2 Foundation and Tools

on ne comprends pas l'intérêt/ce que tu veux passer comme message avec le premier paragraphe ? pourquoi parles-tu de set theory en intro ici ? + quel est le lien avec le sujet de la thèse ? —> MANQUE DE CONTEXTE

Mathematics and logic are at the heart of all formal sciences, including computer science. 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 a medium like 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. 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 (WWI). 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.

acronyme inutile

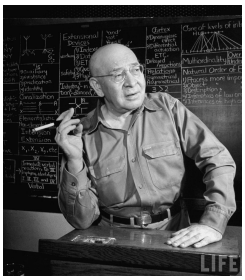

Figure 2.1: Alfred Korzybski

> *"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)

In this first chapter we analyze the issue of expression in mathematics and propose our solution. Then we use our foundation to build some important mathematical notions used later to formalize data structures and algorithms.

c'est un chapitre qui pose les bases (« fondation et outils ») et il semble ici que tu dis qu'il y aura des contributions à toi dans ce chapitre ??? il faut reformuler (à moins qu'il y ait vraiment des réelles contributions, dans ce cas, c'est étonnant que ca soit dans le premier chapitre et le nom du chapitre ne convient pas …) + expliquer de quoi tu vas nous parler dans ce chapitre « concrètement » : les fondations de quoi (pas les fondations des mathémtiques, c'est beaucoup trop large), quels outils ?

2.1 Properties

From this point we can apply some of these ideas to mathematics to analyze what properties a foundation of mathematics must hold.

de quelles idées tu parles ? les éléments cités dans l'intro ? pour moi les éléments de ton intro sont très généraux … « analyse what properties a foundation of math must hold »: un peu ambitieux non ? De plus, quel est le lien avec le sujet de la thèse —> la thèse ne porte pas sur les fondations des mathématiques

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 (potentially infinitely) 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 one that will be used in the context. *This means that abstraction is a loosy process.* 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. Since abstraction is a generalization process, if the host language is expressive enough, abstraction can be applied to already abstracted knowledge. The number of layers abstraction is needed for a term is called its *abstraction level*. Very general notions have a higher abstraction level and we usually 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 or statements).

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 suggest 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 to therefore 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.

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*. This comes from two principal reasons. As abstraction is a loosy process one cannot completely describe a language that can evolve and be infinitely complex. And even when taking a simple language, there is a problem with describing the different levels of rules to process the language using itself. This supposes of the knowledge of the language *a priori*. This is contradictory and therefore impossible to achieve.

je ne comprends pas la phrase (où est le verbe ?)

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)

Defining a term requires using a formula in the host language to express the abstracted properties of the generalization. The problem is that most terms will have *circular* definitions. Like stated this means that every knowledge system will have some circularity in their definitions. This means that it is impossible to have a complete definition of a formal system without needing another formal system to describe its base.

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. This also makes Russel's paradox unavoidable in such systems since there is always an infinite descent in the definition relation. Also, 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 find the smallest possible 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, and typed lambda calculus Awodey (2010).

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 is loosely based on category theory. However, it differs in its priorities and formulation.

those est pluriel non ?

Our theory is axiomatic, meaning it is based on fundamental logical proposition called axioms. Those forms 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 axiom is the mandatory undefinable notion created from natural language. It is the explicit base of the formalism.

Axiom (Function)**.** Let's *everything* be functions that associates a unique function (image) to each function (argument).

manque parenthèse

That axiom states 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 (image which are also functions.

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.

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

2.2.1 Formalism definition

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

Since any formalism has circular definition, we will have two ways of defining each notion: using natural language as host language or by using notions that will be defined later. The main definitions we will express are using natural language but we'll add the formula using future notion as side notes. The goal is to explicitly separate fully defined notions from the circular base while giving the dependencies between notions to ease future improvements.

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 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.

