Proof, reasoning and logic at the interface between Mathematics and Computer Science : toward a framework for analyzing problem solving

Simon Modeste, Sylvain Beauvoir, Jonathan Chappelon, Viviane Durand-Guerrier, Nicolás León, Antoine Meyer

Proof, reasoning and logic at the interface between Mathematics and Computer Science : toward a framework for analyzing problem solving[1]

Simon Modeste[1], Sylvain Beauvoir[1], Jonathan Chappelon[1], Viviane Durand-Guerrier[1], Nicolás León[1] and Antoine Meyer[2]

[1]IMAG, University of Montpellier, CNRS, Montpellier, France; simon.modeste@umontpellier.fr, sylvain.beauvoir@ac-lyon.fr, jonathan.chappelon@umontpellier.fr, viviane.durand-guerrier@umontpellier.fr, nicolas.leon@umontpellier.fr

[2]LIGM (UMR 8049), UPEM, CNRS, ESIEE, ENPC, Université Paris-Est, Marne-la-Vallée, France; antoine.meyer@u-pem.fr

After analyzing the relation between mathematics and computer science and the place given to proof, logic and reasoning, we propose and discuss a framework for the study of their interactions based on the ck¢ model. Then, we exemplify this model in the analysis of a problem, making explicit a mathematical solution and an algorithmic solution.

Keywords: Mathematics, computer science, proof, reasoning, problem solving.

The research presented in this paper is part of a research project funded by the French National Research Agency, called DEMaIn (Didactics and Epistemology of the interactions between Mathematics and Informatics). This project, started in 2017, aims at better understanding the relations between mathematics and computer science, from an epistemological view, in order to tackle didactical issues (Modeste, 2016). It follows from two observations. On the institutional side, computer science enters many curricula at different levels and in many countries (for France, see Gueudet, Bueno-Ravel, Modeste, & Trouche, 2017). On the epistemological side, computer science and mathematics share many concepts and methods (Modeste, 2016) and their frontier is rather blurry. One of the main topics of the project deals with logic, language, proof and reasoning in mathematics, computer science and their interactions. In this paper, we provide epistemological insights regarding this topic and a framework for analyzing proof and reasoning in problem-solving situations in mathematics and computer science, suited to analyze students activities in these fields, including computer-assisted situations.

Proof, reasoning and logic in mathematics and computer science

Mathematics and computer science share common foundations, based on logic. Logic makes explicit the language and the validation rules in both disciplines. This directly relates to the nature of proof and reasoning in mathematics and computer science. In previous works we have shown that the role of proof and reasoning is rather similar in mathematics and computer science, but with an emphasis on different kinds of properties (Meyer & Modeste, 2018; Ouvrier-Buffet, Meyer, & Modeste, 2018). Some specific types of proofs are particularly important at the interface of the two disciplines, like mathematical induction, and its variations.

---

[1] Communication supported by the French National Research Agency <ANR-16-CE38-0006-01>.

We consider that the activities in mathematics and computer science share common aspects, in particular the central role of problem solving. Various models in logic can be used to describe and structure the activities in mathematics or computer science. Following Durand-Guerrier (2008), we consider that First-order logic (namely Predicate Calculus) is a relevant epistemological reference for analyzing mathematical activity in a didactic perspective, allowing to take in consideration the articulations between syntax and semantics. We have provided evidence in (Durand-Guerrier, Meyer, & Modeste, to appear) that it is also the case for computer science.

In this paper, we will focus on four concepts and their relations: proof, formal proof, algorithm and program.

Proof. We call *proof* (in mathematics or computer science) a finite sequence of statements organized according to some determined rules (explicitly or implicitly) in order to convince someone of the truth of a statement (Balacheff, 1987). The level of details of the proof generally depends on the source and the recipient of the proof.

Formal proof. A *formal proof* is a text consisting of a finite sequence of statements, expressed in a well defined formal language, where the statements are deduced from the previous ones or from axioms following predefined deduction rules. Most of the proofs could be expressed as formal proofs (if the axioms and deduction rules were made explicit). Nowadays, formal proofs are often produced by or for software called *proof checkers* which can automatically validate a proof.

