Spotting and Improving Modularity in Large Scale Grammar Development

Simon Petitjean

LIFO, University of Orleans

Abstract. XMG (eXtensible MetaGrammar) is a metagrammar compiler which has already been used for the design of large scale Tree Adjoining Grammars and Interaction Grammars. Due to the heterogeneity in this field (different grammar formalisms, different languages, etc), a particularly interesting aspect to explore is modularity. In this paper, we discuss the different spots where this modularity can be considered in a grammar development, and its integration to XMG.

1 Introduction

Nowadays, a lot a applications have to deal with languages and consequently need to manipulate their descriptions. Linguists are also interested in this kind of resources, for study or comparison. For these purposes, formal grammars production has became a necessity. Our work focuses on large scale grammars, that is to say grammars which represent a significant part of the language.

The main issue with these resources is their size (thousands of structures), which causes their production and maintenance to be really complex and time consuming tasks. Moreover, these resources have some specificities (language, grammatical framework) that make each one unique.

Since a handwriting of thousands of structures represents a huge amount of work, part of the process has to be automatized. A totally automatic solution could consist in a acquisition from treebanks, which is a widely used technique. Semi automatic approaches are alternatives that give an important role to the linguist: they consist in building automatically the whole grammar from information on its structure. The approach we chose is based on a description language, called metagrammar [1]. The idea behind metagrammars is to capture linguistic generalization, and to use abstractions to describe the grammar.

The context that initially inspired metagrammars was the one of Tree Adjoining Grammars (TAG) [8]. This formalism consists in tree rewriting, with two specific rewriting operations: adjunction and substitution. An adjunction is the replacement of an internal node by a an auxiliary tree (one of its leaf nodes is labelled with $\star$ and called foot node) with root and foot node having the same syntactic category as the internal node. A substitution is the replacement of a leaf node (marked with $\downarrow$) by a tree with a root having the same syntactic category as this leaf node. The principle is to apply these operations to a set of elementary trees to match the sentence we want to parse. TAG is said to have a extended domain of locality, because those operations (especially adjunction) and the depth of the trees allow to represent long distance relations between nodes: two nodes of the same elementary tree can after derivation end up at an arbitrary distance from each other. Here, we will only manipulate LTAG (lexicalized-TAG), which means each elementary tree is associated with at least one lexical element.

What can we do to lower the amount of work implied by the conception of the grammar ? Let us take a look at some rules:

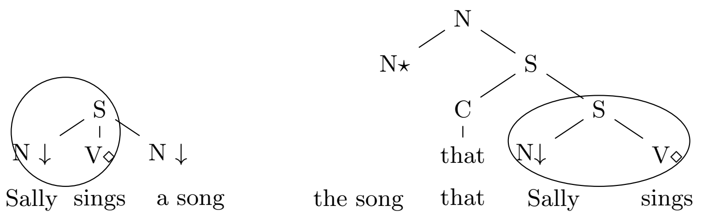


Fig. 1. Verb with canonical subject and canonical or extracted object

Those two trees share some common points: part of the structure is the same (the subject is placed before the verb in both circled parts), and the agreement constraints, given in feature structures associated to nodes (not represented here), are similar. This kind of redundancy is one of the key motivations for the use of abstractions. These abstractions are descriptions of the redundant fragments we can use everywhere they are needed.

Metagrammars are based on the manipulation of those linguistic generalizations. They consist in generating the whole grammar from an abstract description, permitting to reason about language at an abstract level. The metagrammatical language we will deal with here is XMG (eXtensible MetaGrammar) [1], introduced in [4]. A new project, XMG-2 [2], started in 2010 to achieve the initial goal of the compiler, extensibility, which has not been realized yet: XMG-1 only supports tree based grammars (two formalisms, Tree Adjoining Grammars and Interaction Grammars), and includes two levels of description, the syntactic one and the semantic one. Our goal is to go towards two levels of modularity: we want it to be possible to assemble a grammar in a modular way, thanks to a metagrammar assembled in a modular way.

