, search becomes necessary to e.g. enumerate the possible assignments of a non-determined variable x. Which variable to pick and how to enumerate the values is expressed by a distribution (or labeling) strategy. For example the "first-fail" strategy picks a variable that has fewest remaining possible values in its domain; its intended effect is to keep the branching factor in the search tree as small as possible.

3.2 Set Constraints

Modern constraint technology not only supports finite domain variables, denoting integers, but also finite set variables, denoting finite sets of integers (Gervet, 1995; Müller and Müller, 1997). One of our contributions is to demonstrate the elegance, succinctness and efficiency accruing from the use of sets and set constraints.

A set variable S is approximated by a lower bound $\lfloor S \rfloor$ and an upper bound $\lceil S \rceil$:

$$\lfloor S \rfloor \subseteq S \subseteq \lceil S \rceil$$

These bounds can be tightened by constraint propagation. All the usual operations of set theory are supported as constraints. In particular, we often use $S = S_1 \uplus \ldots \uplus S_n$, where $\uplus$ denotes disjoint union, to state that (S_i) forms a partition of S.

Our formulation illustrates many applications of sets. Of special interest is the treatment of attachment ambiguity (or more generally of ambiguous graph connectivity): we use set variables to represent sets of daughters for each node in a tree.

3.3 Selection Constraint

An essential contribution of this paper is a general technique for the compact representation and effective treatment of ambiguity such as lexical ambiguity. Consider a variable X that may be equated with one of n variables (V_i). We represent the choice explicitly using a finite domain variable $I \in \{1..n\}$ and the *selection constraint*:

$$X = \langle V_1, \ldots, V_n \rangle[I] \tag{8}$$

The notation $\langle V_1, \ldots, V_n \rangle[I]$ was chosen to suggest selection of the Ith element of sequence $\langle V_1, \ldots, V_n \rangle$.

The idea is that the constrained value of X can be approximated from the set of variables that may still be selected by I from $\langle V_1, \ldots, V_n \rangle$. Conversely: the remaining domain of I must be restricted to the positions for which V_i is still compatible with X.

This powerful idea was first introduced in CHIP (Dincbas et al., 1988) for finite domain (FD) variables under the name of 'element' constraint. We extend it here to finite set (FS) variables.

The selection constraint can be implemented efficiently and we outline the intuition in the case where X and (V_i) are set variables. We write $\lfloor X \rfloor$ and $\lceil X \rceil$ for the lower and upper approximations of X and $\mathsf{dom}(I)$ for the approximation of I. Given the selection constraint (8), propagation maintains the following invariant:

$$\bigcap_{j \in \mathsf{dom}(I)} \lfloor V_j \rfloor \subseteq \lfloor X \rfloor \subseteq X \subseteq \lceil X \rceil \subseteq \bigcup_{j \in \mathsf{dom}(I)} \lceil V_j \rceil$$

and applies the following rule of inference:

$$\lfloor V_j \rfloor \not\subseteq \lceil X \rceil \ \vee \ \lfloor X \rfloor \not\subseteq \lceil V_j \rceil \quad \Rightarrow \quad I \neq j$$

As a consequence of the above invariant, the selection constraint implements a form of constructive disjunction (lifting of information common to all remaining alternatives). The fact that the choice is made explicit by variable I permits dependent selections. For example in (9) the choice of which V_i to equate with X and which W_i to equate with Y are mutually depend:

$$\begin{aligned} X &= \langle V_1, \ldots, V_n \rangle[I] \\ Y &= \langle W_1, \ldots, W_n \rangle[I] \end{aligned} \tag{9}$$

These two constraints can be viewed as realizing the following dependent (or named) disjunctions (Maxwell and Kaplan, 1989; Dörre and Eisele, 1990; Gerdemann, 1991; Griffith, 1990) both labeled with name I:

$$\begin{aligned} &(X = V_1 \quad \vee \ldots \vee \quad X = V_n)_I \\ &(Y = W_1 \quad \vee \ldots \vee \quad Y = W_n)_I \end{aligned}$$

Notational variations on dependent disjunction have been used to concisely express covariant assignment of values to different features in feature structures. The selection constraint provides the same notational convenience and declarative semantics, but also gives it all the computational benefits that accrue from state-of-the-art constraint technology.

