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. We introduce the new abstract notion of *multi-dimensional lexicalized graph configuration problems*, generalising over many important tasks in computational linguistics such as semantic assembly, surface realization and syntactic analysis, and integrating them. We present *Extensible Dependency Grammar* (XDG) as one instance of this notion.

1 Introduction

A variety of tasks in computational linguistics can be regarded as *configuration problems* (CPs) [1]. In this paper, we introduce the notion of *lexicalised, multi-dimensional* CPs, a particular class of configuration problems that both has a wide range of linguistic applications, and can be solved in a straightforward way using state-of-the-art constraint programming technology. Linguistic modeling based on multi-dimensional CPs brings two major benefits: complete modularity of the developed resources, and tight integration of the individual modules.

We first consider configuration problems where the input is a set of components, and the output is a valid assembly of these components that satisfies certain problem-specific constraints. We then broaden the perspective to permit ambiguity as to the choice of each component. We call a configuration problem *lexicalized* when, as is the case in typical linguistic applications, this ambiguity is stipulated by a lexicon. Finally, to permit modeling with multiple levels of linguistic description (e.g. syntax, linear precedence, predicate/argument structure ...), we introduce the notion of a multi-dimensional configuration problem where we must simultaneously solve several configuration problems that are not independent, but rather constrain each other.

We then provide an introduction to Extensible Dependency Grammar (XDG) which is a development environment for linguistic modeling embracing the approach of lexicalised, multi-dimensional CPs.

The methodology based on lexicalized, multi-dimensional configuration problems is attractive for several reasons: for linguistic modeling, it is possible, for each level of linguistic description, to design a dimension that is especially well suited to it. For constraint-based processing, an inference on any dimension may help disambiguate another.

2 Configuration Problems in NLP

In this section, we develop the point that many tasks in computational linguistics can be usefully regarded as instances of *configuration problems*. We illustrate this point with three representative examples for which we have developed constraint-based processing techniques: semantic assembly, surface realization, 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 is the relative ordering of these 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* [2–4] 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 other. The dashed edges mark *dominance requirements*, where dominance corresponds to ancestorship. 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. 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).

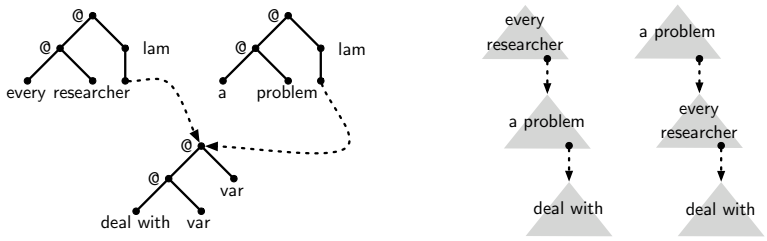


Fig. 1. A dominance constraint for the two readings of (1), and its two solutions

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 as its output a syntax tree containing the 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, namely 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:

$$x = Peter, \quad see(e, x, y), \quad some(x), \quad fat(x), \quad rabbit(x)\,.$$

(Note that all atoms are considered to be ground.) In a first step, we need to choose for each of the semantic atoms an elementary tree from G that verbalises its semantics. A sample selection of trees is shown in the left half of Fig. 2. The dashed arrows in the figure indicate a way to compose the chosen elementary trees by means of substitution and ajunction. For example, the tree realising the semantic atom $fat(x)$ can adjoin into the root node (labelled with N) of the tree realising the semantics of $rabbit(x)$. The right half of Fig. 2 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.

2.3 Syntactic Analysis

Our final example is the parsing of dependency grammars. As we have seen, surface realisation can be reduced to the configuration of a labelled tree in which

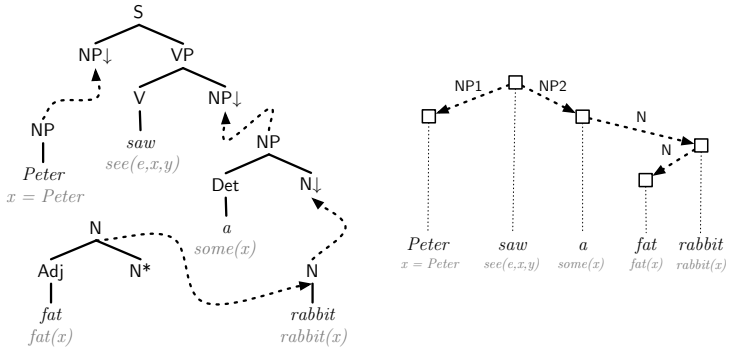
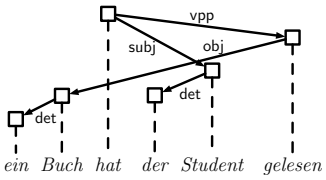


Fig. 2. Generation

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 antisymmetric *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, and v is called the *dependent* of v. 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 both the lexical entries for u and v.



