

Towards computational verification of Self-organising Logic of Structures

Maciej Piasecki and Adam Radziszewski

Institute of Applied Informatics, Wrocław University of Technology, Wrocław
Poland

Abstract

Self-organising Logic of Structures (SLS), a semantic representation language of high expressive power, seems attractive from a theoretical point of view. To check its usefulness for natural language understanding systems, an implementation is required. This article presents our efforts aimed at computational verification of SLS. We discuss some computational problems and propose a solution based on Constraint Logic Programming. An experimental implementation of a model-checker for dynamic formulae is presented. It verifies the satisfaction of formulae representing simple sentences with multiple quantifiers.

Keywords: SLS, semantics, generalised quantifiers, CLP

1 Introduction

In order to let a machine manipulate the meaning of natural language text the machine is usually provided with an algorithm of automated translation of text into a formal language — a metalanguage, with well-defined semantics and inference mechanisms. The expressive power of the metalanguage determines a subset of natural language expressions that can be dealt with. *Self-organising Logic of Structures* (SLS) is a *semantic representation language* proposed by Piasecki (2003b) to solve some of the theoretic problems faced by the approaches originating from DRT (Kamp and Reyle, 1993). Among the distinctive features of SLS, lies a relatively concise and coherent description of natural language quantification as well as handling the discourse semantics without syntactic indexing of noun phrases (a comparison can be found in Piasecki (2005)). Those features seem attractive for natural language processing, including question answering and dialogue systems.

This work is aimed at the computational verification of SLS, a first step towards its practical application. Its semantics is stated in purely declarative manner — the algorithmic issues must be considered. We will show that an effective implementation of SLS's unique approach to quantification is possible. It has been achieved by introducing some technical changes allowing for pruning the search space and optimisation of several operators together. The scope of the implementation has been limited to judging if a given formula is satisfied within a specified model or finding such variable binding that makes the formula satisfied. Only single sentences are analysed (anaphoric resolution is not implemented yet).

2 SLS in a nutshell

As SLS has already been described, e.g. (Piasecki, 2003a), we present only some selected features of SLS, those related directly to the experimental implementation scope. SLS was proposed to provide a consistent formalism of strictly compositional discourse representation. Every occurrence of a nominal phrase (NP) may introduce new entities into the domain of discourse or refer to previously mentioned ones (by anaphoric pronouns). SLS enables those entities to be dynamically resolved without resorting to external aids.

SLS is a logical language defined on the basis of a simple extensional type logic. The meaning of a sentence or multi-sentence discourse is expressed by a *dynamic formula*. Basically, it reflects the observation that after hearing a sentence, the hearer's state of knowledge may change. This state change is identified with the dynamic formula interpretation, i.e. relation on states. What follows, is that given some input state a dynamic formula may relate it to multiple output states. This way non-deterministic interpretation of formulae is introduced.

The state is constituted by:

1. A set of *activated DRs* corresponding to the groups of objects represented by the subsequent NPs.
2. A set of links between DRs (representing anaphoric links).
3. A partial function assigning sets of objects to the activated DRs only.

To give a general picture of states, we will introduce an example. Let us assume a situation described by the statement *John walks*. A possible state k describing this situation contains one discourse referent, d_1 with a set of one entity assigned (*John*). The sentence *He is blind* could be expressed with the following formula:

$$T_1(i)(j) = \downarrow (i)(k_1); \uparrow (\nabla(k_1), \text{male_pron}, k_1, k_2) \\ ; \text{blind}(\#(\nabla(k_2), k_2)) \wedge k_2 = j$$

Formula T_1 can be perceived as a relation transforming input state into a set of output states such that:

1. another DR is activated (\downarrow),
2. it is linked anaphorically to one of the previously activated DRs and denotes the same set of entities; its value is constrained to the set of beings that may be represented by the masculine pronoun *he* (\uparrow),
3. the new DR is constrained to the class of entities being blind.

Applying T_1 to initial state k will result in a set of output states. If John belongs to the class *blind*, there will be one output state with two DRs: d_1 and d_2 , d_2 being linked to d_1 , and both assigned the same singleton set containing John only. Formula T_1 will be then *satisfied* in the state k . More generally, a dynamic formula T is satisfied in the state i if and only if there is a state j , such that $T(i)(j)$ and every DR activated in j has a non-empty set of objects.

