When teaching an elementary logic course to students, who have a general scientific background, but have never been exposed to logic, we have to face the problem that the notions of deduction rule and of derivation are completely new to them, and are related to nothing they already know, unlike, for instance, the notion of model, that can be seen as a generalization of the notion of algebraic structure, or the notion of computable function that is a particular case of the notion of function.

We present, in this paper, a strategy to introduce these notions: start with the notion of inductive definition [1], then, the notion of derivation comes naturally. We also show, with three examples: computability theory, automata theory, and proof theory, that derivations are pervasive in logic—we could have given more examples in formal grammars, rewriting, etc. Thus, defining precisely this notion at an early stage is a good investment to later define other notions. Finally, we show that we need to distinguish two notions of derivation: that of derivation labeled with elements and that of derivation labeled with rule names.

In this paper, no proofs are given, and not even precise definitions. These can be found, for instance, in [2].

1 From inductive definitions to derivations

1.1 A method to define sets: inductive definitions

Inductive definitions are a way to define subsets of a set $A$. The inductive definition of a subset $P$ is formed with a family of functions $f_1$, from $A^{n_1}$ to $A$, $f_2$, from $A^{n_2}$ to $A$, etc. These functions are called rules. For example, the function $f_1 = \langle \rangle \mapsto 0$, from $\mathbb{N}^0$ to $\mathbb{N}$, and $f_2 = \langle x \rangle \mapsto x + 2$, from $\mathbb{N}^1$ to $\mathbb{N}$ are rules.

Instead of writing these rules $f_1 = \langle \rangle \mapsto 0$ and $f_2 = \langle x \rangle \mapsto x + 2$, we often write them

$$\begin{align*}
\emptyset & \vdash f_1 \\
\end{align*}$$
\[
\frac{x}{x+2} f_2
\]

But despite this new notation, rules are things the students already know: functions.

These rules define a function \( F \) from \( P(A) \) to \( P(A) \)

\[
F(X) = \bigcup_i \{ f_i(a_1, ..., a_{n_i}) \mid a_1, ..., a_{n_i} \in X \}
\]

For example, the two rules above define the function

\[
F(X) = \{0\} \cup \{x + 2 \mid x \in X \}
\]

and, for instance, \( F(\{4, 5, 6\}) = \{0, 6, 7, 8\}, F(\emptyset) = \{0\}, F(\{0\}) = \{0, 2\} \), etc.

The function \( F \) is monotonic and continuous. Thus, it has a smallest fixed point \( P \), which is the inductively defined subset of \( A \). This smallest fixed point can be defined in two ways

\[
P = \bigcap_{F(X) \subseteq X} X = \bigcup_i F^i(\emptyset)
\]

The first definition characterizes the set \( P \) as the smallest set closed by \( f_1, f_2 \), etc. the second as the set containing all the elements that can be built with these functions in a finite number of steps.

The notion of monotonicity and continuity of a function from \( P(A) \) to \( P(A) \) can then be introduced and the two fixed point theorems can be proved with mathematically oriented students. They can be admitted otherwise.

Continuing with our example, the set \( P \) of even numbers can be characterized as the smallest set containing 0 and closed by the function \( x \mapsto x + 2 \), or as the union of the sets \( \emptyset, F(\emptyset) = \{0\}, F^2(\emptyset) = \{0, 2\}, F^3(\emptyset) = \{0, 2, 4\} \), etc.

### 1.2 Derivations

A *derivation* is a tree whose nodes are labeled with elements of \( A \) and such that if a node is labeled with \( b \) and its children with \( a_1, ..., a_n \), then there exists a rule \( f \) such that \( b = f(a_1, ..., a_n) \). A *derivation of an element* \( a \) is a derivation whose root is labeled with \( a \). We can then prove, by induction on \( i \), that all the elements of \( F^i(\emptyset) \) have a derivation. The property is trivial for \( i = 0 \). If it holds for \( i \) and \( b \in F^{i+1}(\emptyset) \), then by definition \( b = f(a_1, ..., a_n) \) for some rule \( f \) and \( a_1 \in F^i(\emptyset), ..., a_n \in F^i(\emptyset) \), thus, by induction hypothesis, \( a_1, ..., a_n \) have derivations. Hence, so does \( b \).

