Multi-dimensional Graph Configuration for Natural Language Processing

Ralph Debusmann[1], Denys Duchier[2], and Marco Kuhlmann[1]

[1] Programming Systems Lab, Saarland University, Saarbrücken, Germany
{rade,kuhlmann}@ps.uni-sb.de
[2] Équipe Calligramme, LORIA, Nancy, France
duchier@loria.fr

Abstract Many tasks in computational linguistics can be regarded as *configuration problems*. In this paper, we introduce the notion of *lexicalised multi-dimensional configuration problems* (LMCPs). This class of problems both has a wide range of linguistic applications, and can be solved in a straightforward way using state-of-the-art constraint programming technology. The paper falls into two main parts: We first present examples for linguistic configuration problems and show how to formalise them as constraint satisfaction problems. In the second part, we introduce *Extensible Dependency Grammar* (XDG), a framework for the development of linguistic resources in the context of LMCPs.

1 Introduction

Various tasks in computational linguistics can be regarded as *configuration problems* (CPs). The input to a configuration problem is a set of *components*; the output is a selection from these components and an assembly of the selected components that satisfies certain problem-specific constraints [1].

In this paper, we introduce a particular class of CPs called *lexicalised multi-dimensional configuration problems* (LMCPs). A configuration problem is *lexicalised* when, as is the case in many linguistic applications, the ambiguity involved in choosing the components to assemble is stipulated by a lexicon. A *multi-dimensional* configuration problem is one where several mutually constraining CPs have to be solved at once.

LMCPs are a natural way to formalise linguistic tasks that involve multiple levels of description—e.g., syntactic structure, linear precedence, and predicate/argument structure. They have two major benefits: First, they strike a balance between complete modularity and tight integration of the developed resources—each linguistic dimension is characterised using its own set of well-formedness conditions; interactions between different dimensions are specified at an interface level. Second, LMCPs can be solved in a straightforward way using state-of-the-art constraint programming technology.

Plan of the paper We start by giving three examples for linguistic tasks that can be understood as configuration problems (Section 2). We then sketch how

to encode such problems into constraints on finite sets of integers (Section 3), and how to extend this encoding to LMCPs (Sections 4 and 5). In Section 6, we then provide an introduction to Extensible Dependency Grammar (XDG), a development environment for linguistic modelling that embraces the approach of LMCPs. Section 7 concludes the paper with an outline of future work.

2 Configuration Problems in NLP

In this section, we argue that many tasks in computational linguistics can be usefully regarded as instances of *configuration problems*. We do so by giving three representative examples for which constraint-based processing techniques have been developed: semantic assembly, surface realisation, and syntax analysis.

2.1 Semantic Assembly

We first turn to the task of assembling the semantic representation of a sentence from the individual fragments of representation contributed by its words in the context of *scope ambiguity*. Consider the following sentence:

(1) Every researcher deals with a problem.

This sentence has two readings, which may be disambiguated by the following continuations:

(1a) ... Some of these problems may be unsolvable.
(1b) ... This problem is his funding.

If represented in terms of Montague-style semantics, the two readings could be rendered as follows:

$$\forall x\colon \mathsf{researcher}(x) \rightarrow \exists y\colon \mathsf{problem}(y) \wedge (\mathsf{deal\ with})(x, y) \tag{1a}$$

$$\exists y\colon \mathsf{problem}(y) \wedge \forall x\colon \mathsf{researcher}(x) \rightarrow (\mathsf{deal\ with})(x, y) \tag{1b}$$

Notice that both these terms are made up of exactly the same 'material':

$$\forall x\colon (\mathsf{researcher}(x) \rightarrow \dots)\,, \qquad \exists y\colon (\mathsf{problem}(y) \wedge \dots)\,, \qquad (\mathsf{deal\ with})(x, y)\,.$$

The only difference between the two readings is the relative ordering of the term fragments: in (1a), the universal quantifier takes scope over the existential quantifier; in (1b), it is the other way round. Formalisms for *scope underspecification* [2,3,4] aim for a compact representation of this kind of ambiguity: they can be used to describe the common parts of a set of readings, and to express constraints on how these fragments can be 'plugged together'.

The left half of Fig. 1 shows an underspecified graphical representation of the two readings of (1) in the formalism of *dominance constraints* [4]. The solid edges in the picture mark the term fragments that are shared among all readings. Two fragments can combine by 'plugging' one into an open 'hole' of the

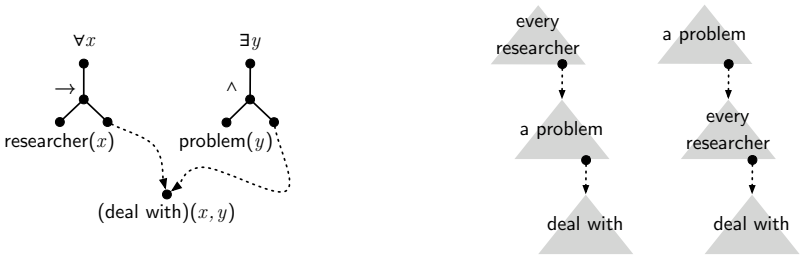


Figure 1. A dominance constraint for the two readings of (1), and its two solutions

other. The dashed edges mark *dominance requirements*, where dominance means ancestorship in the final configuration of the term fragments. For instance, the fragments for 'every researcher' and for 'a problem' dominate the 'deal with' fragment, i.e. both must be ancestors of the latter in any complete term tree. With the given dominance requirements, exactly two configurations of the fragments are possible (shown schematically in the right half of Fig. 1); these two configurations correspond to the readings (1a) and (1b).

