Identifying Objects in Object Determination Logic

Jérôme Cardot
LaLICC – UMR 8139 – CNRS
Université Paris IV Sorbonne
28 rue Serpente
75005 Paris, France

Jerome.Cardot@paris4.sorbonne.fr

Abstract

LDO provides a way of representing objects as built on concepts by two operators: an object builder and an object determinator. Objects built by help of these operators are more or less determinate, LDO giving a model of categorization; objects may accept several descriptions, so that recognizing different descriptions of a same object is actually an important task.

In order to compute identification of these objects, this paper introduces formal languages to represent them, so that they can be processed with either classical (automata, congruence) or newer (like $S-$languages) computer science methods. LDO needs this to model objects of various areas, such as lexical representation, NLP, OO programming...

Keywords: Object Determination Logic, ontology, automata, language, identification.

Introduction

Redundancy of information in the utterance is a general phenomenon. It may surge from repetition of words, of phrases, or from paraphrases…

As LDO (Object Determination Logic) is an intensional formalism designed to represent the building of objects through determinations, and may be used to represent the changes that affect an object along the utterance, a natural question is: how will LDO deal with repetitions, and particularly with all that redundancy.

Indeed, a correct treatment of the redundancy is at least necessary to recognize identical objects. So, in order to give a theory of identity in LDO, this paper studies equivalence between chains of determinations, which does not only involve redundancy, but also commutations of determinations, and other knowledge vehicled by the utterance.

The first section is this *introduction*; then section *LDO* reminds the basis of LDO. Section *LDO and languages* explains how methods of automata and formal languages can be exploited in LDO. Section *Repetition* studies the three main kinds of repetitions, which are *redundancy*, *refocus* and *iteration*, in terms of automata. Some pairs of determinations may commute and still denote the same compound determination, and some other pairs may not. Section *Commutation* studies the cases where commutation is avaible. Section *Global identification* deals with other kinds of knowledge, necessary in NLP and AI, like lexical and pragmatic knowledge, and synthesizes the different kinds of object identification. Section *Applications* shows applications and perspectives of the study developed here.

LDO

A first idea of LDO can be found in (Desclés 1986). The categorization system of LDO, grounded on the fundamental cognitive operation of *determination*, and the way how LDO deals with typical objects of categories, inspired by Rosch (Rosch 1975), have been presented in (Pascu & Carpentier 2002), and is more widely developped in (Desclés 2002); and the notion of contextual inferences in LDO was presented in (Freund *et al.* 2004).

Sources of LDO

The main choice in LDO is to build objects by their intension, that is, by ideas (or concepts) related with terms. The central role of determination in adding ideas in the intension of objects was already pointed out by (Arnauld & Nicole 1662)[1]:

> “ *La détermination est quand ce qu'on ajoûte à un mot général en restreint la signification, & fait qu'il ne se prend plus pour ce mot général dans toute son étendue, mais seulement pour une partie de cette étendue.* ”[2]

> “ *Determination is when something is added to a general word, restricts its meaning, so that it doesn't stand for the general word with all its scope, but only for a part of this scope.* ”

We take concepts as primitive elements, nouns and verbs coming later, built upon these concepts.

More formally, our vision of concepts is inspired by Frege's one: in (Frege 1879), a concept is an “ *unsaturated expression* ” (a function), and saturating such an expression with an object yields a content of thought.

[1]Usually called *Logique de Port-Royal*.

[2](Arnauld & Nicole 1662, I, VIII, p. 66).

Objects and determination in LDO

In LDO objects are built, and may be more or less determinate, by use of operators τ and δ acting upon concepts:

τ: builds, upon a notion f, τf, typical object f, the typical indeterminate object associated to notion f.

δ: builds, upon a notion f, δf, a determination, that is, an operator acting upon an object x and building a more determinate object $(\delta f)x$. So an object may be built by several determinations acting one after the other, and we'll write Δ for a determination chain.