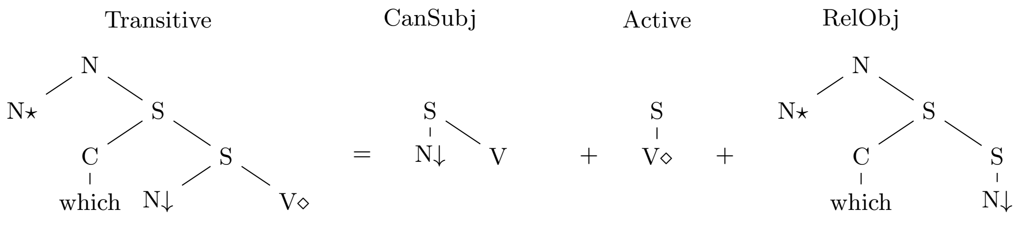
We will begin pointing out the modularity on the grammar side in section 2. In section 3, we will focus on a new level of modularity, a metagrammatical one. In section 4, we will give an overview of what has been done, and what remains to be done. Finally, we will conclude and give some perspectives.

2 Assembling grammars in a modular way

XMG consists in defining fragments of the grammar, and controlling how these fragments can combine to produce the whole grammar. The following figure shows the intuition of the combination of fragments to produce a tree for transitive verbs. It is done by combining three tree fragments, one for the subject (in its canonical form, that

[1]`https://sourcesup.cru.fr/xmg/`
[2]`https://launchpad.net/xmg`

we noticed redundant previously), one for the object (relative) and one for the active form.



To build a lexicon, the metagrammar is first executed in an indeterministic way to produce descriptions. Then these descriptions are solved to produce the models which will be added to the lexicon.

2.1 The control language and the dimension system

The main particularity of XMG is that it allows to see the metagrammar as a logical program, using logical operators.

The abstractions (possibly with parameters) we manipulate are called classes. They contain conjunctions and disjunctions of descriptions (tree fragments descriptions for TAG), or calls to other classes. This is formalized by the following control language:

$$\begin{array}{rcl} \mathit{Class} & := & \mathit{Name}[p_1,\ldots,p_n] \rightarrow \mathit{Content} \\ \mathit{Content} & := & \langle \mathit{Dim}\rangle\{\mathit{Desc}\} \ \mid\ \mathit{Name}[\ldots] \ \mid\ \mathit{Content} \vee \mathit{Content} \\ & & \mid\ \mathit{Content} \wedge \mathit{Content} \end{array}$$

For example, we can produce the two trees of the figure 1 by defining the tree fragments for canonical subject, verbal morphology, canonical object and relativized object, and these combinations:

$$\begin{array}{rl} \mathit{Object} & \rightarrow \mathit{CanObj} \ \vee\ \mathit{RelObj} \\ \mathit{Transitive} & \rightarrow \mathit{CanSubj} \ \wedge\ \mathit{Active} \ \wedge\ \mathit{Object} \end{array}$$

This part of metagrammar says that an object can either be a canonical object or a relative object, and that the transitive mode is created by getting together a canonical subject, an active form and one of the two object realizations.

Notice that descriptions are accumulated within dimensions, which allow to separate types of data. Sharing is still possible between dimensions, by means of another dimension we call interface. In XMG's TAG compiler for example, the *syn* dimension accumulates tree descriptions while the *sem* dimension accumulates predicates representing the semantics. Each dimension comes with a description language, adapted to the type of data it will contain. For each type of description we need to accumulate, we have to use a different description languages. The first version of XMG provides a tree description langague (for TAG or Interaction Grammars) associated with the *syn* dimension and a language for semantics associated with the *sem* dimension.

A tree description language

For trees in TAG, we use the following tree description language:

$$\begin{aligned} \mathit{Desc} \; := \; & x \rightarrow y \;\mid\; x \rightarrow^{+} y \;\mid\; x \rightarrow^{*} y \;\mid\; x \prec y \;\mid\; x \prec^{+} y \;\mid\; x \prec^{*} y \;\mid\; x[f{:}E] \\ & \mid\; x(p{:}E) \mid\; \mathit{Desc} \;\wedge\; \mathit{Desc} \end{aligned}$$

where x and y are node variables, $\rightarrow$ and $\prec$ dominance and precedence between nodes ($^{+}$ and * respectively standing for transitive and reflexive transitive closures). ':' is the association between a property p or a feature f and an expression E. Properties are constraints specific to the formalism (the fact that a node is a substitution node for example), while features contain linguistic information, such as syntactic categories, number or gender.