Thus, from the second property \( P = \bigcup_i F^i(\emptyset) \), we get that all elements of \( P \) have derivations. Conversely, all elements that have a derivation are elements of \( P \).
Continuing with our example the number 4 has the derivation

\[
\begin{array}{c}
\bar{0} \\
\bar{2} \\
\bar{4}
\end{array}
\]

### 1.3 Rule names

There are several alternative definitions of the notion of derivation. For instance, when \( b = f(a_1, ..., a_n) \), instead of labelling the node just with \( b \), we can label it with the ordered pair formed with the element \( b \) and the name of the rule \( f \). For instance, the derivation of 4 above would then be the tree

\[
\begin{array}{c}
\langle 0, f_1 \rangle \\
\langle 2, f_2 \rangle \\
\langle 4, f_2 \rangle
\end{array}
\]

more often written

\[
\begin{array}{c}
\bar{0} f_1 \\
\bar{2} f_2 \\
\bar{4} f_2
\end{array}
\]

Such a derivation is easier to check, as checking the node

\[
\begin{array}{c}
\bar{2} \\
\bar{4}
\end{array}
\]

requires to find the rule \( f \) such that \( f(2) = 4 \), while checking the node

\[
\begin{array}{c}
\bar{2} f_2 \\
\bar{4} f_2
\end{array}
\]

just requires to apply the rule \( f_2 \) to 2 and check that the result is 4.

But these rules names are redundant, as soon as the relation \( \cup_i f_i \) is decidable. So, in general, they can be omitted.
1.4 Derivations labeled with rule names

Instead of omitting the rule names, it is possible to omit the elements of $A$. The derivation of 4 is then the tree

$$
\begin{array}{c}
\bar{f}_1 \\
\bar{f}_2 \\
\bar{f}_2 \\
\bar{f}_2 \\
\end{array}
$$

that can also be written

$$
\begin{array}{c}
\vdash f_1 \\
\vdash f_2 \\
\vdash f_2 \\
\vdash f_2 \\
\end{array}
$$

We introduce this way a second kind of derivations labeled with rule names. In contrast, the previous derivations can be called labeled with objects.

Although it is not explicit in the derivation, the element 4 can be inferred from this derivation with a top-down conclusion inference algorithm, because the rules $f_i$ are functions. The conclusion of the rule $f_1$ can only be $f_1(\langle \rangle) = 0$, that of the first rule $f_2$ can only be $f_2(\langle 0 \rangle) = 2$, and that of the second can only be $f_2(\langle 2 \rangle) = 4$.

More generally, for each derivation labeled with rule names $D$, there is at most one $a$ such that $D$ is a derivation of $a$, the existence of such an $a$ can be decided and, when it exists, this $a$ can be computed from $D$. As a consequence, the set of ordered pairs $D : a$ such that $D$ is a derivation of $a$ is decidable.

1.5 Proof-terms and type systems

Derivations labeled with rules names are often written as a term, that is in a linear form. For instance the derivation

$$
\begin{array}{c}
\vdash f_1 \\
\vdash f_2 \\
\vdash f_2 \\
\vdash f_2 \\
\end{array}
$$

is often written $f_2(f_2(f_1))$. Such a term is called a proof-term.

