Syntax and semantics in algebra

Jean-François Nicaud, Denis Bouhineau, Jean-Michel Gélis

Syntax and Semantics in Algebra

Jean-François Nicaud, Denis Bouhineau
IRIN, University of Nantes, France
{bouhineau, nicaud}@irin.univ-nantes.fr

Jean-Michel Gélis
IUFM of Versailles, France
gelis@inrp.fr

This paper is the first chapter of a cognitive, didactic and computational theory of algebra that presents, in a formal way, well known elements of mathematics (numbers, functions and polynomials) as semantic objects, and expressions as syntactic constructions. The link between syntax and semantics is realised by morphisms. The paper highlights preferred semantics for algebra and defines formally algebraic problems.

Keywords: syntax, semantics, formalisation, algebraic problem.

Introduction

This paper describes a theory of algebra with cognitive, didactic and computational features. We (the three authors) have a background in mathematics and computer science, and some knowledge in mathematics education. Two of us have been secondary mathematics school teachers. We all have been for long designers of computer systems helping students to learn algebra, involved in the Aplusix project [Nicaud 1994, Nguyen-Xuan et al. 1995, Nicaud et al. 1999]. We have been engaged by this activity to elaborate a theory of algebra with cognitive, didactic and computational features. The reason why we present this theory here is that we think that the interest of the theory goes beyond the design of systems and that the theory can be a contribution to mathematics education. The first phase of this theoretical work is described in the J.M. Gélis PhD dissertation [Gélis 1994]. It is a technical (hard to read) document specialized for polynomial factoring. We are now enlarging the problem domain and rewriting the theory in a more easy to read style. This paper is the first chapter.

Syntax and semantics are present everywhere in algebra. This is emphasized by researchers in mathematics education, sometimes explicitly, sometimes implicitly, i.e., in describing situations. Until now, no formalisation of these concepts has been built. This chapter is a tentative to build such formalisation. It is based of a theory of computer science called *algebraic specification* [Wirsing 1990] or *formal specifications*. Several aspects of signification have been identified [Arzarello et al. 2000, Drouhard 1992, Lins 2000], this chapter is only devoted to the one generally called *denotation* by the two formers.

This theory is presented to the Twelfth ICMI Study in the section *Approaches to algebra*. It is mainly a formal approach for syntax, semantics and problem solving in a general sense. Some aspects also concern the sections *Why algebra?* and *Language aspects of algebra*.

Informal significations. Before entering the subject, we will say a little about some words that will be used in this paper, words like object, concept, abstract, concrete, language, syntax, semantics. These words will occur in formal and informal discourses. In formal discourses, they will be defined explicitly (e.g., we call concept…) or supposed to be defined elsewhere (e.g., let us consider a set of concepts…). In the informal discourses, these words will more or less have the significations that follow.

Objects are things we manipulate explicitly, physically or mentally. There are *concrete* objects that we can see and touch, there are *abstract* objects that are well-formed mental constructions which are explicitly manipulated. As the readers of this paper are researchers and teachers, the authors call objects what they would like the readers to consider as objects. This

does not mean that they are objects for students. A major cognitive and didactic problem consists of deciding when and how some of these objects are supposed to become objects for students. Objects are basically *semantic*. They have a *raison d'être* that gives them signification.

Concepts are mental constructions of various sorts related to objects or to other concepts (e.g., piece of furniture, temperature, number).

Words and *sentences* are assemblages of signs (letters, symbols, sounds, etc.) on particular supports (paper, blackboard, screen, voice, etc.). A *language* is a set of *syntactic* rules that indicates what well-formed words and sentences are. The role of a language is to allow to describe objects, concepts and relationships between them with signs. For mental constructions, this role is fundamental, because it is the only way to communicate about objects and concepts. We will also use *forms* for syntactic constructions. Words, forms and sentences are concrete (we can see or hear them). They are basically *syntactic*. The sentence *x represents y* means that the syntactical construction x represents the semantic object y.

Words, forms, sentences become objects when we describe them with sentences. This is a necessary confusion. In the case of algebra, algebraic expressions are basically syntactic: they are made for describing numbers, polynomials, functions, etc. But an important part of the algebraic activities consists of reasoning on expressions. That means that expressions also are objects. In this chapter, algebraic expressions only appear as syntactic constructions, they will also appear as semantic objects in further chapters.

Abstraction has two complementary meanings. First, abstract is used to qualify non-concrete objects, second, it is used to qualify objects obtained from others by omitting a part of them (see models below). There are several degrees of abstraction for each meaning.

Theories and models. We consider *theories* as frameworks for scientific works. There are informal theories, particularly in human sciences. There are formal theories, particularly in mathematics, physics, computer sciences. Theories have: (1) cognitive features when they concern human behaviour and knowledge, (2) didactic features when they concern human learning and teaching, and (3) computational features when they include mechanisms for calculating certain objects and concepts. The theory described in this paper has cognitive features, but there will be no discourse on that point, we simply always consider algebra for humans. The theory has no advanced didactic features described in this chapter. It is a framework we think adequate for didactic constructions, but we do not make propositions about the teaching of algebra, except a general one in the conclusion. The computational features of the theory are indicated.

Models can be viewed in two different ways. First, as often in science, a model is an abstraction of a concrete object. The electric schema of a car is a model in this view. It does not include mechanic, aerodynamic aspects; it does not include the details of electric components. Second, as it is in computer science, a model is a construction verifying a formal system. These two views are complementary: a model may be, at the same time, a construction verifying a formal system, and an abstraction of a concrete object. In this paper, we will mainly adopt the second view of models (construction verifying a formal system).