Interpretation of NP activates a new DR; it is assigned *some* value (multiple output states) by the SLS ' \downarrow ' operator. Additionally, referential and anaphoric NP generate links between the 'new' DR and some of the appropriate activated DRs — the \uparrow operator. The existential presupposition is represented by the constraint

on the number of links. For a referential NP only one link should be created, pointing to some DR activated in the input context. The *Property of Indeterministic Interpretation of Reference* (PIIR) constrains that in each state at least one from the DRs linked to the ‘new’ DR must be assigned the identical value.

In SLS it is assumed that one of the NP meaning aspects is quantification. The SLS interpretation of NP quantification was based on the formalisation of Bellert’s graph-based representation of quantification (1989) and the concept of quantifier varieties of van der Does (1994). The interpretation of NPs in a sentence is formulated in terms of *possible structures of the relation* they can express; practically all possible readings of NPs, concerning quantification and plurality can be expressed in one dynamic formula.

Determiner representation is based on a generalised quantifier (GQ) in a way following (Barwise and Cooper, 1981). A *proto-quantifier* is a functor taking a set and generating a set of its subsets (a GQ), satisfying a cardinality condition, e.g.:

$$\mathbf{all}(A) = \{A\}$$

$$\mathbf{no}(A) = \{\emptyset\}$$

$$\mathbf{at\ least\ } \mathbf{n}(A) = \{X : |X| \geq n \wedge X \subseteq A\}$$

Proto-quantifier generates a set of sets (a GQ) based on the input set. This way, a situation corresponding to a one-place verb can be described.

In SLS-based interpretation, a proto-quantifier is applied to the value of the corresponding DR or the denotation of NP predicate. This corresponds to the division of proto-quantifiers into *relative* (marking out a set from a whole class of beings, as in the phrase *most men*) and *absolute* (*three men*). Moreover, proto-quantifiers are subjected to *variety modifiers* of van der Does (1994) — type-lifting functors transforming resulting GQs into sets of sets of *collections* (modelled as sets). This further level of nesting accounts for the way entities are grouped together. *Collective* variety assumes that all objects are acting together; in *distributive* every objects acts separately; in *neutral* all combinations are allowed.

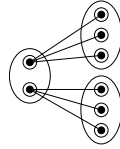
In multi-quantifier sentences the relation between GQs contributes an important part to the meaning. SLS introduces *cardinality dependencies* between pairs of GQs, defining the way in which the collections belonging to a pair must be connected in the output relation. There are two cardinality dependency operators, each operating on two modified GQs (i.e. sets of sets of collections), generating a structure of relation Z between collections (definitions are in the appendix).

Cardinality dependency ($\mathbf{Q}_1 < \mathbf{Q}_2$) Z satisfies the following conditions:

1. for each collection x from \mathbf{Q}_1 there exists a set of collections Y belonging to \mathbf{Q}_2 , such that x is related to every collection from the set Y ,
2. for each *set of collections* Y from \mathbf{Q}_2 there exists a collection x from \mathbf{Q}_1 , such that x is related to every collection in set Y .

Cardinality independency ($\mathbf{Q}_1 : \mathbf{Q}_2$), a less restricted structure built from two sets of collections, respectively: one belonging to \mathbf{Q}_1 and one to \mathbf{Q}_2 , i.e. Z belongs to $\mathbf{Q}_1 : \mathbf{Q}_2$ if there is $X \in \mathbf{Q}_1$ and $Y \in \mathbf{Q}_2$ such that $Z \subseteq X \times Y$.

The dependency operators overgenerate, thus ‘illegal’ structures need to be filtered out. The final configurations are sets of n -tuples drawing the shape of

FIGURE 1: Cardinality dependency **two** < **three**

n -place relation (n is the number of quantifiers). The ultimate satisfaction check is performed by the *intersection operator*, which verifies if the final configurations share at least one configuration with the verb denotation.