Lexical entry	*incoming*	*outgoing*
ein	{det}	{}
Buch	{subj, obj}	{det}
hat	{}	{subj, vpp}
der	{det}	{}
Student	{subj, obj}	{det}
gelesen	{vpp}	{obj}

Fig. 3. A dependency tree for the German sentence

The left half of Fig. 3 shows a dependency tree for the sentence

(2) Ein Buch hat der Student gelesen.

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: the task is always one where we are given a number of tree (or graph) fragments, and we must plug them together (under constraints) to obtain a complete assembly. We call this class of problems *graph configuration problems*.

For such problems, it is often convenient to represent the plugging of a fragment w into another w' by means of a directed labeled edge $w{-}\ell{\rightarrow}w'$ which makes explicit that a resource of type ℓ supplied by w' is being matched with a corresponding need in w.

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

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

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

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

By taking advantage of the constraint programming support available in a language such as Mozart/Oz, this formal characterization of finite labeled trees can be given straightforwardly a 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, $|\ell(w)| = 1$ for a resource offered exactly once, etc... This is how we can model subcategorization in the application to dependency parsing.

4 Lexicalized Configuration Problems

The view presented sofar, where we are given a fixed number of fragments to be plugged together into a fully configured assembly, is not yet sufficient to capture realistic tasks. In the surface realization problem, there may be several alternative ways to verbalize the same semantic element. Similarly, in the syntax analysis problem, one word may have several readings, alternative subcategorization frames, alternative linearization constructions.

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

The challenge is now to adapt the constraint-based approach outlined earlier to gracefully handle lexical ambiguity.

4.1 Selection constraints

Let's consider again the dependency parsing application. As described in the previous section, for a given set V of words, we can (a) model the possible dependency trees as the solutions of constraint system (3), and (b) we can enforce subcategorization 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 selection for the given words. Instead, we would like to take advantage of constraint propagation to avoid having to make non-deterministic choices.

What we want is an underspecified representation of the lexical entry that is ultimately chosen in a form that integrates well in a constraint-based formulation. This is the purpose of the *selection constraint*:

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

Its 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 integer variables (FD) and integer set variables (FS). This support can easily be lifted over feature structures as follows:

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

and, in this manner, can apply to 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 I.

5 Multi-dimensional Configuration Problems

Sofar, we have informally introduced the notion of lexicalized graph configuration problems and suggested how they can be encoded into systems of contraints that are adequately supported by corresponding constraint technology.

However, when modeling language, we must contend with multiple descriptive levels (e.g. syntax, linear precedence, predicate/argument structure, information structure, etc...). Each is concerned with a different aspect of linguistic description, yet they are not independent: they interact and constrain each other.

Our notion of lexicalized configuration problems is an excellent tool for modeling each individual descriptive level, but, as we said, these levels are not independent. For this reason, we now propose a generalized approach where we have several configuration problems, all simultaneously constrained by the same lexical entries.

This is what we call a *multi-dimensional configuration problem*. For each descriptive level, there is one dedicated dimension. Every lexical entry now contains a sub-lexical entry for each dimension. In this manner, each lexical entry simultaneously constrains all dimensions. Furthermore, we also permit *inter-dimensional principles*, i.e. global constraints that relate one dimension with another. In this manner, they are not merely coupled through the lexicon, but also by linguistically motivated well-formedness relations.

Figure 4 illustrates the selection constraint operating simultaneously on 3 dimensions over 3-dimensional lexical entries.