Assimilation. Very often, we are making two sorts of assimilation in algebra. The first one is the *syntax/semantics assimilation*: the word representing an object is assimilated to the object. It is the case for representations of natural numbers: 12 is usually seen as a natural number, although it is a succession of two digits. This sort of assimilation is important: it avoids saying continuously 'represents' ; it allows to reason on the semantic objects through the syntax, as mathematicians use to do [Arzarello et al. 2000]. However sometimes it causes trouble because it hides syntactic constructions on which the reasoning applies. In education, this assimilation seems to be always done implicitly. We are not sure this is the best way to teach algebra. The second sort of assimilation is the *inclusion assimilation*. It is a conscious assimilation that occurs when elements of a set are assimilated with elements of another, for

example, when numbers are assimilated with polynomials with degree zero. This sort of assimilation is important too, it avoids the continuous presentation of relations between the assimilated objects. It is sometimes necessary to undo for a while an assimilation, when one wants to be clear on a particular point.

Plan. The first section of the paper describes shortly well known elements of mathematics that are numbers, functions and polynomials. They are described as semantic objects, with a language for representing them. In section 2, a formal system is presented: numbers, function and polynomials appear as semantic models of this formal system while a unified syntax for the expressions appears as a formal model of the formal system. Section 3 extends the constructions of section 2. In section 4, we present our preferred semantics for algebra. Last, in section 5, we define algebraic problems.

Numbers, functions and polynomials

In this section, we describe shortly numbers, functions and polynomials. They are semantic objects, with a language for representing images of functions. We avoid here general assimilation. Functions are distinguished, for example $+_{\mathcal{N}}$ is the addition of natural numbers, $+_{\mathcal{K}}$ the addition of the numbers of a set $\mathcal{K}$ and $+_{\mathcal{F}}$ the addition of functions.

Natural numbers. Let $\mathcal{N}$ be the set of natural numbers. Natural numbers are *abstract semantic objects* represented here by successions of digits in base ten. Each succession of digits represents a unique natural number. Several successions of digits represent the same number, e.g., 12, 012, 0012 represent the same number. We call *canonical forms of natural numbers*, successions of digits which are 0 or do not have 0 on their left, and we note $\mathcal{N}_{\mathcal{C}}$ the set of syntactic constructions they form, $\mathcal{N}_{\mathcal{C}}$ = {0, 1, 2, 3 …}. There is a canonical bijection between $\mathcal{N}_{\mathcal{C}}$ and $\mathcal{N}$ so we make the syntax/semantics assimilation and write : $\mathcal{N}$= {0, 1, 2, 3 …}.

Variables. A variable is a symbol made of characters (most of the time a unique letter). A variable is a *concrete syntactic element* that is used to represent many objects, but only one at a time. An assignation [Drouhard 1992] is a mapping between a set of variables and objects. For example, applying to sentence *n divides p* the assignation (n:3 ; p:6) provides *3 divides 6*.

Functions. A function from A to B with arity k is a function from A^k to B for a given k. A variadic function from A to B is a function from A^k to B for any k>1. A binary associative function f is often extended to a variadic function. Functions from A to B are semantic objects when A and B are semantic sets, which is the only case we will consider in this chapter (for other functions, like morphisms, the syntactic/semantic qualification is not pertinent).

Numbers. Let $\mathcal{K}$ be a set of *abstract semantic objects* called numbers, which is, for the beginning, the set $\mathcal{Z}$ of integers (later, we will consider larger sets). Elements of $\mathcal{K}$ have canonical forms that constitute the set $\mathcal{K}_{\mathcal{C}}$: they are elements of $\mathcal{N}_{\mathcal{C}}$, and elements of the form $\overline{x}$ (representing the opposite of x) where $x \in \mathcal{N}_{\mathcal{C}} - \{0\}$. Over $\mathcal{K}$ we consider the variadic functions $+_{\mathcal{K}}$ and $*_{\mathcal{K}}$ (addition and multiplication), the unary function $-_{\mathcal{K}}$ (opposite), and the power notation a^n for representing a $*_{\mathcal{K}}$a $*_{\mathcal{K}}$ … $*_{\mathcal{K}}$a (n times). As $\mathcal{K}_{\mathcal{C}}$ includes $\mathcal{N}_{\mathcal{C}}$, $\mathcal{K}$ includes $\mathcal{N}$. The above function symbols allow to represent the image for $+_{\mathcal{K}}$ of 5 and 9 as 5 $+_{\mathcal{K}}$ 9, so 5 $+_{\mathcal{K}}$ 9 is 14. There is a canonical bijection between $\mathcal{K}_{\mathcal{C}}$ and $\mathcal{K}$ so we will make the syntax/semantics assimilation between these two sets.

The computational aspect is: for each of these functions, we have a mechanism that calculates the canonical form of the image when we have the canonical forms of the arguments.

The mechanism is called *calculation procedure*. For $+_{\mathcal{K}}$ and elements of $\mathcal{N}$ the mechanism is an addition procedure like the one taught in primary school. Forms like $5 +_{\mathcal{K}} 9$ are fundamentally written to be replaced by the correspondent canonical forms: ***this is arithmetic***.

Numerical functions. Let $\mathcal{F}$ be the set of functions from $\mathcal{K}^n$ to $\mathcal{K}$ for any n. From $+_{\mathcal{K}}$, $-_{\mathcal{K}}$, $*_{\mathcal{K}}$ we define $+_{\mathcal{F}}$, $-_{\mathcal{F}}$, $*_{\mathcal{F}}$ and the power notation over $\mathcal{F}$, x being a finite sequence of variables: $(f +_{\mathcal{F}} g)(x) = f(x) +_{\mathcal{K}} g(x)$ and $(f *_{\mathcal{F}} g)(x) = f(x) *_{\mathcal{K}} g(x)$ and $(-_{\mathcal{F}} f)(x) = -_{\mathcal{K}} f(x)$ and $f^m = f *_{\mathcal{F}} f *_{\mathcal{F}} \dots *_{\mathcal{F}} f$ (m times). Numerical functions are *abstract semantic objects*.