The ‘illegal’ structures correspond to the violation of one of the two structural properties, which discard the readings impossible in natural languages. The *Property of the Root of Distribution* (PRD) states that for any \mathbf{Q}_2 if there is no \mathbf{Q}_1 such that \mathbf{Q}_2 is cardinally dependent on \mathbf{Q}_1 , directly or by a dependency chain, then every configuration of collections may contain collections from *exactly one* set from \mathbf{Q}_2 . In such cases, \mathbf{Q}_2 will be referred to as the *root of distribution*. The *Property of Maximal Distribution* (PMD) states that *either* there is no \mathbf{Q}_1 , such that $\mathbf{Q}_1 < \mathbf{Q}_2$ *or* there is \mathbf{Q}_1 , such that $\mathbf{Q}_1 < \mathbf{Q}_2$ and the number of sets of collections from \mathbf{Q}_2 involved in the output relation is not greater than the overall number of collections from \mathbf{Q}_1 involved in the output relation. Consider the reading $\mathbf{Q}_{baboon} < \mathbf{Q}_{banana}$ of the sentence *Two baboons ate three bananas* (fig. 1). PRD requires that there is no more than one group of baboons. PMD rules out the readings with the overall number of bananas greater than six.

3 Computational feasibility problems

The baseline approach was a naïve implementation of SLS operators, however the search space grows up dramatically with the introduction of every new operator. First of all, to check if a formula is satisfied, we have to find out whether for the given input state there is an output state such that every activated DR has a non-empty denotation. A naïve approach would require checking output states one by one. As any occurrence of NP introduces a new discourse referent with some value assigned (‘↓’ operator) the number of output states for it is equal to the number of all possible subsets of the domain ($2^{|D_e|}$), e.g. for a small domain of 10 entities, we have 1024 output states for the first formula. The following NP formulae multiply the number of states further.

The reference operator (‘↑’) poses a similar problem. Whenever there are more candidates for an anaphoric antecedent of given DR, PIIR comes into operation. It basically ensures that at least one of the candidates must have the same denotation as the given referent. Evaluating a dynamic formula from left to right, when we come across a ‘↑’, we know no further constraints on the denotation of referents. It may happen that so far assumed constraints are mutually exclusive and we have to backtrack and proceed with another arbitrarily chosen variant. The inconsistency may be found after having performed a considerable amount of useless calculations.

Proto-quantifiers also present implementation challenges. Let us assume a sen-

tence with two quantifiers, both based on a 10-element input set. If each proto-quantifier generates a set of 4-element subsets, we result in two families of 120 sets; thus, we have to generate at least 120×120 configurations selecting different subsets from both quantifiers. What is more, if the sets on which the proto-quantifier is applied are denotations of DRs, we have to face another non-deterministic choice; a typical constraint on DR is inclusion in a set representing a class of beings (e.g. students), which means that we have to examine all possible subsets of given class. This sort of combinatorial problem would arise when evaluating an SLS formula for simple sentences like *Four students ate four sandwiches*.

4 Processing set constraints

The starting point was the following observation: to judge the satisfiability of a given formula there is no need of explicit generation of all the structures output by SLS operators; we should rather strive for early rejection of inconsistencies and limiting the search space. What is more, the nature of dynamic formulae suggests that they can be easily expressed in terms of *variables* and successively applied *constraints*, e.g. constraining the denotation of a DR to given class of objects or constraining the cardinality of subsets generated by a proto-quantifier.

We adopted the paradigm of *Constraint Programming*, e.g. (Apt, 2003). A dynamic formula can be represented as a *Constraint Satisfaction Problem* \mathcal{P} :

$$\mathcal{P} = \langle \mathcal{C}; M_1 \in 2^{D_e}, \dots, M_k \in 2^{D_e} \rangle$$

The set of constraints \mathcal{C} expresses the semantics of the dynamic formula, while the domain expressions (M_1, \dots, M_k) define the denotations of all DRs.

Let us consider the domain of SLS operators. The domain of discourse D_e is a finite set, but to handle DRs and GQs, we need quite an expressive constraint language dealing with the domain 2^{D_e} , i.e. constraints on subsets.