2.2 Surface realisation

Surface realisation is the sub-task of natural language generation that maps a semantic representation to a grammatical surface string. More specifically, for some given grammar, surface realisation takes as its input a bag of semantic descriptions, ϕ, and returns a syntax tree containing a verbalisation of ϕ.

Here we discuss surface realisation for Tree Adjoining Grammar (TAG) [5]. One of the underlying design principles of many TAG grammars is *semantic minimality*: each lexical entry (*elementary tree*) of a TAG grammar corresponds to an atomic semantics. Surface realisation then can be reduced to the problem of *selecting* for each semantic atom a matching elementary tree, and *assembling* these trees into a *derivation tree* using the standard TAG operations that combine grammatical structures: substitution and adjunction [6].

We illustrate this by means of an example. Assume that we want to realise the following (event-based) input semantics using some given TAG grammar G:[3]

$$x = \mathsf{Peter},\ \mathsf{see}(e, x, y),\ \mathsf{indef}(y),\ \mathsf{fat}(y),\ \mathsf{rabbit}(y)\,.$$

In a first step, we need to choose for each of the semantic atoms an elementary tree from G that verbalises their semantics. A sample selection of trees is shown in Fig. 2. The dashed arrows in the figure indicate a way to compose the chosen elementary trees by means of substitution and adjunction; for example,

[3] Note that all atoms are considered to be ground.

the tree realising the semantic atom $\mathsf{fat}(y)$ can adjoin into the root node (labelled with N) of the tree realising the semantics of $\mathsf{rabbit}(y)$. Fig. 3 shows the resulting derivation tree for the sentence. In a post-processing step, this derivation tree can be transformed into a *derived tree*, whose yield is a possible realisation of the intended semantics.

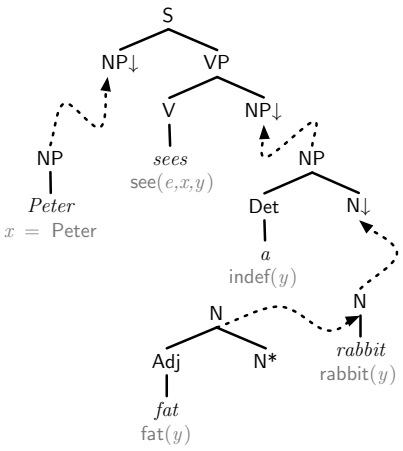


Figure 2. Configuring TAG elementary trees for surface realisation

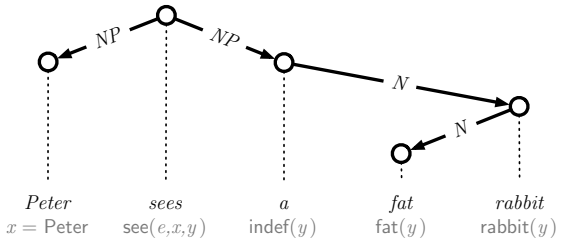


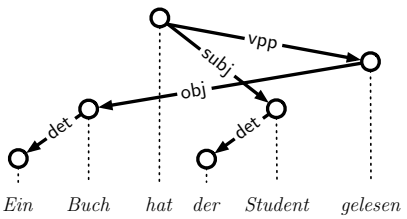
Figure 3. TAG derivation tree for the configuration in Fig. 2

2.3 Syntactic Analysis

Our final example for a linguistic configuration problem is the parsing of dependency grammars. As we have seen, surface realisation can be reduced to the configuration of a labelled tree in which the nodes are labelled with lexical entries

(elementary trees), and the edges are labelled with sites for substitution and adjunction. It has often been noted that these derivation trees closely resemble dependency trees.

A *dependency tree* for a sentence s is a tree whose nodes are labelled with the words of s, and whose (directed) edges are labelled with assymmetric *grammatical relations* (like *subject-of* or *adverbial-modifier-of*). Given an edge $u - \rho \rightarrow v$ in a dependency tree, u is called the *head* of u, v is called the *dependent* of v, and ρ is the grammatical relation between the two. A *dependency grammar* consists of a lexicon and a *valency* assignment for the lexical entries that specifies the grammatical relations a given entry must or may participate in as head and as dependent. A dependency tree is *licensed* by a given grammar if for every edge $u - \rho \rightarrow v$, the relation ρ is a grammatical relation licensed by the lexical entries for u and v.



Lexical entry	*function*	*subcat*
ein	det	–
Buch	subj ∨ obj	det
hat	–	subj ∧ vpp
der	det	–
Student	subj ∨ obj	det
gelesen	vpp	obj

Figure 4. A dependency tree for (2), and a licensing lexicon

The left half of Fig. 4 shows a dependency tree for the German sentence

(2) *Ein Buch hat der Student gelesen.*
a book has the student read
'The student has read a book.'