When accumulated, the tree description in the syntactic dimension is still partial. The TAG elementary trees that compose the grammar are the models for this partial description. They are built by a tree description solver, based on constraints to ensure the well-formedness of the solutions. XMG computes minimal models, that is to say models where only the nodes of the description exist (no additional node is created). Here is a toy metagrammar, composed of three description classes (representing canonical subject, relative object, active form) and one combination class (transitive mode):

$$\begin{aligned} \mathit{CanSubj} \rightarrow & \langle \mathit{syn} \rangle \{ (s_1[\mathit{cat} : S] \rightarrow v_1[\mathit{cat} : V]) \wedge (s_1 \rightarrow n_1(\mathit{mark} : \mathit{subst})[\mathit{cat} : N]) \\ & \wedge (n_1 \prec v_1) \} \\ \mathit{RelObj} \rightarrow & \langle \mathit{syn} \rangle \{ (n_2[\mathit{cat} = N] \rightarrow n_3(\mathit{mark} = \mathit{adj})[\mathit{cat} = N]) \wedge (n_2 \rightarrow s_2[\mathit{cat} = S]) \\ & \wedge (n_3 \prec s_2) \wedge (s_2 \rightarrow c) \wedge (s_2 \rightarrow s_1[\mathit{cat} = S]) \wedge (c \prec s_1) \\ & \wedge (c \rightarrow \mathit{wh}[\mathit{cat} = \mathit{wh}]) \wedge (s_1 \rightarrow n_1[\mathit{cat} = n]) \} \\ \mathit{Active} \rightarrow & \langle \mathit{syn} \rangle \{ (s_1 \rightarrow v_2[\mathit{cat} : V]) \} \\ \mathit{Transitive} \rightarrow & \mathit{CanSubj} \wedge \mathit{RelObj} \wedge \mathit{Active} \end{aligned}$$

The minimal models for the classes named CanSubj, Active and Object are the trees with matching names on the previous figure. The tree Transitive is a minimal model for the description accumulated in class Transitive.

A language for semantics

To describe semantics, we use another description language, which is:

$$\mathit{SemDesc} \; := \ell : p(E_1, ..., E_n) \mid \neg\ell : p(E_1, ..., E_n) \mid E_i << E_j \mid E$$

where ℓ is a label for predicate p (of arity n) and $<<$ is a scope-over relation for dealing with quantifiers. To add binary relations to the semantic dimension, we can use a class of this type:

$$\mathit{BinaryRel}[\mathit{Pred}, X, Y] \rightarrow \langle \mathit{sem} \rangle \{ \mathit{Pred}(X, Y) \}$$

When instantiated with $\mathit{Pred}{=}\mathit{love}$, $X{=}\mathit{John}$, $Y{=}\mathit{Mary}$, calling the class $\mathit{BynaryRel}$ accumulates the predicate $\mathit{love}(\mathit{John}, \mathit{Mary})$.

2.2 Principles

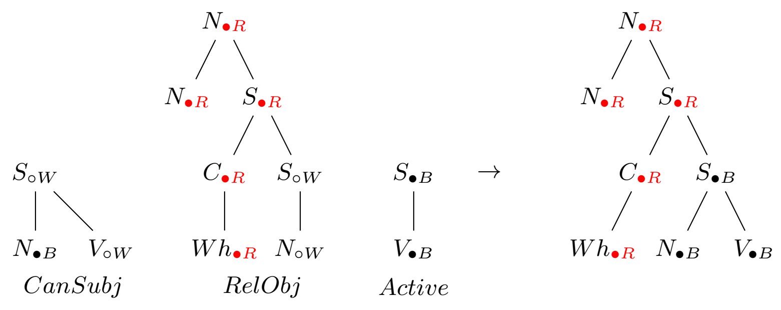
Some additional sets of constraints we call principles are available. Their goal is to check some properties in the resulting models of the compilation, they are consequently dependent from the target formalism. For example, in TAG, the color principle is a way to forbid some fragments combination, by associating colors to each node.

When unifying nodes, their colors are merged: a red node must not unify, a white node has to unify with a black node, creating a black node, and a black node can only unify with white nodes. The only valid models are the ones in which every node is colored either in red or black. The following table shows the results of colors unifications.