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

$$(\!|f|\!) = (x \to (\!|f(x)|\!))$$

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 $(\!)$ so that $f(x, y)$ behaves the same way as $(\!|h|\!)(x)(y)$.

$$(\!|f|\!) = (\!|x, y \to f(x)(y)|\!)^+$$

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 $f(x, y, z, ...)$ 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. $x\,f\,y$ for its application.

Example 3. 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 (Association)**.** Let's be the *association function* $(\to)$ such that, for any two functions x and $f(x)$, the expression $x \to f(x)$ is equal to f.

$$(\to) = x, f(x) \to f$$

Definition 5 (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"*.

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

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.

Definition 6 (Null)**.** The *null function* is the function between nothing and nothing. We note it $\gg$.

$$\gg = \gg \to \gg$$

$(=) = x \to x$

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

Definition 7 (Identity)**.** The *identity function* is the function that associates any function to itself noted $(=)$. This function is therefore transparent as by definition $= (x)$ is the same as x. A more useful form of this function is as a binary predicate (see **LATER**). The predicate can work in two ways: *equation* and *affectation*.

- As an **equation**, the predicate tests whether two functions are the same.
- The **affectation** allows to assert that two functions have the same identity making them an alias to one another. An affectation is often used to define a new function.

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

2.2.2 Literals and Variables

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

$l =\gg\to l$

Definition 8 (Literal)**.** A literal is a function that associates null to its value. This consists of any function l written as $\gg\to l$. 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 4. A good example of that would be the yet to be defined natural numbers. We can define the literal 3 as $3 =\gg\to 3$. This is a fonction that takes no argument and is always valued to 3.

$x = x \to\gg$

Definition 9 (Variable)**.** A variable is a function that associates itself to null. This consists of any function x written as $x \to\gg$. 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.

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 $\gg\to\gg$ and fullfil both definitions.

When defining currying, we annotated with the formalism $(\!|f|\!) = f \to (x \to (\!|f(x)|\!))$. 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.

2.2.3 Functional algebra

pourquoi plus expressif et compact que les existants ? tu ne peux pas juste dire ça, il faut le justifier

Inspired by relational algebra and by category theory, we present a highly expressive and compact functional algebra. In the figure 2.2 we illustrate the different operators of this algebra and their properties.

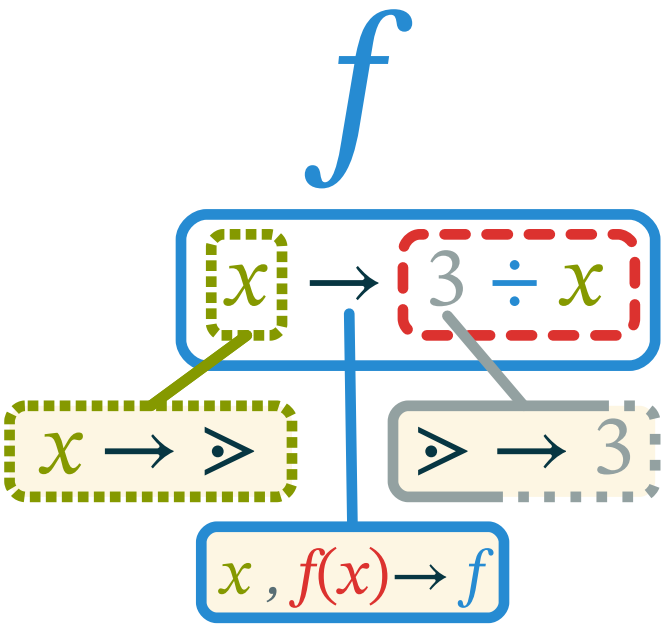

Figure 2.2: Illustration of basic functional operators and their properties.

FIXME: legend & explanation

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.

c'est pas clair du tout la définition 10

Definition 10 (Combination). The *combination function* $\bowtie$ is the function that associates any two functions to a new function that associates any functions associated by **either** functions.

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

In other words, combination allows the union of two function by adding all their definitions into one function.