Algorithm. An *algorithm* is a finite sequence of organized instructions, that describes how to solve a problem, that is, how to obtain a defined goal starting from given data. The steps must be considered as elementary by the recipient of the algorithm, and the algorithm must not be ambiguous. In other words, the producer of the algorithm and its recipient must agree on the granularity of the details of the algorithm.

Program. A *program* is a text consisting of a finite sequence of instructions, written in a well defined (programming) language, that is, having a precise syntax (structure of the language) and semantics (effect of each instruction). An algorithm can be described with a program.

These definitions make clear the fact that the relations between proof and formal proof and between algorithm and program are pretty similar, with, on one side, an informal description, more suitable to human interactions (but also driven by some rationality) and on the other side, a formal language with a precise syntax and semantic, that can be interpreted and checked by computer.

In previous work about algorithmics (Modeste, 2012; Modeste & Ouvrier-Buffet, 2011), we have made explicit the links between algorithm and proof. In particular, any (constructive) proof can be interpreted as an algorithm. In a more formal context, the *Curry–Howard isomorphism* states there is a strict correspondence between programs and formal proofs.

We hypothesize that it is valuable taking into consideration these four concepts for questioning the place of proof and reasoning in mathematics, computer science and their interactions.

Finally, let us precise what we will consider as *reasoning*. We enclose in reasoning, in mathematics and computer science, all the human activities that permit to solve problems and increase the epistemic value of properties or problem solutions. Reasoning is clearly at the origin of the building of proofs, formal proofs, algorithms and programs, and is strongly related to logic. Indeed, logic is

built in order to model the way of reasoning and reasoning is based on a (not completely explicit) set of logical rules (e.g. Mesnil, 2017).

This leads us to formulate the following research questions: What place and role do proof, formal proof, algorithm and program have in the teaching and learning of mathematics, computer science and their interactions? What is the nature of reasoning in computer science in comparison to mathematics and how can we analyze this reasoning in problem-solving and proving activities?

To answer these questions, we are currently developing a framework that permits to analyze problems, problem-solving and proving activities in mathematics and computer science. We will first introduce our framework and then illustrate its possible use by an example. Finally, we will briefly discuss the future development of this framework, in relation with our project's goals.

A framework for analyzing reasoning and proving in problem-solving activity

Our framework is based on a specific definition of *problem*, on a framework called *concept-problem* and on the ck¢ model.

The central notion of problem

The notion of problem and problem solving is central in mathematics and computer science. In the literature, problems attest of the questions of the two fields and, in practice, they structure the research activity. The notion of problem also carries the issue of generality, important in mathematics and computer science. For our purpose, we will use a definition of problem, based on theoretical computer science and computability and complexity theory (see, for instance, Garey & Johnson, 1979). We consider a problem as a pair (I, Q) where I is a set of instances and Q a question that can be instantiated on any of the element of I (Modeste, 2013). Solving a problem P=(I,Q) is finding the answer to the question Q for any element *i* of I. This answer can be given by a formula depending on *i*, any characterizations of the subsets of I for which the answer is a given value, an algorithm that permits to construct the answer for any *i*, etc. In all cases, a proof can be given that the proposed solution to P is correct, that is, for all *i* in I, the answer given to Q(*i*) is correct. This definition is general enough to describe any problem.

In Modeste (2012, 2013) we have shown that this definition allows to conveniently analyze curricula, textbooks and activities in algorithmics, in particular concerning the place given to proof. Giroud (2011), has used a similar definition to develop what he called the "concept-problem", in order to study the problem-solving activities of students, mainly based on the Theory of Conceptual Fields (Vergnaud, 2009). We will rely on Giroud's idea that different problems can be related to the main problem p studied (called by Giroud the situations "giving meaning to p") and follow his idea of representing problem solving with flowcharts between related problems.

The ck¢ model to analyze proof and reasoning

The ck¢ model (Balacheff, 2013) is an enrichment of Vergnaud's Theory of Conceptual Fields. It considers a concept as composed of four elements: a set P of the problems that give meaning to the concept and a representation system L (the *signifier*), similarly to the model of Vergnaud; but it separates the invariants in two types: a set R of operators that permit to transform a problem in another one; and a set Σ of controls that permit to decide whether an operator *r* applies to a given problem *p*, and to determine whether or not a problem is solved.

