1 Introduction

In antiquity, philosophy, mathematics and logic were considered as a single discipline. Since Aristotle we have realized that the world is not just black and white but full of nuance and colour. The inspiration for this thesis comes from one of the most influential philosopher and scientist of his time: Alfred Korzibsky. He founded a discipline he called *general semantics* to deal with problems of knowledge representation in humans. Korzibsky then found that complete knowledge of reality being inaccessible, we had to abstract. This abstraction is then only similar to reality in its structure. In these pioneering works, we find notions similar to that of modern descriptive languages.

It is from this inspiration that this document is built. We then start, off the beaten track and away from computer science by a brief excursion into the world of mathematical and logical formalism. This makes it possible to formalize a language that allows to describe itself by structure and that evolves with its use. The rest of the work illustrates the possible applications through specific fields such as automatic planning and intention recognition.

1.1 Motivations

The social skills of modern robots are rather poor. Often, it is that lack that inhibits human-robot communication and cooperation. Humans being a social species, they require the use of implicit social cues in order to interact comfortably with an interlocutor.

In order to enhance assistance to dependent people, we need to account for any deficiency they might have. In our case we chose to address cognitive deficiencies. The main issue is that the patient is unable or unwilling to express their needs. That is a problem even with human caregivers as the information about their intents needs to be inferred from their past actions.

These aspect of social communication often eludes the understanding of Artificial Intelligence (AI) systems. This is the reason why intent recogniton is such a complicated problem. The primary goal of this thesis is to address this issue and create the formal foundations of a system able to help dependent people.

1.2 Problem

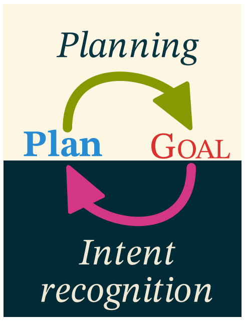
First, what exactly is intent recogntion ? The problem is simple to express: finding out what other agents want to do before they do. It is important to distinguish between

oui on peut citer sans parenthèses comme dis dans ton mail,
mais c'est quand on mentionne l'auteur directement dans le texte
donc ici, 2 options:
- the Theory of mind (Baker et al., 2011)
- the Theory of mind of Baker et al. (2011)

several notions. *Plans* are the sequence of actions that the agent is doing to achieve a *goal*. This goal is a concrete explanation of the wanted result. However, the *intent* is more of a set of abstract goals, some of which may be vague or impossible (e.g. drink something, survive forever, etc.).

Some approaches use trivial machine learning methods, along with a hand made plan library to match observations to their most likely plan using statistics. The issue with these common approaches is that they require an extensive amount of training data and need to be trained on each agent. This makes the practicality of such system quite limited. To address this issue, some works proposed hybrid approaches using logical constraints on the probabilistic method. These constraints are made to guide the resolution toward a more coherent solution. However, all probabilistic methodds require an existing plan library that can be quite expensive to create.

A work from Ramırez and Geffner (2009), added an interesting method to solve this issue. Indeed, they noticed an interesting parallel between that problem with the field of automated planning. This analogy was made by using the Theory of mind Baker *et al.* (2011), which states that any agent will infer the intent of other agents using a projection of their own expectation on the observed behavior of the other agent.



This made the use of planning techniques possible to infer intent without the need for extensive and well-crafted plan libraries. Now only the domain of the possible actions, their effects and prerequisites are needed to infer the logical intent of an agent.

The main issue of planning for that particular use is computation time and search space size. This prevents most planners to make any decision before the intent is already realized and therefore being useless for assistance. This time constraint leads to the search of a real-time planner algorithm that is also expressive and flexible.

1.3 Contributions

In order to achieve such a planner, the first step was to formalize what is exactly needed to express a domain. Hierarchical and partially ordered plans gave the most expressivity and flexibility but at the cost of time and performance. This is why, a new formalism of knowledge representation was needed in order to increase the speed of the search space exploration while restricting it using semantic inference rules.

While searching for a knowledge representation model, some prototypes were done using standard ontology tools but all proved to be too slow and inexpressive for that application. This made the design of a lighter but more flexible knowledge representation model, a requirement of the project.

Then the planning formalism has to be adapted to our general knowledge representation tool. Since automated planning has a very diverse ecosystem of approaches and paradigms, its standard, the Planning Domain Description Language (PDDL) needs use of various extensions. However, no general formalism has been given for PDDL and some approaches often lack proper extensions (hierarchical planning, plan representation, etc). This is why a new formalism is proposed and compared to the one used as standard of the planning community.

Then finally, a couple of planners were designed to attempt answering the speed and flexibility requirements of human intent recognition. The first one is a prototype that aims to evaluate the advantages of repairing plans to use several heuristics. The second is a more complete prototype derived from the first (without plan repairs), which also implements a Breadth-First Search (BFS) approach to hierarchical decomposition of composite actions. This allows the algorithm to provide intermediaries plan that, while incomplete, are an abstraction of the result plans. This allows for anytime probability computation using existing techniques of invert planning.

1.4 Plan

In this document we will describe a few contributions from the new mathematical formalism to intent recognition. Each chapter builds on the previous one.

First we will present a new mathematical fondation, highlighting the flaws of the current ones. This fondation is used to create a formalism capable of describing all the classical mathematics.

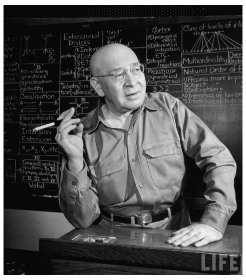
In the third chapter, a new knowledge description system is presented as well as the associated grammar and inference framework.

The fourth chapter is an application of that knowledge description system to automated planning. Existing formalism and languages are compared to this new general one. This allows us to design a general planning framework that can express any existing type of domain.

Using this framework, two online planning algorithms are presented in chapter five: one that uses repairs on existing plans and one that uses hierarchical domains to create intermediary abstract plans.

The final chapter is about intent recognition and its link to planning. Existing works are presented as well as a technique called *inverted planning*.

2 Foundation and Tools



(Korzybski 1933, ch. 4 pp. 58)

> *"A map* is not *the territory it represents, but, if correct, it has a* similar structure *to the territory, which accounts for its usefulness."*

Mathematics and logic are at the heart of all formal sciences, including computer science. The boundary between mathematics and computer science is quite blurry. Indeed, computer science is applied mathematics and mathematics are abstract computer science. Both cannot be separated when needing a formal description of a new model.

In mathematics, a fondation is a axiomatic theory that is consistent and well defined. For the fondation to be generative of all mathematics it only need to either define all basic notions of mathematics or simply to define another existing fondation. needS ?

Since we need to use existing mathematical structures that are beyond the expressivity of the classical mathematics, we need a non-classical fondation. There are several existing ones but since we only use a subset of their possibilities, it is more efficient to create a simpler fondation.

In this chapter, we define a new formalism as well as a proposed fondation that lays on the bases of type theory and lambda calculus. From this formalism we define the classical set theory (which is the most commonly used fondation). The contribution is mainly in the axiomatic system and functional formalism. The rest is simply an explanation, using our formalism, of existing mathematical domains and structures commonly use in computer science. **This formalism is used for all the contributions later on this document.**

useD

il faut donc que tu justifies / prouves le manque d'expressivité des maths classiques
et que tu justifies pourquoi tu as besoin de plus d'expressivité :
(où dire où ca sera expliqué en donnant la ref de la section)

2.1 Issues with some existing fondation

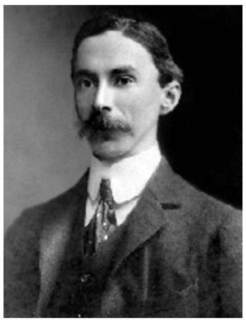
Some of the most important problems in mathematics are the consistency and formalization issues. Research on these issues starts at the end of the 19th century, with Cantor inventing set theory. Then after a crisis in the beginning of the 20th century with Russel's paradox and Gödel's incompletude theorem, revised versions of the set theory become one of the foundations of mathematics. The most accepted version is the Zermelo-Fraenkel axiomatic set theory with the axiom of Choice (ZFC). This effort leads to a formalization of mathematics itself, at least to a certain degree.

Cantor (1874)

Fraenkel *et al.* (1973, vol. 67); Ciesielski (1997)

Any knowledge must be expressed using an encoding support (medium) like a language. Natural languages are quite expressive and allow for complex abstract ideas to be communicated between individuals. However, in science we encounter the first issues with such a language. It is culturally biased and improperly convey formal notions and proof constructs. Indeed, natural languages are not meant to be used for rigorous mathematical proofs. This is one of the main conclusions of the works of Korzybski (1958) on "general semantics". The original goal of Korzybski was to pinpoint the errors that led humans to fight each other in World War 1. He affirmed that the language is unadapted to convey information reliably about objective facts or scientific notions. There is a discrepancy between the natural language and the underlying structure of the reality. This issue is exacerbated in mathematics as the ambiguity in the definition of a term can be the cause for a contradiction and make the entire theory inconsistent.