The right half of the figure shows a valency assignment with which this tree would be licensed: it specifies possible incoming edges and required outgoing edges. For example, the lexical entry for *Student* can act both as a subject and an object dependent of its head, and itself requires a dependent determiner.

3 Graph Configuration Problems

The problems described in the previous section have this in common: in each task, we are given a number of graph fragments, and have to plug them together (under constraints) to obtain a complete assembly. We call this class of problems *graph configuration problems*.

For such problems, we represent the plugging of a fragment w into another fragment w' by means of a directed labelled edge $w - \ell \rightarrow w'$, which makes explicit that a resource of type ℓ supplied by w' is being matched with a corresponding

resource requirement in w. In the dependency parsing example, fragments can provide resources whose types are grammatical functions like *subject-of*.

We are thus led to the notion of (finite) labelled graphs. Consider given a finite set $\mathcal{L}$ of labels. A $\mathcal{L}$-labelled graph (V, E) consists of a set V of nodes and a set $E \subseteq V \times V \times \mathcal{L}$ of directed labelled edges between them. We can interpret each label ℓ as a function from nodes to sets of nodes defined a follows:

$$\ell(w) = \{\, w' \mid w -\ell\rightarrow w' \in E \,\}$$

(In graph theoretical terms, the set $\ell(w)$ is the set of immediate successors of w that can be reached by traversing an edge labelled with ℓ.) Duchier [7] developed this set-based approach and showed e.g. how to formulate a constraint system that precisely characterises all labelled trees which can be formed from the finite set of nodes V and the finite set of edge labels $\mathcal{L}$. This system is expressed in terms of finite set variables $\ell(w)$, $\mathsf{daughters}(w)$, $\mathsf{down}(w)$ and $\mathsf{eqdown}(w)$, and a global variable roots:

$$\begin{aligned}
& V = \mathsf{roots} \;\uplus\; \uplus\{\, \mathsf{daughters}(w) \mid w \in V \,\} \\
\wedge\;& |\mathsf{roots}| = 1 \\
\wedge\;& \forall w \in V \\
& \quad (\forall \ell \in \mathcal{L} \quad \ell(w) \subseteq V) \\
& \quad \wedge \quad \mathsf{eqdown}(w) = \{w\} \uplus \mathsf{down}(w) \\
& \quad \wedge \quad\;\; \mathsf{down}(w) = \cup\{\, \mathsf{eqdown}(w') \mid w' \in \mathsf{daughters}(w) \,\} \\
& \quad \wedge\; \mathsf{daughters}(w) = \uplus\{\, \ell(w) \mid \ell \in \mathcal{L} \,\}
\end{aligned} \tag{3}$$

By taking advantage of the constraint programming support available in a system such as Mozart/Oz, this formal characterisation of finite labelled trees can be given a straightforward computational interpretation.

Using this approach as a foundation, we can encode graph configuration problems with additional constraints. For example, each node typically offers a specific subset of resources. Such a restriction can be enforced by posing constraints on the cardinality of $\ell(w)$ (for $\ell \in \mathcal{L}$), e.g. $|\ell(w)| = 0$ for a resource not offered by w and $|\ell(w)| = 1$ for a resource offered exactly once. This is how we can model subcategorisation in the application to dependency parsing.

4 Lexicalised Configuration Problems

For linguistic configuration tasks, it is not realistic to assume that the fragments to be plugged together are given explicitly right from the start. For example, in the surface realisation problem, there may be several alternative ways to verbalise the same atomic semantics. Similarly, in the syntax analysis problem, one word may have several readings, alternative subcategorisation frames, or alternative linearisation constructions.

We therefore generalise the previous view by replacing each fragment with a finite collection of fragments from which one must be selected. Since in linguistic contexts, the mapping from nodes to collections of alternative fragments is often stipulated by a lexicon, we call such problems *lexicalised configuration problems*.

The challenge now is to adapt the constraint-based approach outlined in Section 3 to gracefully handle lexical ambiguity. Let's consider again the dependency parsing application. As described earlier, for a given set V of words, we can model the possible dependency trees as the solutions of the constraint system (3), and enforce subcategorisation frames using cardinality constraints on ℓ-daughter sets $\ell(w)$. If we now assume that w has k lexical entries, each one may stipulate a different cardinality constraint for $\ell(w)$. Clearly, in order to avoid combinatorial explosion, we do not want to try all possible combinations of selections for the given words. Instead, we would like to take advantage of constraint propagation to avoid non-deterministic choices.

What we want is an underspecified representation of the lexical entry that is ultimately chosen in a form that can be easily integrated into a constraint-based formulation. This is the purpose of the *selection constraint*

$$X = \langle Y_1, \ldots, Y_n \rangle [I] \,,$$

whose declarative semantics is $X = Y_I$. All of X, Y_i and I may be variables, and propagation takes place in both directions: into X in a manner similar to constructive disjunction, and into I whenever a Y_k becomes incompatible with X (and thus k can be removed from the domain of I).