4 Constraint Model

In this section, we develop a constraint model for the formal framework of Section 2: we introduce FD and FS variables to encode the quantities and mappings it mentions, and formulate constraints on them that precisely capture the conditions the formal model stipulates. What motivates this reduction is the desire to take advantage of the powerful and very efficient technological support for constraint propagation on finite domains and finite sets.

For our application to parsing, we assume that we are given an input sentence consisting of n words $s_1\, s_2\, \ldots\, s_n$. The solutions to the CSP obtained as its constraint model correspond to all admissible parse trees of the sentence.

Our approach takes advantage of the following observations: (1) there are finitely many edges labeled with *Roles* that can be drawn between n nodes,[3] (2) for each word there are finitely many lexical entries. Thus the problem is to pick a set of edges and, for each node, to pick a lexical entry so that (a) the result is a tree, (b) none of the grammatical conditions are violated. Viewed in this light, dependency parsing reduces to a configuration problem.

4.1 Representation

Nodes and Lexical Attributes: we identify a node with the integer representing its position in the sentence; thus index is simply the identity function. The lexical entry assigned to each node w is represented by its attributes. We write $\mathsf{cat}(w)$, $\mathsf{agr}(w)$, and $\mathsf{roles}(w)$ for the variables denoting their values.

Domains: each category in *Cats* is encoded by a distinct integer. Similarly for each agreement tuple in *Agrs*[4] and each role in *Roles*. Thus every value is represented either by an integer or a set of integers.

Daughter Sets: for each node $w \in \mathcal{V}$ and each role $\rho \in \mathit{Roles}$, we write $\rho(w)$ for the set of immediate daughters of w whose dependency edge is labeled ρ, i.e. $\rho(w) = \{w' \mid w \xrightarrow{\rho} w' \in \mathcal{E}\}$. Thus, if there is no edge labeled ρ emanating from w, $\rho(w)$ is the empty set. Since $\rho(w)$ is a subset of $\mathcal{V}$, i.e. a finite of set of integers in the constraint model, we represent it by a FS variable.

Lexicon: we consider the function Lex from words to sets of lexicon entries.

$$\mathsf{Lex}(s) = \{e \in \mathit{Lexicon} \mid \mathsf{string}(e) = s\}$$

Without loss of generality, we can assume that Lex returns a sequence rather than a set which allows us to identify the lexicon entries of a given word by position in this sequence.

The lexical entry assigned to w is generated by one of its lexicon entries as described in Section 2.3. We say that the latter is "selected" and introduce variable $\mathsf{entryIndex}(w)$ to denote its position in the sequence returned by Lex for w.

4.2 Lexical Constraints

We now explicate the assignment of lexical attributes to a node w and demonstrate how lexical ambiguity can be axiomatized by reduction to the selection constraint.

Lexical attribute assignment proceeds in two steps: (a) selection of a lexicon entry (b) generation of a lexical entry from this lexicon entry as described in Section 2.3. Let us write E for the lexicon entry selected for w and I for its position in the sequence returned by Lex. They are abstractly defined by the following equations:

$$\begin{array}{rcl} \langle e_1, \ldots, e_n \rangle & = & \mathsf{Lex}(\mathsf{string}(w)) \\ I & = & \mathsf{entryIndex}(w) \\ E & = & \langle e_1, \ldots, e_n \rangle[I] \end{array}$$

the lexical attributes are obtained as solutions of formula (7) which translates into the following constraints:

$$\begin{array}{c} \mathsf{cat}(w) \in \mathsf{cats}(E) \\ \mathsf{agr}(w) \in \mathsf{agrs}(E) \\ \lfloor\mathsf{roles}\rfloor(E) \subseteq \mathsf{roles}(w) \subseteq \lceil\mathsf{roles}\rceil(E) \end{array}$$

For practical reasons of implementation, the selection constraint is only provided for finite domains and finite sets, but E and (e_i) are AVMs. We overcome this difficulty by pushing attribute

[3] $|\mathcal{V} \times \mathcal{V} \times \mathit{Roles}| = n^2\,|\mathit{Roles}|$

[4] In German, there are 72 distinct agreement tuples: 4 cases × 3 genders × 3 persons × 2 numbers

access into the selection:

$$\begin{aligned} \mathsf{cat}(w) &\in \langle \mathsf{cats}(e_1), \ldots, \mathsf{cats}(e_n) \rangle [I] \\ \mathsf{agr}(w) &\in \langle \mathsf{agrs}(e_1), \ldots, \mathsf{agrs}(e_n) \rangle [I] \\ \langle \lfloor \mathsf{roles} \rfloor (e_1), \ldots, \lfloor \mathsf{roles} \rfloor (e_n) \rangle [I] &\subseteq \mathsf{roles}(w) \\ \langle \lceil \mathsf{roles} \rceil (e_1), \ldots, \lceil \mathsf{roles} \rceil (e_n) \rangle [I] &\supseteq \mathsf{roles}(w) \end{aligned}$$