The CSP systems are usually realised as libraries implemented in a particular programming language referred to as the *host language*; the choice of the host language determines the extent to which the CSP algorithms can be controlled. A very popular host language for dealing with CSPs is Prolog — *Constraint Logic Programming* (CLP) (Apt, 2003). Prolog enables easy symbol manipulation and integration with parsing modules. Dover *et al.* (1999) propose a CLP language that allow for constraints on sets of finite-domain entities, named CLP(\mathcal{SET}). The number of similar systems seems to be quite limited. Two systems which seem suitable for implementation of SLS are the Oz language (Müller, 2001) and Setlog (Rossi, 2000). The latter was chosen for its easy integration with Prolog code and good code readability.

Setlog is a language of the CLP(\mathcal{SET}) paradigm implemented in Prolog. It handles constraints on finite sets of atomic or set elements; more precisely, *hereditarily finite hybrid sets* (Dover *et al.*, 1999). The sets can be introduced extensionally (by enumerating) or intensionally (constructs like $\{e : p(e)\}$). All standard set operators, such as equivalence, inclusion, membership, cardinality, are supported as constraints, and may be applied to constrain DR denotations, e.g. the membership operator can be used either to constrain a class of sets to those having at least

given element or to find an element satisfying given conditions. There are also more complex constraints, e.g. Restricted Universal Quantifiers, that are formulae of the form $\forall X \in S.G(X)$, where X is a logic variable, S is a set, and $G(X)$ is any Setlog goal (analogous to Prolog goal) containing X . RUQs can be nested.

Setlog integrates well with Prolog. Setlog goals can be called from within Prolog and vice-versa. Some complex routines that need explicit control can be written in Prolog.

5 Scheme of implementation

The task is to verify given formulae for their satisfaction in the specified model. The scope has been limited to SLS constructs related to verb predicates, their arguments and quantification structures. Although current version supports only distributive variety of quantification, extending it to remaining varieties seems quite straightforward and will be discussed later. The system was implemented in Sicutus Prolog 3.12 (Carlsson, 1995) and Setlog and consists of:

1. The main module loading the Setlog interpreter and the remaining modules.
2. Model checker and quantification structure analyser — written in Setlog.
3. Prolog predicates for early quantification structure analysis (PRD, PMD).
4. The model specification consisting of Prolog clauses defining entity classes and verb predicate denotations (by explicit enumerating ‘situations’).

Dynamic formulae are expressed in Setlog as first order predicates. Formulae containing references to DRs are implemented as predicates whose arguments are set variables. The ‘non-empty denotation’ condition was incorporated into the representation of formulae, i.e. they are required to contain this sort of constraints, e.g. a formula activating a new DR and constraining its denotation to a class:

$$F_1(i)(j) = \downarrow (i)(k) ; student(\#(\nabla(k), k)) \wedge k = j$$

is represented as:

```
f1(M) :-
    forall(E in M, class(E, student))
    & M neq {}.
```

Expressive power of Setlog makes the implementation of proto-quantifiers quite straightforward, e.g. the implementation of the absolute quantifier **exactly n**:

```
quant(Quant, Spec, DR) :-
    set(Quant)
    & ( integer(Spec)
        & subset(Quant, DR)
        & size(Quant, Spec) ).
```

The predicate `quant` is satisfied when the set `Quant` satisfies the cardinality specification `Spec` with DR value of `DR`. This means that `Quant` belongs to the GQ generated from the set `DR`, e.g. the query `quant(Q, 3, {1, 2, 3, 4})` would be answered with four alternatives: `Q = {2, 3, 4}`, `Q = {1, 3, 4}`, `Q = {1, 2, 4}`, `Q = {1, 2, 3}`.