> *"Mathematics may be defined as the subject in which we never know what we are talking about, nor whether what we are saying is true."*


Russell (1917)

Mathematics are meant to be above those definition issues. Indeed, most of mathematics are defined with a well formed fondation and their validity has been tested over decades if not centuries.

In this part we expose some issues about the way mathematics are formulated in the original theories and how it may be possible to improve the way notions are defined to lift more ambiguity. This is done by explaining the properties of laguages and their ability to formalize other languages or even themselves. We expose the issues to be able to correctly identify and isolate them once we do a formalism of our own. This formalism will allow us to define precisely a new endomorphic meta-language for knowledge description that is used for encoding planning domains (see chapter 3 and chapter 4).

2.1.1 Abstraction

> **abstraction** (n.d.): *The process of formulating generalized ideas or concepts by extracting common qualities from specific examples*

Collins English Dictionary (2014)

The idea behind abstraction is to simplify the representation of complex instances. This mechanism is at the base of any knowledge representation system. Indeed, it is unnecessarily expensive to try to represent all properties of an object. An efficient way to reduce that knowledge representation is to prune away all irrelevant properties while also only keeping the ones that will be used in the context. *This means that abstraction is a loosy process* since information is lost when abstracting from an object.

Since this is done using a language as a medium, this language is a *host language*. Abstraction will refer to an instance using a *term* (also called symbol) of the host language. If the host language is expressive enough, it is possible to do abstraction on an object that is already abstract. The number of layers abstraction needed for a term is called its *abstraction level*. Very general notions have a higher abstraction level and we represent reality using the null abstraction level. In practice abstraction uses terms of the host language to bind to a referenced instance in a lower abstraction level. This forms a structure that is strongly hierarchical with higher abstraction level terms on top.

Example 1. We can describe an individual organism with a name that is associated to this specific individual. If we name a dog "Rex" we abstract a lot of information about a complex, dynamic living being. We can also abstract from a set of qualities of the specimen to build higher abstraction. For example, its species would be *Canis lupus familiaris* from the *Canidae* family. Sometimes several terms can be used at the same abstraction level like the commonly used denomination "dog" in this case.

Terms are only a part of that structure. It is possible to combine several terms into a *formula* (also called proposition, expression or statement).

2.1.2 Formalization

formal (adj.): *Relating to or involving outward form or structure, often in contrast to content or meaning.*

American Heritage Dictionary (2011a)

A formalization is the act to make formal. The word "formal" comes from Latin *fōrmālis*, from *fōrma*, meaning form, shape or structure. This is the same base as for the word "formula". In mathematics and *formal sciences* the act of formalization is to reduce knowledge down to formulas. Like stated previously, a formula combines several terms. But a formula must follow rules at different levels:

- *Lexical* by using terms belonging in the host language.
- *Syntactic* as it must follow the grammar of the host language.
- *Semantic* as it must be internally consistent and meaningful.

The information conveyed from a formula can be reduced to one element: its semantic structure. Like its etymology suggests, a formula is simply a structured statement about terms. This structure holds its meaning. Along with using abstraction, it becomes possible to abstract a formula and therefore, to make a formula about other formulae should the host language allowing it.

Example 2. The formula using English "dog is man's best friend" combines terms to hold a structure between words. It is lexically correct since it uses English words and grammatically correct since it can be grammatically decomposed as (n. v. n. p. adj. n.). In that the (n.) stands for nouns (v.) for verbs (adj.) for adjectives and (p.) for possessives. Since the verb "is" is the third person singular present indicative of "be", and the adjective is the superlative of "good", this form is correct in the English language. From there the semantic aspect is correct too but that is too subjective and extensive to formalize here. We can also build a formula about a formula like "this is a common phrase" using the referential pronoun "this" to refer to the previous formula.

Any language is comprised of formulas. Each formula holds knowledge about their subject and states facts or belief. A formula can describe other formulas and even *define* them. However, there is a strong limitation of a formalization. Indeed, a complete formalization cannot occur about the host language. It is possible to express formulae about the host language but *it is impossible to completely describe the host language using itself* (Klein 1975). This comes from two principal reasons. As abstraction is a loosy process one cannot completely describe a language while abstracting its definition. If the language is complex enough, its description requires an even more complex *meta-language* to describe it. And even for simpler language, the issue stands still while making it harder to express knowledge about the language itself. For this we need knowledge of the language *a priori* and this is contradictory for a definition and therefore impossible to achieve.

When abstracting a term, it may be useful to add information about the term to define it properly. That is why most formal system requires a *definition* of each term using a formula. This definition is the main piece of semantic information on a term and is used when needing to evaluate a term in a different abstraction level. However, this is causing yet another problem.

2.1.3 Circularity

> **circularity** (n.d.): *Defining one word in terms of another that is itself defined in terms of the first word.*

American Heritage Dictionary (2011b)

Circularity is one of the issues we explore in this section about the limits of formalization languages. Indeed, defining a term requires using a formula in the host language to express the abstracted properties of the generalization (Korzybski 1933). The problem is that most terms will have *circular* definitions.

Example 3. Using definitions from the American Heritage Dictionary (2011b), we can find that the term "word" is defined using the word "meaning" that is in turn defined using the term "word". Such circularity can happen to use an arbitrarily long chain of definition that will form a cycle in the dependencies.

This problem is very important as it is overlooked in most foundations of mathematics. Since a formalization cannot fully be self defined, another host language is generally used, sometimes without being acknowledged. This causes cycles in the dependencies of languages and theories in mathematics.

The only practical way to make some of this circularity disappear is to base a foundation of mathematics using natural language as host language for defining the most basic terms. This allows to acknowledge the problem in an instinctive way while being aware of it while building the theory.

2.2 Functional theory of mathematics

We aim to reduce the set of axioms allowing to describe a foundation of mathematics. The following theory is a proposition for a possible foundation that takes into account the previously described constraints. It is inspired by category theory (Awodey 2010), and typed lambda calculus (Barendregt 1984).

2.2.1 Category theory

This theory is based, as its name implies, on *categories*. A category is a mathematical structure that consist in two components: S

- B A set of **ojects** that can bee any arbitrary mathematical entities.
- S A set of **morphism** that are functional monomes. They are often represented as *arrows*.

Many definitions of categories pas de s exists ~~(~~(Barr and Wells 1990, vol. 49) but they are all in essence similar to this explanation. The best way to see the category theory is as a general theory of functions. Even if we can use any mathematical entity for the types of the components, the structure heavily implies a functional connotation.

2.2.2 Axioms

In this part, as an introduction to the fundamentals of maths and logic, we propose another view of that foundation based on functions. The unique advantage of it lays in its explicit structure that have emergent reflexive properties. It also holds a coherent ~~algebra that~~ has a strong expressive power. This approach can b be described as a special case of category theory. However, it differs in its priorities and formulation. For example, since our goal is to build a fondation of mathematics, it is impossible to fully specify the domain or co-domain of the functions and they are therefore weakly specified (Godel and Brown 1940).

ok mais tu le montres où ca ?

Our theory is axiomatic, meaning it is based on fundamental logical proposition called axioms. Those form~~s~~ the base of the logical system and therefore are accepted without any need for proof. In a nutshell, axiomes are true prior hypotheses.

The following axioms are the explicit base of the formalism. It is mandatory to properly state those axioms as all the theory is built on top of it.

$(=) = x \to x$

Axiom (Identity)**.** Let's the identity function be $=$ that associates every function~~s~~ to itself. This function is therefore transparent as by definition $=(x)$ is the same as x. It can be described by using it as **affectation** or aliases to make some expressions shorter or to define any function.

In the rest of the document, classical equality and the identity function will refer to the same notion.

That axiom implies that the formalism is based on *functions*. Any function can be used in any way as long as it has a single argument and returns only one value at a time (named image, which is also a function).

It is important to know that **everything** in our formalism *is* a function. Even notions such as literals, variables or set from classical mathematics are functions.

Using definition 1 and definition 2, we can note $(\to) = x \to (f(x) \to f)$

Axiom (Association)**.** Let's the term $\to$ be the function that associates two expression S to another function that associates those expressions. This special function is derived from the notation of *morphisms* of the category theory.

The formal definition uses currying to decompose the function into two. It associates a parameter to a function that takes an expression and returns a function.

Next we need to lay off the base definitions of the formalism.

2.2.3 Formalism definition

This functional algebra at the base of our foundation is inspired by *operator algebra* (Takesaki 2013, vol. 125) and *relational algebra* (Jónsson 1984). The problem with the operator algebra is that it supposes vectors and real numbers to work properly. Also, relational algebra, like category theory supposes of set theory.

Here we define the basic notions of our functional algebra that dictates the rules of the formalism we are defining.