4.3 Valency Constraints

Every daughter set $\rho(w)$ is a finite set of nodes occurring in the tree:

$$\forall w \in \mathcal{V}\ \forall \rho \in \mathit{Roles}\quad \rho(w) \subseteq \mathcal{V}$$

A complement daughter set $\rho(w)$ is of cardinality at most 1 (e.g. a verb has at most 1 subject), and it is non-empty iff ρ appears in w's valency. We write $|\rho(w)|$ for $\rho(w)$'s cardinality.

$$\begin{aligned} &\forall \rho \in \mathit{Comps} \\ &\qquad 0 \leq |\rho(w)| \leq 1 \\ &\quad \wedge \quad |\rho(w)| = 1 \ \equiv\ \rho \in \mathsf{roles}(w) \end{aligned}$$

A modifier daughter set has no cardinality restriction (e.g. a noun can have any number of adjectives).

4.4 Role Constraints

For each role type $\rho \in \mathit{Roles}$ our grammar specifies a binary predicate Γ_ρ expressing a grammatical condition between mother and daughter. We assume that $\Gamma_\rho(w, w')$ is defined by a formula of a constraint language that can be interpreted over dependency trees. We will not make this language explicit here, but detail below a few examples. These examples are intended to be illustrative rather than normative and we extend for them no claim of linguistic adequacy.

According to well-formedness condition (4), for every edge $w \xrightarrow{\rho} w'$, grammatical condition $\Gamma_\rho(w, w')$ must hold. Therefore the tree must satisfy the following condition:

$$\forall w, w' \in \mathcal{V}\ \forall \rho \in \mathit{Roles}\quad w' \in \rho(w) \Rightarrow \Gamma_\rho(w, w')$$

Modern constraint technology has efficient support for such implicative conditions.[5] In particular, if constraint propagation can show that $\Gamma_\rho(w, w')$ is inconsistent in T, then $w' \notin \rho(w)$ is inferred which improves the approximation of $\rho(w)$.

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:

$$\begin{aligned} \Gamma_{\texttt{subject}}(w, w') \equiv \quad & \mathsf{cat}(w') \in \{\texttt{n}, \texttt{pro}\} \\ \wedge \quad & \mathsf{agr}(w) = \mathsf{agr}(w') \\ \wedge \quad & \mathsf{agr}(w') \in \text{NOM} \end{aligned} \tag{10}$$

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

$$\begin{aligned} \Gamma_{\texttt{adj}}(w, w') \equiv \quad & \mathsf{cat}(w) = \texttt{n} \\ \wedge \quad & \mathsf{cat}(w') = \texttt{adj} \\ \wedge \quad & \mathsf{agr}(w) = \mathsf{agr}(w') \end{aligned} \tag{11}$$

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

$$\begin{aligned} \Gamma_{\texttt{det}}(w, w') \equiv \quad & \mathsf{cat}(w') = \texttt{det} \\ \wedge \quad & \mathsf{agr}(w) = \mathsf{agr}(w') \\ \wedge \quad & w' = \mathsf{min}(\mathsf{yield}(w)) \end{aligned} \tag{12}$$

The yield of w is the set of nodes (including w) reachable from w by traversing downwards any number of dependency edges. We introduce variable $\mathsf{yield}(w)$ for this quantity and develop, in Section 4.6, its constraint-based axiomatization. Since $\mathsf{yield}(w)$ contains w, it is a non-empty set of nodes i.e. integers (Section 4.1). Thus it is meaningful to speak of its minimum element, which we write $\mathsf{min}(\mathsf{yield}(w))$.[6]

4.5 Treeness Constraints

Our formal framework simply assumed the "usual" definition of treeness: (a) every node has a unique mother except for a distinguished node, called the root, which has none, (b) there are no cycles. We must now provide an explicit axiomatization of this notion. For this purpose we introduce variables $\mathsf{daughters}(w)$ and $\mathsf{mother}(w)$ for each node w.