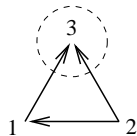
Many computational problems were avoided by analysing quantification structures instead of generating them. In our solution the search starts from the model — situations from the verb predicate denotation are consecutively checked against given quantification description. A considerable optimisation was achieved by early verification of quantification structure that allows for rejection of many situations before the proper analysis. Given the verb predicate, DRs (implemented as logical variables) and quantification structure description (proto-quantifier types and the dependency relations between them), the model checker works as follows:

1. The input parameters are fed to the intersection operator implementation. Any situation from the verb denotation that along with given quantification description violates PRD or PMD is immediately rejected.
2. The situation (a set of n -tuples) is projected onto sets of pairs corresponding to each pair of quantifiers.
3. Every set of pairs is verified by the matrix operator implementation. It analyses pairs of quantifiers under the assumption that the steps 1 and 2 have already been performed. The implementation incorporates all possible combinations of dependency relations between two quantifiers and optimises each combination separately. If each set of pairs is consistent with the model, the formula is judged to be satisfied.

The optimisation of dependency operators was based on the observation that there are seven possible mutual combinations of operators and roots of distribution. One-way dependency ($\mathbf{Q}_1 < \mathbf{Q}_2$ or $\mathbf{Q}_1 > \mathbf{Q}_2$), either with the dominating quantifier being a root or not altogether result in four combinations. Bidirectional dependency ($\mathbf{Q}_1 <> \mathbf{Q}_2$) with two roots or without any (single root is not possible in this case) give two combinations. The last case is that of mutual independency ($\mathbf{Q}_1 : \mathbf{Q}_2$). There are no more possible combinations (Radziszewski, 1997). For instance, the bidirectional dependency with two roots is evaluated as follows:

```
% operator <> (bidirectional) with PRD (roots at Q1 and Q2)
% requirements: Z1 = Z12|1, Z2 = Z12|2
mtxZ12(Z12, Z1, Z2, QSpec1, QSpec2, dep_on_both, 1, 1) :-
    Z12 neq {}
    & quant(Z1, QSpec1, Z1) % Z1,Z1 satisfy QSpec
    & quant(Z2, QSpec2, Z2) % (root <> root, so 1 set at both sides)
    & forall(E1Z1 in Z1, % Z12 equals the full product Z1 x Z2
            forall(E1Z2 in Z2, [E1Z1,E1Z2] in Z12) ).
```

The early analysis of quantification structures finds all distribution roots and checks against PMD. The root finding algorithm was implemented in Prolog; it simply checks all the quantifiers and marks those not dependent (directly or indirectly). If there is no root, the situation is rejected. PMD was implemented indirectly, following the idea that the number of entities introduced into a relation by a quantifier is constrained by the quantifiers it is dependent on, e.g. in Fig. 1, (**two** < **three**), the number of entities from Q_{three} will be constrained to $2 \times 3 = 6$. More generally, the number of objects from a quantifier is constrained by the chain of quantifiers it is depending on. In case of non-determinacy, the weaker constraint is binding (Radziszewski, 1997). Consider Fig. 2, \mathbf{Q}_3 is the root, therefore it is

FIGURE 2: Dependency schema for $Q_1 < Q_2$, $Q_1 > Q_3$, $Q_1 > Q_2$

constrained only by itself. Q_1 is directly dependent on the root. In case of Q_2 there are two paths; we have to traverse both and calculate the constraints' integer values and find the weaker one (the greater value).

6 Evaluation

The implemented system was thoroughly tested and no inconsistency with the formal definitions was found. The tests consisted of specifying different models, posing queries and checking if the answers were consistent with the SLS formal definitions as well as the language intuition. The experiments were divided into four parts: one-quantifier formulae (testing the implementation of proto-quantifiers), two-quantifier formulae (testing cardinality dependencies), multi-quantifier formulae (testing PMD, PRD and the overall correctness) and experiments finding all possible configurations of roots of distribution (checking the rationale of PRD optimisation). For instance, consider the following denotations of the verb *give*:

$$\begin{aligned} \text{give1} &= \{\langle \text{prof}_1, p_1, st_1 \rangle, \langle \text{prof}_1, p_2, st_2 \rangle\} \\ \text{give2} &= \{\langle \text{prof}_1, p_1, st_1 \rangle, \langle \text{prof}_1, p_2, st_2 \rangle, \langle \text{prof}_2, p_3, st_3 \rangle, \langle \text{prof}_2, p_4, st_4 \rangle\} \end{aligned}$$

give1 corresponds e.g. to the sentence *Profesor dał dwóm studentom po piórze* (*The professor gave two students-DISTRIB-a pen*¹). In *give2* there are two such professors. A query corresponding to the description $Q_{1prof} < Q_{1pen}$, $Q_{1pen} > Q_{2stud}$, $Q_{1prof} : Q_{2stud}$ can be posed with the following Setlog goal:

```
isect3(give,prof,pen,stud,X1,X2,X3,1,1,2,dep_on_1,dep_on_2,indep).
```