Definition 1 (Currying)**.** Currying is the operation named after the mathematician Haskell Brooks Curry (1958) that allows multiple argument functions in a simpler monoidal formalism. A monome is a function that has only one argument and has only one value, as in our main axiom. rappeler l'axiome

$\llparenthesis f \rrparenthesis = (x \to \llparenthesis f(x) \rrparenthesis)$

The operation of *currying* is a function $\llparenthesis\rrparenthesis$ that associates to each function f another function that recursively partially applies f with one argument at a time.

$\lbag f \rbag = \lbag x, y \to f(x)(y) \rbag^{+}$

If we take a function h such that when applied to x gives the function g that takes an argument y, *unCurrying* is the function $\lbag\rbag$ so that $f(x,y)$ behaves the same way as $h(x)(y)$. We note $h = \lbag f \rbag$.

Definition 2 (Application)**.** We note the application of f with an argument x as $f(x)$. The application allows to recover the image y of x which is the value that f associates with x.

$y = f(x)$

Along with Currying, function application can be used *partially* to make constant some arguments.

Definition 3 (Partial Application)**.** We call *partial application* the application using an insufficient number of arguments to any function f. This results in a function that has fewer arguments with the first ones being locked by the partial application. It is interesting to note that currying is simply a recursion of partial applications.

From now on we will note $f(x, y, z, \ldots)$ any function that has multiple arguments but will suppose that they are implicitly Curryied. If a function only takes two arguments, we can also use the infix notation e.g. xfy for its application.

Example 4. Applying this to basic arithmetic for illustration, it is possible to create a function that will triple its argument by making a partial application of the multiplication function $\times(3)$ so we can write the operation to triple the number 2 as $\times(3)(2)$ or $\times(2,3)$ or with the infix notation 2×3.

$\times(3) = x \to 3 \times x$

Definition 4 (Null)**.** The *null function* is the function between nothing and nothing. We note it $\gtrdot$.

$\gtrdot = \gtrdot \to \gtrdot$

The notation $\gtrdot$ was chosen to represent the association arrow ($\to$) but with a dot instead of the tail of the arrow. This is meant to represent the fact that it inhibits association.

2.2.4 Literals and Variables

As everything is a function in our formalism, we use the null function to define notions of variables and literals.

$l = \gtrdot \to l$ **Definition 5** (Literal)**.** A literal is a function that associates null to its value. This consists of any function l written as $\gtrdot \to l$. This means that the function has only itself as an immutable value. We call *constants* functions that have no arguments and have as value either another constant or a literal.

Example 5. A good example of that would be the yet to be defined natural numbers. We can define the literal 3 as $3 = \gtrdot \to 3$. This is a fonction that takes no argument and is always valued to 3.

$x = x \to \gtrdot$ **Definition 6** (Variable)**.** A variable is a function that associates itself to null. This consists of any function x written as $x \to \gtrdot$. This means that the function requires an argument and has undefined value. Variables can be seen as a demand of value or expression and mean nothing without being defined properly.

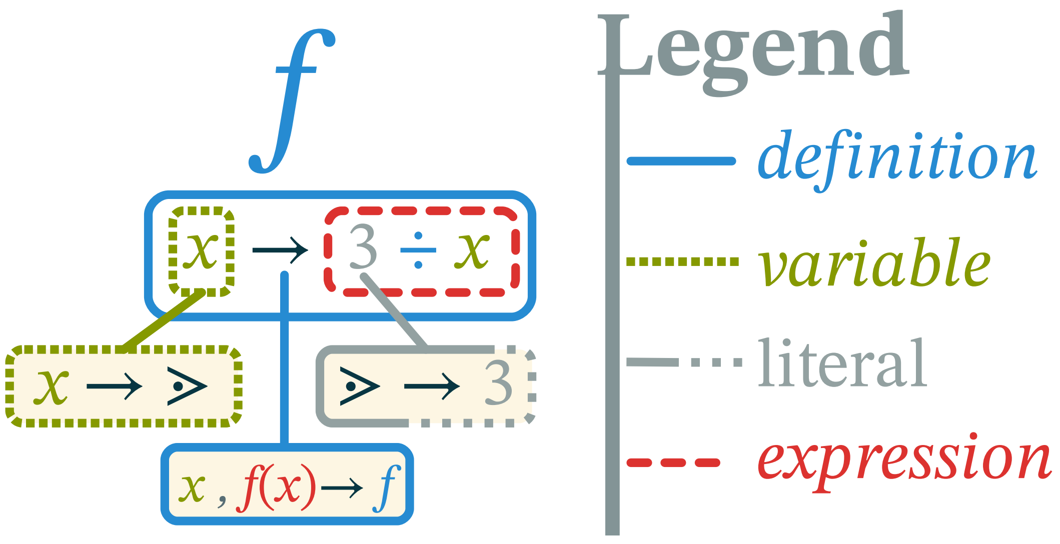


Figure 2.1: Illustration of basic functional operators and their properties.

Example 6. The function f defined in figure 2.1 associates its argument to an expression. Since the argument x is also a variable, the value is therefore dependant on the value required by x. In that example, the number 3 is a literal and $3 \div x$ is therefore an expression using the function $\div$.

An interesting property of this notation is that $\gg$ is both a variable and a constant. Indeed, by definition, $\gg$ is the function that associates $\gg \rightarrow \gg$ and fullfil both definitions.
S

it
When defining currying, we annotated with the formalism $(\!|f|\!) = f \rightarrow (x \rightarrow (\!|f(x)|\!))$. The obvious issue is the absence of stopping condition in that recursive expression. While the end of the recursion doesn't technically happen, in practice from the way variables and literals are defined, the recursion chain either ends up becoming a variable or a constant because it is undefined when currying a nullary function.

2.2.5 Functional algebra

Inspired by relational algebra and by category theory, we present a new functional algebra. The first operator of this algebra allows to combine several functions into one. This is very useful to add to the definition of a function by specifying special cases using a reduced notation.

Definition 7 (Combination). The *combination function* $\bowtie$ associates any two functions to a new function that is the union of the definition of **either** functions.

$f_1 \bowtie f_2 = x \rightarrow y$ with $f_1(x) = y$ **or** $f_2(x) = y$.

Example 7. For two functions f_1 and f_2 that are defined respectively by:

- $f_1 = (1 \rightarrow 2)$
- $f_2 = (2 \rightarrow 3)$

the combination $f_3 = f_1 \bowtie f_2$ will behave as follows:

- $f_3(1) = 2$
- $f_3(2) = 3$

Definition 8 (Superposition). The *superposition function* $\vartriangle$ is the function that associates any two functions to a new function that associates any function associated by **both** functions.

$f_1 \vartriangle f_2 = x \rightarrow y$ with $f_1(x) = f_2(x) = y$.

We can say that the superposition is akin to an intersection where the resulting function is defined when both functions have the same behavior.

Example 8. Reusing the functions of the previous example, we can note that $f_3 \vartriangle f_1 = f_1$ because $1 \rightarrow 2$ is the only combination that both functions associates.

Definition 9 (Subposition). The *subposition function* is the function $\triangledown$ that associates any two functions f_1 and f_2 to a new function $f_1 \triangledown f_2$ that associates anything associated by the first function f_1 but **not** by the second function f_2.

The subposition is akin to a substraction of function where we remove everything defined by the second function to the definition of the first.

We can also note a few properties of these functions:
tu ne peux pas intégrer une table dans ton discours comme cela. Tu dois la citer.

Table 2.1: Example properties of superposition an subposition

Formula	Description
$f \vartriangle f = f$	A function superposed to itself is the same.
$f \vartriangle \gtrdot = \gtrdot$	Any function superposed by null is null.
$f_1 \vartriangle f_2 = f_2 \vartriangle f_1$	The superposition order doesn't affect the result.
$f \triangledown f = \gtrdot$	A function subposed by itself is always null.
$f \triangledown \gtrdot = f$	Subposing null to any function doesn't change it.

These functions are intuitively the functional equivalent of the union, intersection and difference from set theory. In our formalism we will define the set operations from these.

The following operators are the classical operations on functions.

Definition 10 (Composition)**.** The *composition function* is the function that associates any two functions f_1 and f_2 to a new function such that: $f_1 \circ f_2 = x \to f_1(f_2(x))$.

Definition 11 (Inverse)**.** The *inverse function* is the function that associates any function to its inverse such that if $y = f(x)$ then $x = \bullet(f)(y)$.

We can also use an infix version of it with the composition of functions: $f_1 \bullet f_2 = f_1 \circ \bullet(f_2)$.

These properties are akin to multiplication and division in arithmetic.

Table 2.2: Example of function composition and inverse with their properties.

Formula	Description
$f \circ \gtrdot = \gtrdot$	This means that $\gtrdot$ is the absorbing element of the composition.
$f \circ (=) = f$	Also, $=$ is the neutral element of the composition.
$\bullet(\gtrdot) = \gtrdot \wedge \bullet(=) = (=)$	This means that $\gtrdot$ and $=$ are commutative functions.
$f_1 \circ f_2 \neq f_2 \circ f_1$	However, $\circ$ is not commutative.