The set of immediate daughters of w is simply defined as the union of its daughter sets:

$$\mathsf{daughters}(w) = \bigcup_{\rho \in \mathit{Roles}} \rho(w)$$

[5] In Oz, this is expressed by the construct: **or** $w' \in \rho(w) \wedge \Gamma_\rho(w, w')$ [] $w' \notin \rho(w)$ **end**

[6] Oz supports constraints of the form $I = \mathsf{min}(S)$ between a finite domain variable I and a finite set variable S.

We model the notion of 'mother' of a node as a set of cardinality at most 1. Thus, we can account for both the presence or absence of a mother. The latter case is needed only for the root of the dependency tree.

$$\mathsf{mother}(w) \subseteq \mathcal{V} \quad 0 \leq |\mathsf{mother}(w)| \leq 1$$

w is a mother of w' iff w' is an immediate daughter of w:

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

Further, treeness requires the existence of a unique root. We therefore introduce the new variable ROOT to denote the root of the dependency tree. This is the only node without a mother:

$$\begin{array}{c} \textsc{root} \in \mathcal{V} \\ \forall w \in \mathcal{V} \ \ w = \textsc{root} \ \equiv \ |\mathsf{mother}(w)| = 0 \end{array}$$

Each node must fill precisely one role in the sentence; it must either be the root or an immediate daughter of another node:

$$\mathcal{V} = \{\textsc{root}\} \uplus \biguplus_{\substack{w \in \mathcal{V} \\ \rho \in \mathit{Roles}}} \rho(w)$$

In order to guarantee well-formedness, we must additionally enforce acyclicity. We do this below in the axiomatization of yields.

4.6 Axiomatization of Yield

The notion of *yield* of a lexical node, i.e. the set of 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 the full dependency tree rooted at this node 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 of constraint propagation.

Weighted Set. as a preliminary, we introduce the *weighted set* constraint, between boolean variable B and FS variables $S1, S2$:

$$S_1 = B \star S_2$$

which has the declarative semantics:

$$\begin{array}{rcl} B & \Rightarrow & S_1 = S_2 \\ \neg B & \Rightarrow & S_1 = \emptyset \end{array}$$

Note that, if we make the classical identification of false with 0 and true with 1, we can also express it with the selection constraint:

$$S_1 = B \star S_2 \quad \equiv \quad S_1 = \langle \emptyset, S_2 \rangle [B+1]$$

The strict yield $\mathsf{yield!}(w)$ is the set of nodes strictly below w in the dependency tree. In other words, it is formed by the disjoint union of the yields of its immediate daughters:

$$\mathsf{yield!}(w) = \biguplus_{w' \in \mathsf{daughters}(w)} \mathsf{yield}(w')$$

But $\mathsf{daughters}(w)$ is not statically known; so, instead, we reformulate the above as a union of weighted sets:

$$\mathsf{yield!}(w) = \biguplus_{w' \in \mathcal{V}} (w' \mathrel{\ddot{\in}} \mathsf{daughters}(w)) \star \mathsf{yield}(w')$$

where $w' \mathrel{\ddot{\in}} \mathsf{daughters}(w)$ is a "reified" constraint: it denotes a boolean which is true iff $w' \in \mathsf{daughters}(w)$ is satisfied. To obtain the yield of w, we simply add w to its strict yield:

$$\mathsf{yield}(w) = \{w\} \uplus \mathsf{yield!}(w) \tag{13}$$

Disjoint union in (13) enforces the condition that w must not appear in its own strict yield, thus ruling out loops.

4.7 Word-order constraints

Although the treatment of word-order constraints lies well beyond the scope of this article, we give here an idea of how they may be accommodated. The technique exploits powerful constraints on sets, such as "sequentiality" and "convexity" (no holes).

Consider that adjectives must be placed between the determiner (if any) and the noun, and, for simplicity of presentation, ignoring the possibility of PPs, nothing else is allowed to land between the determiner and the noun. That constraint can be expressed as follows:

$$\begin{array}{rl} & \mathsf{Seq}(\mathtt{det}(w), \mathtt{adj}(w), \{w\}) \\ \wedge & \mathsf{Convex}(\mathtt{det}(w) \cup \mathtt{adj}(w) \cup \{w\}) \end{array}$$

where $\mathsf{Seq}(S_1, \ldots, S_m)$ is satisfied whenever for all $i < j$, all elements of S_i are strictly smaller than all elements of S_j, and $\mathsf{Convex}(S)$ when S is an interval (with no holes).