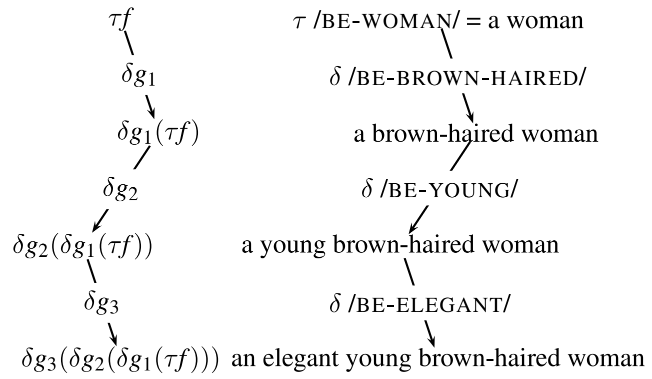
τf
δg_1
$\delta g_1(\tau f)$
δg_2
$\delta g_2(\delta g_1(\tau f))$
δg_3
$\delta g_3(\delta g_2(\delta g_1(\tau f)))$
τ /BE-WOMAN/ = a woman
δ /BE-BROWN-HAIRED/
a brown-haired woman
δ /BE-YOUNG/
a young brown-haired woman
δ /BE-ELEGANT/
an elegant young brown-haired woman

Figure 1: Determinations of objects.

The class of concepts is denoted by $\mathcal{F}$, the class of objects by $\mathcal{O}$.

LDO as a Leśniewskian ontology

The sole axiom of Leśniewskian system of ontology is the following (we give here the 1920 version of the axiom, longer but easier to read than the 1929 version). Our transcription follows Leśniewski's choice of writing single objects in capitals, and what may be several objects in lower case:

Axiom 1 (AO - 1920)
The object A is a if and only if:

i. there is an object B such that B is A;
ii. if the object B is A and the object C is A, then the object B is C;
iii. if an object B is A, then B is a.

$$\begin{aligned}(\forall Aa)(\varepsilon\{Aa\} \equiv (&\sim((\forall B)(\sim(\varepsilon\{BA\})))\\ &\wedge(\forall BC)((\varepsilon\{BA\}\wedge\varepsilon\{CA\})\Rightarrow\varepsilon\{BC\})\\ &\wedge(\forall B)(\varepsilon\{BA\}\Rightarrow\varepsilon\{Ba\})))\end{aligned}$$

Leśniewskian ontology involves ε as a prefixed primitive symbol, representing the *copula* (to be). In LDO we will introduce an analogue to ε, denoted by $\underline{\varepsilon}$ (infixed), in the following definition:

Definition 2 (Specification of an object)
$x\underline{\varepsilon}y$ if, and only if, there is a chain of determinations Δ, possibly empty – that is, Δ would be reduced to identity – such that $x = \Delta y$.
We read $x\underline{\varepsilon}y$ " x is a *specification* of *y ", or " x* is a *y ".*

With the specification relation we can characterize completely determinate objects:

Definition 3 (Completely determinate objects)
We say that an object x is completely determinate if, and only if, one of the two equivalent following properties is verified:

i. x is a fixed point of any applicable determination;
ii. from any two specifications of x, each one is a specification of the other one.

Denoting the class of completely determinate objects by $\mathcal{O}_{det}$, we have:

$$\forall A, A \in \mathcal{O}_{det} \equiv \forall BC((B\underline{\varepsilon}A \wedge C\underline{\varepsilon}A) \Rightarrow B\underline{\varepsilon}C)$$

We also denote by $\mathcal{O}_{ind}$ the class of objects which are not in $\mathcal{O}_{det}$, that is the class of not completely determinate objects, so we have $\mathcal{O} = \mathcal{O}_{ind} \cup \mathcal{O}_{det}$.

Interpreting here $\varepsilon\{Ab\}$ as " *A is a specification of b* ", we get the theorem:

Theorem 4 *Objects of LDO build a Leśniewskian system of ontology.*

Proof : Property *ii*) of definition 3 leads to the second of the three requisites in axiom 1:
$(\forall BC)((\varepsilon\{BA\} \wedge \varepsilon\{CA\}) \Rightarrow \varepsilon\{BC\})$.
The two other requisites are straightforward. □

The class of all specifications of τf (the scope of f) is denoted by Étendue f. Among these specifications, some objects are completely determinate: they constitute the extension of f, denoted by Extension f.

So we have Extension f = Étendue $f \cap \mathcal{O}_{det}$.

LDO and languages

Further in this article, our aim will be to recognize when the objects $\delta g_k \cdots \delta g_1 \tau f$ and $\delta g'_\ell \cdots \delta g'_1 \tau f'$ may be identified, in other words when $\delta g_k \cdots \delta g_1 \tau f = \delta g'_\ell \cdots \delta g'_1 \tau f'$. Indeed, we shall only examine the case $f = f'$, that is, we restrict the problem to identification of objects within Étendue f.