The ck¢ model has been designed to study proof and reasoning (this motivated the separation between operators and controls) and to be appropriate for analyzing computer-assisted learning situations. This leads us to use it in our research project for describing concepts and conceptions.

Description of the framework for analyzing problems

We use the ck¢ model, to describe concept-problems. For us, a concept-problem on a given problem *p* will be described with:

- a set P of the problems that give meaning to the problem *p*, that is having a link with *p*,
- a set R of operators that transform a problem in another problem, we will denote $p_2=r(p_1)$ if the operator r transforms p_1 in p_2 or $p_1 \rightarrow_r p_2$,
- a set Σ of controls that describe if an operator r is relevant to apply on a problem or if a problem is solved;
- a representation system L, that permits to describe elements of P, R and Σ.

In (Durand-Guerrier et al., to appear; Modeste, 2012) we have used such a framework to analyze the concept "algorithm" and its relation to proof. We proposed to distinguish two levels of problems, to bring to light a tool-object dialectic, and to differentiate the situations where proof concerns the controls level (elements of Σ) and the situations where the studied problem p consists in proving something. In this second case, the operators concern proving strategies and the controls of Σ concern logic rules (Σ can remain mostly implicit while controlling the use of the operators).

In this paper, we extend this framework to any situation in mathematics and computer science, including problem-solving and proving situations, assisted or not with a computer. The framework will allow us to focus on proof and reasoning in these activities.

Finally, we consider that the operators and controls occur at two (intertwined) levels: syntactic and semantic. It is clear that the presence of a computer (programming tool, proof assistant or many other tools[2]), brings some new controls (including feedback) that have a strong syntactic dimension. For example, a very basic control can come from what is called *syntactical analysis* which checks if an expression (formula, program, logic statement…) is well-formed with respect to the grammar of the language. This kind of feedback from the computer can be considered as a purely syntactic control. On the other hand, any interpretation (by the user) of the expression in terms of the objects represented or for specific values of the variables would be considered as a semantic control.

Example of problem analysis

Tin this section, we illustrate the use of our framework for problem analysis, focusing on analyzing the problem itself, the *a priori* analysis (analysis of students' solving of the problem is one of our next perspectives). We have selected a problem from a well-known website called Project Euler: "Project Euler is a series of challenging mathematical/computer programming problems that will require more than just mathematical insights to solve. Although mathematics will help you arrive at elegant and efficient methods, the use of a computer and programming skills will be required to solve most problems." (https://projecteuler.net/). These problems, at the interface between

[2] For example CAS or softwares like Aplusix: www.aplusix.com/

mathematics and computer science, seem interesting to us to confront our framework. We will consider the following problem:

Problem 1. If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

In our framework, the problem can be described as an instance of the general problem $P_{3\text{-}5}=(I,Q)$ with I the set of all natural numbers and Q(i) the question "What is the sum of all the multiples of 3 or 5 below i?". Although the problem consists in solving $P_{3\text{-}5}(1000)$, the value 1000 is large enough to require to think about the general problem while solving it.

Here, we will describe with flowcharts two ways of solving the problem3, one considered as more "mathematical" and the other considered as more "algorithmic". In the flowcharts, problems are represented in blue and arrows between them represent the transformations under the action of an operator (in green) selected according to a control in red (controls on operators and on the status of problems). Sometimes, an operator can generate several problems. We will not give details about the representation system L in this paper.