From now on, we will use numbers and classical arithmetic as we had defined them. However, we consider defining them from a foundation point of view, later using set theory and Peano's axioms.

In classical mathematics, the inverse of a function f is often written as f^{-1}. Therefore we can define the transitivity of the n^{th} degree as the power of a function such that $f^n = f^{n-1} \circ f$. Figure 2.2 shows how the power of a function is behaving at key values.

By generalizing the formula, we can define the *transitive cover* of a function f and its inverse respectively as $f^+ = f^{+\infty}$ and $f^- = f^{-\infty}$. This cover is the application of the function to its result infinitely. This is useful especially for graphs as the transitive cover of the adjacency function of a graph gives the connectivity function (see section 2.5).

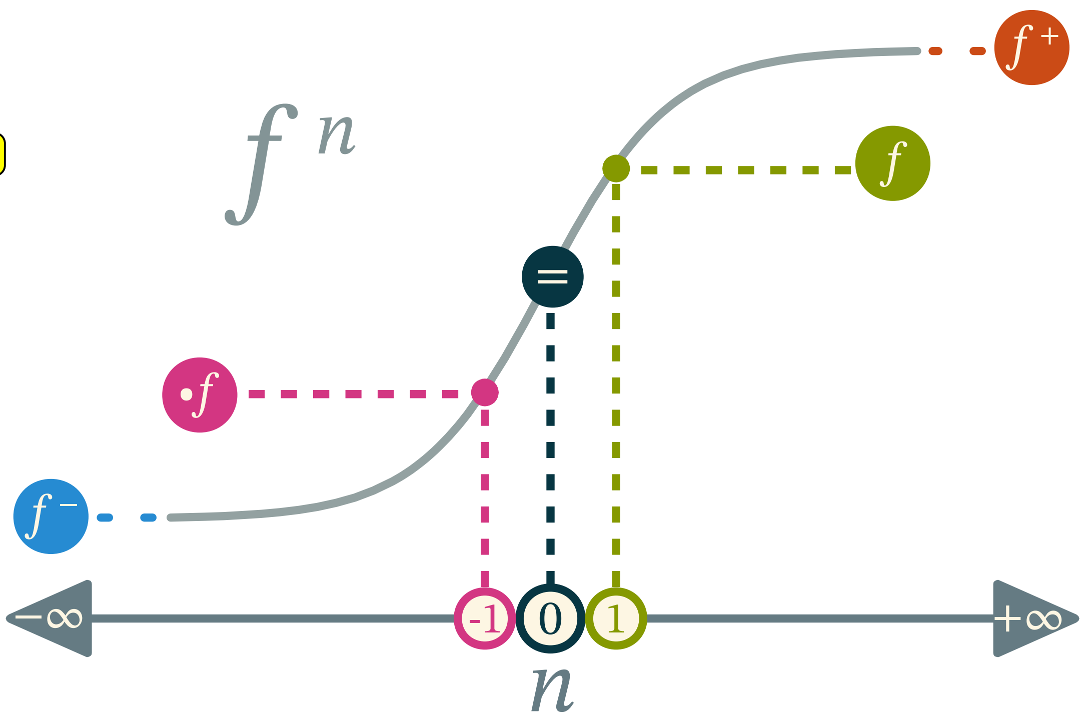


Figure 2.2: Illustration of how the functional equivalent of the power function is behaving with notable values (in filled circles)

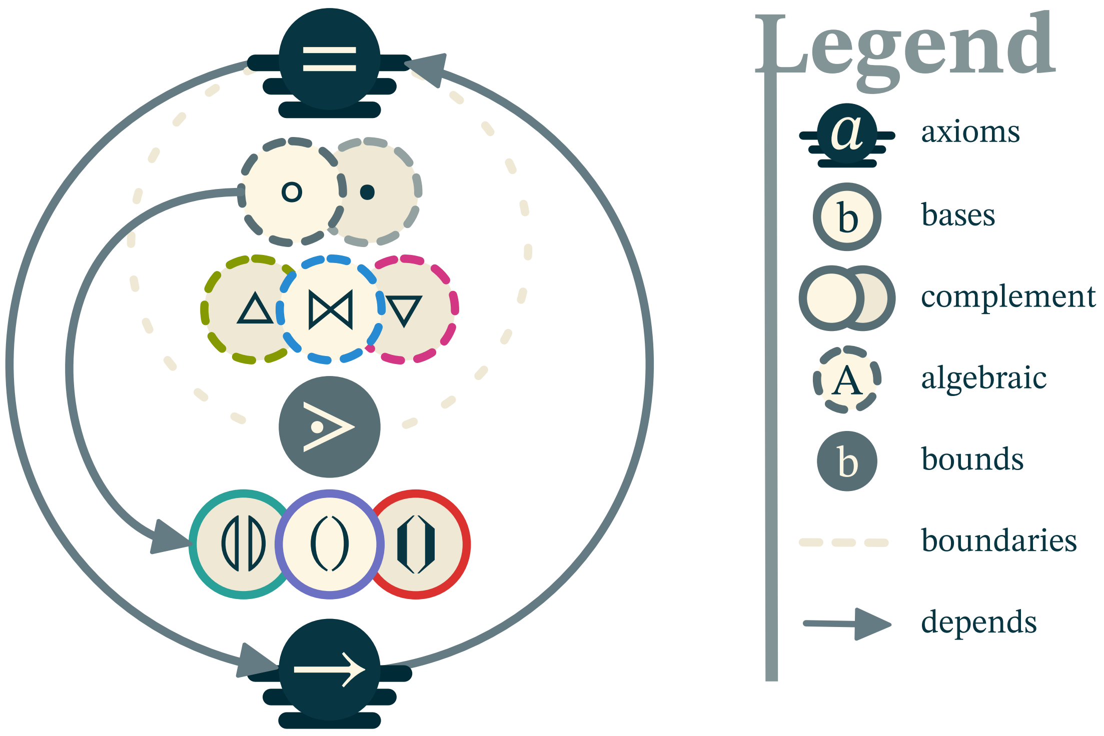
We also call *arity* the number of arguments (or the currying order) of a function noted $|f|$.

2.2.6 Properties

A modern approach of mathematics is called *reverse mathematics* as instead of building theorems from axioms, we search the minimal set of axioms required by a theorem. Inspired by this, we aim to minimize the formal basis of our system as well as identifying the circularity issues, we provide a dependency graph in figure 2.3. We start with the axiom of Association at the bottom and the axiom of Identity at the top. Everything depends on those two axioms but drawing all the arrows makees the figuree way less leggible. vérifier 'l'anglais

Then we define the basic application function $()$ that has as complement the Currying $(\!|\,|\!)$ and unCurrying $\{\!|\,|\!\}$ functions. Similarly, the combination $\bowtie$ has the superposition $\vartriangle$ and the subpostion $\triangledown$ functions as complements. The bottom bound of the algebra is the null function $\gtrdot$ and the top is the identity function $=$. Composition $\circ$ is the the main operator of the algebra and allows it to have an inverse element as the inverse function $\bullet$. The composition function needs the application function in order to be constructed.

The algebra formed by the previously defined operations on functions is a semiring $(\mathbb{F}, \bowtie, \circ)$ with $\mathbb{F}$ being the set of all functions.



ok le début de 2.2.6 explique bien ta figure, il faudrait faire aussi bien pour toutes tes autres figures !

Figure 2.3: Dependency graph of notions in the functional theory

Indeed, $\bowtie$ is a commutative monoid having $\gtrdot$ as its identity element and $\circ$ is a monoid with $(=)$ as its identity element.

Also the composition of the combination is the same as the combination of the composition. Therefore, $\circ$ (composition) distributes over $\bowtie$ (combination).

At last, using partial application composing with null gives null: $\circ(\gtrdot) = ((\gtrdot) \to \gtrdot) = \gtrdot$.

This foundation is now ready to define other fields of mathematics. We start with logic as it is a very basic formalism in mathematics. ~~Then we redefine the ZFC set theory using our formalism as a base.~~ And finally we will present derived mathematical tools to represent data structures and their properties.

2.3 First Order Logic

In this section, we present First Order Logic (FOL). FOL is based on boolean logic with the two literals ue and $\bot$ *false*.

$\mathcal{D}(\bullet ?) = \{\bot, \top\}$ A function noted $(?)$ that have as only values either $\top$ or $\bot$ is called a **predicate**.

We define the classical logic *entailment*, the predicate that holds true when a predicate (the conclusion) is true if and only if the first predicate (the premise) is true.

$$\vdash = (\bot, x \to \top) \bowtie (\top, x \to x)$$

Then we define the classical boolean operators $\neg$ *not*, $\wedge$ *and* and $\vee$ *or* as:

- $\neg = (\bot \to \top) \bowtie (\top \to \bot)$, the negation associates true to false and false to true.
- $\wedge = x \to ((\top \to x) \bowtie (\bot \to \bot))$, the conjunction is true when all its arguments are simultaneously true.
- $\vee = x \to ((\top \to \top) \bowtie (\bot \to x))$, the disjunction is true if all its arguments are not false.