Intuitively, this is not really a restriction, because if the two objects $a = \delta g_k \cdots \delta g_1 \tau f$ and $a' = \delta g'_\ell \cdots \delta g'_1 \tau f'$ may be identified, there must be a common ancestor to τf and $\tau f'$, otherwise a and a' would simply be far too different from each other for the question of identifying them to be askable.

There is a simple order morphism between words of $\mathcal{F}^*$ (the free monoid language built on the alphabet $\mathcal{F}$), ordered by the prefix order, and objects of Étendue f, ordered by the converse of $\underline{\varepsilon}$:

$$\begin{aligned}\varphi :\ & \mathcal{F}^* \longrightarrow \mathcal{O}\\ & g_1 g_2 \cdots g_n \longmapsto \delta g_n \circ \cdots \delta g_2 \circ \delta g_1 \circ \tau f\end{aligned}$$

Recalling the example of figure 1, *an elegant young brown-haired woman*, described in Étendue /BE-WOMAN/, is: δ/BE-ELEGANT/ $\circ$ δ/BE-YOUNG/$\circ$ δ/BE-BROWN-HAIRED/ $\circ$ τ/BE-WOMAN/

This object of Étendue /BE-WOMAN/ will be represented by the word of $\mathcal{F}^*$:

/BE-BROWN-HAIRED/ /BE-YOUNG/ /BE-ELEGANT/.

So φ maps words of $\mathcal{F}^*$ to objects of Étendue$f$. We will say that a word $g_1 \cdots g_k$ refers in Étendue$f$ to the object $\delta g_k \cdots \delta g_1 \tau f$.

Definition 5 *Let x be an object of* Étendue$f$*. The set of all words of* $\mathcal{F}^*$ *referring to x in* Étendue$f$ *is called the language referring to x, and denoted by* $\mathscr{L}_x$.

$$\mathscr{L}_x = \varphi^{-1}\{x\} = \{m \in \mathcal{F}^*; \varphi(m) = x\}$$

Repetition

In this section we examine the link between repetition of determinations and the language referring to an object. We have to distinguish between three kinds of repetitions:

- pure redundancy: the repetition doesn't give any new information;
- refocus: the repetition focuses on the center of an idea;
- iteration: repetition makes one determination more, after the first determination (and changes the changed object).

Indeed, determination is a constructive operation on objects. Pure redundancy corresponds to an idempotent determination, refocus realizes idempotency after the second step, and with iteration the building goes on.

Repetition as redundancy

Let us examine the possibility of repeating one or all determinations, and still referring to the same object. We can transform the automata of figure 2, which recognizes only the word *abc*, to express for example that if *b* is repeated, the new word still refers to the same object as *abc* does. This transformation simply consists in adding a new arc, tagged with the empty word[3]. Such an arc will be written a 1−arc.

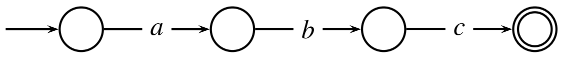

Figure 2: Automata recognizing the word *abc*

Figure 3 shows the transformed automata. Allowing repetitions of *b* is translated in terms of regular expression: the *b* of the initial form *abc* is replaced by bb^*, or by b^n in terms of formal languages.

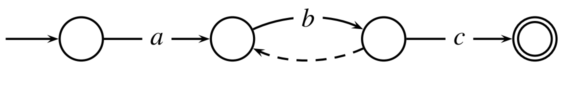

Figure 3: Automata recognizing the language ab^nc

[3]We make here the choice of denoting the empty word by 1, as does J.-M. Autebert in (Autebert 1994), to avoid another use of ε in this article. Arcs tagged by 1 will be dotted.

Now, if every repetition is allowed, we transform the automata by adding a return 1−arc for each arc of the initial automata. This leads to automata drawn figure 4.

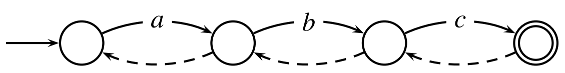

Figure 4: Automata recognizing all repetitions on the pattern *abc*

1−arcs are easily removed: each path tagged by $1^n\alpha$ is to be replaced by an arc tagged by α, with the same extremities than the initial path. Applied to the automata of figure 4, this leads to the one of figure 5.