The decidable set of ordered pairs $\pi : a$ such that $\pi$ is a proof-term of $a$ can itself be defined by an inference system obtained from the original one by replacing each rule

$$
\frac{a_1 \ldots a_n}{b} R
$$
with the rule

\[ \frac{\pi_1 : a_1 \ldots \pi_n : a_n}{R(\pi_1, \ldots, \pi_n) : b} \]

In our example, we get the rules

\[ \frac{f_1 : 0}{f_1} \]

\[ \frac{\pi : a}{f_2(\pi) : a + 2} \]

and the ordered pair \( f_2(f_2(f_1)) : 4 \) has the derivation

\[ \frac{f_1 : 0}{f_2(f_1) : 2} \]

\[ \frac{f_2(\pi) : a + 2}{f_2(f_2(f_1)) : 4} \]

This second inference system is called a type system. It defines a decidable set—it is even an automaton in the sense of [4].

Moreover the conclusion inference algorithm transforms into a type inference algorithm. For each proof-term \( \pi \), there is at most one \( a \) such that \( \pi : a \) is derivable, the existence of such an \( a \) can be decided and, when it exists, this \( a \) can be computed from \( \pi \).

### 1.6 Making the rules functional

Natural deduction proofs [6, 5], for instance, are often labeled both with sequents and rule names, for instance

\[ \frac{P, Q, R \vdash P}{P, Q, R \vdash P \land Q} \text{ axiom} \]

\[ \frac{P, Q, R \vdash Q}{P, Q, R \vdash P \land Q} \text{ \&-intro} \]

but they can be labeled with sequents only

\[ \frac{P, Q, R \vdash P}{P, Q, R \vdash P \land Q} \]

\[ \frac{P, Q, R \vdash Q}{P, Q, R \vdash P \land Q} \]

and proof-checking is still decidable. They can also be labeled with rule names only, but we have to make sure that all the deduction rules are functional, which is often not the case in the usual presentations of Natural deduction. The rule

\[ \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} \text{ \&-intro} \]
is functional: there is only one possible conclusion for each sequence of premises, but the axiom rule

\[ \Gamma, A \vdash A \] _axiom

is not. To make it functional, we must introduce a different rule axiom\(\langle \Gamma, A \rangle\) for each ordered pair \(\langle \Gamma, A \rangle\). Thus, the proof above must be written

\[
\frac{P, Q, R \vdash P}{P, Q, R \vdash P \land Q} \text{ axiom}_\langle\{Q, R\}, P\rangle
\]
\[
\frac{P, Q, R \vdash Q}{P, Q, R \vdash P \land Q} \text{ _\land-intro}
\]

And from the derivation labeled with rule names

\[
\frac{\text{axiom}_\langle\{Q, R\}, P\rangle}{P, Q, R \vdash P \land Q} \quad \frac{\text{axiom}_\langle\{P, R\}, Q\rangle}{P, Q, R \vdash P \land Q}
\]

the conclusion \(P, Q, R \vdash P \land Q\) can be inferred.

In a linear form, this derivation is \(_\land-intro(\text{axiom}_\langle\{Q, R\}, P\rangle, \text{axiom}_\langle\{P, R\}, Q\rangle)\) and its type, \(P, Q, R \vdash P \land Q\), can be inferred.

## 2 Derivations in elementary computability theory

### 2.1 A pedagogical problem

The set of computable functions is often defined inductively, as the smallest set containing the projections, the null functions, and the successor function, and closed by composition, definition by induction, and minimization.

But to study the computability of properties of computable functions, we need a secondary notion of _program_, that is we need a way to express each computable function with a expression of a finite language, to which a Gödel number can be assigned. A usual solution is to introduce Turing machines, λ-calculus, or any other language at this point.

This solution however is not pedagogically satisfying as, while the students are still struggling to understand the inductive definition of the set of computable functions, we introduce another, that is often based on completely different ideas, letting them think that logic made of odds and ends. Moreover, the equivalence of the two definitions requires a tedious proof.

Such a second definition is in fact not needed as the inductive definition itself already gives a notion of program, through the notion of derivation.
2.2 Programs already exist

The function $x \mapsto x + 2$ is computable because it is the composition of the successor function with itself. But the derivation labeled with objects

$$
\frac{x \mapsto x + 1}{x \mapsto x + 1}
$$

cannot be used as a program, because to label its nodes, we would need a language to express all the functions, and there is, of course, no such language.