We have implemented support for selection constraints for finite-domain integer (`FD`) and finite integer set variables (`FS`). This support can easily be lifted over feature structures as follows:

$$\left\langle \begin{bmatrix} \mathsf{f}_1 = v_1^1 \\ \vdots \\ \mathsf{f}_p = v_p^1 \end{bmatrix}, \ldots, \begin{bmatrix} \mathsf{f}_1 = v_1^k \\ \vdots \\ \mathsf{f}_p = v_p^k \end{bmatrix} \right\rangle [I] \quad = \quad \begin{bmatrix} \mathsf{f}_1 = \langle v_1^1, \ldots, v_1^k \rangle [I] \\ \vdots \\ \mathsf{f}_p = \langle v_p^1, \ldots, v_p^k \rangle [I] \end{bmatrix} \tag{4}$$

In this manner, we can use the selection constraints for complex lexical entries. Notice how features f_1 through f_p are constrained by concurrent selection constraints which are all covariant because they share the same selector variable I.

5 Multi-dimensional Configuration Problems

In the previous section, we have introduced the notion of lexicalised graph configuration problems, and suggested how they can be encoded into systems of contraints that are adequately supported by corresponding constraint technology. In the present section, we generalise this notion to *multi-dimensional configuration problems*—problems that consist of several individual configuration tasks that mutually constrain each other. The motivation for this generalisation is the insight that many phenomena in natural language take advantage from splitting them up into more than one dimension of linguistic description. We illustrate this idea with the treatment of German word order phenomena in Topological Dependency Grammar (TDG).

Consider again the sentence from the dependency parsing example presented in Section 2, here repeated for convenience:

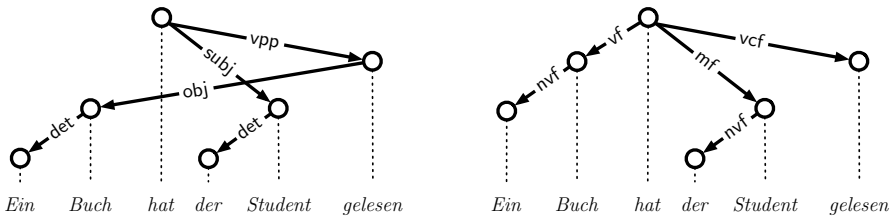
(5) *Ein Buch hat der Student gelesen.*
a book has the student read
'The student has read a book.'

The sentence illustrates object topicalisation in German. Starting from the 'canonically' ordered sentence

(6) *Der Student hat ein Buch gelesen.*
The student has a book read
'The student has read a book.'

sentence (5) may be construed as the result of the fronting of *ein Buch* (the object of the verb), and a successive movement of *der Student* (the subject) to the now vacant object position. However, a far more natural and perspicuous account is obtained when one separates constituency and linear precedence, and describes word order variation as a relation between those two structures.

Fig. 5 shows the analysis of (5) using Topological Dependency Grammar (TDG) [8]). TDG dedicates one dimension to *immediate dominance* (ID) and another to *linear precedence* (LP). Each dimension has its own set of well-formedness conditions (called *principles*): both the ID and the LP structures are required to be trees, but LP structures must also be ordered and projective. Moreover, a *multi-dimensional principle* called *climbing* constrains the LP tree to be a flattening of the ID tree.



(a) Immediate Dominance (b) Linear Precedence

Figure 5. Topicalisation

The parsing task of TDG is an example for a multi-dimensional graph configuration problem according to the characterisation above: to find the structures licensed by a given input, one has to find all triples $(V, T_{\mathsf{id}}, T_{\mathsf{lp}})$ in which T_{id} is a well-formed ID structure on V, T_{lp} is a well-formed LP structure on V, and the

climbing relation holds between them. Section 6 describes an extended version of this model with additional dimensions for deep syntax, predicate/argument structure, and scope.

The constraint encoding presented in the previous section can be extended to multi-dimensional configuration problems as follows:

- Every lexical entry now contains a sub-lexical entry for each dimension. In this manner, lexical entries simultaneously constrain all dimensions. Figure 6 illustrates the selection constraint operating simultaneously on 3 dimensions over 3-dimensional lexical entries.
- Multi-dimensional principles are expressed as global constraints that relate several dimensions to one another.

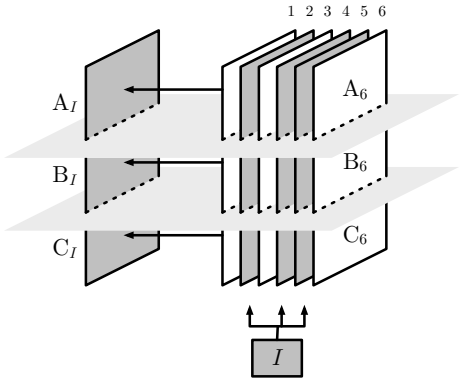


Figure 6. Multi-dimensional selection

Lexicalised multi-dimensional configuration problems yield a modular and scalable framework for modelling linguistic phenomena. How do they relate to other approaches? LMCPs most closely resemble the parallel grammar architecture proposed by Sadock and Jackendoff [9,10], where a number of semi-autonomous modules (the dimensions in an LMCP) operate in parallel and interact through bi-directional interfaces (multi-dimensional principles). With various formalisms like Lexical Functional Grammar (LFG) [11] and Meaning-Text Theory (MTT) [12], LMCPs share the idea of distinguishing several representational structures. In contrast to these formalisms, however, LMCPs do not presuppose a *layered* representation, where only adjacent layers can share information directly, nor do they assume that information can flow only in one direction: multi-dimensional principles can connect arbitrary dimensions, and they are not required to be functional. Given that all dimensions share the same set of nodes, multi-dimensional graphs also exhibit many of the benefits that arise through the tight integration of information that is obtained in formalisms like HPSG [13].