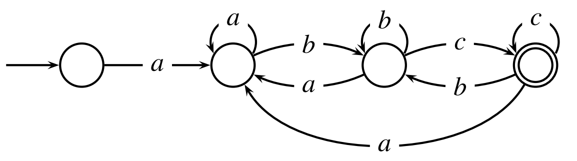

Figure 5: Automata without 1−arc, recognizing all repetitions on the pattern *abc*

This process builds, on a word, the language of all words obtained by repetitions of parts of the primer word. Although rational expressions are perfectly suitable to represent the modified language, the easier representation, even for studying properties of such languages, is that of automata with 1−arc.

A converse part of the problem is, from a given word, to eliminate all repetitions. We can try to do this with a rewriting rule like the following one: $uv^2w \longrightarrow uvw$.

Each application of this rule reduces the length of the word. So the relation associated to this rule is clearly *nœtherian*, that is, from a given word the graph of the relation does not show infinite path – in other words the rule can not be iterated infinitely.

Unfortunately this relation does not have the Church-Rosser property of convergence. To win the convergence, we have to use the following rule: $uvwvuvw \longrightarrow uvw$.

Hence we get:

Proposition 6 *The reduced word by the relation associated with the former rewriting rule is the shortest word of a language allowing all repetitions.*

Proof : The relation is nœtherian, convergent and length decreasing. □

Repetition as refocus

In natural language not all repetitions are redundant. For example, speaking of *a black, black dog*, we don't only refer to *a black dog*, but to a *a really black dog*, a dog that doesn't have even a white tuft at the end of its tail.

That is what we call *refocus*, and we will express it in terms of words of $\mathcal{F}^*$ by a rewriting rule: $aa \longrightarrow \mathbb{T}a$ where $\mathbb{T}a$ denotes " *be a typical a* ".

Of course we require a dictionary to express which concepts obey this rule, and more important, this rule must be applied before the reduction of repetitions presented above (see *Redundancy*), otherwise there wouldn't be any sequence *aa* left in the word to be transformed into $\mathbb{T}a$. And the rule has to be applied only once: once the object has been recognized as *a typical f*, it is not relevant to check again whether it is typical.

Let us recall that, given $V = \{a_1, a_2, \cdots, a_k\}$ a set of letters, or *alphabet* (written here in an alphabetic order), and $w \in V^*$ a word on that alphabet, the *Parikh vector* of w is the list $(|w|_{a_1}, \cdots, |w|_{a_k})$ where $|w|_{a_i}$ is the number of occurrences of a_i in w.

Then a more efficient way to recognize refocus is to compute the Parikh vector of the analyzed word: we just have to find letters appearing more than once in the word, and search them in the dictionary of refocusing. Once possible typicality has been recognized, remaining repetitions may be reduced, expressing redundancy.

Repetition as iteration

Some determinations may be applied and still change the object, even after several steps. Let us recall a stupid high school joke:

Example 7 *A scientist studies how fleas jump. He tells a flea to jump, and the flea jumps. He cuts one leg of the flea, and tells it to jump, and the flea jumps. He cuts one leg of the flea, tells it to jump, and the flea hardly jumps. He cuts one leg of the flea, tells it to jump, and the flea does not move. He concludes: when we cut legs of a flea, the flea goes deaf.*

As for *Refocus*, the right tool to treat that kind of repetition is to compute the Parikh vector of the analyzed word, storing the occurrence number of each letter.

Commutation

This section deals with the order of determinations applied to build an object. Clearly, if determinations were commutative (as we'll see later, some application domains of LDO may accept that), letters of a word referring to an object could be written in an arbitrary order, say for example an alphabetical order chosen on $\mathcal{F}$. In that case the set of letters of the word, possibly associated with the Parikh vector, would be enough to refer to the object.

This is not the general case: determinations do not commute in general, but some pairs of determinations may commute. Intuitively, if concepts are independent, then the associated determinations commute. For example, when speaking of *a big black dog* or of *a black big dog* we refer to the same object.

So we have to identify the objects that are identical because of some determination commutation.

Commutative and non-commutative determinations

The problem here is, knowing that δb and δc commute, to recognize that *abcd* and *acbd* refer to the same object. And still we may want to choose one of the two words to represent canonically the object.