Example 5. For two functions f_1 and f_2 that are defined respectively by:

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

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

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

Definition 11 (Superposition). The *superposition function* $\triangle$ is the function that associates any two functions to a new function that associates any function associated by **both** functions.

$f_1 \triangle f_2 = x \to y$ 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 6. Reusing the functions of the previous example, we can note that $f_3 \vartriangle f_1 = f_1$ because $1 \to 2$ is the only combination that both functions associates.

Definition 12 (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:

- $f \vartriangle f = f$, a function superposed to itself is the same.
- $f \vartriangle \gg = \gg$, 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.
- $f \triangledown f = \gg$, a function subposed by itself is always null.
- $f \triangledown \gg = f$, 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 13 (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 14 (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.

- $f \circ \gg = \gg$ ($\gg$ is the absorbing element)
- $f \circ == f$ ($=$ is the neutral element)
- $\bullet(\gg) = \gg$ and $\bullet(=) ==$
- $f_1 \circ f_2 \neq f_2 \circ f_1$

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:

- $f^{-1} = \bullet f$
- $f^0 = (=)$
- $f^1 = f$
- $f^n = f^{n-1} \circ f$

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.

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

All this allows for a powerful notation system used throughout the rest of the document.

l'intérêt de tout ça est uniquement d'avoir une notation puissante ?

2.2.4 Properties

A modern approach of mathematics is called *reverse mathematics* as instead of builing 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.

building ?

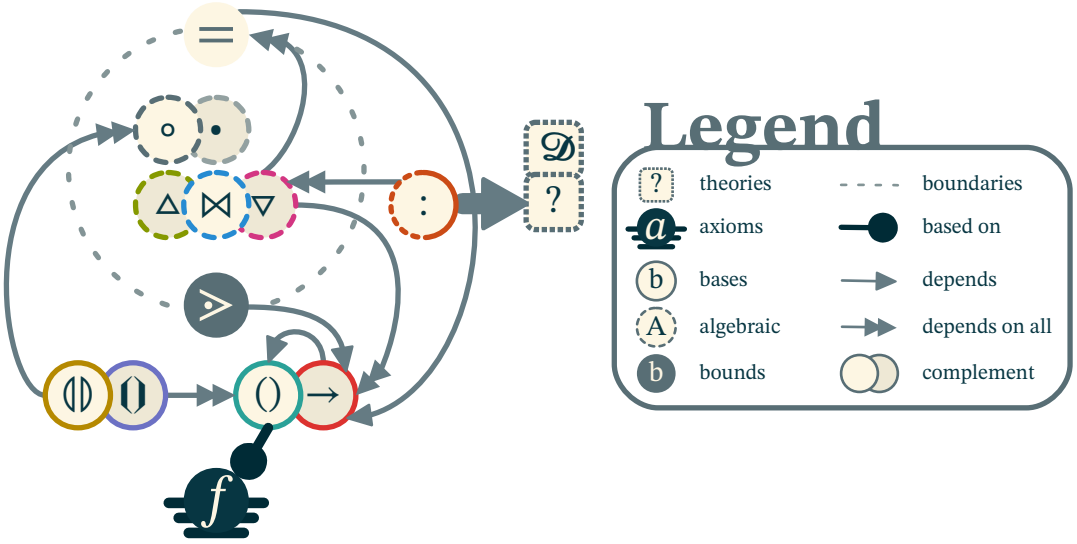

Figure 2.3: Dependency graph of notions in the functional theory

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.

Indeed, $\bowtie$ is a commutative monoid having $\gg$ 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: $\circ(\gg) = ((\gg) \to \gg) = \gg$.

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 $\top$ *true* 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.

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:

$\forall = \S(\wedge)$
- The *universal quantifier* $\forall$ meaning *"for all"*.

$\exists = \S(\vee)$
- The *existential quantifier* $\exists$ meaning *"it exists"*.