Polynomials. Let $\mathcal{P}$ be the set of polynomials of one indeterminate over $\mathcal{K}$. They are *abstract semantic objects*. A polynomial is an infinite sequence $(a_0, a_1, a_2 \dots)$ of elements of $\mathcal{K}$ such as only a finite subset of them are different from 0 [Bourbaki 1967]. The functions $+_{\mathcal{K}}$, $-_{\mathcal{K}}$, $*_{\mathcal{K}}$ allow to define associated functions $+_{\mathcal{P}}$, $-_{\mathcal{P}}$, $*_{\mathcal{P}}$ and the power notation over $\mathcal{P}$ by:

$(a_0, a_1, a_2 \dots) +_{\mathcal{P}} (b_0, b_1, b_2 \dots) = (a_0 +_{\mathcal{K}} b_0 , a_1 +_{\mathcal{K}} b_1 , a_2 +_{\mathcal{K}} b_2 \dots)$

$-_{\mathcal{P}} (a_0, a_1, a_2 \dots) = (-_{\mathcal{K}} a_0 , -_{\mathcal{K}} a_1 , -_{\mathcal{K}} a_2 \dots)$

$(a_0, a_1 \dots) *_{\mathcal{P}} (b_0, b_1 \dots) = (c_0, c_1 \dots)$ with $c_i = a_0 *_{\mathcal{K}} b_i +_{\mathcal{K}} a_1 *_{\mathcal{K}} b_{i-1} +_{\mathcal{K}} \dots +_{\mathcal{K}} a_i *_{\mathcal{K}} b_0$

$(a_0, a_1, a_2 \dots)^m = (a_0, a_1, a_2 \dots) *_{\mathcal{P}} (a_0, a_1, a_2 \dots) *_{\mathcal{P}} \dots *_{\mathcal{P}} (a_0, a_1, a_2 \dots)$ (m times).

The function from $\mathcal{K}$ to the set of polynomials with degree 0 that associates the polynomial (b, 0, 0 …) to the number b is a bijection and a morphism for +, –, *, ^ (e.g., $b +_{\mathcal{K}} c$ is associated to $(b, 0, 0 \dots) +_{\mathcal{P}} (c, 0, 0 \dots)$), so we make an *inclusion assimilation* considering that $\mathcal{K}$ is included in $\mathcal{P}$ and we note b the polynomial (b, 0, 0 …). The polynomial (0, 1, 0, 0 …) is noted X and is called the indeterminate. With that, we get the canonical representation of polynomials: $(a_0, a_1, a_2, a_3 \dots) = a_0 +_{\mathcal{P}} a_1 *_{\mathcal{P}} X +_{\mathcal{P}} a_2 *_{\mathcal{P}} X^2 +_{\mathcal{P}} a_3 *_{\mathcal{P}} X^3 +_{\mathcal{P}} \dots$

Polynomials of several indeterminates over $\mathcal{K}$ are defined the same way.

Before using these constructions, we define models and expressions in the following section.

Models

We define here a formal system (the set of expressions): numbers, functions and polynomials appear as semantic models of this formal system. A general equivalence between expressions is defined for chosen semantic models.

The formal system described below is a *signature* as defined in *formal specifications* [Wirsing 1990, Sanuella & Tarlecki 1999]. All the symbols of this signature are formal symbols not linked to previous constructions, except $\mathcal{N}_c$ and $\mathcal{K}_c$ which are the sets of canonical forms presented above. A signature describes *types* in a formal way, 'Type Tnat' means 'here is a description of a type named Tnat' and 'x : Tnat' means 'x has the type Tnat'. When models will be associated to the signature, sets will be associated to types and '∈' will correspond to ':'. In some models, Tnat will be associated to natural numbers, Tnum to numerical expressions. But, in the signature, symbols are formal.

The signature $\Sigma_{num,s}(v)$ is defined below. In this notation: (1) 'num' is for 'numeric' which does not mean 'all is number' but which is opposed to Boolean (or logical), (2) s is for 'small'. (there will be larger signatures later); (3) v is a finite set of variables that may be empty.

Type Tnat	$\forall a \in \mathcal{N}_c$, a : Tnat	*elements of $\mathcal{N}_c$ are Tnat*
Type Tnum	$\forall a \in \mathcal{K}_c$, a : Tnum	*elements of $\mathcal{K}_c$ are Tnum*
	$\forall x \in v$, x : Tnum	*variables that are Tnum*
	+, * : Tnum × Tnum × … × Tnum → Tnum	*two variadic function symbols*
	– : Tnum → Tnum	*– is a unary function symbol*
	^ : Tnum × Tnat → Tnum	*^ is a binary function symbol*

Models of a signature. In *formal specifications*, models are associated to a signature Σ. For that purpose, sets (called domains) are associated to the types of Σ; constants of these sets to constant symbols of Σ; variables to variable symbols of Σ; and functions to function symbols of Σ. Of course, all these associations have to respect the correspondence between domains and types.

The formal model $FM_{num,s}(v)$ of $\Sigma_{num,s}(v)$, also called *term algebra* [Sanuella & Tarlecki 1999], is described below, defining expressions that are coherent with $\Sigma_{num,s}(v)$. In this definition, we make an important assimilation: we use the same constant symbols and the same function symbols for the signature and for the model. The reader must be aware of this assimilation. Note that the letter 'E' is for 'Expression'.