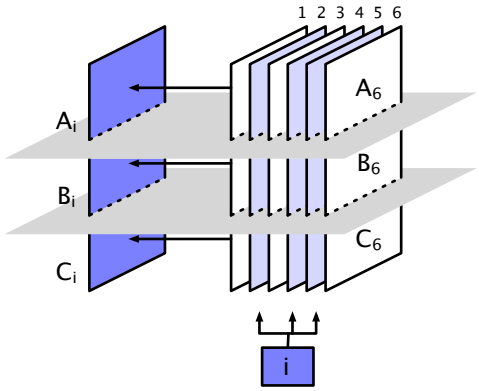


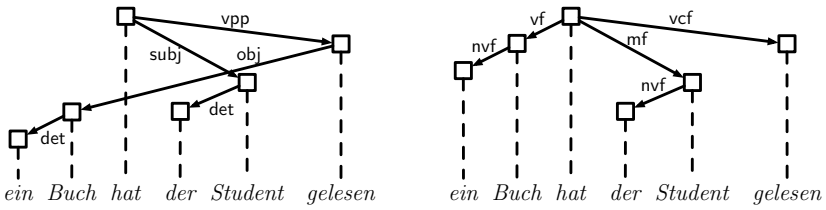
Fig. 4. Multi-dimensional selection

An important methodological motivation for the generalisation to multi-dimensional graph configuration problems is the desire to obtain a modular and scalable framework for modeling linguistic phenomena. We now illustrate the idea with the dependency analysis of word order phenomena in German. Consider again the parsing example sentence from Section 2:

(5) Ein Buch hat der Student gelesen.

It illustrates object topicalisation in German. Starting from the "canonically" ordered sentence *Der Student hat ein Buch gelesen,* it 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 – but a far more natural and perspicuous account is obtained when separating constituency and linear precedence, and describing word order variation as a relation between those two structures. Fig. 5 shows the analysis of the sentence using Topological Dependency Grammar (TDG, [8]) which dedicates one dimension to *immediate dominance* (ID) and another to *linear precendence* (LP).

Each dimension has its own set of well-formedness conditions (*principles*): both ID and 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

Fig. 5. Topicalisation

The parsing task of Topological Dependency Grammar 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 licensed ID tree, T_{lp} is a licensed LP tree, the two trees share the same node set V, and the climbing relation holds between them. Section 6 describes XDG which is an extended version of this model with additional dimensions for deep syntax, predicate/argument structure, and scope.

How does the multi-dimensional graph model relate to other approaches? With various *multi-stratal* formalisms, like Lexical Functional Grammar (LFG) [9] and Meaning-Text Theory (MTT) [10], it shares the idea of distinguishing several representational structures. In contrast to these formalisms, however, it does not presuppose a *layered* representation (where only adjacent layers can share information directly), nor does it assume that information can only flow in one direction: multi-dimensional principles can connect arbitrary dimensions. However, 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 as it is obtained in *mono-stratal* formalisms like HPSG [11].

6 Extensible Dependency Grammar

In this section, we introduce Extensible Dependency Grammar (XDG) [12], our flagship instance of a lexicalized multi-dimensional configuration problem. XDG is a generalization of TDG. It supports the use of arbitrary many dimensions of linguistics representation, and of arbitrary *principles*[3] regulating the well-formedness conditions. XDG was devised to be maximally general, and allow the grammar writer the freedom to decide how to split up linguistic modeling into separate dimensions.

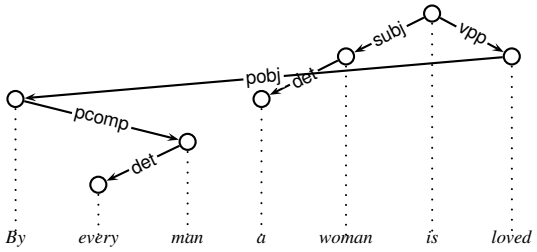
6.1 Example

As an illustrative example, we propose a five-dimensional grammar and illustrate it on the following example, an English passive construction paraphrasing the ubiquitous linguistic example "Every man loves a woman":

By every man, a woman is loved. (6)

In the following, we give a quick tour through the five dimensions of our example grammar to see what aspects of the linguistic analysis of this sentence they cover.

ID dimension The ID dimension (where ID stands for *immediate dominance*) was already introduced for TDG, and represents the linguistic aspect of *grammatical function*. We display the ID analysis of the example below:

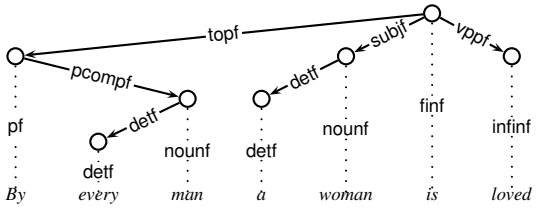


(7)

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". Moreover, "by" is the prepositional object (pobj) of "loved", "man" the prepositional complement (pcomp) of "by", and finally "every" the determiner of "man".

[3] Currently instantiated from a library of parametric principles.