They are sometimes extended with :

$\exists! = \S(= (1) \circ +)$
- The *uniqueness quantifier* $\exists!$ meaning *"it exists a unique"*.

$\nexists = \S(\neg \circ \wedge)$
- The *exclusive quantifier* $\nexists$ meaning *"it doesn't exist"*.

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

$\S = f, x, ? \to \{\!|f(x) : ?|\!\}$
- The *solution quantifier* $\S$ meaning *"those"*.

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

Georg Cantor (1895)

Since we need to represent knowledge, we will handle more complex data than simple booleans. The first theory we will define is the set theory. It is used as the classical foundation of mathematics. It is so important to mathematics that 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 funding 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.*"

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 inspired by the work of Cantor (1895) and the set theory. The goal is to define the notions of set theory using our formalism.

Definition 15 (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.

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 16 (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 17 (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 18 (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 19 (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. In figure 2.4 we present the classical set operations.

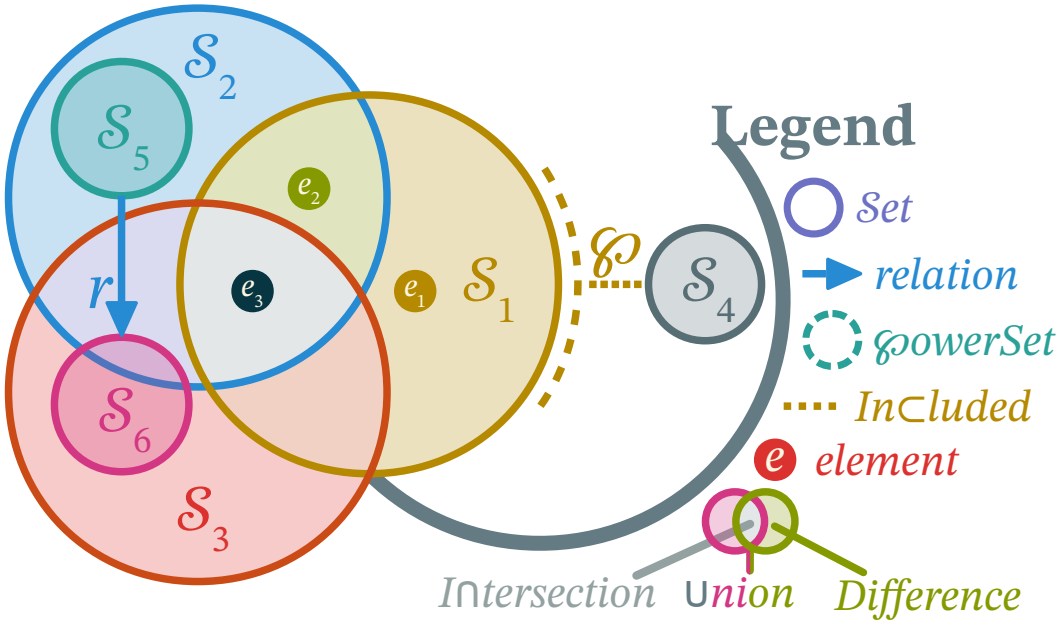

Figure 2.4: Example of an upgraded Venn diagram to illustrate operations on sets.

These diagrams 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.

Definition 20 (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 21 (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 7. 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). $\forall ?\forall \mathcal{T} \exists \mathcal{S} : (\forall e \in \mathcal{S} : (e \in \mathcal{T} \wedge ?(e)))$

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

as-tu besoin plus tard de toutes les notions présentées dans la section Graph ? y'a t il un intérêt à en présenter autant si certaines ne sont pas réutilisées ? quel est le lien avec ton formalisme précédemment introduit (je ne vois pas en quoi ton formalisme est réutilisé/utile ici)

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. Most of these structures use graphs and isomorphic derivatives.