	$\bullet_{B}$	$\bullet_{R}$	$\circ_{W}$	$\perp$
$\bullet_{B}$	$\perp$	$\perp$	$\bullet_{B}$	$\perp$
$\bullet_{R}$	$\perp$	$\perp$	$\perp$	$\perp$
$\circ_{W}$	$\bullet_{B}$	$\perp$	$\circ_{W}$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Fig. 2. Unification rules for colors.

For example, if we consider our previous example, the colored trees of the metagrammar are the following:


$N_{\bullet R}$
$N_{\bullet R}$ $S_{\bullet R}$
$S_{\circ W}$ $C_{\bullet R}$ $S_{\circ W}$ $S_{\bullet B}$ $\rightarrow$
$N_{\bullet B}$ $V_{\circ W}$ $Wh_{\bullet R}$ $N_{\circ W}$ $V_{\bullet B}$
CanSubj
RelObj
Active
$N_{\bullet R}$
$N_{\bullet R}$ $S_{\bullet R}$
$C_{\bullet R}$ $S_{\bullet B}$
$Wh_{\bullet R}$ $N_{\bullet B}$ $V_{\bullet B}$

The tree description solver (ignoring the colors) will produce models where the nodes labelled S of CanSubj and Active unify with any of the two nodes labelled S in RelObj, where the nodes labelled V do not unify, etc. But when filtering with the colors principle, the only remaining model is the one of the right, which is linguistically valid, contrary to the others.

We can also cite the rank principle: we use it to add constraints on the ordering of nodes in the models of the description. In French for example, clitics are necessarily ordered, so we associate a rank property to some nodes, with values that will force the right order.

3 Assembling metagrammars in a modular way

The main aim of the XMG-2 project is to make it possible for the linguist to design new metagrammatical scopes, that can accomodate any linguistic theory. A simple way to realize this ambition is to provide a set of bricks the user can pick to build the compiler he needs. Those bricks could be used to design new dimensions, with new description languages or new principles.

3.1 A modular architecture

XMG compiler comes with a modular processing chain. Most of this chain is a standard compiling chain, including a tokenizer for the metagrammar, a parser, an unfolder, etc.

The particularity of XMG is to make it possible to chose the modules that suits the best his metagrammar. By this mean, descriptions accumulated in different dimensions can be handled differently. For example, the end of the processing chain for TAG is a tree description solver, that builds the grammar's elementary trees from the descriptions accumulated in the syntactic dimension. The user can chose the kind of output the compiler will produce: he can interactively observe the grammar he produced, or produce an XML description of the grammar. This description can be used by a parser (for example TuLiPA [9] [3] for TAG, or LeoPar [4] for IG).

3.2 Representation modules

As we wish to build a tool which is as universal as possible, being independent from the formalism is a priority. To achieve this goal, we need to be able to describe any type of structure into XMG. We saw the dimension system was useful to separate syntax from semantics. It could also be used to separate tree descriptions from constraints based descriptions, as long as we have a dedicated dimension, with a dedicated description language.

In [6], description languages for two formalisms, namely Lexical Functional Grammars (LFG) and Property Grammars (PG), are proposed. Here, we will focus on Property Grammars, because they differ from TAG in many aspects. PG are not based on tree rewriting but on a local constraints system: the properties. A property concerns a node and applies constraints over its children nodes. One of the interesting aspects of PG is the ability to analyse non grammatical utterances. When parsing a utterance, its grammaticality score is lowered at every violated property. Here, we will consider these six properties:

Obligation	$A : \triangle B$	at least one B child
Uniqueness	$A : B!$	at most one B child
Linearity	$A : B \prec C$	B child precedes C child
Requirement	$A : B \Rightarrow C$	if a B child, then also a C child
Exclusion	$A : B \not\Leftrightarrow C$	B and C children are mutually exclusive
Constituency	$A : S$	children must have categories in S

A real size PG consists in a inheritance hierarchy of linguistic constructions. These constructions are composed of feature structures and a set of properties. Variables are manipulated on both sides, and can be used to share data between them. Figure 3 represents a part of the hierarchy built in [7] for French. The V-n construction of the figure says that in verbs with negation in French, negation implies the presence of an adverb *ne* labelled with category $Adv - ng$ (*ne*) and/or an adverb labelled with category $Adv - np$ (like *pas*). We also have a uniqueness obligation over these adverbs,