A "mathematical" solution

This solution is based on the observation that if we want to sum all the multiples of 3 and 5 under 1000, we can count the multiples of 3 and the multiples of 5, and only multiples of 15 will be counted twice. Since we can derive from the formula of the sum of the first integers a formula for the sum of the first multiples of any n, we can solve $P_{3\text{-}5}(1000)$. In the end, we can write:

$$\sum_{\substack{k=1 \\ 3|k \text{ or } 5|k}}^{999} k = \sum_{\substack{k=1 \\ 3|k}}^{999} k + \sum_{\substack{k=1 \\ 5|k}}^{999} k - \sum_{\substack{k=1 \\ 15|k}}^{999} k = 3\sum_{k=1}^{333} k + 5\sum_{k=1}^{199} k - 15\sum_{k=1}^{66} k = 233168$$

An "algorithmic" solution

This solution is based on an enumeration of the numbers below n, a test of the property "being a multiple of 3 or 5" and an accumulation of the values satisfying the property. This can lead to this algorithm (in a pseudo-code, close to the programming language Python):

```
Function Sum_3-5 (n)          % We define a function returning the sum
                                  of the multiples of 3 or 5 below n
  S = 0                       % S will contain the sum of the multiples
  for k from 1 to n-1:        % We enumerate the integers below n
    if k mod 3==0 or k mod 5 = 0   % if they are multiples of 3 or 5
      then S = S+k            % we add them to S
  return(S)
```

[3] Due to space constraints, we cannot show all the solutions that we have identified in our *a priori* analysis.

The solving process is described more precisely by the flowchart in figure 2.