Definition 22 (Graph). A graph is a mathematical structure g which is defined by its *connectivity function* χ that

2.5.1 Adjacency, Incidence and Connectivity

Definition 23 (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:

Also: $\chi_{\multimap\circ} = \bullet\chi_{\multimap}$

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

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

$$\chi = \chi_{\multimap\circ} \bowtie \chi_{\multimap}$$

$E \subseteq V^2$

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: $\mathcal{D}(\chi_{\multimap}) = E$

Example 8. 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.

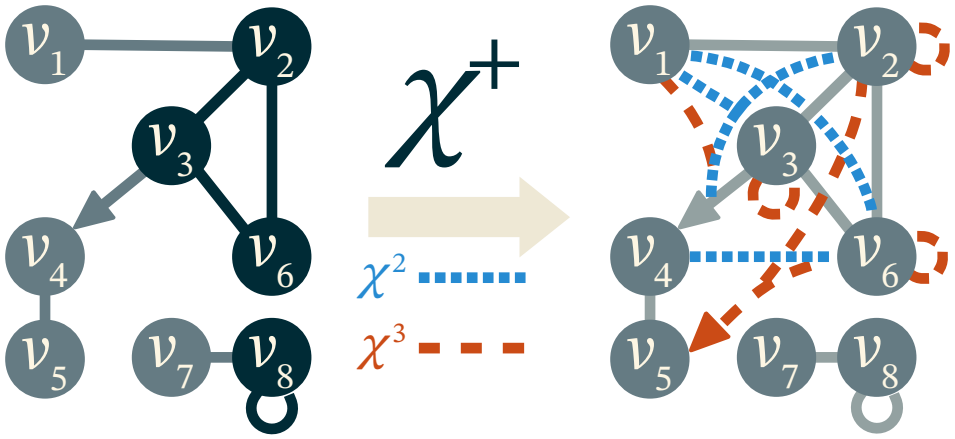

Figure 2.5: Example of the recursive application of the transitive cover to a graph.

2.5.2 Digraphs

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:

$$\chi_{\rightarrow} = \chi_{\succ\!\!\rightarrow} \bowtie \chi_{\rightarrow\!\!\prec}$$

We note $\chi_{\succ\!\!\rightarrow}$ the **incoming relation** and $\chi_{\rightarrow\!\!\prec}$ 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 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 $g = (V, E)$.

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 24 (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.

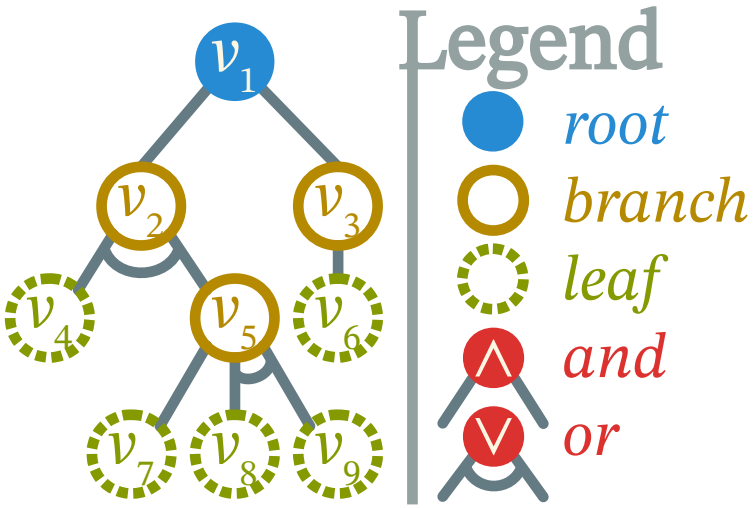

Figure 2.6: Example of and/or tree.

2.5.5 Quotient

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

Definition 25 (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.