But if we use a derivation labeled with rule names instead

$$
\frac{\neg Succ}{\neg Succ} \quad \frac{\neg Succ}{\circ_1}
$$

and write the derivations in a linear form: $\circ_1(Succ, Succ)$, we obtain a simple variable-free functional programming language, to express the programs. We can introduce this way a symbol $\pi^n_i$ for the $n$-ary $i$-th projection, $\circ_p^n(f, g_1, \ldots, g_p)$ for the composition of the $n$-ary functions $g_1, \ldots, g_p$ with the $p$-ary function $f$, and $\mu^n(f)$ for the minimization of the $n + 1$-ary function $f$ over its last argument, etc.

For instance, introducing a Gödel numbering $\land, \lor$ for these programs, and assuming there is an always defined function $h$ such that

- $h(p, q) = 1$ if $p = \land f \land$ and $f$ defined at $q$
- and $h(p, q) = 0$ otherwise,

we get a contradiction: the function

$$
k = \circ_1(\mu^1(\pi^1_2), \circ_2(h, \pi^1_1, \pi^1_1))
$$

is defined at $\land k \land$ if and only if it is not.

We get this way a proof of the undecidability of the halting problem that requires nothing else than the inductive definition of the set of computable functions.

3 Derivations in elementary automata theory

When introducing the notion of finite automaton, we often introduce new notions, such as those of transition rules and recognizability. Having introduced the notion of derivation from the very beginning of the course permits to avoid introducing these as new notions.
Consider for instance the finite state automaton

\[
\text{odd } \xrightarrow{a} \text{even} \quad \text{even } \xrightarrow{a} \text{odd}
\]

where the state even is final. In this automaton, the word \text{aaa} is recognized in odd. Indeed

\[
\text{odd } \xrightarrow{a} \text{even} \xrightarrow{a} \text{odd } \xrightarrow{a} \text{even}
\]

If, instead of introducing a new notion of transition rule, we just define transition rules as deduction rules

\[
\frac{}{\text{even } \xrightarrow{a} \text{odd}} \quad \frac{}{\text{odd } \xrightarrow{a} \text{even}} \quad \frac{}{\text{even } \xrightarrow{} \epsilon}
\]

then, the element odd has a derivation

\[
\frac{}{\text{even } \xrightarrow{\epsilon}} \frac{}{\text{odd } \xrightarrow{a}} \frac{}{\text{even } \xrightarrow{a}} \frac{}{\text{odd } \xrightarrow{a}}
\]

If we label this derivation with rule names we obtain

\[
\frac{}{\epsilon} \frac{}{a} \frac{}{a} \frac{}{a}
\]

which can be written in linear form \text{a(a(a(\epsilon)))}, or \text{aaa}. Thus, a word \text{w} is recognized in a state \text{s} if and only if it is a derivation, labeled with rule names, of \text{s}.

Transforming this inference system into a type system, like in Section 1.5, we get

\[
\frac{w : \text{even}}{aw : \text{odd } a} \quad \frac{w : \text{odd}}{aw : \text{even } a} \quad \frac{\epsilon : \text{even}}{\epsilon : \text{even } \epsilon}
\]

And a word \text{w} is recognized in a state \text{s} if and only \text{w} : \text{s} is derivable.