4.8 Creating and Solving the CSP

The CSP for a given sentence is defined by the variables introduced above and by the conjunction of all constraints presented in the preceding sections. It is important to notice that all quantification is of the form $\forall x \in D\ \phi(x)$ where D is a finite statically known set of integers. Such formulae are expanded in the CSP into $\bigwedge_{x \in D} \phi(x)$.

To solve the CSP, we need a "labeling" strategy. We have only experimented with the following obvious one: first apply the default naïve labeling strategy on the collection of mother sets $\{\mathsf{mother}(w) \mid w \in \mathcal{V}\}$, then apply first-fail to the collection of lexicon entry selectors $\{\mathsf{entryIndex}(w) \mid w \in \mathcal{V}\}$, and finally use again the default naïve labeling strategy on the collection of daughter sets $\{\rho(w) \mid \rho \in \mathit{Roles}\ w \in \mathcal{V}\}$.

5 Results

We developed several prototype parsers, for both German and English, using the techniques described in this paper. In addition to what has been presented, we also support PPs (however, in the case of ambiguous attachments, we explicitly enumerate all possibilities), relative clauses, infinitive clauses, separable verb prefixes in German (our techniques are very effective in dealing with the ambiguity arising from the multiplicity of possible verb prefixes for V2 sentences). We cover the following phenomena: topicalization, fronting (including partial fronting), extraposition (although not in its full generality) and scrambling. However we are still missing many important notions such as coordination and nested extraposition fields, and our practical coverage is still quite far from what is possible in more mature grammatical frameworks.

We have experimented with both relatively small hand-crafted lexicons and one lexicon automatically derived from an annotated corpus (quite large, $\approx$ 18000 entries, but in practice disappointingly sparse). While moderate, our coverage nonetheless permits experimentation with fairly intricate sentences. Our experience so far has been quite positive: performance is good and scales up well with sentence length and number of readings, but a systematic evaluation remains to be done.

Since our framework does not assume any a-priori word order, the challenge is not to *permit* difficult linearizations to account for such phenomena as fronting, extraposition, or scrambling, but rather to *rule out* those that are ungrammatical and to avoid over-generation. Our most complete parser relies on set constraints to express complex principles of linear precedence, and borrows ideas from the theory of topological fields as well as from Reape's (1994) word order domains. While we have developed powerful *techniques* to express complex word-order constraints, we have not yet arrived at a satisfactory formal account for them.

Our current research proceeds along 4 lines: (a) extend the grammatical coverage, (b) develop a formal framework for word-order constraints in the style of the present article, (c) push beyond the limits of dependency grammar to account for e.g. "headless" constructions, (d) integrate a constraint-based treatment of semantics (Egg et al., 1998) with our treatment of syntax.

6 Conclusion

In this article, we contributed a new presentation of dependency grammar that follows modern linguistic practice, and provided an axiomatization of syntactic well-formedness whose economy and elegance accrues primarily from the expressivity of set and selection constraints. This axiomatization regards dependency parsing as a configuration problem and expresses it as a CSP.

Within the paradigm of concurrent constraint programming, our axiomatization also has a direct computational reading as a program. As could be observed in Figure 1, this program is quite efficient in two respects:

1. Constraint propagation is very effective at pruning the search space. The example of (Fig 1) makes precisely once choice to enumerate the two possible readings of the sentence. In (Fig 4), parsing of an 18 word sentence with ambiguous PP attachment is demonstrated. Again, constraint propagation is strong enough to permit optimal

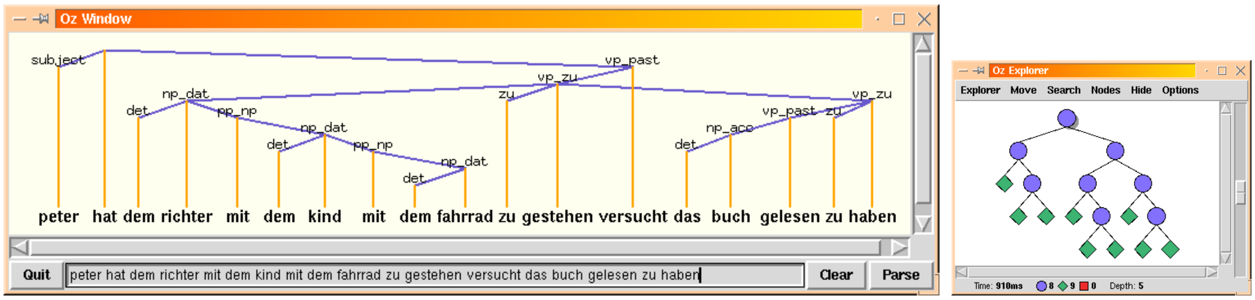


Figure 4: Enumerating PP Attachments

enumeration of all parses without any failure.