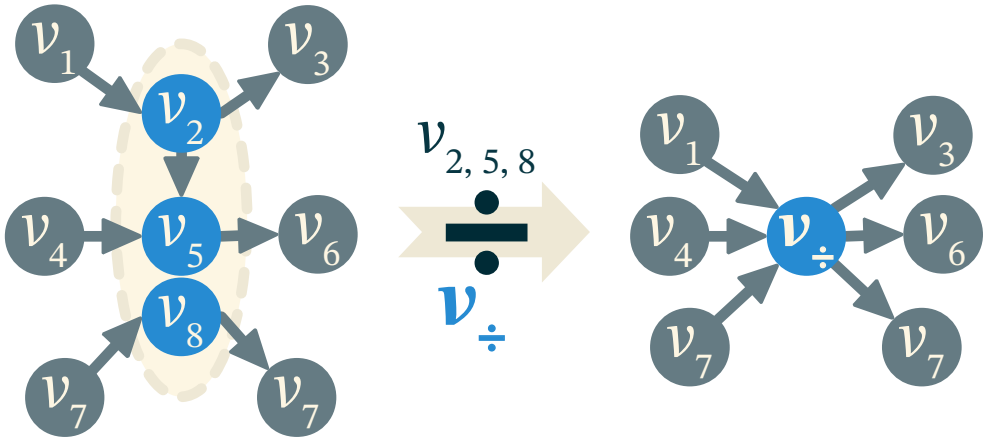

Figure 2.7: Example of graph quotient.

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.

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 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.

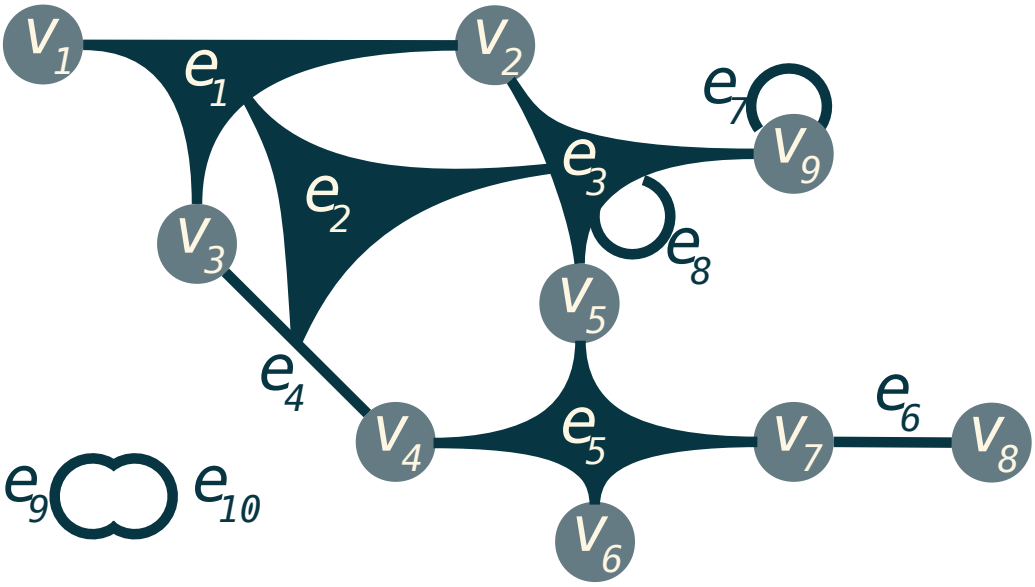

Figure 2.8: Example of hypergraph with total freedom on the edges specification.

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.

Definition 26 (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 27 (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