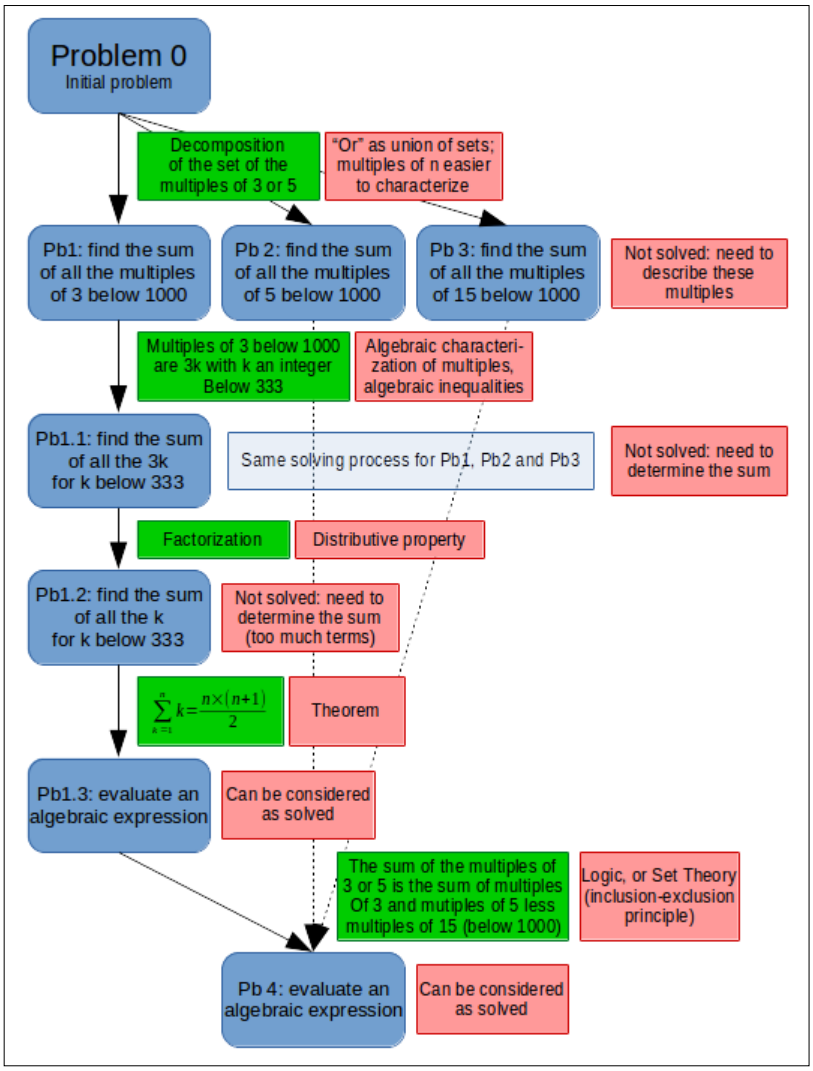


Figure 1: A mathematical solving

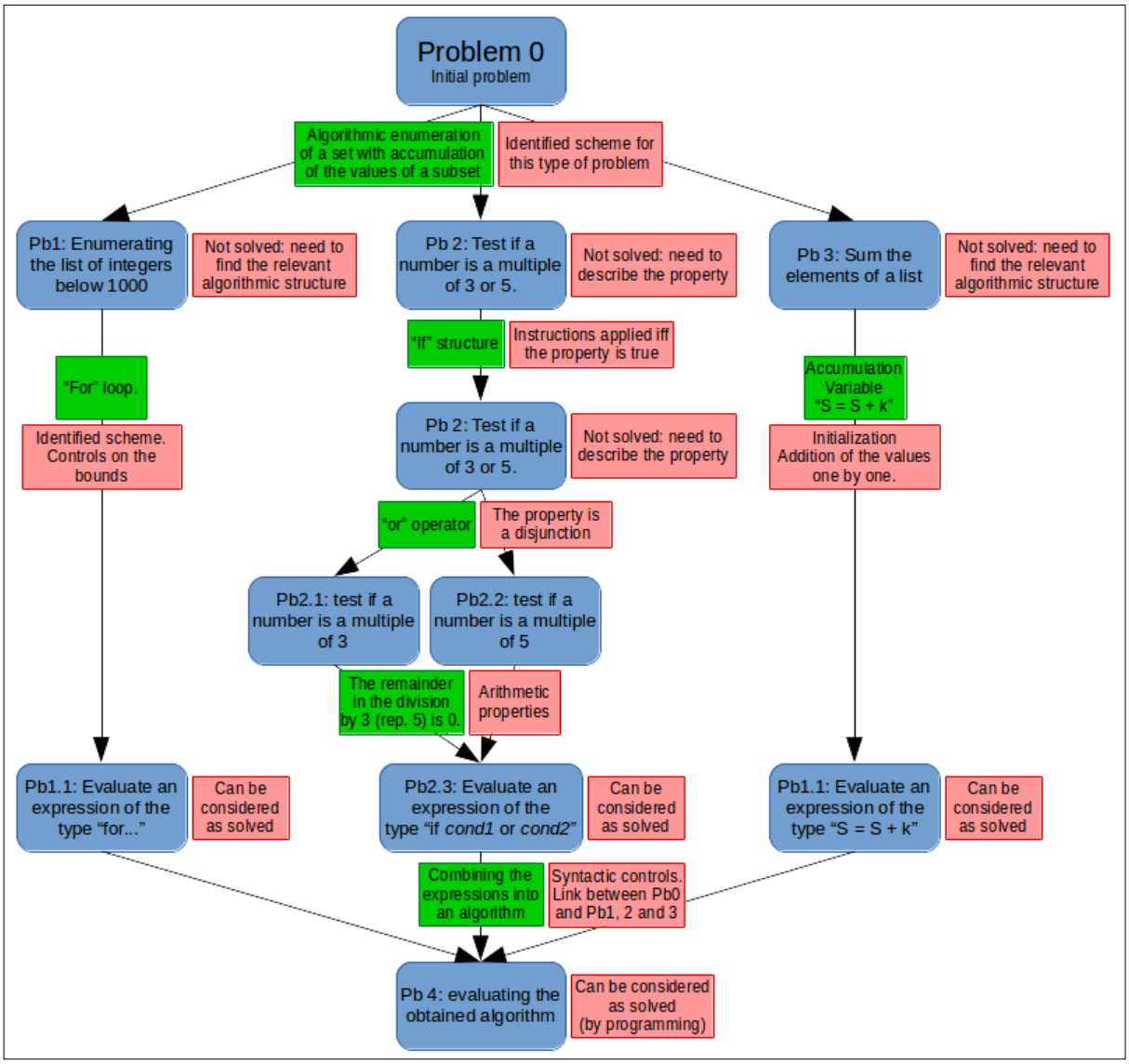
Discussion and comparison of the 2 solutions

The two solving processes analyzed show two very different (correct) strategies. We can clearly distinguish the types of operators and controls that take part in the solutions. In the mathematical solving process, we can notice how the use of algebraic operators (factorization, formula...) are controlled and chosen regarding the sub-problem studied. In the algorithmic solving process, we can notice operators and controls that make clearer the way decisions can be taken about choosing algorithmic tools ("for" loop, "if" structure). Although operators and controls are different between the two solving processes, we can notice that the framework allows to describe finely both of them. This supports our claim that there are many common aspects of the problem-solving process in mathematics and computer science.