Each constant symbol of $\mathcal{N}_c$, $\mathcal{K}_c$ is associated to itself. Each variable symbol is associated to itself. To Tnat is associated $\mathcal{N}_c$. To Tnum is associated the sets Enum defined below. To each function symbol is associated a function between the sets associated to the types and represented with the same symbol (e.g., to the function symbol + is associated a function from Enum × Enum to Enum named +). In this model, when f is a function, the notation $f(u_1, u_2, \ldots u_n)$ means a tree having f as root and $u_1, u_2, \ldots u_n$ as ordered immediate sub-trees of this root.
Enum is the smallest set such as:
- if u is constant symbol of $\mathcal{K}_c$ or a variable symbol of v then $u \in$ Enum
- if $u_1, u_2, \ldots u_n \in$ Enum then $+(u_1, u_2, \ldots u_n) \in$ Enum and $*(u_1, u_2, \ldots u_n) \in$ Enum
- if $u \in$ Enum then $-(u) \in$ Enum
- if $u_1 \in$ Enum and $u_2 \in$ Enat then $\hat{}(u_1, u_2) \in$ Enum

In figure 1, examples of expressions are drawn as trees and in the usual two-dimension notation. For $FM_{num,s}(v)$, we call 'usual notation' a representation in which function symbols of +, * are drawn in an infixed mode, – in an prefixed mode, ^ is just represented with a shift of the exponent, parentheses indicate the sub-expressions, priorities of function symbols allow to withdraw some parentheses, * is omitted when it is possible, + is omitted before –.

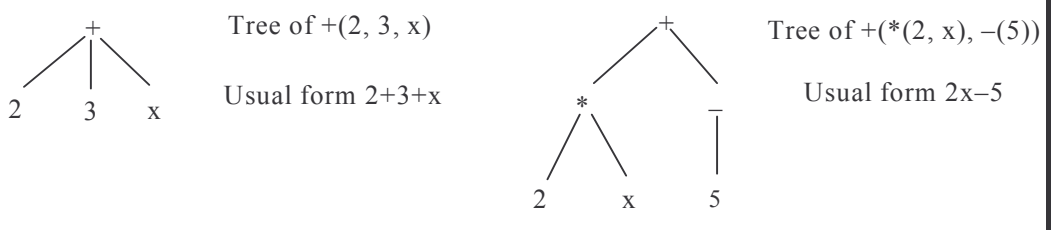

Figure 1, examples of expressions (the trees) with their associated usual form

$FM_{num,s}(v)$ provides expressions which are objects (trees) we can manipulate and which constitute the object language of algebra, according to the terminology of logic, the

syntax being: *well-formed expressions are built by applying function symbols to well-formed expressions, respecting types and arity*. However, in many situations, in particular ***in this chapter, we will consider, most of the time, expressions as syntactic constructions that aim at representing semantic objects***.

Models of a signature (following). The Algebraic Specification theory considers different models of a given signature. A model is a set of elements with functions corresponding to the function symbols of the signature. Models are linked by morphisms and constitute a category [Wirsing 1990]. As a consequence, there is a particular model, called initial model, such as there is a morphism from this initial model to any other model. The initial model is unique in the sense that all the initial models are isomorphic. The initial model is actually the formal model. We name *semantic model* of a signature $\mathbf{\Sigma}$, any model M of $\mathbf{\Sigma}$ that *is not isomorphic* to its formal model FM. We will also say that M is a *semantic model* of FM. We call *evaluation* the morphism from FM to M. Formal models are denumerable because they are generated by finite constructions from denumerable sets. So, when the morphism from the formal model FM to a semantic model M is a surjection, M is denumerable, otherwise it may be not denumerable.

First major point: Given a semantic model M, the semantic of an expression is obtained by applying the evaluation function which is the morphism from the formal model to M.

$\mathcal{K}$ is a semantic model of $FM_{num,s}(\varnothing)$. The proof consists of exhibiting a morphism f:

$\forall a \in \mathcal{K}_c$, $f(a) = a$ $\quad \forall u_1, u_2, \ldots u_n \in$ Enum, $f(u_1+u_2+\ldots+u_n) = f(u_1) +_{\mathcal{K}} f(u_2) +_{\mathcal{K}} \ldots +_{\mathcal{K}} f(u_n)$

$\forall u_1, u_2, \ldots u_n \in$ Enum, $f(u_1*u_2*\ldots*u_n) = f(u_1) *_{\mathcal{K}} f(u_2) *_{\mathcal{K}} \ldots *_{\mathcal{K}} f(u_n)$

$\forall u \in$ Enum, $f(-u) = -_{\mathcal{K}} f(u)$ $\quad \forall u_1 \in$ Enum, $\forall u_2 \in$ Enat $f(u_1 \wedge u_2) = f(u_1) \wedge_{\mathcal{K}} f(u_2)$

This morphism is defined in a recursive way. It is a surjective morphism: any element of $\mathcal{K}$ represented by x in $\mathcal{K}_c$ is the image of x. Remember that f is called an *evaluation* morphism, examples of the mapping are: $f(2+4) = 6$ $\quad f(-2) = \overline{2}$ $\quad f((2*2-3^2)-3) = \overline{8}$

Now we are able to simplify the notations. First, we use –2 instead of $\overline{2}$, so a canonical form is now an expression. We can do that because we can build a mechanism that recognises when an expression is a canonical form of an integer. Second, instead of considering forms to be calculated, e.g., $2 *_{\mathcal{K}} (8 -_{\mathcal{K}} 12)$, we can now consider the corresponding expressions, e.g., 2(8–12). As said above, they are also objects that can be observed and manipulated, and we can calculate them with the evaluation morphism.

It is natural to take $\mathcal{K}$ as *preferred* semantic model of $FM_{num,s}(\varnothing)$, to call *denotation* of an expression of $FM_{num,s}(\varnothing)$ its evaluation and to define the equivalence over $FM_{num,s}(\varnothing)$ by : a ~ b if and only if a and b have the same denotation. This equivalence is generally noted =.