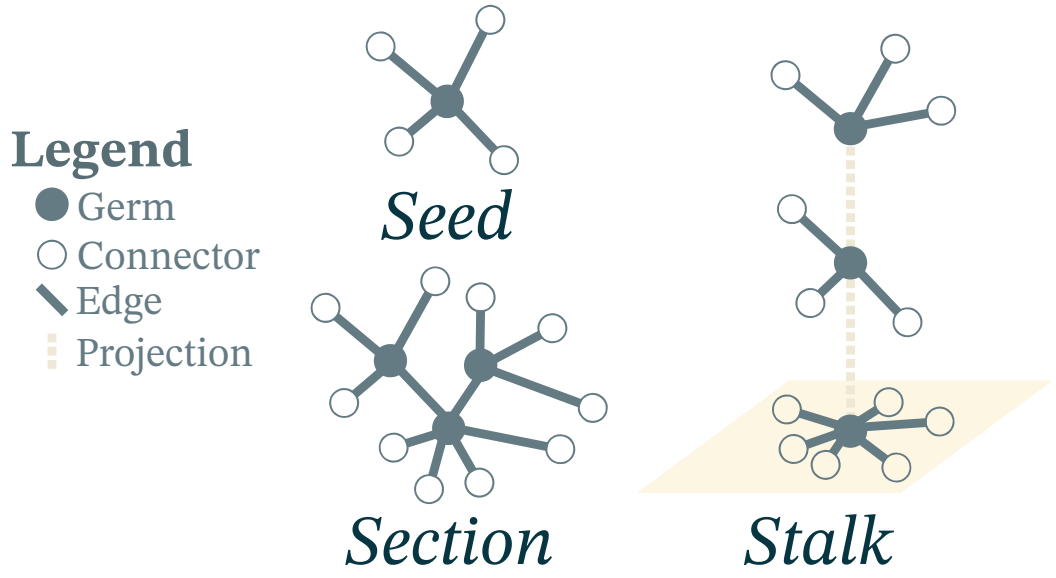

Figure 2.9: Example of a seed, a section and a stalk.

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.

Definition 28 (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 29 (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 merging common vertices into a section, we can build stack fields. These fields are simply a subcategory of sheaves. Illustrated in figure 2.10, a sheaf is a set of section with a projection relation.

pas de conclusion ????

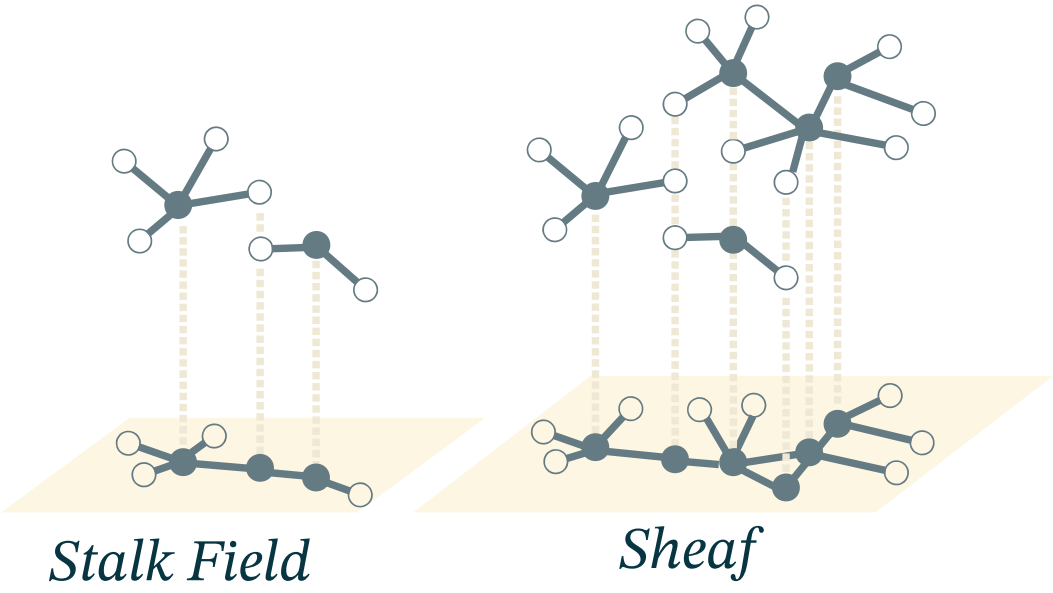

Figure 2.10: Example of sheaves.