This example introduces a point that needs to be discussed: the rules

\[
\frac{}{\text{even } \xrightarrow{a} \text{odd}} \quad \frac{}{\text{odd } \xrightarrow{a} \text{even}}
\]

are labeled with the same name. If the automaton is deterministic, we can replace these two rules with one: a function such that \text{a(\text{even})} = \text{odd} and \text{a(\text{odd})} = \text{even}.
But for non deterministic automata, we either need to extend the notion of rule name, allowing different rules to have the same name, or to consider two rule names

\[
\begin{align*}
\text{even} & \quad a_1 \\
\text{odd} & \quad a_2 \\
\text{even} & \quad \varepsilon
\end{align*}
\]

and map the derivation \(a_1(a_2(a_1(\varepsilon))))\) to the word \(a(a(a(\varepsilon))))\) with the function \(|.|\) defined by: \(|\varepsilon| = \varepsilon\), \(|a_1(t)| = a(|t|)\), and \(|a_2(t)| = a(|t|)\).

4 Introducing the Brouwer-Heyting-Kolmogorov correspondence

4.1 A radical change in viewpoint?

The Brouwer-Heyting-Kolmogorov interpretation, and its counterpart, the Curry-de Buijn-Howard correspondence, are often presented as a radical change in viewpoint: proofs are not seen as trees anymore, but as algorithms.

But, of course, these algorithms must be expressed in some language, often the lambda-calculus. Thus, proofs are not really algorithms, but terms expressing algorithms, and such terms are nothing else than trees. So, it is fairer to say that, in the Brouwer-Heyting-Kolmogorov interpretation, proofs are not derivation trees, but trees of a different kind. For instance, the tree

\[
\frac{P \land Q \vdash P \land Q \\ P \land Q \vdash P}{P \land Q \vdash Q \land P \\ P \land Q \vdash P} \vdash (P \land Q) \Rightarrow (Q \land P)
\]

is replaced by the tree

\[
\frac{x}{\text{snd}} \quad \frac{x}{\text{fst}} \\
\overline{\lambda x : P \land Q \langle \text{snd}(x), \text{fst}(x) \rangle}
\]

often written in linear form: \(\lambda x : P \land Q \langle \text{snd}(x), \text{fst}(x) \rangle\).

4.2 Derivation trees labeled with rule names

Instead of using this idea of expressing proofs as algorithms, let us just try to label the derivation above with rule names. Five rules are used in this proof. Three of
them are functional

\[
\begin{align*}
\Gamma \vdash A & \quad \Gamma \vdash B \\
\hline
\Gamma \vdash A \land B & \quad \land\text{-intro}
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A \land B & \quad \quad \quad \Gamma \vdash A \quad \land\text{-elim1} \\
\hline
\Gamma \vdash B & \quad \land\text{-elim2}
\end{align*}
\]

Let us just give them shorter names: \(\langle,\rangle\), \text{fst}, and \text{snd}. The rule

\[
\begin{align*}
\Gamma, A \vdash B & \quad \Rightarrow\text{-intro} \\
\hline
\Gamma \vdash A \Rightarrow B & \quad \Rightarrow\text{-intro}
\end{align*}
\]

is functional, as soon as we know which proposition \(A\) in the left-hand side of the antecedent is used. So, we need to supply this proposition \(A\) in the rule name, let us call this rule \(\lambda A\). Finally, the rule

\[
\begin{align*}
\Gamma, A \vdash A & \quad \text{axiom} \\
\hline
\Gamma ; A \vdash A & \quad \text{axiom}
\end{align*}
\]

is functional, as soon as we know \(\Gamma\) and \(A\). We could supply \(\Gamma\) and \(A\) in the rule name. However, we shall just supply the proposition \(A\) and infer the context \(\Gamma\). Let us call this rule \([A]\). So, the proof above can be written

\[
\begin{align*}
\begin{array}{c}
\vdash P \land Q \land P \land Q \\
\hline
\vdash P \land Q \land P \land Q
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\vdash [P \land Q] \\
\vdash [P \land Q] \\
\vdash \text{snd} \\
\vdash \text{fst} \\
\vdash \langle,\rangle \\
\vdash \lambda P \land Q
\end{array}
\end{align*}
\]

and if we keep rule names only

\[
\begin{align*}
\begin{array}{c}
\vdash P \land Q \\
\vdash P \land Q \\
\vdash \text{snd} \\
\vdash \text{fst} \\
\vdash \langle,\rangle
\end{array}
\end{align*}
\]

which, in linear form is the proof-term \(\lambda P \land Q \langle \text{snd}(P \land Q), \text{fst}(P \land Q) \rangle\).