The last two operators are curried function and can take any number of arguments as necessary and recursively apply their definition.

D e

Another basic preicate is the **equation**. It is th identity function $=$ but as a binary predicate that is true whenever the two arguments are the same.

pas de s

Functions that takes an expression as parameters are called *modifiers*. FOL introduces a useful kind of modifier used to modalize expressions: *quantifiers*. Quantifiers take an expression and a variable as arguments. Classical quantifiers are also predicates: they restrict the values that the variable can take.

The classical quantifiers are:

- The *universal quantifier* $\forall$ meaning *"for all"*.
- The *existential quantifier* $\exists$ meaning *"it exists"*.

$\forall = \S(\wedge)$
$\exists = \S(\vee)$

They are sometimes extended with :

- The *uniqueness quantifier* $\exists!$ meaning *"it exists a unique"*.
- The *exclusive quantifier* $\nexists$ meaning *"it doesn't exist"*.

$\exists! = \S(= (1) \circ +)$
$\nexists = \S(\neg \circ \wedge)$

Another exotic quantifier that isn't a predicate can be proven useful (Hehner 2012):

- The *solution quantifier* $\S$ meaning *"those"*.

$\S = f, x, ? \to \{\!\{ f(x) : ? \}\!\}$

The last three quantifiers are optional in FOL but will be conducive later on. It is interesting to note that most quantified expression can be expressed using the set builder notation discussed in the following section.

2.4 Set Theory

Since we need to represent knowledge, we will handle more complex data than simple booleans. One such way to describe more complex knowledge is by using set theory. It is used as the classical foundation of mathematics. Most other proposed foundation of mathematics invoke the concept of sets even before their first formula to describe the kind of notions they are introducing. The issue is then to define the sets themselves. At the beginning of his founding work on set theory, Cantor wrote:

"*A set is a gathering together into a whole of definite, distinct objects of our perception or of our thought–which are called elements of the set.*"

problème avec photo

For Cantor, a set is a collection of concepts and percepts. In our case both notions are grouped in what we call *objects*, *entities* that are all ultimately *functions* in our formalism.

2.4.1 Base Definitions

This part is based on the work of Cantor (1895) and the set theory. The goal is to define the notions of set theory using our formalism.

Definition 12 (Set). A collection of *distinct* objects considered as an object in its own right. We define a set one of two ways (always using braces):

- In extension by listing all the elements in the set: $\{0, 1, 2, 3, 4\}$
- In intention by specifying the rule that all elements follow: $\{n : ?(n)\}$

Using our functional foundation, we can define any set as a predicate $\mathcal{S} = e \to \top$ with e being a member of $\mathcal{S}$. This allows us to define the member function noted $e \in \mathcal{S}$ to indicate that e is an element of $\mathcal{S}$.

$\in = e, \mathcal{S} \to \mathcal{S}(e)$

Another, useful definition using sets is the *domain* of a function f as the set of all arguments for which the function is defined. We call *co-domain* the domain of the inverse of a function. We can note them $f : \mathcal{D}(f) \mapsto \mathcal{D}(\bullet f)$. In the case of our functional version of sets, they are their own domain.

Definition 13 (Specification). The *function of specification* (noted $:$) is a function that restricts the validity of an expression given a predicate. It can ~~be~~ intuitively be read as *"such that"*.

$(:) = f, ? \to f \triangledown (\mathcal{D}(? = \bot) \mapsto \mathcal{D}(\bullet f))$

supprimer premier be

The specification operator is extensively used in classical mathematics but informally, it is often seen as an extension of natural language and can be quite ambiguous. In the present document any usage of $(:)$ in any mathematical formula will follow the previously discussed definition.

2.4.2 Set Operations

Along with defining the domains of functions using sets, we can use function on sets. ~~This is very important in order to define ZFC and is extensively used in the rest of the document.~~

In this section, basic set operations are presented. The first one is the subset.

Definition 14 (Subset). A subset is a part of a set that is integrally contained within it. We note $\mathcal{S} \subset \mathcal{T} \vdash ((e \in \mathcal{S} \vdash e \in \mathcal{T}) \wedge \mathcal{S} \neq \mathcal{T})$, that a set $\mathcal{S}$ is a proper subset of a more general set $\mathcal{T}$.

Definition 15 (Union). The union of two or more sets $\mathcal{S}$ and $\mathcal{T}$ is the set that contains all elements in *either* set. We can note it:

$$\mathcal{S} \cup \mathcal{T} = \{e : e \in \mathcal{S} \vee a \in \mathcal{T}\}$$

Definition 16 (Intersection)**.** The intersection of two or more sets $\mathcal{S}$ and $\mathcal{T}$ is the set that contains only the elements member of *both* set. We can note it:

$$\mathcal{S} \cap \mathcal{T} = \{e : e \in \mathcal{S} \wedge e \in \mathcal{T}\}$$

Definition 17 (Difference)**.** The difference of one set $\mathcal{S}$ to another set $\mathcal{T}$ is the set that contains only the elements contained in the first but not the last. We can note it:

$$\mathcal{S} \setminus \mathcal{T} = \{e : e \in \mathcal{S} \wedge e \notin \mathcal{T}\}$$

An interesting way to visualize relationships with sets is by using Venn diagrams (Venn 1880). In figure 2.4 we present the classical union, intersection and difference operations. It also introduce a new way to represent more complicated notions such as the cartesian product by using a representation for powerset and higher dimensionality inclusion that a 2D Venn diagram cannot represent.

S

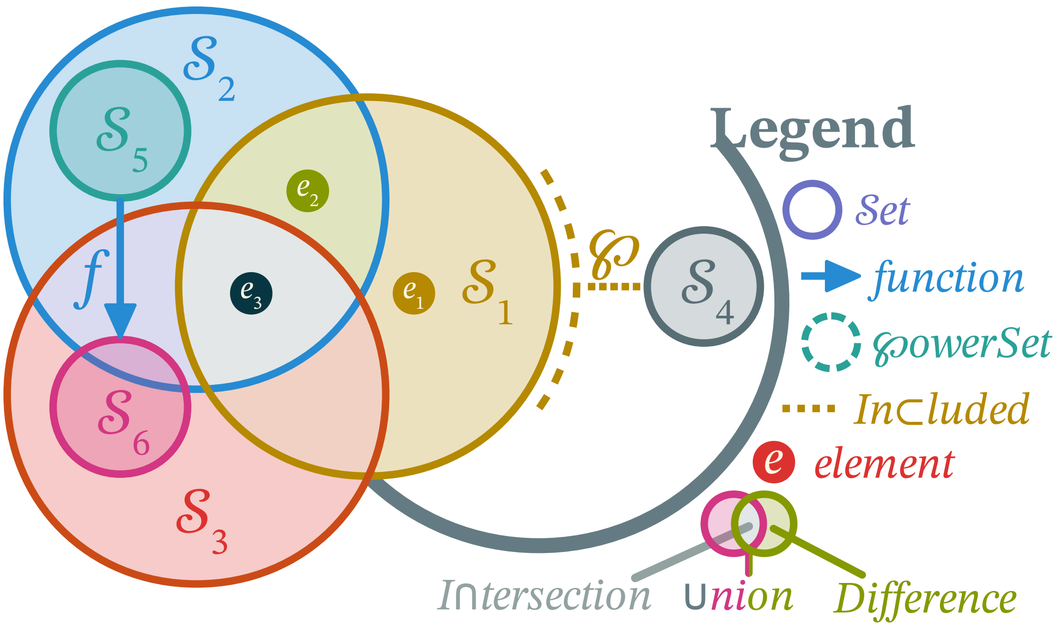


Figure 2.4: Example of an upgraded Venn diagram to illustrate operations on sets.

Example 9. Figure 2.4 is the graphical representation of these statements:
the statements presented in Table 2.3

Table 2.3: Statements represented in the extended Venn diagram in figure 2.4.

Formula	Description
$e_1 \in \mathcal{S}_1$	e_1 is an element of the set $\mathcal{S}_1$.
$e_2 \in \mathcal{S}_1 \cap \mathcal{S}_2$	e_2 is an element of the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$.
$e_3 \in \mathcal{S}_1 \cap \mathcal{S}_2 \cap \mathcal{S}_3$	e_3 is an element of the intersection of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$.
$\mathcal{S}_5 \subset \mathcal{S}_2$	$\mathcal{S}_5$ is a subset of $\mathcal{S}_2$.