6 Extensible Dependency Grammar

In this section, we introduce Extensible Dependency Grammar (XDG) [14], our flagship instance of a development environment for lexicalised multi-dimensional configuration problems. XDG is a generalization of TDG from the previous section. It supports the use of an arbitrary number of dimensions of linguistic representation, and of arbitrary *principles* stating the well-formedness conditions of these structural dimensions.

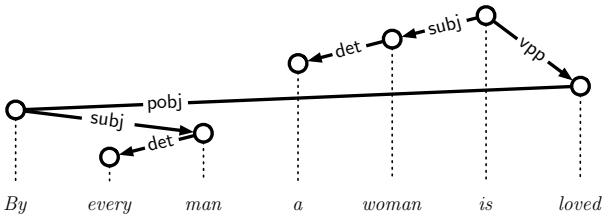
6.1 Example

Rather than giving a formal definition of XDG, we propose a five-dimensional sample grammar and illustrate it with the following sentence, an English passive construction paraphrasing the ubiquitous linguistic example 'Every man loves a woman':

(7) By every man, a woman is loved.

In the following, we give a quick tour through the five dimensions of our sample grammar to show what aspects of linguistic analysis they cover.

ID dimension The ID dimension (ID stands for *immediate dominance*) represents the linguistic aspect of *grammatical function*. It was introduced already for TDG in Section 5. We display the ID analysis of (7) below:

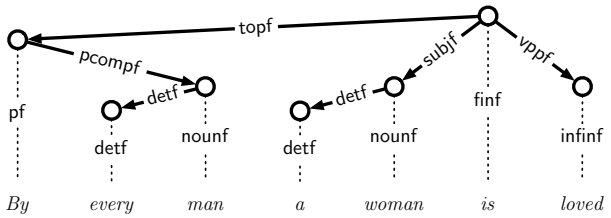


(8)

Here, 'woman' is the subject (edge label subj) of the finite verb 'is'; 'a' is the determiner (edge label det) of 'woman'; 'loved' is the verbal past participle (vpp) of 'is'; 'by' is the prepositional object (pobj) of 'loved'; 'man' is the prepositional complement (pcomp) of 'by'; 'every' is the determiner of 'man'.

LP dimension The LP dimension (LP for *linear precedence*) describes *word order*, and has also already been introduced in Section 5. On the LP dimension, edge labels name word positions.[4]

[4] Following work on TDG, we adopt the convention to suffix LP edge labels with an 'f' for 'field' to better distinguish them from ID edge labels.

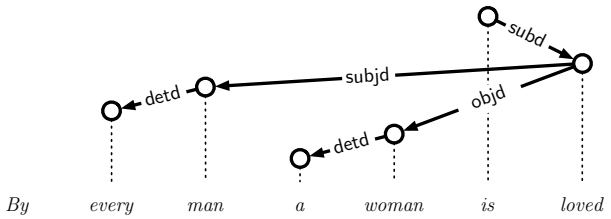


(9)

In the LP analysis, the finite verb 'is' is the root of the sentence; 'by' is in the topicalisation position (topf); 'woman' is in the subject position (subjf); 'loved' is in the verbal past participle position (vppf). Furthermore, 'man' is in the prepositional complement position of 'by' (pcompf); 'every' is in the determiner position of 'man' (detf); 'a' is in the determiner position of 'woman' (detf).

On the LP dimension, we additionally annotate the nodes with node labels (displayed on the dotted projection edges connecting nodes and words). These are required for specifying the relative order of mothers and their daughters. For example, a determiner in the determiner position of a noun (edge label detf) must precede the noun itself (node label nounf).

DS dimension The DS dimension (DS for *deep syntax*) represents an intermediate structure between syntax and semantics: this In dimension, constructions such as control, raising and passive are already resolved to enable a more seamless transition to semantics. Furthermore, function words such as the preposition 'by' and 'to'-particles are not connected, since they have no impact on the semantics. Below is an example DS analysis:[5]



(10)

Here, 'loved' is subordinated to 'is' (edge label subd); 'man' is the deep subject (subjd) of 'loved'; 'woman' is the deep object (objd); 'every' is the determiner of 'man'; 'a' is the determiner of 'woman'.

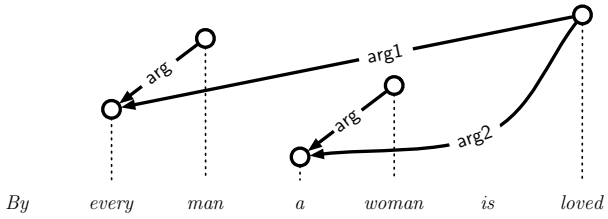
Notice that in this example, the relations of deep subject and deep object do not match the relations of subject and prepositional object on the ID dimension, due to the passive construction: whereas in the ID analysis, 'woman' is the subject of the auxiliary 'is' and 'by' is the prepositional object of 'loved', the DS analysis

[5] We adopt the convention to suffix DS edge labels with 'd' for 'deep' to better distinguish them from ID edge labels.

mirrors the underlying predicate/argument structure more closely by making 'woman' the deep object of 'loved', and 'man' its deep subject.