This can be done, provided we define an alphabetic order on $\mathcal{F}$, by giving the following rule:

$cb \longrightarrow bc$ (where $b < c$ in that $\mathcal{F}$ alphabetic order).

Let us gather such commutative rewriting rules together in a set, a *Thue system* (Autebert 1994) that will be written κ (for commutation). Application of $\kappa-$rules is a nœtherian relation, decreasing the alphabetic order of the word without changing its length. The associated congruence identifies objects differing only by order of commuting determinations, and $\kappa-$rules lead to the canonic word representing the object.

$S-$languages

S. R. Schwer (Schwer 2001) presented the formalism of $S-$languages based on an alphabet X: the letters of the $S-$alphabet associated with X are the non-empty subsets of X. That $S-$alphabet is denoted by $\widehat{X}$, and an $S-$language is a language on the $S-$alphabet. She introduced that formalism in order to abstract unknown temporal information, that is to abstract possibilities in an order.

$S-$languages proved to be an efficient way of representing commuting determinations: in Étendue f, identifying objects that are identical because of some determination commutation is precisely abstracting an information in the order induced by determination.

Thus, if δb and δc commute, both words *abcd* and *acbd* may be abstracted into the same $S-$word $a \begin{pmatrix} b \\ c \end{pmatrix} d$.

This $S-$word gives an easy access to the class of words it abstracts, and to the canonic word representing the object, whereas mapping the class from the canonic word, although computable, is less intuitive.

Global identification

We still have to deal with other reasons of identification, but this will not require new kinds of tools. Then we have to mix all these methods.

Other knowledge

Concerning other kinds of knowledge leading to different ways of identifying objects, we shall treat them as we did for commutation: we build a different Thue system for each kind of knowledge (of course they could be merged together, but keeping them apart increases readability and maintenability).

These systems are:

λ for lexicon. Rules of this system are like:
/BE-RECTANGLE//BE-RHOMBUS/ $\longrightarrow$ /BE-SQUARE/

$\mathcal{U}$ for incompatibility. Up to this point, when writing a word of $\mathcal{F}^*$, we did not care whether the referred object in Étendue f exists or not. Incompatibility rules allow to treat impossible objects. We just have to add a symbol for non-existing objects, for example Λ, and an incompatibility rule will look like:
/BE-RAINY//BE-DOG/ $\longrightarrow \Lambda$.

π for pragmatics. Pragmatic rules allow to introduce knowledge about the real world (or about a fictive world we would want to represent).

Synthesis

Figure 6 shows how the different ways of identifying objects of Étendue f are chained: ψ for repetitions, κ for commutations, λ for lexicon, $\mathcal{U}$ for incompatibility and π for pragmatics.

$$\begin{tikzcd}
 & \mathcal{F}^*/\overset{*}{\underset{\psi,\kappa}{\longleftrightarrow}} \arrow[r, "\lambda"] & \mathcal{F}^*/\overset{*}{\underset{\psi,\kappa,\lambda}{\longleftrightarrow}} \arrow[dr, "\mathcal{U}"] & \\
\mathcal{F}^*/\overset{*}{\underset{\psi}{\longleftrightarrow}} \arrow[ur, "\kappa"] & & & \mathcal{F}^*/\overset{*}{\underset{\psi,\kappa,\lambda,\mathcal{U}}{\longleftrightarrow}} \arrow[d, "\pi"] \\
\mathcal{F}^* \arrow[u, "\psi"] \arrow[rrr, "\varphi"'] & & & \mathcal{O}
\end{tikzcd}$$

Figure 6: Diagram of objects as classes of $\mathcal{F}^*$

Applications

We present here two domains of possible applications of identity in LDO: applications to Computer Science, and applications to Natural Language Processing.

Model of OO-computation

In an OO-language where every class and every object is specified from a *general class*, the requisite we accepted in section *LDO and languages*, that is identifying objects, provided they have a common ancestor, is clearly fulfilled.

Then we can interpret OO-operations that build entities (classes or objects) of an OO-program as follows:

class specification is a determination, where the specification consists in defining a new method, a new attribute or in setting a class attribute;

class specification with overriding makes the more determinate class behave differently than the general class; it does not have the default behaviour, and may be considered as an atypical class;

multiple inheritance corresponds to the use of two different chains of determinations; if no conflict occurs, the two chains are independent and commute; if there is a conflict, one order between the two determinations may be more suitable than the other possible one;

instanciation of an object of a class is a determination; we even can consider that this determination gives to the object its memory address as an identificator, so that another instance of the same class cannot be identified with it;

setting an attribute is also a determination.