Transforming this inference system into a type system, like in Section 1.5, we get

\[
\begin{align*}
\begin{array}{c}
\Gamma \vdash \pi : A \\
\vdash \pi' : B
\end{array}
\end{align*}
\]

\[
\begin{align*}
\vdash \langle \pi, \pi' \rangle : A \land B \\
\vdash \langle \pi, \pi' \rangle : A \land B \quad \land\text{-intro}
\end{align*}
\]
\[ \Gamma \vdash \pi : A \land B \]  
\[ \Gamma \vdash \text{fst}(\pi) : A \land\text{elim1} \]  
\[ \Gamma \vdash \pi : A \land B \]  
\[ \Gamma \vdash \text{snd}(\pi) : B \land\text{elim2} \]  
\[ \Gamma, A \vdash \pi : B \]  
\[ \Gamma \vdash \lambda A \pi : A \Rightarrow B \Rightarrow\text{intro} \]

in which the ordered pair \( \lambda P \land Q \langle \text{snd}([P \land Q]), \text{fst}([P \land Q]) \rangle : (P \land Q) \Rightarrow (Q \land P) \) is derivable. This is the scheme representation \( [3] \) of this proof.

Let us show that the conclusion can be inferred, although we have not supplied the context \( \Gamma \) in the axiom rule. The conclusion inference goes in two steps. First we infer the context bottom-up, using the fact that the conclusion has an empty context, and that all rules preserve the context, except \( \lambda A \) that extends it with the proposition \( A \)

\[
\frac{P \land Q \vdash [P \land Q]}{P \land Q \vdash \text{snd}} \quad \frac{P \land Q \vdash [P \land Q]}{P \land Q \vdash \text{fst}} \quad \frac{P \land Q \vdash \langle \cdot, \cdot \rangle}{\vdash \lambda P \land Q}
\]

Then, the right-hand part of the sequent can be inferred with a usual conclusion top-down inference algorithm, using the fact that the rules are functional

\[
\frac{P \land Q \vdash P \land Q}{P \land Q \vdash \text{snd}} \quad \frac{P \land Q \vdash P \land Q}{P \land Q \vdash \text{fst}} \quad \frac{P \land Q \vdash Q \land P}{\vdash (P \land Q) \Rightarrow (Q \land P) \lambda P \land Q}
\]

4.3 Brouwer-Heyting-Kolmogorov interpretation

In the rule

\[
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B \Rightarrow\text{intro}}
\]

instead of supplying just the proposition \( A \), we can supply the proposition \( A \) and a name \( x \) for it. Then, in the axiom rule

\[ \Gamma, A \vdash A \text{ axiom} \]
instead of supplying the proposition $A$, we can just supply the name that has been introduced lower in the tree for it. We obtain this way the tree

\[
\begin{array}{c}
\vdash x \\
\vdash \text{snd} \\
\hline
\vdash \text{fst} \\
\hline
\vdash \lambda x : P \land Q \langle \text{snd}(x), \text{fst}(x) \rangle \\
\end{array}
\]

in linear form $\lambda x : P \land Q \langle \text{snd}(x), \text{fst}(x) \rangle$, which is exactly the representation of the proof according to the Brouwer-Heyting-Kolmogorov interpretation.

So, the Brouwer-Heyting-Kolmogorov interpretation boils down to use of derivations labeled with rule names plus two minor modifications: context inference and the use of variables. These two modifications can be explained by the fact that Natural deduction does not really deal with sequents and contexts: rather with propositions, but, following an idea initiated in \[7\], some rules such as the introduction rule of the implication dynamically add new rules, named with variables.

References