LP dimension The LP dimension (LP stands for *linear precedence*) represents the linguistic aspect of *word order*, and has also already been introduced in the preceding section for TDG. On the LP dimension, the edge labels are names for word positions[4]:

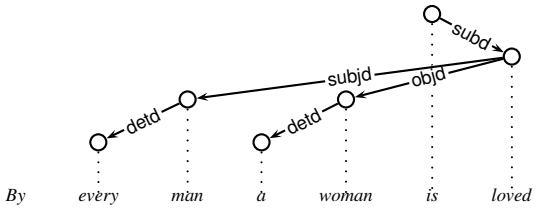


(8)

Here, the finite verb "is" is the root. "by" is in the topicalization position (topf), "woman" in the subject position (subjf), and "loved" in the verbal past participle position (vppf). Furthermore, "man" is in the prepositional complement position of "by" (pcompf), and "every" in the determiner position of "man" (detf). Similarly, "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, e.g. that a determiner in the determiner position (edge label detf) of a noun must precede the noun (node label nounf).

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



(9)

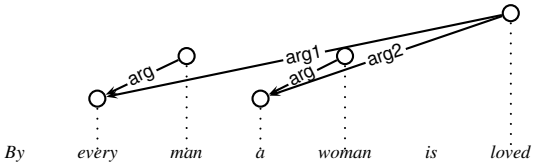
Here, "loved" is subordinated to "is" (edge label subd). "man" is the deep subject (subjd) of "loved", and "woman" the deep object (objd). "every" is the

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

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

determiner of "man", and "a" 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" the prepositional object of "loved", the DS analysis mirrors the underlying predicate-argument structure much more closely. Here, "woman" is the deep object of "loved", whereas "man" is its deep subject.

PA dimension The PA dimension (PA for *predicate-argument structure*) represents semantic predicate-argument structure, or, in terms of logic, *variable binding*. The idea is that quantifiers such as the determiners "every" and "a" introduce variables, which can then be bound by predicates (e.g. common nouns or verbs):

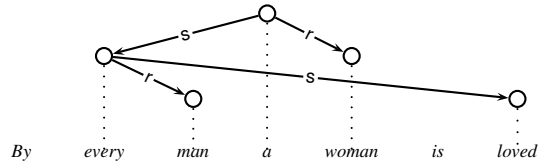


(10)

In the example analysis, think of the quantifier "every" as introducing variable x and "a" as introducing the variable y. Now, "man" binds the x, and "woman" the y. Finally, the x is the first argument of the predicate expressed by "loved", and y is the second argument. I.e. the PA analysis represents a flat semantics in the sense of [13], where the semantics of a sentence is represented by a multiset of first order predicates, and scopal relationships are completely omitted:

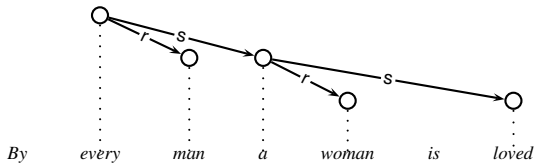
$$\{every(x), man(x), a(y), woman(y), love(x, y)\} \tag{11}$$

SC dimension The SC dimension (SC for *scope structure*) represents semantic scope structure. Contrary to [13], who advocates a semantic representation which completely omits scopal relationships, we follow of Minimal Recursion Semantics (MRS) [3] and have both: the PA dimension represents a flat semantics, and the SC dimension the scopal relationships. Here, e.g. quantifiers such as "every" and "a" have a restriction and a scope:



(12)

"a" has "woman" in its restriction (edge label r), and "every" in its scope (edge label s), and "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)

6.2 Principles and the lexicon

XDG describes the well-formedness conditions of an analysis by the interaction of *principles* and the *lexicon*. The principles stipulate restrictions on one or more of the dimensions, and are controlled by the feature structures assigned to the nodes from the lexicon. The principles are drawn from an extensible *principle library*. The principle 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 principle. Dimension i must be a tree. In the example above, we use this principle on the ID, LP and SC dimensions.

DAG principle. Dimension i must be a directed acyclic graph. We use this principle on the DS and PA dimensions, which need not necessarily be trees.

Valency principle. For each node on dimension i, the incoming edges must be licensed by the in specification, and the outgoing edges by the out specification. This is a key principle in XDG, and used on all dimensions. It is also lexicalized (cf. the lexical entries for TDG in the preceding section).

Order principle. For each node v on dimension i, the order of the daughters depends on their 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 ("a") precede nouns ("woman").