Formula	Description
$\mathcal{S}_6 \subset \mathcal{S}_2 \cup \mathcal{S}_3$	$\mathcal{S}_6$ is a subset of the union of $\mathcal{S}_2$ and $\mathcal{S}_3$.
$f = \mathcal{S}_5 \mapsto \mathcal{S}_6$	f is a function which domain is $\mathcal{S}_5$ and co-domain is $\mathcal{S}_6$.
$\mathcal{S}_4 \subset \wp(\mathcal{S}_1)$	$\mathcal{S}_4$ is a combination of elements of $\mathcal{S}_1$.

These Venn diagrams, originally have a lack of expressivity regarding complex operations on sets. Indeed, from their dimensionality it is complicated to express numerous sets having intersection and disjunctions. For example, it is difficult to represent the following notion.

S **Definition 18** (Cartesian product). The Cartesian product of two sets $\mathcal{S}$ and $\mathcal{T}$ is the set that contains all possible combination of an element of both sets. These combinations are a kind of ordered set called *tuples*. We note this product:

$$\mathcal{S} \times \mathcal{T} = \{\langle e_{\mathcal{S}}, e_{\mathcal{T}} \rangle : e_{\mathcal{S}} \in \mathcal{S} \wedge e_{\mathcal{T}} \in \mathcal{T}\}$$

From this we can also define the set power recursively by $\mathcal{S}^1 = \mathcal{S}$ and $\mathcal{S}^n = \mathcal{S} \times \mathcal{S}^{n-1}$.

The Cartesian product is the set equivalent of currying as:

$$\mathcal{S} \times \mathcal{T} = e_{\mathcal{S}}, e_{\mathcal{T}} \to \mathcal{S}(e_{\mathcal{S}}) \wedge \mathcal{T}(e_{\mathcal{T}})$$

The angles $\langle\rangle$ notation is used for tuples, those are another view on currying by replacing several arguments using a single one as an ordered list. A tuple of two elements is called a *pair*, of three elements a *triple*, etc. We can access elements in tuples using their index in the following way $e_2 = \langle e_1, e_2, e_3 \rangle_2$.

Definition 19 (Mapping). The mapping notation $\{\!\!\{\}\!\!\}$ is a function such that $\{\!\!\{ f(x) : x \in \mathcal{S} \}\!\!\}$ will give the result of applying all elements in set $\mathcal{S}$ as arguments of the function using the unCurrying operation recursively. If the function isn't specified, the mapping will select a member of the set non deterministically. The function isn't defined on empty sets or on sets with fewer members than arguments of the provided function.

Example 10. The classical sum operation on numbers can be noted:

$$\sum_{i=1}^{3} 2i = \{\!\!\{ +(2 * i) : i \in [1,3] \}\!\!\} = +(2 * 1)(+(2 * 2)(2 * 3))$$

2.4.3 The ZFC Theory

The most common axiomatic set theory is ZFC (Kunen 1980, vol. 102). In that definition of sets there are a few notions that come from its axioms. By being able to distinguish elements in the set from one another we assert that elements have an identity and we can derive equality from there:

Axiom (Extensionality). $\forall \mathcal{S} \forall \mathcal{T} : \forall e((e \in \mathcal{S}) = (e \in \mathcal{T})) \vdash \mathcal{S} = \mathcal{T}$

~~This means that two sets are equal if and only if they have all their members in common.~~

~~Another axiom of ZFC that is crucial in avoiding Russel's paradox ($\mathcal{S} \in \mathcal{S}$) is the following:~~

~~**Axiom** (Foundation). $\forall \mathcal{S} : (\mathcal{S} \neq \varnothing \vdash \exists \mathcal{T} \in \mathcal{S}, (\mathcal{T} \cap \mathcal{S} = \varnothing))$~~

~~This axiom uses the empty set $\varnothing$ (also noted $\{\}$) as the set with no elements. Since two sets are equal if and only if they have precisely the same elements, the empty set is unique.~~

~~The definition by intention uses the set builder notation to define a set. It is composed of an expression and a predicate $?$ that will make any element e in a set $\mathcal{T}$ satisfying it part of the resulting set $\mathcal{S}$, or as formulated in ZFC:~~

~~**Axiom** (Specification). $\forall ? \forall \mathcal{T} \exists \mathcal{S} : (\forall e \in \mathcal{S} : (e \in \mathcal{T} \land ?(e)))$~~

~~The last axiom of ZFC we use is to define the power set $\wp(\mathcal{S})$ as the set containing all subsets of a set $\mathcal{S}$:~~

~~**Axiom** (Power set). $\wp(\mathcal{S}) = \{\mathcal{T} : \mathcal{T} \subseteq \mathcal{S}\}$~~

~~With the symbol $\mathcal{S} \subseteq \mathcal{T} \vdash (\mathcal{S} \subset \mathcal{T} \lor \mathcal{S} = \mathcal{T})$. These symbols have an interesting property as they are often used as a partial order over sets.~~

2.5 Graphs

With set theory, it is possible to introduce all of standard mathematics. A field of interest for this thesis is the study of the structure of data. This interest arrises from the need to encode semantic information in a knowledge base using a very simple language (see chapter 3). Most of these structures use graphs and isomorphic derivatives.

OK là tu justifies pourquoi tu présentes des notions liées au Graph en faisant le lien avec la suite de ta thèse.

Definition 20 (Graph). A graph is a mathematical structure g which is defined by its *connectivity function* χ that

il manque une fin à ta phrase !!!!

2.5.1 Adjacency, Incidence and Connectivity

Definition 21 (Connectivity). The connectivity function is a combination of the classical adjacency and incidence functions of the graph. It is defined using a circular definition in the following way:

- *Adjacency*: $\chi_{\circ\!-\!\circ} = v \to \{e : v \in \chi_{\bullet\!-}(e)\}$
- *Incidence*: $\chi_{\bullet\!-} = e \to \{v : e \in \chi_{\circ\!-\!\circ}(v)\}$

Also: $\chi_{\circ\!-\!\circ} = \bullet\chi_{\bullet\!-}$

Defining either function defines the graph. For convenience, the connectivity function combine the adjacency and incidence:
S

$$\chi = \chi_{\circ\!-\!\circ} \bowtie \chi_{\bullet\!-}$$

$E \subseteq V^2$ Usually, graphs are noted $g = (V, E)$ with the set of vertices V (also called nodes) and edges E (arcs) that links two vertices together. Each edge is classically a pair of vertices ordered or not depending on if the graph is directed or not. It is possible to go from the set based definition to the functional relation using the following equation: $\mathcal{D}(\chi_{\bullet\!-}) = E$
whether

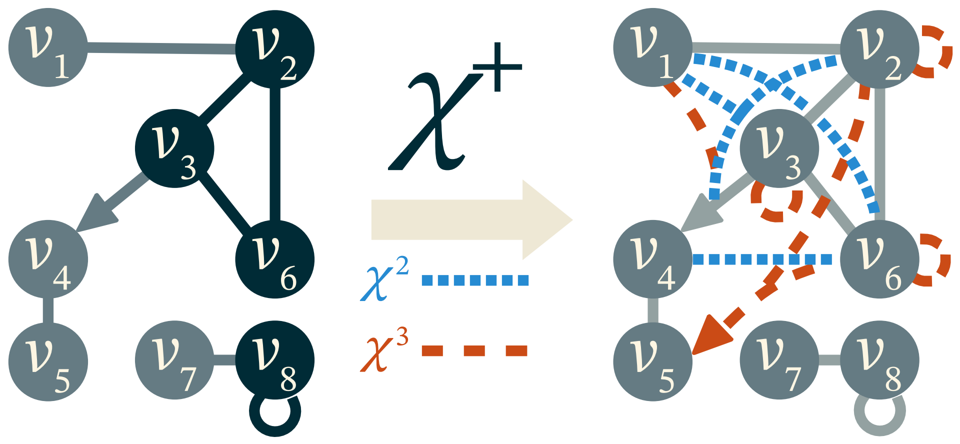


Figure 2.5: Example of the recursive application of the transitive cover to a graph.

Example 11. A graph is often represented with lines or arrows linking points together like illustrated in figure 2.5. In that figure, the vertices v_1 and v_2 are connected through an undirected edge. Similarly v_3 connects to v_4 but not the opposite since they are bonded with a directed edge. The vertex v_8 is also connected to itself.
expliquer plus ce qui est nouveau, la tu expliques des choses que tout le monde connait (directed and undirected edge) sur la partie gauche. Par contre, sur la partie droite , tu n'expliques rien et c'est ça qu'on ne comprends pas !!!!

2.5.2 Digraphs

je ne comprens pas tu parles déjà des graphs dirigés juste avant

supprimer
~~In~~ digraphs or *directional graphs* are a specific case of graphs where edges have a direction. This means that we can have two vertices v_1 and v_2 linked by an edge and while it is possible to go from v_1 to v_2, the inverse is impossible. For such case the edges are ordered pairs and the incidence function can be decomposed into:

$$\chi_{\bullet\!-} = \chi_{\succ\!\bullet} \bowtie \chi_{\bullet\!\prec}$$

We note $\chi_{\succ\!\bullet}$ the **incoming relation** and $\chi_{\bullet\!\prec}$ the **outgoing relation**.