An object can not be atypical of the class it is an instance of, but through this class it may be atypical of an ancestor class.

So, the main structural operations on OO-entities have been modeled in terms of LDO. It is now possible to check the hierarchy of classes, trying to identify some abstract classes, in order to factorize some determinations (*i.e.* some specification of methods) and to optimize the hierarchy.

Determination and the lexicon

As we have shown in (Cardot 2003), LDO may be used to represent a structured lexicon. Indeed, as we showed with theorem 4, LDO follows the structure of Leśniewski's system of ontology (Leśniewski 1929), also called by Twardowski *the proper calculus of names*.

So the names are directly represented by LDO (see above *Other knowledge*), whereas the semantic of verbs may be represented by a scheme of their action on the terms they receive as arguments. This action is given in terms of determination, for example *die* transforms a subject with a determination (which can be implicit) δ/BE-ALIVE/ into a subject with the determination δ/NOT BE-ALIVE/.

Determination and text analysis

When objects are introduced in a text, they are not determinate; then the text brings more and more determinations. Determiners follow the same scheme, so when we speak of a new character we can say *a man*, and when we later refer to the same one, we have to use the definite form *the man*, otherwise *a man* would be understood as *another man*. Using an indeterminate noun phrase introduces a new object in the text.

So the different determiners – or quantifiers – in natural language act to restrict the scope of the noun, precisely as determinations act in LDO.

LDO, with a method of identifying objects along the text, allows to model how new informations build the object further. Thus the text may be seen as a list of constructive instructions about the objects it deals about.

Note that the information given along the text may be non-monotonic, and may therefore need some revision if a new determination conflicts with older ones.

Conclusion

We have developed in this article methods for computing a formal identity of objects in LDO. This step was necessary to get the benefit of the intensional constructive way objects are built in that system.

Hence LDO promises wide domains of application. The determination of objects is involved in structuring a lexicon, in understanding a text… We have shown a model of class hierarchies of OO programming languages, and this model may be adapted to any knowledge representation system allowing inheritance, or be used to check and improve class hierarchies.

We still have to emphasize that the way we compute identity between objects does not require that these objects are completely determinate: the system deals with general classes the same way than with determinate objects, so that general or abstract classes can be identified too.

Acknowledgements

Thanks are due to the referees whose suggestions helped in improving a former version of this article. As one of them suggested, I give more detailed information on the subject online, available at `http://www.lalic.paris4.sorbonne.fr/cardot/research/LDO/`

And I am particularly grateful to S. R. Schwer and J.-P. Desclés for numerous discussions about LDO and languages.

References

Arnauld, A., and Nicole, P. 1662. *La logique ou l'art de penser*. Paris: Vrin, 1993 edition.

Autebert, J.-M. 1994. *Théorie des langages et des automates*. Paris: Masson.

Cardot, J. 2003. Logique de la détermination d'objets: un formalisme pour la représentation des objets du discours. In *Actas del VIII^{vo} symposio internacional de comunicación social*, 115–120.

Desclés, J.-P. 1986. Implication entre concepts: la notion de typicalité. *Travaux de linguistique et de littérature* (XXIV).

Desclés, J.-P. 2002. Categorisation: a logical approach to a cognitive problem. *Journal of Cognitive Science* 3(2):85–137.

Frege, G. 1879. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle.

Freund, M.; Desclés, J.-P.; Pascu, A.; and Cardot, J. 2004. Typicality, contextual inferences and object determination logic. In *Proceedings of FLAIRS 2004*, 491–495.

Leśniewski, S. 1929. Grundzüge eines neuen Systems der Grundlagen der Mathematik. *Fundamenta Mathematicae* 14.

Pascu, A., and Carpentier, F.-G. 2002. LDO, a categorization system. In *Proceedings of FLAIRS 2002*, 178–180.

Rosch, E. 1975. Cognitive representations of semantic categories. *Journal of Experimental Psychology* (104):192–233.

Schwer, S. R. 2001. $S-$arrangements avec répétitions. *Comptes rendus de l'Académie des Sciences de Paris* (334):261–266.