Conclusion and perspectives

Or framework permits to analyze problem-solving strategies in mathematics and computer science, with an emphasis on proof and reasoning. On the epistemological side, it puts light on the controls in mathematics and computer science and will permit to discuss their differences and common

points, which can be taken into account in a didactical perspective. The framework should also permit to describe and analyze problem-solving strategies developed by students, in particular faced with problems that can be solved with mathematics and computer science. Another goal is to be able to use this framework for situations that integrate formal tools (computer-assisted situations) – by detailing syntactic controls and their use, which needs to detail the role of the system of representations L – and for situations of proof and proving (by taking into account different levels of control, as aforementioned). In the next step of the DEMaIn project, we will use this framework in the design, analysis and experimentation of didactical situations at the interface of mathematics



and computer science.

Figure 2: A algorithmic solving

References

Balacheff, N. (1987). Processus de preuve et situations de validation. *Educational Studies in Mathematics*, *18*, 147–176.

Balacheff, N. (2013). cK¢, a model to reason on learners' conceptions. In M. Martinez & A. Castro Superfine (Eds.), *Proceedings of the 35th annual meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education* (pp. 2–15). Chicago, IL: University of Illinois at Chicago. Retrieved from https://hal.archives-ouvertes.fr/hal-00853856/

Durand-Guerrier, V. (2008). Truth versus validity in mathematical proof. *ZDM The International Journal on Mathematics Education*, *40/3*, 373–384.

Durand-Guerrier, V., Meyer, A., & Modeste, S. (to appear). Didactical issues at the interface of mathematics and computer science. In G. Hanna, D. Reid, & de V. Michael (Eds.), *Proof Technology in Mathematics Research and Teaching*. Springer. Postprint at https://hal.archives-ouvertes.fr/hal-01912885v1

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Giroud, N. (2011). *Étude de la démarche expérimentale dans les situations de recherche pour la classe*. Université de Grenoble. Retrieved from http://tel.archives-ouvertes.fr

Gueudet, G., Bueno-Ravel, L., Modeste, S., & Trouche, L. (2017). Curriculum in France. A national frame in transition. In D. R. Thompson, M. A. Huntley, & C. Suurtmamm, *International Perspectives on mathematics Curriculum*. IAP.

Mesnil, Z. (2017). A reference for studying the teaching of logic. In *CERME 10*. Dublin, Ireland. Retrieved from https://hal.archives-ouvertes.fr/hal-01865656

Meyer, A., & Modeste, S. (2018). Recherche binaire et méthode de dichotomie, comparaison et enjeux didactiques à l'interface mathématiques - informatique. EMF Conference 2018, Paris.

Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* (Manuscrit de thèse). Université de Grenoble. Retrieved from https://tel.archives-ouvertes.fr/tel-00783294/

Modeste, S. (2013). Modelling algorithmic thinking : the fundamental notion of problem. In *proceedings of CERME 8*. Antalya (Turkey).

Modeste, S. (2016). Impact of Informatics on Mathematics and Its Teaching. In F. Gadducci & M. Tavosanis (Eds.), *History and Philosophy of Computing* (pp. 243–255). Springer.

Modeste, S, & Ouvrier-Buffet, C. (2011). The appearance of algorithms in curricula, a new opportunity to deal with proof? In *Proceedings of CERME 7*. Poland: University of Rzeszów. Retrieved from http://www.cerme7.univ.rzeszow.pl/index.php?id=wg1

Ouvrier-Buffet, C., Meyer, A., & Modeste, S. (2018). Discrete mathematics at university level. Interfacing mathematics, computer science and arithmetic. Presented at the Second INDRUM Conference. Retrieved from https://hal.archives-ouvertes.fr/hal-01849537

Vergnaud, G. (2009). The Theory of Conceptual Fields. *Human Development*, *52*(2), 83–94. https://doi.org/10.1159/000202727