With the model above, we get the following answer (the DRs are instantiated):

```
Satisfied with give1
X1 = {prof1},
X2 = {pen1,pen2},
X3 = {st1,st2}
```

The further experiments showed that *give1* can be described either with the quantifiers 1–1–2 or with 1–2–2. *give2* can be described with the sequence 2–1–2, while 2–2–2 is not applicable. Interestingly, it is consistent with the language intuition: there seems to be no clause employing only the quantifiers **two** that would correspond to *give2*, e.g. it certainly cannot be described by *Two professors gave*

¹The Polish preposition *po* marks overtly the distributive reading, resulting in an interpretation closer to *The professor gave a pen to each of the students*.

two pens to two students. This is an important advantage of the implementation: some language phenomena can be observed.

It has also been shown that there are seven possible schemas of root placement in three-quantifier sentences, three of which violate PRD (Radziszewski, 1997). The implemented strategy seems a good heuristics: it allows for early rejection of those situations without proceeding with the quantification structure analysis.

Moreover, some mistakes in the formal definitions of SLS operators have been spotted. The corrected definitions are given in the appendix.

7 Conclusions and further work

The achieved results in verifying the SLS correctness or analysing natural language phenomena are encouraging but obviously much further work on extending the implementation scope is necessary. The implementation provided also a better insight into the theory. The present implementation supports only distributive variety of quantification. However, extending it also to collective and neutral readings seems quite straightforward. Currently, situations are described by entity tuples; a possible fix assumes tuples of *sets* of entities (for collections). To avoid further manipulations on such complex structures, we propose a separate initial stage of checking the conformance of a collection with given variety and, when passed, flattening the structures to plain entity tuples (as already implemented). Although this extension seem feasible, it needs to be formally checked and implemented.

A Appendix: the corrected SLS definitions

1. $\| < \| := \lambda \mathbf{Q}_1. \lambda \mathbf{Q}_2. \lambda Z. \begin{cases} Z \neq \emptyset \\ \forall X \in Z|_1. \exists \mathbf{X} \in \mathbf{Q}_1. \begin{cases} X \in \mathbf{X} \\ \forall X \in \mathbf{X}. \exists \mathbf{Y} \in \mathbf{Q}_2. \forall Y \in \mathbf{Y}. \langle X, Y \rangle \in Z \end{cases} \\ \forall Y \in Z|_2. \exists \mathbf{Y} \in \mathbf{Q}_2. \begin{cases} Y \in \mathbf{Y} \\ \exists \mathbf{X} \in Z|_1. \forall Y \in \mathbf{Y}. \langle X, Y \rangle \in Z \end{cases} \end{cases}$
 $\| : \| := \lambda \mathbf{Q}_1. \lambda \mathbf{Q}_2. \lambda Z. \exists S_1 \in \mathbf{Q}_1. \exists S_2 \in \mathbf{Q}_2. [Z \subseteq S_1 \times S_2]$
 Dependency operators ' $<$ ', ' $:$ ' are of type $((((et)t)t)((((et)t)t), ((et)^2t)))$.
2. $\|\mathbf{M}^2\| := \lambda o_{1,2}. \lambda o_{2,1}. \lambda \mathbf{Q}_1. \lambda \mathbf{Q}_2. \lambda W. \begin{cases} W \subseteq (o_{1,2}(\mathbf{Q}_1, \mathbf{Q}_2) \cap o_{2,1}(\mathbf{Q}_2, \mathbf{Q}_1)) \\ \mathbf{PRD}^2(\langle o_{1,2}, o_{2,1} \rangle, \langle \mathbf{Q}_1, \mathbf{Q}_2 \rangle, W) \\ \mathbf{PMD}^2(\langle o_{1,2}, o_{2,1} \rangle, \langle \mathbf{Q}_1, \mathbf{Q}_2 \rangle, W) \end{cases}$
 $W|_{i,j}$ is the projection of Cartesian product on i -th and j -th set.
3. $\|\mathbf{M}^i\| :=$ a generalisation of $\|\mathbf{M}^3\|$, where \mathbf{M}^i is of type $\underbrace{op(\dots op}_{i \times i - i}(\underbrace{(((et)t)t}_{i}(((et)t)t) \dots ((et)^i t) \dots))}$
4. $\|\mathbf{PMD}^i\| := \lambda \mathbf{S}_O. \lambda \mathbf{S}_Q. \lambda W. \mathbf{S}_O = \langle o_1, \dots, o_{i \times i - i} \rangle \wedge \mathbf{S}_Q = \langle \mathbf{Q}_1, \dots, \mathbf{Q}_i \rangle \wedge$