$\mathcal{P}$ is a semantic model of $FM_{num,s}(x)$. The proof consists of exhibiting a morphism f:

$f(x) = X$ $\quad \forall u_1, u_2, \ldots u_n \in$ Enum, $f(u_1+u_2+\ldots+u_n) = f(u_1) +_{\mathcal{P}} f(u_2) +_{\mathcal{P}} \ldots +_{\mathcal{P}} f(u_n)$

$\forall u_1, u_2, \ldots u_n \in$ Enum, $f(u_1*u_2*\ldots*u_n) = f(u_1) *_{\mathcal{P}} f(u_2) *_{\mathcal{P}} \ldots *_{\mathcal{P}} f(u_n)$

$\forall u \in$ Enum, $f(-u) = -_{\mathcal{P}} f(u)$ $\quad \forall u_1 \in$ Enum, $\forall u_2 \in$ Enat $f(u_1 \wedge u_2) = f(u_1) \wedge_{\mathcal{P}} f(u_2)$

Let $a_0 +_{\mathcal{P}} a_1 *_{\mathcal{P}} X +_{\mathcal{P}} a_2 *_{\mathcal{P}} X^2 +_{\mathcal{P}} \dots +_{\mathcal{P}} a_n *_{\mathcal{P}} X^n$ be the generic element of $\mathcal{P}$. This element is the image of the expressions $a_0 + a_1 * x + a_2 * x^2 + \dots + a_n * x^n$. So, f is surjective. An example of the mapping of this morphism is: $x(x+4) \rightarrow 4 *_{\mathcal{P}} x +_{\mathcal{P}} x^2$.

Again we simplify the notations by replacing function symbols of $\mathcal{P}$ (e.g., $+_{\mathcal{P}}$) by function symbols of the formal model (e.g., +). So the polynomial $4 *_{\mathcal{P}} X +_{\mathcal{P}} X^2$ becomes $4X + X^2$. We can do that because we can build a mechanism that recognises when an expression is a canonical form of a polynomial. As any element of $FM_{num,s}(x)$ has an image in $\mathcal{P}$ with the above morphism, we call ***polynomial expressions*** the elements of $FM_{num,s}(x)$.

$\mathcal{P}$ *may be chosen* (see functions below) as preferred semantic model of $FM_{num,s}(x)$, which defines the denotation and the equivalence of expressions of $FM_{num,s}(x)$.

$\mathcal{F}$ is a semantic model of $FM_{num,s}(x)$. A morphism proving that can be built from the above proof for $\mathcal{P}$ by replacing the first equality by $f(x) = x \rightarrow x$ and by replacing everywhere $_{\mathcal{P}}$ by $_{\mathcal{F}}$. An important difference is that f is not surjective ($FM_{num,s}(x)$ is denumerable and $\mathcal{F}$ is not). An example of the mapping of this morphism is: $f(x(x+4)) = x \rightarrow 4 *_{\mathcal{K}} x +_{\mathcal{K}} x^2$.

The image of $FM_{num,s}(x)$ in $\mathcal{F}$ defines the set of polynomial functions. We know that, in our framework, as $\mathcal{K}$ is infinite, $\mathcal{P}$ is isomorphic to that image [Bourbaki 1967].

Most of the time, $\mathcal{F}$ is chosen as preferred semantic model of $FM_{num,s}(x)$, which defines the denotation and the equivalence of expressions of $FM_{num,s}(x)$. Note that, in our framework, this equivalence is identical to the previous one defined on $\mathcal{P}$. Later (see below), when the image of $FM_{num,s}(x)$ will grow and will no more be isomorphic to $\mathcal{P}$ (when we will have functions that are not polynomial functions), it will be essential to choose $\mathcal{F}$ as preferred semantic model.

This is algebra. *Now we have variables and well-formed expressions, that may appear as objects, and evaluation morphisms providing their meanings: we are in algebra.*

Extensions of the signature and of the models

In this section, we consider successive extensions of the above signature and models (necessary to reach usual frameworks of algebra). Functions are added. Sometimes $\mathcal{K}$ evolves which implies an evolution of $\mathcal{P}$ and $\mathcal{F}$. A new type Tbool is added to the signature in order to introduce equalities and inequalities, the corresponding domain being the Boolean values.

Adding functions without changing $\mathcal{K}$. Here we add at the same time a function symbol h in the signature and an associated function $h_{\mathcal{K}}$ between the corresponding domains. For example, we add $max : Tnum \times \dots \times Tnum \rightarrow Tnum$ to the signature and we associate max to the function $max_{\mathcal{K}}$ that provides the greatest number. The computational aspect consists of having a calculation procedure that calculates $h_{\mathcal{K}}(a, \dots c)$ when a, .. c are canonical forms. A new function on $\mathcal{K}$ induces a new function on $\mathcal{F}$ but not (in general) a new function on $\mathcal{P}$. In such situation, $\mathcal{F}$ is still a semantic model of $FM_{num,s}(x)$ and $\mathcal{P}$ is no more. It becomes a partial semantic model, i.e., a model of a subset of $FM_{num,s}(x)$: the polynomial expressions.

Adding functions AND changing $\mathcal{K}$. Let us take an example of this situation: going from the integers $\mathcal{Z}$ to the rational numbers $\mathcal{Q}$. We add the function symbol / in the signature, we

change $\mathcal{K}$ (from $\mathcal{Z}$ to $\mathcal{Q}$). New canonical forms are defined, they are a/b and –a/b where a, b ∈ $\mathcal{N}_C$ – {0}, a and b having only 1 as common divisor. This defines $\mathcal{Q}_C$, the new $\mathcal{K}_C$. Previous functions (+ – * ^) are extended to $\mathcal{Q}$ and calculation procedures are built for these functions. In such kind of extension, we may have functions that cannot be extended (like *div* from $\mathcal{Z}$ to $\mathcal{Q}$ or *max* from the real to the complex numbers). These functions become partial functions.