In digraphs, classical edges can exist if allowed and will simply be bi-directional edges.

2.5.3 Path, cycles and transitivity

Most of the intrinsic information of a graph is contained within its structure. Exploring
S its properties require to study the "shape" of a graph and to find relationships between vertices. That is why graph properties are easier to explain using the transitive cover χ^+ of any graph $g = (V, E)$.

ah ok c'est là que tu expliques la partie droite de ta figure. Mais ca n'est pas clair, détaille plus.

This transitive cover will create another graph in which two vertices are connected through an edge if and only if it exists a path between them in the original graph g. We illustrate this process in figure 2.5. Note how there is no edge in $\chi^2(g)$ between v_5 and v_6 and the one in $\chi^3(g)$ is directed towards v_5 because there is no path back to v_6 since the edge between v_3 and v_4 is directed.

Definition 22 (Path)**.** We say that vertices v_1 and v_2 are *connected* if it exists a path from one to the other. Said otherwise, there is a path from v_1 to v_2 if and only if $\langle v_1, v_2 \rangle \in \mathcal{D}(\chi^+(g))$.

The notion of connection can be extended to entire graphs. An undirected graph g is said to be *connected* if and only if $\forall e \in V^2 (e \in \mathcal{D}(\chi^+(g)))$.

Similarly we define *cycles* as the existence of a path from a given vertex to itself. For example, in figure 2.5, the cycles of the original graph are colored in blue. Some graphs can be strictly acyclical, enforcing the absence of cycles.

2.5.4 Trees

A **tree** is a special case of a graph. A tree is an acyclical connected graph. If a special vertex called a *root* is chosen, we call the tree a *rooted tree*. It can then be a directed graph with all edges pointing away from the root. When progressing away from the root, we call the current vertex *parent* of all exterior *children* vertices. Vertex with no children are called *leaves* of the tree and the rest are called *branches*.

An interesting application of trees to FOL is called *and/or trees* where each vertex has two sets of children: one for conjunction and the other for disjunction. Each vertex is a logic formula and the leaves are atomic logic propositions. This is often used for logic problem reduction. In figure 2.6 we illustrate how and/or trees are often depicted.

2.5.5 Quotient

Another notion often used for reducing big graphs is the quotiening as illustrated in figure 2.7.

Definition 23 (Graph Quotient)**.** A quotient over a graph is the act of reducing a subgraph into a node while preserving the external connections. All internal structure becomes ignored and the subgraph now acts like a regular node. We note it $\div_f(g) = (\{f(v) : v \in V\}, \{f(e) : e \in E\})$ with f being a function that maps any vertex either toward itself or toward its quotiened vertex.

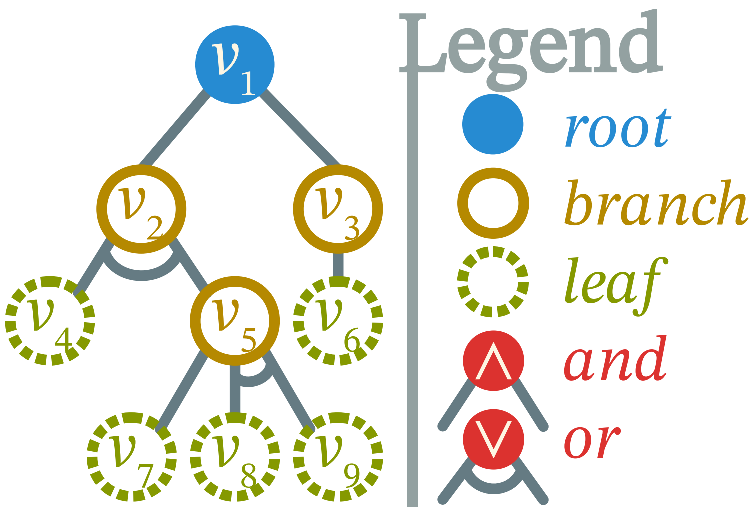


Figure 2.6: Example of and/or tree.

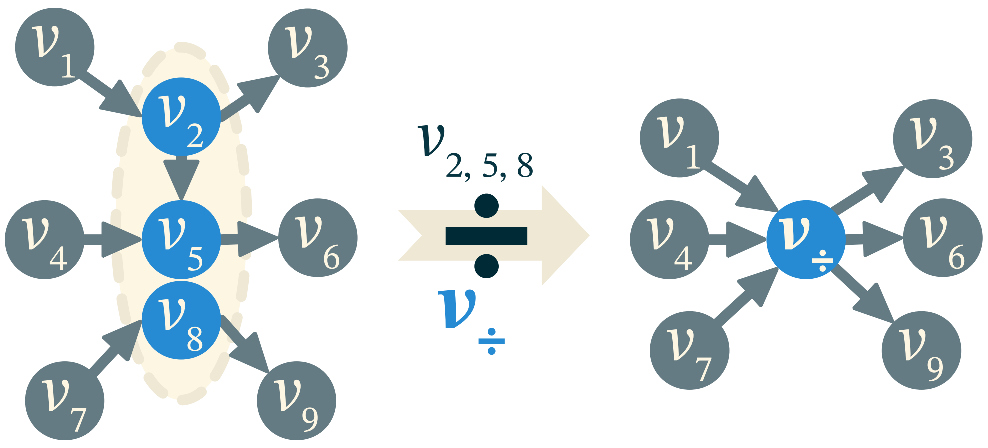


Figure 2.7: Example of graph quotient.

?

A quotion can be thought of as the operation of merging several vertices into one while ? keeping concerving their connections with other vertices.

Example 12. Figure 2.7 explains how to do the quotient of a graph by merging the vertices v_2, v_5 and v_8 into $v_\div$. The edge beetween v_2 and v_5 is lost since it is inside the quotienned part of the graph. All other edges are now connected to the new vertex $v_\div$.

2.5.6 Hypergraphs

A generalization of graphs are **hypergraphs** where the edges are allowed to connect to more than two vertices. They are often represented using Venn-like representations but can also be represented with edges "gluing" several vertex like in figure 2.8.

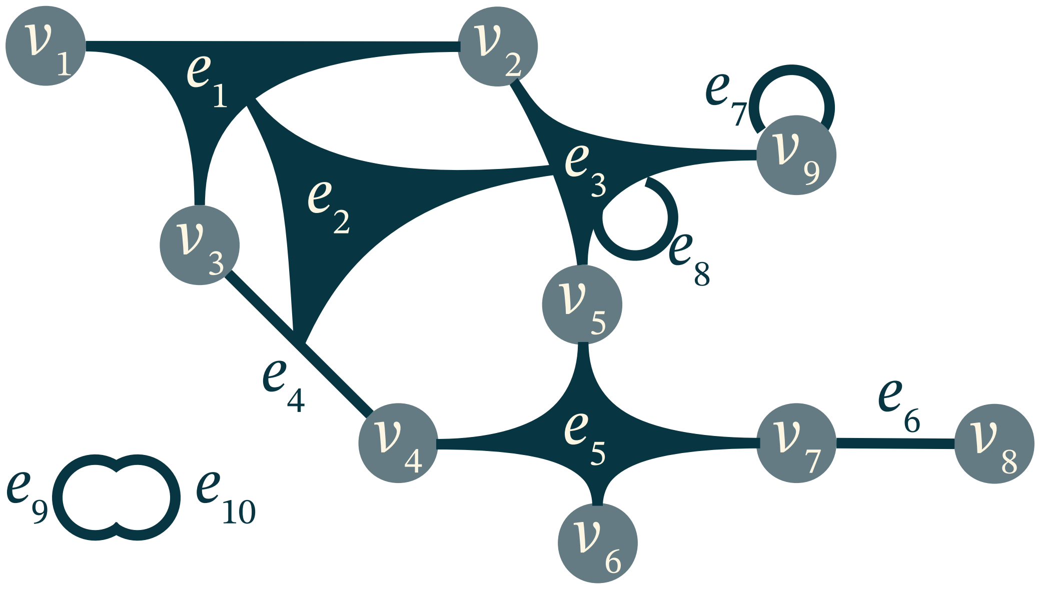


Figure 2.8: Example of hypergraph with total freedom on the edges specification.

Example 13. In figure 2.8, vertices are the discs and edges are either lines or gluing surfaces. In hypergraph, classical edges can exist like e_4, e_6 or e_7. Taking for example e_1, we can see that it connects 3 vertices: v_1, v_2 and v_3. It is also possible to have an edge connecting edges like e_8 that connects e_3 to itself. Edges can also "glue" more than two edges like e_2 connects e_1, e_3 and e_4. The more exotic structures allows are ? edge-loops as seen with e_9 and $e_1 0$ which allows edge only graphs.