Semantics disentangled The last two dimensions of our sample grammar represent semantics. Sticking to our dictum of modularity, we use them to disentangle two aspects of semantics which are usually conflated in representations based on predicate logic: the introduction and binding of individual variables (PA dimension) and the hierarchical relationships between the predicates (SC dimension). The PA dimension (PA for *predicate/argument structure*) tells us which individual variables are introduced by which quantifiers, and how these variables fill the argument slots of the predicates. It does not tell us anything about the hierarchical structure in which the predicates are arranged in the formula; this is relegated to the SC dimension (SC for *scope structure*).

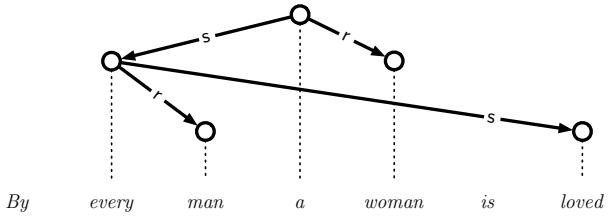
Here is the PA analysis for sentence (7):



(11)

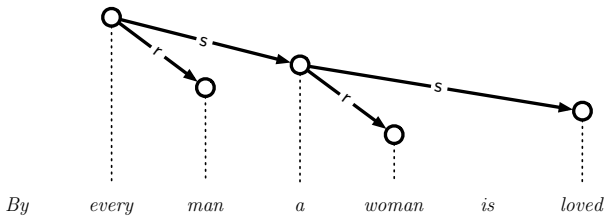
In this example analysis, think of the quantifier 'every' as introducing a variable x, and 'a' introducing a variable y. The predicate 'man' binds x, and 'woman' binds y. Finally, x is the first argument of the predicate expressed by 'loved', and y is the second argument.

Here is a corresponding SC analysis:



(12)

Here, 'a' has 'woman' in its restriction (edge label r) and 'every' in its scope (edge label s); 'every' has 'man' in its restriction and 'loved' in its scope. Notice that this is only one of the two possible scope readings of the sentence—the 'strong' reading where the existential quantifier outscopes the universal quantifier. The SC analysis representing the other, 'weak' reading is depicted below.



(13)

With predicate/argument structure and scope structure disentangled, we are able to easily construct a flat semantic representation useful e.g. for machine translation [15]. To this end, we extract the information present in the PA dimension, without taking the SC dimension into account. To obtain a 'complete' semantic representation (reflecting both variable binding *and* the hierarchical structure of the formula), we have to combine the information from the PA and the SC dimension. We will give an example of how to do this below.

Semantics construction Suppose that the lexicon contains for each semantically contentful word a PA expression and an SC expression as follows:

	PA	SC
every	$\forall\langle\_\rangle$	$\mathsf{every}(\langle\mathsf{r}\rangle \rightarrow \langle\mathsf{s}\rangle)$
a	$\exists\langle\_\rangle$	$\mathsf{some}(\langle\mathsf{r}\rangle \wedge \langle\mathsf{s}\rangle)$
man	$\mathsf{man}(\langle\mathsf{arg}\rangle)$	man
woman	$\mathsf{woman}(\langle\mathsf{arg}\rangle)$	woman
love	$\mathsf{love}(\langle\mathsf{arg1}\rangle, \langle\mathsf{arg2}\rangle)$	love

Starting from these expressions, we can make use of the PA and SC analyses to construct the final semantics of the sentence.

First, we instantiate the underscores in the PA expressions with fresh individual variables, and the arguments of the predicates with the variables associated to the quantifiers their argument edges point to. This yields the following description of the PA dimension:

$$\text{every} : \forall x\,,\ \text{a} : \exists y\,,\ \text{man} : \mathsf{man}(x)\,,\ \text{woman} : \mathsf{woman}(y)\,,\ \text{love} : \mathsf{love}(x, y)$$

We then replace the slot arguments of the SC expressions by the respective dependents in an SC analysis, say (12). This yields the following SC description:

$$\text{every} : \mathsf{every}(\mathsf{man} \rightarrow \mathsf{love})\,,\ \text{a} : \mathsf{some}(\mathsf{woman} \wedge \mathsf{every}(\mathsf{man} \rightarrow \mathsf{love}))\,,$$
$$\text{man} : \mathsf{man}\,,\ \text{woman} : \mathsf{woman}\,,\ \text{love} : \mathsf{love}$$

Now we combine the two descriptions to obtain a complete formula: we simply replace the constants in the SC description by the corresponding PA expressions.

$$\text{every} : \forall x(\mathsf{man}(x) \rightarrow \mathsf{love}(x, y))\,,$$
$$\text{a} : \exists y(\mathsf{woman}(x) \wedge \forall x(\mathsf{man}(x) \rightarrow \mathsf{love}(x, y)))\,,$$
$$\text{man} : \mathsf{man}(x)\,,\ \text{woman} : \mathsf{woman}(x)\,,\ \text{love} : \mathsf{love}(x, y)$$

The expression at the root of the SC analysis (12)—the expression corresponding to the quantifier a—is the formula we are after: the 'strong' reading of the sentence, in which the existential quantifier outscopes the universal quantifier.