Another way to manage such evolution consists of adding a new type in the signature. We change Tnum in Tint (means integer type) and we introduce Trat (means rational type). We extend some functions to Trat (here: + – * ^) and we do not extend others. We define Tnum as the union of Tint and Trat.

When $\mathcal{K}$ evolves, the semantic models $\mathcal{F}$ and $\mathcal{P}$ evolve too. In the previous example, $\mathcal{F}$ evolves toward functions from $\mathcal{Q}$ to $\mathcal{Q}$ while $\mathcal{P}$ evolves toward polynomials over $\mathcal{Q}$.

Adding objects without canonical forms. This situation occurs, for example, when we introduce the set **R** of real numbers. We can introduce **R** in the previous framework (with $\mathcal{K}$ = $\mathcal{Q}$) just by saying that **R** is a semantic model of the signature (and the formal model). The morphism built for $\mathcal{Q}$ is also valid for **R**. The difference is that it is no more surjective. We have now unreachable objects in this semantic model.

Adding objects without calculation procedure. This situation occurs, for example, when we have introduced **R** as the model for numbers and we introduce the sinus function with the symbol sin. There is no problem for extending the formal model, so we have now expressions like 3sin(2x+1). This extends the set of number that can be represented but we do not have canonical forms for this extended set. We may want to extend $\mathcal{K}_C$ with forms like sin(1), cos(3/4). This is possible, but $\mathcal{K}_C$ appear now as a set described with forms, several forms corresponding to the same element without a mechanism telling when two forms correspond to the same element. We are exiting a strict computational framework. Note that this kind of extension enlarges the reachable elements of **R**. But this set remains denumerable.

Adding the Boolean values. In order to reach equations and inequalities, we define a signature $\Sigma_{bool,s}(\emptyset)$ for Boolean values without variables ($\mathcal{B}ool_C$ is the set {*false*, *true*} of the canonical forms of the Boolean values that constitute the semantic set $\mathcal{B}ool$):

Type Tbool	∀a∈ $\mathcal{B}ool_C$, a : Tbool	*elements of $\mathcal{B}ool_C$ are Tbool*
	not : Tbool → Tbool	*a unary function symbol*
	and, or : Tbool × Tbool × … × Tbool → Tbool	*variadic function symbols*

and we define a more general signature $\Sigma_s(v)$ as the union of $\Sigma_{num,s}(v)$ and $\Sigma_{bool,s}(\emptyset)$ plus:

Type Tbool	<, ≤, =, ≠, >, ≥ : Tnum × Tnum → Tbool	*function symbols form Tnum to Tbool*
Type Tgen	Tgen is Tnum ∪ Tbool	*gen is for general*

Let us consider the set $\mathcal{B}$ of unary functions from $\mathcal{K}$ to $\mathcal{B}ool$. We can prove that **$\mathcal{F} \cup \mathcal{B}$ is a semantic model of $FM_s(x)$**. The morphism associates to the expression 2x+1=0 the function $x \to 2 *_{\mathcal{K}} x +_{\mathcal{K}} 1 =_{\mathcal{K}} 0$ from $\mathcal{K}$ to $\mathcal{B}ool$. It is natural to chose $\mathcal{F}$ as preferred semantic model of $FM_s(x)$, which defines the denotation and the equivalence of expressions of $FM_s(x)$. The equivalence is generally noted = for numerical expressions and ⇔ for Boolean expressions. For example, when $\mathcal{K}$ is **R**, $(x+5)^2 = x \Leftrightarrow x^2+x+1 = 0$ because the functions $x \to (x+5)^2 = x$ and $x \to x^2+x+1 = 0$ are equal (equal to the function $x \to$ *false*).

The preferred semantics of algebra

Our framework defines, in a formal way, a formal model (the expressions) and semantic models for algebra for a given formal system, the signature. In a lot of contexts, expressions are seen as syntactic constructions and elements of semantic models as semantic objects. These structures are linked by a morphism from the expressions to each set of semantic objects that is called evaluation. We have defined the denotation of an expression and the equivalence between expressions from the semantic models. There are several ways to do that, but it is usual in algebra to have a unique denotation and a unique equivalence for a class of expressions, as it is evoked by researchers who write "the denotation" [Arzarello et al. 2000, Drouhard 1992]. So, we established a set of preferred semantic models: the set $\mathcal{K}$ of numbers of the framework for expressions without variable, the set $\mathcal{F}$ of functions from $\mathcal{K}^n$ to $\mathcal{K}$ for numerical expressions with n variables, the set $\mathcal{B}$ of functions from $\mathcal{K}^n$ to $\mathcal{Bool}$ for Boolean expressions with n variables (including equations and inequalities). We consider these sets as being ***the preferred semantics of algebra*** and when there is no ambiguity as being ***the semantics of algebra***. In this conception, the semantics depend only of the set $\mathcal{K}$ of numbers, which means that :

> *When someone in an algebraic context indicates a set of numbers, (s)he also indicates all the semantics, i.e., the denotation and the equivalence for each sort of algebraic expressions.*

This framework is compatible with conceptions of researchers in psychology and mathematics education. Cauzinille et al. [1990] consider an arithmetic and an algorithmic microworld that corresponds to what we called arithmetic with $+_{\mathcal{K}}$, $*_{\mathcal{K}}$ … functions. They consider a syntactic and formal microworld that corresponds to our formal model. Sfard [1989] and Kieran [1991] distinguish a structural approach corresponding to our formal model and an operational (or procedural) approach corresponding to arithmetic.

This framework fundamentally differs from Lins' conception [Lins 2000] when he says: *Characterising algebra as a text, rather than a knowledge, allowed us to account positively for different meanings produced for it, without having to slide into a hierarchy in which "official" ways of producing meaning are at the top*. Actually, we consider in the same time a syntax that is more than text and precise semantic sets, and we do not indicate what knowledge is. However, although our theory establishes *official meanings*, it does not contradict Lins' Theory of Semantic Fields, it just considers the various semantic fields proposed by Lins as being close to algebra and not as being algebra.