[3] `https://sourcesup.cru.fr/tulipa/`

[4] `http://wikilligramme.loria.fr/doku.php?id=leopar:leopar`

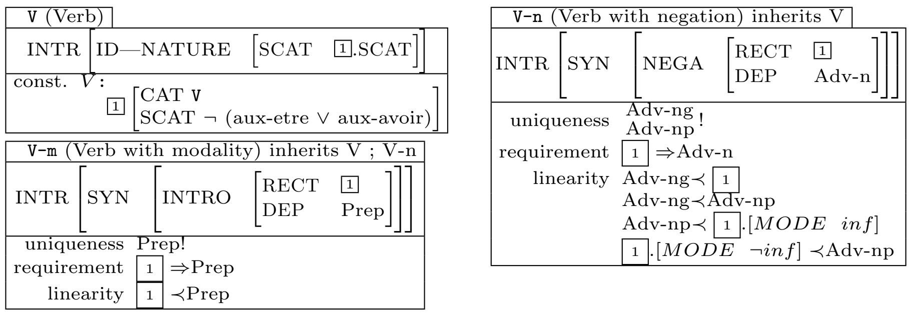


Fig. 3. Fragment of a PG for French (basic verbal constructions)

and an linear order must be respected (*ne* must come before *pas*). When the mode of the verb is infinitive, the verb must be placed after the adverbs.

To describe a PG, we need to be able to represent encapsulations, variables, feature structures, and properties. We can notice that XMG classes can be seen as encapsulations, and that variables and features structures were already used for TAG descriptions. Considering that, the XMG description language for PG can be formalized this way:

$$\begin{aligned} Desc_{PG} &:= x = y \mid x \neq y \mid [f{:}E] \mid \{P\} \mid Desc_{PG} \wedge Desc_{PG} \\ P &:= A : \triangle B \mid A : B! \mid A : B \prec C \mid A : B \Rightarrow C \mid A : B \not\Leftrightarrow C \mid A : B \end{aligned}$$

where x, y correspond to unification variables, $=$ to unification, $\neq$ to unification failure, : to association between the feature f and some (possibly complex) expression E, and $\{P\}$ to a set of properties. Note that E and P may share unification variables.
The translation of the linguistic construction for V-m in XMG would be:

$$\begin{aligned} V{-}m \;\; \rightarrow \;\; & (Vclass \vee V{-}n) \wedge \langle PG \rangle \{[\mathsf{INTR}{:}[\mathsf{SYN}{:}[\mathsf{INTRO}{:}[\mathsf{RECT}{:}X, \mathsf{DEP}{:}\texttt{Prep}]]]] \\ & \wedge (V : \texttt{Prep}!) \wedge (V : X \Rightarrow \texttt{Prep}) \wedge (V : X \prec \texttt{Prep})\} \end{aligned}$$

Here, inheritance is made possible by calls of classes. The control language even allows to do disjunctive inheritance, like it happens in class V-m. The end of the compilation process for PG will differ from TAG's one. We don't need any solver for descriptions, the accumulation into PG dimension is the grammar. To get the properties solved for a given sentence, the solution is to use a parser as a post processor for the compiler.

Nevertheless, including a specific representation module to the compiler can be seen as an ad-hock solution. That is why allowing the linguist to build his own description language (for example, choosing to use feature structures, dominance relations between nodes, open unification, etc), would be an essential feature.

3.3 Principle bricks

The notion of principles defined in XMG was too restrictive for our aims. Their specificity for the target formalism, for example, is incompatible with the multi-formalism ambition. An interesting way to handle principles is the one of [3], both allowing the linguist to create his own principles or to use a subset of the ones already defined. An example is the tree principle, which states that the solution models must be trees. What we aim to provide is a meta-principles library: generic and parametrizable principles the user can pick and configure. For example, the color principle provided for TAG could be an implementation of a generic polarity principle, parametrized with the table of figure 2. Another example of meta-principle is called unicity and was already implemented in XMG-1. It is used to check the uniqueness of a specific attribute-value pair in each solution, and thus is not specific to any linguistic theory.