2. Absolute performance is also quite satisfactory, even without any sort of optimization: the two readings of (Fig 1) are enumerated in 190ms and the 9 readings of (Fig 4) in 910ms. Our preliminary results appear to be typically about 1 order of magnitude faster than those reported by Harper (Harper et al., 1999). We intend to look into this more closely.

We described an effective treatment of ambiguity resting on set and selection constraints. In particular we showed how selection constraints provided a CP account of dependent disjunctions with the added benefits of constructive disjunction.

In closing, it is our hope that this paper will help promote awareness of recent advances in concurrent constraint technology and of their relevance to computational linguistics, and that the techniques we described will find applications in other grammatical frameworks.

Acknowledgements. The author is especially grateful to Joachim Niehren for his extensive help in revising and improving this paper.

References

Abderrahamane Aggoun, David Chan, Pierre Dufresne, Eamon Falvey, Hugh Grant, Alexander Herold, Geoffrey Macartney, Micha Meier, David Miller, Shyam Mudambi, Bruno Perez, Emmanuel Van Rossum, Joachim Schimpf, Periklis Andreas Tsahageas, and Dominique Henry de Villeneuve. 1995. ECL^{i}PS^e 3.5. User manual, European Computer Industry Research Centre (ECRC), Munich, Germany, December.

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

Philippe Codognet and Daniel Diaz. 1996. Compiling constraints in `clp(FD)`. *The Journal of Logic Programming*, 27(3):185–226, June.

Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahamane Aggoun, Thomas Graf, and F. Berthier. 1988. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88*, pages 693–702, Tokyo, Japan, December.

Jochen Dörre and Andreas Eisele. 1990. Feature logic with disjunctive unification. In *COLING-90*, volume 2, pages 100–105.

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

Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. 1998. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING/ACL'98)*, pages 353–359, Montreal, Canada, August.

Dale Gerdemann. 1991. *Parsing and Generation of Unification Grammars.* Ph.D. thesis, University of Illinois.

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

John Griffith. 1990. Modularizing contexted constraints. In *COLING-96*.

Mary P. Harper, Stephen A. Hockema, and Christopher M. White. 1999. Enhanced constraint dependency grammar parsers. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, Honolulu, Hawai USA, August.

Johannes Heinecke, Jürgen Kunze, Wolfgang Menzel, and Ingo Schröder. 1998. Eliminative parsing with graded constraints. In *Proceedings of the Joint Conference COLING-ACL*, pages 526–530.

Randall A. Helzerman and Mary P. Harper. 1993. Muse csp: An extension to the constraint satisfaction problem. *Journal of Artificial Intelligence Research*, 1.

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

ILOG. 1996. ILOG Solver: User manual, July. Version 3.2.

Ronald M. Kaplan and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press.

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

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

John T. Maxwell and Ronald M. Kaplan. 1989. An overview of disjunctive constraint satisfaction. In *Proceedings of the Internation Workshop on Parsing Technologies*, pages 18–27.

John T. Maxwell and Ronald M. Kaplan. 1998. Unification-based parsers that automatically take advantage of context freeness. in preparation.

Wolfgang Menzel and Ingo Schröder. 1998. Decision procedures for dependency parsing using graded constraints. In *Proceedings of the COLING-ACL98 Workshop "Processing of Dependency-based Grammars"*.

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

The Mozart Consortium. 1998. The Mozart Programming System. http://www.mozart-oz.org/.

Tobias Müller and Martin Müller. 1997. 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.

Peter Neuhaus and Norbert Bröker. 1997. 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.

Carl Pollard and Ivan Sag. 1987. *Information-Based Syntax and Semantics*, volume 1 of *CSLI Lecture Notes*. CSLI.

Mike Reape. 1994. Domain union and word order variation in german. In John Nerbonne, Klaus Netter, and Carl Pollard, editors, *German in Head-Driven Phrase Structure Grammar*, number 46. CSLI Publications.

Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. 1991. 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,. ACM SIGACT-SIGPLAN, ACM Press. Preliminary report.

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