6.2 Principles and the lexicon

Parsing sentences using XDG amounts to solving a lexicalised multi-dimensional configuration problem in the sense presented in Section 5: The well-formedness of an analysis is determined by the interaction of grammatical principles and the lexicon. The principles stipulate restrictions on one or more dimensions. They are controlled by and interact through the feature structures assigned to the nodes from the lexicon.

In the current setup of XDG, concrete principles are taken from an extensible library of parametric principles. This library already contains the necessary principles to model the syntax and semantics for large fragments of German and English, and smaller fragments of Arabic, Czech and Dutch. We present a representative subset of it below.

tree(d) stipulates that dimension d must be a tree. In the example above, we use this principle on the ID, LP and SC dimensions.

dag(d) stipulates that dimension d must be a directed acyclic graph. We use this principle on the DS and PA dimensions, which need not necessarily be trees.

valency(d) stipulates that, for each node on a dimension d, incoming and outgoing edges must be licensed by the lexical valency (cf. the valency lexicon given in Fig. 4). This is a key principle in XDG, and is used on all dimensions. Note that in contrast to the previous principles, it is a lexicalised principle.

ordered($d, \prec$) stipulates that, for each node w on a given dimension, the set containing w and its daughters is ordered according to the order $\prec$, defined on node and edge labels. We use this principle on the LP dimension to constrain the order of the words in a sentence; we can use it e.g. to require that determiners (detf) precede nouns (nounf).

projective(d) stipulates that dimension d must be a projective graph. We use this principle on the LP dimension to exclude LP trees with crossing edges.

climbing(d_1, d_2) The climbing principle is two-dimensional; it stipulates that a given dimension d_1 must be flatter than a given dimension d_2. We use it to state that the LP dimension (9) must be flatter than the ID dimension (8).

linking(d_1, d_2) Like the climbing principle, the linking principle relates two dimensions. It allows us to specify, for each node on the participating dimensions, what edges on dimension d_1 leaving w correspond to what edges on dimension d_2 leaving w. In particular, we use it to stipulate that e.g. the first argument of 'loved' (arg1) on the PA dimension (11) must be realised by the deep subject (subjd) on the DS dimension (10), and the second argument (arg2) by the deep object (objd).

contradominant(d_1, d_2) The contra-dominance principle is yet another lexicalised two-dimensional principle. We use it to stipulate that, for each pair (w_1, w_2) of nodes on the participating dimensions, an edge $w_1 \ -\rho\rightarrow\ w_2$ on dimension d_1 must correspond to a (contravariant) sequence of edges $w_2 \ -\sigma_1\rightarrow \cdots -\sigma_k\rightarrow\ w_2$ on dimension d_2. In particular, this allows us to stipulate that the semantic arguments of verbal predicates on the PA dimension must dominate (be an ancestor of) these predicates on the SC dimension. For instance, the first semantic argument of 'loved' on the PA dimension—in (11), this is the determiner 'every' of the noun phrase 'every man'—must dominate 'loved' on the SC dimension (analyses (12) and (13)).

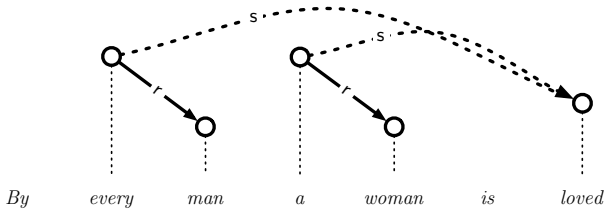
6.3 Solving XDG descriptions

Extending the encoding presented in Sections 3 and 4, we have implemented a constraint solver for XDG descriptions using the MOZART/OZ programming system [16,17]. One of the major benefits of using constraint programming to solve LMCPs like the ones that can be specified using XDG is that constraint solving can operate concurrently on all dimensions: the solver can infer information about one dimension from information on any other dimension. In this way, syntactic information can trigger inferences in semantics, and vice versa. Moreover, the same solver can be used for parsing and generation with XDG grammars: the only difference between the two tasks is the form of the input (words for the parsing task, semantic atoms for generation).

Because XDG allows us to write grammars with completely free word order, XDG solving is an NP-complete problem [6]. However, the runtime behaviour of the solver on hand-written natural language grammars is excellent in practice: constraint propagation is both fast and effective, and permits the enumeration of solutions to the configuration problem with few or no failures. This indicates that there may be fragments of the general problem of solving XDG descriptions that can be solved in polynomial time.

6.4 Underspecification

Similar to MRS [3] and CLLS [4], XDG supports the use of *underspecification*. Indeed, underspecification is a very natural concept in the context of XDG constraint solving: an underspecified XDG analysis is a partial XDG dependency graph where not all of the edges are fully determined. We show an underspecified XDG analysis for the SC dimension below.