3.4 Dynamic definition of a metagrammar

To build his own metagrammatical scope, one should only have to select the dimensions he needs and the properties he wants to check on them. Building a dimension would consist in picking bricks out from a library to create a new description language. With this feature, a user could redefine the property grammars description we proposed earlier. The advantage here is that the specific part of the compiler is written automatically, and new features could be added just for experiments. Defining the principles would just consist in taking meta-principles out from the library and instantiate them.

Building a metagrammar compiler in this way allows to deal with a large range of linguistic theories, or even to quickly experiment while creating a new grammar formalism.

4 Current state of the work

XMG project started in 2003 with a first tool, that has been used to produce large TAG grammars for French [2], German [10] and English, and a large Interaction Grammar for French [11]. The compiler was written is Oz/Mozart, a language which is not maintained any more and not compatible with today's architectures (64 bits). It was also important to restart from scratch, in order to build a compiler more in adequation with its ambitions : modularity and extensibility.

Consequently, a new implementation started in 2010, in YAP (Yet Another Prolog) with bindings with Gecode for constraints solving. XMG-2 is currently the tool used for modeling the syntax and morphology of Ikota, a bantu language [5], and is getting close to total compatibility with the previous large metagrammars. It also includes a dimension for basic property grammar description. The work focuses now on a parser generator which, from a description of a description language, produces the parser rules for this language. The first application could be the dynamic generation of a language dedicated to morphologic descriptions. We also wish to implement quickly some generic principles, beginning with the tree principle.

5 Conclusion

In this paper, we showed how modularity, together with a metagrammatical approach, eases the development of a large scale grammar. This modularity is essential for reaching the main goal of XMG, that is to say extensibility. Getting to that means taking a big step towards multi-formalism and multi-language grammar development, and then offers new possibilities for sharing data between different types of grammar, or even for comparing them.

Now, what we would like to create is a way to express the definition of dimensions and meta-principles. This could begin by formalizing a description language for description languages. We also aim to provide more checking tools to the user, beginning with the type checking of the properties and the feature structures we manipulate in a lot of grammar formalisms.

References

1. Candito, M.: A Principle-Based Hierarchical Representation of LTAGs. In: Proceedings of COLING 96. Copenhagen, Denmark (1996)
2. Crabbé, B.: Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints. Ph.D. thesis, Université Nancy 2 (2005)
3. Debusmann, R.: Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description. Ph.D. thesis, Saarland University (4 2006)
4. Duchier, D., Le Roux, J., Parmentier, Y.: The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In: Proceedings of the 2nd Oz-Mozart Conference, MOZ 2004. Charleroi, Belgium (2004)
5. Duchier, D., Magnana Ekoukou, B., Parmentier, Y., Petitjean, S., Schang, E.: Describing Morphologically-rich Languages using Metagrammars: a Look at Verbs in Ikota. In: Workshop on "Language technology for normalisation of less-resourced languages", 8th SALTMIL Workshop on Minority Languages and the 4th workshop on African Language Technology. Istanbul, Turkey (2012), http://hal.archives-ouvertes.fr/hal-00688643/en/
6. Duchier, D., Parmentier, Y., Petitjean, S.: Cross-framework grammar engineering using constraint-driven metagrammars. In: CSLP'11. Karlsruhe, Allemagne (2011), http://hal.archives-ouvertes.fr/hal-00614661/en/
7. Guénot, M.L.: Éléments de grammaire du français pour une théorie descriptive et formelle de la langue. Ph.D. thesis, Université de Provence (2006)
8. Joshi, A.K., Schabes, Y.: Tree adjoining grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Springer Verlag, Berlin (1997)
9. Kallmeyer, L., Maier, W., Parmentier, Y., Dellert, J.: Tulipa - parsing extensions of tag with range concatenation grammars (June 2009), first Polish-German Workshop on Research Cooperation in Computer Science
10. Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., Dellert, J.: Developing a tt-mctag for german with an rcg-based parser. In: LREC. ELRA (2008)
11. Perrier, G.: A French Interaction Grammar. In: RANLP. Borovets, Bulgaria (2007), http://hal.inria.fr/inria-00184108/en/