Algebraic problems

Problem type, solution predicate, problem, solution. We consider a set $\mathcal{K}$ of numbers and the preferred semantics induced by $\mathcal{K}$. We define a problem type *Tpb* (like *simplify*, *factor*, *solve-equation*) by the data of a set of expressions $\mathcal{E}$ and of a function *Solved* from $\mathcal{E}$ to $\mathcal{Bool}$ that indicates when an expression is solved (we term *Solved* a solution predicate). $\mathcal{E}$ is a formal model as defined above. A problem is a couple <*Tpb*, e> where Tpb is a problem and e an element of $\mathcal{E}$. A solution of <*Tpb*, e> is any u ∈ $\mathcal{E}$ which is equivalent to e and such as *Solved*(u)=*true*. A problem may have several solutions. An algebraic problem type *Tpb* is deeply plunge into the semantics by the following constraint: *Two problems <Tpb, e> and <Tpb, u> such as e and u are equivalent must have the same solutions*.

With this definition, solving a problem <*Tpb*, e> can be realised by replacing e by successive equivalent expressions until a solved expression is obtained.

First example: *solve-equation.* This problem type can be defined over $\mathcal{Q}$ for a chosen set $\mathcal{E}$ of expressions over $\mathcal{Q}$ with one variable x, including numerical and Boolean expressions, and for the solution predicate *EqSolved* defined by *EqSolved*(e)=true if and only if e has the form $x=a_1$ or $x=a_2$ or … $x=a_n$ where a_1, a_2 … a_n are distinct canonical forms of rational numbers. In this problem type, the variable has the status of *unknown*.

Solve-equation is an algebraic problem type. We prove this fundamental property just by telling that the set of numbers $\{a_1,a_2, \dots a_n\}$ occurring in a solution of an equation E is the coimage of *true* for the function $x \rightarrow E$ from $\mathcal{E}$ to $\mathcal{Bool}$. By definition, equivalent expressions are associated to the same function, so the coimage is unique for equivalent expressions.

This definition is a syntactic one, as it is based on the expression. Someone may prefer a more semantic definition, something like *solving an equation consists of providing its set of solutions which is a set of numbers*. However, if no syntax is imposed on the *set of solutions*, it is easy to solve any equation, for example, $4x^5+5x^3-x^2+6x-1 = 0$ has for set of solutions $\{x\in \mathcal{Q} \mid 4x^5+5x^3-x^2+6x-1 = 0\}$ and nobody will consider that the problem is solved like this. If we add that *the set of solutions must be described as a list of canonical forms*, then we add syntactic constraints and we are close to the proposed definition. Note that we have trouble for defining *solve-equation* over **R** as there is no canonical form on **R**.

A non-example. Let us consider *how_many_x_are_there_in* as a problem type over a set $\mathcal{E}$ of expressions over $\mathcal{Q}$ where the solution of the problem <*how_many_x_are_there_in*, (x–3)(x+3)> is 2. This is not an algebraic problem type because two equivalent expressions do not have the same solution: (x–3)(x+3) and x^2-9 are equivalent and the solution for x^2-9 is 1.

Second example: *Simplify*. There are several *simplify* problem types in algebra. We consider here a basic concept of simplification for a very large set of expressions. Let $\mathcal{K}$ be a set of numbers and $\mathcal{E}$ a set of expressions over $\mathcal{K}$. We choose a denumerable set RM of replacement mechanisms that are able to replace sub-expressions (sub-trees) by equivalent expressions. In RM, we place: replace x+0 by x, replace 5+6 by 11, replace 1*x by x, replace x^2+2x^2 by $3x^2$, replace 0 = 0 by *true*, etc. The solution predicate *Simplified* of this problem type is defined by '*Simplified*(e) is true if and only if there is no replacement mechanism of RM that applies to any sub-expression of e'. In this problem type, the variables may have various status like *unknown*, *parameter*, *variable describing a set of values*. When RM is well chosen, the simplifications are the usual simplifications we make on expressions whatever they come from. For example, when a physicist writes a formula from a situation in physics, (s)he forgets physics for a while, simplifies the formula with the above problem type, then returns to physics.

Simplify is an algebraic problem type. This fundamental property comes from the fact that, as replacements are defined, any replacement changes an expression to an equivalent one.

But *simplify* has trouble if RM does not have the *termination property*, i.e., if, for some expression e, there is an infinite sequence of replacements. In this case, the solutions do not exist or are unreachable. So we will add the termination property as a constraint for RM. This problem type will be widely developed in the second chapter: the mechanisms will be called *rewrite rules* and the results of the *rewrite rule theory* (a computer science theory [Dershowitz & Jouannaud 1990]) will be applied to algebra.

Second major point: Algebraic problem types are based on the preferred semantics of algebra that define the equivalence of expressions.

Conclusion

In this paper, we have built algebraic expressions and their semantics. This construction is solid, as it is defined formally and as morphisms link algebraic expressions to semantic objects. It maps well the usual things of algebra, in particular the numerous problem types.

When a problem of a domain D is modelled as an algebraic problem P, P can be solved without any interpretation of the steps of the resolution in D, because algebra is autonomous (as problem types are defined here). There is a suspension of the semantics of the original domain and a replacement by the algebraic semantics during the resolution. This point joins observations made by Vergnaud et al. [1987] and Drouhard [1992]. Of course, this is not a conception for novices. For students, it is a conception to acquire and the learning may benefit from the interaction with other semantic fields as emphasised by Kaput [1989] and Lins [2000].