An hypergraph is said to be *n-uniform* if the edges are restricted to connect to only n vertices together. In that regard, classical graphs are 2-uniform hypergraphs.

Hypergraphs have a special case where $E \subset V$. This means that edges are allowed to connect to other edges. In figure 2.8, this is illustrated by the edge e_3 connecting to je vois e8 et e2, qui est le 3 eme ?

three other edges. Information about these kinds of structures for knowledge representation is hard to come by and rely mostly on a form of "folk wisdom" within the mathematics community where knowledge is rarely published and mostly transmitted orally during lessons. One of the closest information available is this forum post (Kovitz 2018) that associated this type of graph to port graphs (Silberschatz 1981). Additional information was found in the form of a contribution of Vepstas (2008) on an encyclopedia article about hypergraphs. In that contribution, he says that a generalization of hypergraph allowing for edge-to-edge connections violate the axiom of Foundation of ZFC by allowing edge loops. Indeed, like in figure 2.8, an edge $e_9 = \{e_{10}\}$ can connect to another edge $e_{10} = \{e_9\}$ causing an infinite descent inside the $\in$ relation in direct contradiction with ZFC.

This shows the limits of standard mathematics especially on the field of knowledge representation. Some structures needs higher dimensions than allowed by the structure of ZFC and FOL. However, it is important not to be mistaken: such non-standard set theories are more general than ZFC and therefore contains ZFC as a special case. All is a matter of restrictions.

2.6 Sheaf

In order to understand sheaves, we need to present a few auxiliary notions. Most of these definitions are adapted from (Vepštas 2008). The first of which is a seed.

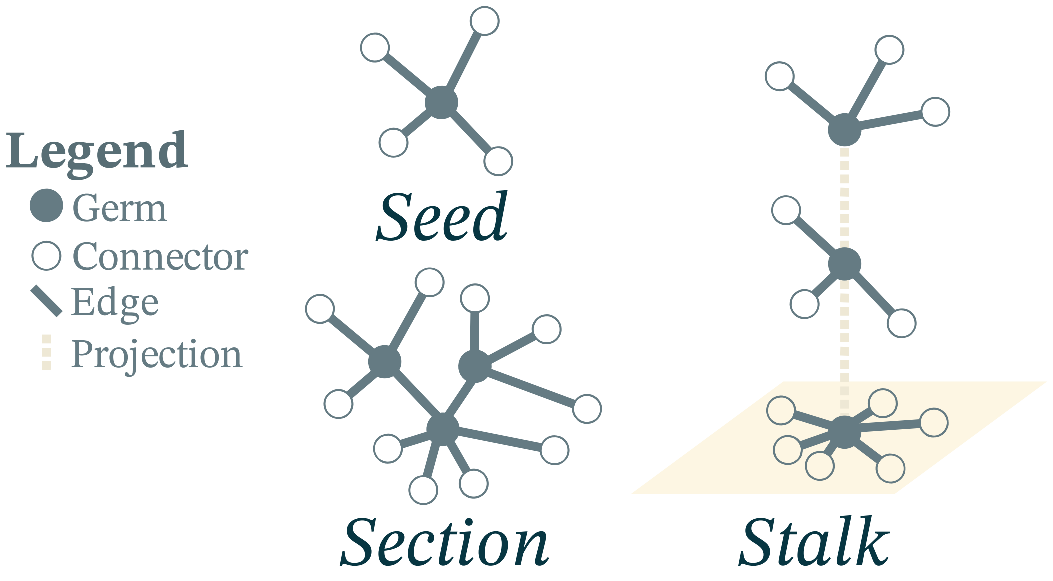


Figure 2.9: Example of a seed, a section and a stalk.

Definition 24 (Seed). A seed corresponds to a vertex along with the set of adjacent edges. Formally we note a seed $\star = (v, \chi_g(v))$ that means that a seed build from the vertex v in the graph g contains a set of adjacent edges $\chi_g(v)$. We call the vertex v the *germ* of the seed. All edges in a seed do not connect to the other vertices but keep the information and are able to match the correct vertices through typing (often a type of a single individual). We call the edges in a seed *connectors*.

Seeds are extracts of graphs that contain all information about a vertex. Illustrated in the figure 2.9, seeds have a central germ (represented with discs) and connectors leading to a typed vertex (outlined circles). Those external vertices are not directly contained in the seed but the information about what vertex can fit in them is kept. It is useful to represent connectors like jigsaw puzzle pieces: they can match only a restricted number of other pieces that match their shape.

From there, it is useful to build a kind of partial graph from seeds called sections.

Definition 25 (Section)**.** A section is a set of seeds that have their common edges connected. This means that if two seeds have an edge in common connecting both germs, then the seeds are connected in the section and the edges are merged. We note $g_\star = (V, \{\!\{\cup : E_{section}\}\!\})$ the graph formed by the section.

In figure 2.9, a section is represented. It is a connected section composed of seeds along with the additional seeds of any vertices they have in common. They are very similar to subgraph but with an additional border of typed connectors. This tool was originally mostly meant for big data and categorization over large graphs. As the graph quotient is often used in that domain, it was ported to sections instead of graphs allows us to define stalks.

vérifier, je n ecomprends pas la phrase

Definition 26 (Stalk)**.** Given a projection function $f : V \to V'$ over the germs of a section $\star$, the stalk above the vertex $v' \in V'$ is the quotient of all seeds that have their germ follow $f(v) = v'$.

The quotienning is used in stalks for their projection. Indeed, as shown in figure 2.9, the stalks are simply a collection of seeds with their germs quotiened into their common projection. The projection can be any process of transformation getting a set of seeds in one side and gives object in any base space called the image. Sheaves are a generalization of this concept to sections.

Definition 27 (Sheaf)**.** A sheaf is a collection of sections, together with a projection. We note it $\mathcal{F} = \langle g_\star, glue \rangle$ with the function $glue$ being the gluing axioms that the projection should respect depending on the application. The projected sheaf graph is noted as the fusion of all quotiened sections:

$$glue_{\mathcal{F}} = \{\div_{glue_\star} : \{glue_\star \in g_\star\}$$

By puting several sections into one projection, we can build stack fields. These fields are simply a subcategory of sheaves. Illustrated in figure 2.10, a sheaf is a set of sections with a projection relation that usually merge similarly typed connectors.

S

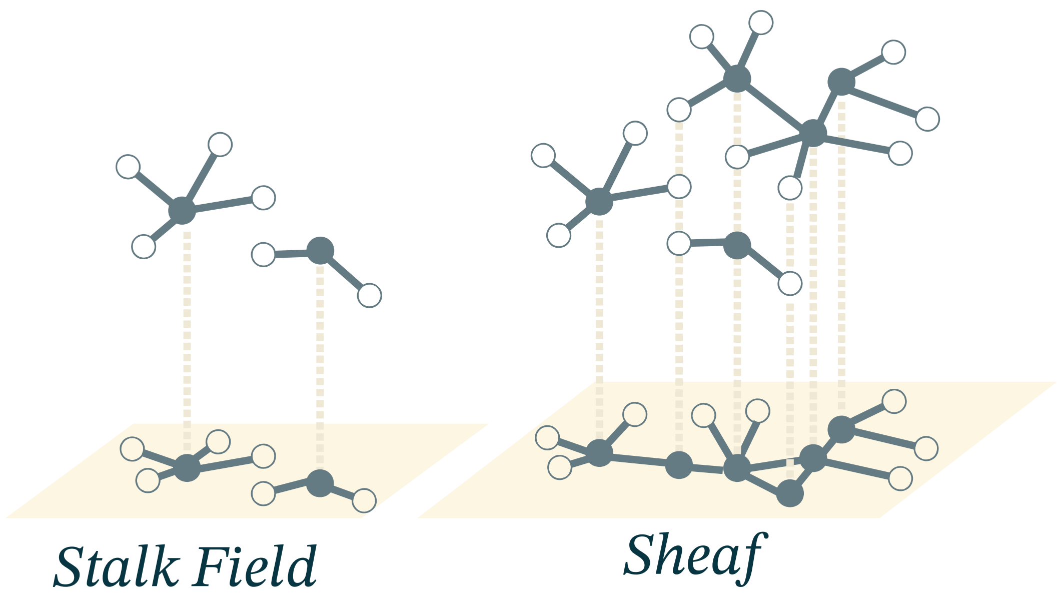


Figure 2.10: Example of sheaves.

have shown

In this chapter, we shown the limits of the formalization in mathematics and the possible ways to mitigatee them. We also presented a fondation of mathematics derived from category theory as well as defining mathematical tools very useful to formalize computer science notions. The last notion, the sheaf, is at the heart of the data structure for the knowledge describtion model presented in the next chapter.

??? 1- il faut une section 2.7 conclusion,
2- tu n'as pas montré les limites de la formalisation classique des maths ! et tu n'as pas montré en quoi ton formalisme est mieux vs les issues présentées en 2.1