Projectivity principle. Dimension i must be a projective graph. We use this principle on the LP dimension to ensure that LP trees do not have crossing edges.

Climbing principle. The climbing principle is two-dimensional, and allows us to constrain the relation between two dimensions. It stipulates that dimension i must be flatter than dimension j. We use it to state that the LP dimension (8) must be flatter than the ID dimension (7).

Linking principle. The linking principle relates two dimensions i and j, and in particular allows us to specify how semantic arguments must be realized in the syntax. It is lexicalized, and we use it to stipulate e.g. that the first argument (arg1) of "loved" on the PA dimension (10) must be realized by the deep subject (subjd), and the second argument (arg2) by the deep object (objd) on the DS dimension (9).

Contra-dominance principle. The contra-dominance is also two-dimensional. We use it to constrain the relation between the two semantic dimensions PA and SC, in particular to stipulate that the semantic arguments of verbal predicates (on the PA dimension) must dominate them on the SC dimension. For instance, the first semantic argument of "loved" on the PA dimension (in (10), this is the determiner "every" of the NP "every man") must dominate (be an ancestor of) "loved" on the SC dimension (12) and (13).

6.3 Parsing and generation

Parsing and generation with XDG grammars is done using constraint solving by the *XDG solver*. XDG solving has a natural reading as a constraint satisfaction problem (CSP) on finite sets of integers, where well-formed analyses correspond to the solutions of the CSP [7]. We have implemented the XDG solver with the Mozart/Oz programming system [14], [15].

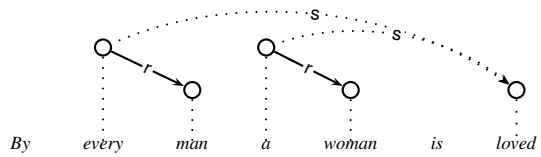
XDG solving operates on all dimensions concurrently. This means that the solver can infer information about one dimension from information on any other, e.g. by the multi-dimensional principles (climbing, linking, contra-dominance). For instance syntactic information can trigger inferences in semantics, and vice versa.

Because XDG allows us to write grammars with completely free word order, XDG solving is an NP-complete problem [6]. This means that the worst-case complexity of the solver is exponential at present. However, the behaviour of the solver on NL grammars is excellent in practice. Constraint propagation is both fast and very effective, and permits to enumerate solutions with few or no failures.

6.4 Underspecification

Similar to MRS and also CLLS, XDG supports the use of *underspecification*. 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" (labeled r) and from "a" to "woman" (also labeled r) are already determined, i.e. we know already 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 dominate the verb "loved" (as indicated by the dotted "dominance edge"). 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 goes one step further and allows to underspecify *any* of its dimensions, i.e. not only the scopal but also the syntactic dimensions. This can be used e.g. to compactly represent PP-attachment ambiguities.

7 Conclusion

We proposed a notion of *lexicalized multi-dimensional configuration problems* as a metaphor and a practical constraint-based approach for a wide range of tasks in computational linguistics, including semantic assembly, surface realization and syntactic analysis, and how it can be used to integrate them. We then presented XDG as an instance of this approach, and showed how to use it to integratively handle syntax and semantics of natural language. We think that multi-dimensional lexicalized configuration problems can play an important role in the future, as an overarching framework for computational linguistics research, on the theoretical and on the algorithmic side. For instance, research on configuration problems for semantic assembly have already yielded highly efficient algorithms for satisfiability and enumeration of dominance constraints [16]. For surface realization, an efficient algorithm was presented in [6]. At the moment, we are working on making the integrated processing of syntax and semantics in XDG more efficient.

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. 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
10. Mel'čuk, I.: Dependency Syntax: Theory and Practice. State Univ. Press of New York, Albany/USA (1988)
11. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago/USA (1994)
12. Debusmann, R., Duchier, D., Koller, A., Kuhlmann, M., Smolka, G., Thater, S.: A relational syntax-semantics interface based on dependency grammar (2004)
13. Trujillo, I.A.: Lexicalist Machine Translation of Spatial Prepositions. PhD thesis, University of Cambridge, Cambridge/USA (1995)
14. 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
15. Mozart Consortium: The Mozart-Oz website (2004) http://www.mozart-oz.org/.
16. Fuchss, R., Koller, A., Niehren, J., Thater, S.: Minimal recursion semantics as dominance constraints: Translation, evaluation, and analysis. In: Proceedings of ACL 2004, Barcelona/ESP (2004)