$$\wedge \left(\forall n. (1 \leq n \leq i) \longrightarrow \left(\begin{array}{l} \neg \exists m. \left\{ \begin{array}{l} 1 \leq m \leq i \\ m \neq n \\ o_{num}(m, n) = \langle \cdot \rangle \end{array} \right. \\ \vee \\ \exists m. \left\{ \begin{array}{l} 1 \leq m \leq i \\ m \neq n \\ o_{num}(m, n) = \langle \cdot \rangle \\ |W|_m| \geq \left| \lambda \mathbf{Y}. \left(\begin{array}{l} \mathbf{Q}_n(\mathbf{Y}) \wedge \\ \exists X. \forall Y \in \mathbf{Y}. \\ \langle X, Y \rangle \in W|_{m,n} \end{array} \right) \right| \end{array} \right. \end{array} \right) \right)$$

References

- Krzysztof APT (2003), *Principles of Constraint Programming*, Cambridge University Press.
- J. BARWISE and R. COOPER (1981), Generalized Quantifiers and Natural Language, *Linguistics and Philosophy*, 4:159–219.
- Irena BELLERT (1989), *Feature system for quantification structures in natural language*, Foris Publ.
- Mats CARLSSON (1995), *SICStus Prolog User's Manual*, ISBN 91-630-3648-7, Swedish Institute of Computer Science.
- Agostino DOVIER, Carla PIAZZA, Enrico PONTELLI, and Gianfranco ROSSI (1999), Sets and Constraint Logic Programming, in *DPS'99, Workshop on Declarative Programming with Sets*.
- Hans KAMP and Uve REYLE (1993), *From Discourse to Logic: Introduction to Modeltheoretic Semantics in Natural Language, Formal Logic and Discourse Representation Theory, Vol. 1*, Kluwer Academic Publishers, Dordrecht.
- Tobias MÜLLER (2001), *Constraint Propagation in Mozart*, Ph.D. thesis, Universität des Saarlandes, Fachrichtung Informatik, Saarbrücken.
- Maciej PIASECKI (2003a), Dynamic Representation of Nominal Anaphora without Syntactic Indexing, in H. Bunt ET AL., editor, *Proc. of the 5th International Workshop on Computational Semantics*, pp. 321–336, Tilburg University Computational Linguistics and AI Group, Tilburg.
- Maciej PIASECKI (2003b), *Język modelowania znaczenia polskiej frazy nominalnej*, Ph.D. thesis, Faculty of Computer Science and Management, Wrocław University of Technology, Wrocław.
- Maciej PIASECKI (2005), Discourse Interpretation Based on Dynamic Constraints, *Archives of Control Science*, 15(LI)(3):403–414.
- Adam RADZISZEWSKI (1997), *Analiza znaczenia wypowiedzi w języku polskim w oparciu o Samoorganizującą się Logikę Struktur*, Master's thesis, Faculty of Computer Science and Management, Wrocław University of Technology.
- Gianfranco ROSSI (2000), *{log} User's Manual — Version 3.3*, Dipartimento di Matematica, Università di Parma.
- Jaap VAN DER DOES (1994), *Applied Quantifier Logic*, Ph.D. thesis, ILLC, Univ. of Amsterdam, Amsterdam.