As many problems are modelled as algebraic problems, and as algebra has efficient methods for solving a lot of problem types, algebra is a very powerful tool and framework for solving problems. Note that, in this conception, problem solving is larger than in the usual one. It is not limited to equation solving and the meaning of variables is not limited to unknowns. It encompasses, in particular, the two frameworks defined by Chevallard [1990], the *formal* and the *functional* frameworks. It is obvious for the former; almost every step of the study of a function is an algebraic problem as defined is this paper.

Our general proposition about the teaching of algebra consists of trying to have didactic constructions in accordance with the theory presented in this paper. We suggest: (1) to separate more clearly objects and representation (semantics and syntax) and to present explicitly assimilations; indicating sometimes that the work made on expressions has its fundamental signification on numbers and functions may help students to learn algebra (2) to announce the power of algebraic problem solving (in a large sense), in order to justify the learning of algebra, and to explicit the suspension of the semantics of a domain and the replacement by the algebraic semantics. We think that the general ideas can be presented with simple words, which does not mean that algebra will become easy to learn just with such an explanation.

As said in the introduction, this paper is a first chapter mainly devoted to syntax, one aspect of semantics, the *denotation*, and algebraic problem solving in a general sense. In further chapters, we will formalise other elements of algebra in the same general framework (cognitive, didactic and computational). First, we will consider rewrite rules that transform expressions with unification and an inference mechanism. We will refer for that to the *rewrite rule theory* [Dershowitz & Jouannaud 1990]. Second we will consider concepts with a separation between concepts on expressions, that correspond to the *algebraic sense* [Arzarello et al. 2000, Drouhard 1992], concepts on semantic objets and concepts on rules. Expressions will often appear as objects in these constructions. We will exhibit weak and strong rules for a given problem type. Third, we will consider particularly two linked problem types: factoring polynomials and solving polynomial equations. For them, we will prove a termination theorem of a particular set of rewrite rules (defined by formal concepts on rules) and provide a strategic principle for solving these problems. This framework applies as well for a low level of knowledge as for a high one, because the rules are not directly handled but are handled through concepts.

References

Arzarello, F., Bazzini, L., Chiappini, G. (2000). *A Model for Analysing Algebraic Processes of Thinking*. In R. Sutherland, T. Rojano, A. Bell, R. Lins (Eds.), *Perspectives on School Algebra*. Kluwer Academic Publishers, Netherlands.

Bourbaki, N. (1967). *Eléments de mathématiques*. Book XI, algèbre, chapter 4. Paris: Herman.

Cauzinille-Marnèche, E., Mathieu, J., (1990). *Experimental data for the design of a microworld-based system for algebra*. In *Learning Issues in ITS*. Berlin: Springer-Verlag.

Chevallard, Y. (1990). Le passage de l'arithmétique à l'algèbre dans l'enseignement des mathématiques au collège. Perspectives circulaires, la notion de modélisation. *Petit x, 5*, Paris.

Dershowitz, N., Jouannaud, J.P. (1990). *Rewriting systems*. In J. Van Leeuwen (Ed.), *Handbook of Theoretical Computer Science, vol. B*, Amsterdam: Elsevier, pp. 245-320.

Drouhard, J.P. (1992). *Les écritures symboliques de l'algèbre élémentaire*. PhD thesis. University of Paris VII.

Gélis, J.M. (1994). *Eléments d'une théorie cognitive et computationnelle de l'algèbre. Application au cas de la factorisation d'expressions polynomiales*. PhD thesis. University of Paris XI.

Kaput, J. (1989). *Linking Representations in the Symbol Systems of Algebra*. In S. Wagner & C. Kieran (Eds.), *Research Issues in the Learning and the Teaching of Algebra*. Lawrence Erlbaum.

Kieran, C. (1991). *A Procedural-Structural Perspective on Algebra Research*. In F. Furinghetti *Proceedings of Psychology of Mathematics Education*, Assise.

Lins, R., C. (2000). *The Production of Meanings for Algebras: a perspective based on a Theoretical Model of Semantic Fields*. In R. Sutherland, T. Rojano, A. Bell, R. Lins (Eds.), *Perspectives on School Algebra*. Kluwer Academic Publishers, Netherlands.

Nguyen-Xuan, A., Nicaud, J.F., Gélis, J.M. (1995). An experiment in learning algebra with an intelligent learning environment. *Instructional Science, 23*, 25-45.

Nicaud, J.F. (1994). *Building ITSs for use: Lessons Learned from the APLUSIX Project*. In R. Lewis et P. Mendelshon (Eds.), *Lessons From Learning*, IFIP, North Holland, pp. 181-198.

Nicaud, J.F., Bouhineau, D., Varlet, C., Nguyen-Xuan, A. (1999). *Towards a product for teaching formal algebra*. In S. Lajoie & M. Vivet (Eds.), proceedings of *Artificial Intelligence in Education*, pp. 207-214, Le Mans. Amsterdam: IOS Press.

Sanuella, D., Tarlecki, A. (1999). *Algebraic preliminaries*. In E. Astesiano and H.-J. Kreowski and B. Krieg-Brückner (Eds.), *Algebraic Foundations of Systems Specification*. Berlin: Springer-Verlag, pp. 13-30.

Sfard, A., (1989). Transition from operational to structural conception: the notion of function revisited. In G. Vergnaud, J. Rogalski, M. Artigue (Eds.), *Proceedings of Psychology of Mathematics Education*, Paris..

Vergnaud, G., Cortès, A., Favre-Artigue, P. (1987). *Introduction de l'algèbre auprès de débutants faibles*. Proceedings of *Colloque de Sèvre*. Grenoble: La Pensée Sauvage.

Wirsing, M. (1990). *Algebraic specification*. In J. Van Leeuwen (Ed.), *Handbook of Theoretical Computer Science, vol. B*, Amsterdam: Elsevier, pp. 677-788.