(14)

In this analysis, the edges from 'every' to 'man' and from 'a' to 'woman' (both labelled r) are already determined, i.e. we know that 'man' is in the restriction of 'every', and that 'woman' is in the restriction of 'a'. However, the scopal relationship between the two quantifiers is yet unknown. Still, the XDG constraint solver has already inferred that both quantifiers dominate the verb 'loved', as indicated by the dotted dominance edges. This partial analysis abstracts over both fully specified analyses (12) and (13) above.

Whereas in MRS and CLLS, only scopal relationships can be underspecified, XDG allows to underspecify *any* of its dimensions. For example, underspecification on the syntactic dimensions can be used to compactly represent PP-attachment ambiguities.

7 Conclusion and future work

We proposed lexicalised multi-dimensional configuration problems (LMCPs) as a common metaphor for a wide range of tasks in computational linguistics, including semantic assembly, surface realisation, and syntactic analysis. We then presented Extensible Dependency Grammar (XDG) as a development environment for LMCPs, and showed how to use it to model the syntax and semantics of natural language in an integrated fashion. We think that LMCPs provide an interesting framework for research in computational linguistics both on the theoretical and on the algorithmic side. In the remainder of this section, we outline possible directions for future work in this area.

Formal aspects Graph configuration takes a descriptional approach to modelling natural language: problems are characterised in terms of description languages (like the lexicon in XDG); solutions to the problems are valid interpretations of such descriptions with respect to the intended models (like finite labelled graphs in the case of XDG).

One of the major questions is how this description-based approach relates to other—in particular, generative—approaches to NLP. More specifically, we want to find out how standard tasks like parsing CFGs, TAGs, or CCGs can be encoded as graph configuration problems. This will give us a better understanding of the expressive power of graph configuration, and might also allow us to compare distinct syntactic and semantic formalisms on the basis of their 'configurational core'—i.e., their underlying configuration problems.

Polynomial fragments While unrestricted graph configuration is NP-complete, the constraint propagation techniques employed in the XDG solver usually do an excellent job on hand-written grammars. This indicates that those grammars might belong to a natural fragment of XDG for which specialised polynomial algorithms can be found. Furthermore, following up on the encoding issue mentioned above, we will explore whether the polynomial complexity of e.g. the parsing problems of CFGs and TAGs is mirrored by special properties of their formulation as LMCPs.

Import of linguistic resources Using the encodings of other syntactic and semantic formalisms, it will be possible to import existing linguistic resources like large-scale grammars and syntactic and semantic corpora into the graph configuration framework. We will then be able to evaluate the feasibility of the constraint-based processing architecture outlined above on real-world data, and see if the descriptional and methodological benefits that LMCPs provide are relevant in linguistic practice. This line of research has already led to very interesting results with respect to the development of syntax/semantics interfaces for dependency grammar [14].

References

1. Mittal, S., Frayman, F.: Towards a generic model of configuration tasks. In: Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1989) 1395–1401
2. Bos, J.: Predicate logic unplugged. In: Proceedings of the 10th Amsterdam Colloquium. (1996) 133–143
3. Copestake, A., Flickinger, D., Pollard, C., Sag, I.: Minimal recursion semantics. an introduction. Journal of Language and Computation (2004) To appear.
4. Egg, M., Koller, A., Niehren, J.: The constraint language for lambda structures. Journal of Logic, Language, and Information (2001)
5. Abeillé, A., Rambow, O.: Tree Adjoining Grammar: An Overview. In: Tree Adjoining Grammars: Formalisms, Linguistic Analyses and Processing. CSLI Publications (2000)
6. Koller, A., Striegnitz, K.: Generation as dependency parsing. In: Proceedings of ACL 2002, Philadelphia/USA (2002)
7. Duchier, D.: Configuration of labeled trees under lexicalized constraints and principles. Research on Language and Computation **1** (2003) 307–336
8. Duchier, D., Debusmann, R.: Topological dependency trees: A constraint-based account of linear precedence. In: Proceedings of ACL 2001, Toulouse/FRA (2001)
9. Sadock, J.M.: Autolexical Syntax. University of Chicago Press (1991)
10. Jackendoff, R.: Foundations of Language. Oxford University Press (2002)
11. Bresnan, J., Kaplan, R.: Lexical-functional grammar: A formal system for grammatical representation. In Bresnan, J., ed.: The Mental Representation of Grammatical Relations. The MIT Press, Cambridge/USA (1982) 173–281
12. Mel'čuk, I.: Dependency Syntax: Theory and Practice. State Univ. Press of New York, Albany/USA (1988)
13. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago/USA (1994)
14. Debusmann, R., Duchier, D., Koller, A., Kuhlmann, M., Smolka, G., Thater, S.: A relational syntax-semantics interface based on dependency grammar. In: Proceedings of COLING. (2004)
15. Trujillo, I.A.: Lexicalist Machine Translation of Spatial Prepositions. PhD thesis, University of Cambridge, Cambridge/USA (1995)
16. Smolka, G.: The Oz Programming Model. In van Leeuwen, J., ed.: Computer Science Today. Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin (1995) 324–343
17. Mozart Consortium: The Mozart-Oz website (2004) http://www.mozart-oz.org/.