Utrecht University

Formalization of finite groups of Lie Type

Corentin Cornou

Under the supervision of Johan Commelin

Internship report in the context of M1 Hadamard of ENS Paris-Saclay

Abstract

Mathematics Subject Classification: 68V15 Theorem proving, 20D05 Finite simple groups and their classification, 20D06 Simple groups: alternating groups and groups of Lie type

Français :
Un groupe de type Lie est un groupe dont la structure est proche de celle d'un groupe de Lie, permettant de généraliser des résultats de la théorie de Lie à des groupes construits sur des corps finis. Ces groupes forment la majorité de la classification des groupes finis simples [5], ce qui les rend particulièrement dignes d'intérêt.

Nous formalisons dans ce rapport la notion de groupes de type Lie à l'aide du prouveur de théorèmes L∃∀N . Pour ce faire, nous utilisons la notion de BN-paire introduite par Jacques Tits dans [2]. Une BN-paire est une paire (B, N) de sous-groupes d'un groupe G de type Lie répondant à certaines propriétés et qui permet de déduire d'importants résultats sur la structure du groupe.

Nous décrivons ici l'implémentation en L∃∀N de cette notion ainsi que celle de la construction de la structure de BN-paire sur le groupe linéaire général sur un corps. Nous montrons également que cette dernière permet de doter les groupes projectif linéaire général, spécial linéaire et projectif spécial linéaire d'une telle structure. De plus, nous implémentons une méthode permettant de démontrer la simplicité de certains groupes possédant une BN-paire. Ce résultat est ensuite appliqué au cas du groupe projectif linéaire sur un corps F, $PSL_n(F)$, dans les cas où $n \geq 3$ ou $n = 2$ et $|F| \geq 4$, démontrant ainsi sa simplicité dans L∃∀N .

English

A group of Lie type is a group whose structure is close to that of a Lie group, allowing the generalization of results from Lie theory to groups constructed over finite fields. These groups form the majority of the classification of finite simple groups [5], which makes them particularly noteworthy.

In this report, we formalize the notion of groups of Lie type using the theorem prover L∃∀N . To achieve this, we utilize the concept of a BN-pair, introduced by Jacques Tits in [2]. A BN-pair is a pair (B, N) of subgroups of a group G of Lie type that satisfy certain properties, allowing us to derive significant results about the group's structure.

We describe here the implementation in L∃∀N of this notion, as well as the construction of the BN-pair structure on the general linear group over a field. We also show that this structure can be used to endow the projective general linear, special linear, and projective special linear groups with such a structure. Additionally, we implement a method to demonstrate the simplicity of certain groups possessing a BN-pair. This result is then applied to the case of the projective linear group over a field F, $PSL_n(F)$, in the cases where $n \geq 3$ or $n = 2$ and $|F| \geq 4$, thus proving its simplicity in L∃∀N .

Introduction

The *classification of finite simple groups* (CFSG) is one of the most important achievements of 20^{th} century's mathematics. It states (see [5] for the exact statement) that the finite simple simple[1] groups can be sorted into 4 families :

- the cyclic groups of prime order,
- the alternating groups of degree at least 5,
- the *groups of Lie type*,
- the 26 sporadic groups.

Recent progress in formalization of mathematics has shown abundantly that large and complex theories of mathematics can be formalized in a computer proof assistant (see [6] or [7]). The complexity of both the statement and proof of the CFSG makes them perfect candidates for formalization. Formalizing the notion of *group of Lie type* would be an important step in that direction.

The goal of the internship was to formalize in the L∃∀N theorem prover the definition of groups of Lie type, some of their properties and build e concrete examples of such goups.

L∃∀N is a functional programming language as well as a proof assistant initially developed by Leonardo de Moura in 2013. In 2017, the L∃∀N mathematical library `mathlib` was created. It is actively maintained by a motivated community and currently contains more 1.6 millions of lines of code and nontrivial objects such as schemes were formalized in this library. Recently, the liquid tensor experiment [7], a challenge posed by reknowed mathematician Peter Scholtze to formalize one of its theorem proved to be a success.

This makes of L∃∀N the perfect framework for formalizing the notion of groups of Lie type.

If some of these groups have been known for a long time (Galois constructed the projective special linear groups $PSL_2(F)$ over fields F of prime orders in 1830) the greatest advancement in this theory happened in the 1950s when Claude Chevalley generalized results about complex simple Lie groups to over arbitrary (and especially finite) fields (see [3]). Examples of such groups are general linear, special linear, symplectic and orthogonal groups and some of their subgroups and quotients (such as special, projective general and projective special linear groups). There exists however no consensual definition of what a group of Lie type is, yet, there is no confusion about the class of finite *simple* groups of Lie type. That last fact makes the formalization of these groups even more important as it could allow fix a clear definition for the term.

To achieve this formalization we used the concept of BN-pairs, first introduced by Jacques Tits in [2]. BN-pairs are structure over groups of Lie type which can be used to prove important results about the structure of the groups, such as simplicity (see §2 of [4]). Moreover every *finite* group of Lie type is endowed with a BN-pair. That makes them perfect candidate to be used as foundation for the formalization of groups of Lie type. This is why most of this report consists in formalization of properties of groups with BN-pairs. This porperties includes the above-mentioned simplicity theorem, construction of the BN-pair structure over the general linear group over a field and proof of the simplicity of the special projective general linear groups $PSL_n(F)$ over a field F, for $n \geq 3$ or $n = 2$ and $|F| \geq 4$.

This report starts by a basic introduction to L∃∀N . The goal of section 1 is the provide to the reader unfamiliar with L∃∀N the knowledge necessary to understand the code scattered throughout the report. Reader accustomed with L∃∀N can simply ignore that part as only basic concept of the language are exposed. It begins with a global presentation of the type system of L∃∀N then describes with more in detail the types of functions, and of structures as well as the concept of type class. Afterward, follows a brief presentation of two important types of L∃∀N , **`Prop`** and `Subgroup` .

[1] We recall here that a groups is said *simple* if it has exactly two normal subgroups : the trivial subgroup and itself.

Thereafter, in section 2, BN-pairs are defined and the new notion of a $BN\pi$-triplets is introduced. A $BN\pi$-triplet is a structure equivalent to a BN-pair but simpler to use in formalizing. Then results about double cosets, sets of the form BgB for B a subgroup and g an element of the group under consideration, are exposed and then used in the next subsection to prove that the general linear group over a field is endowed with BN-pair.

Afterwards, the section 3 presents some of the properties of groups with BN-pairs which were formalized during the internship, such as the notion of quotient $BN\pi$-triplet which allows to endow the projective general linear groups with a $BN\pi$-pair. These properties are then used to prove the simplicity theorem 4.1.1 in section 4. This theorem is then applied to the projective special linear group. Finally, we construct the BN-pair over the special linear and projective special Linear groups.

This document was written both as an internship report and as a documentation of the code written during the internship.

Therefore, certain parts of the report might not be of the greatest interest for the reader depending on what they view this document as. Thus, the author would advise, as stated above, for readers familiar with L∃∀N to ignore the first section and for reader viewing this document as an internship report not to pay too much attention to the technical lemmas specific to the L∃∀N implementation.

The code produced during this internship is available on github.

Aknowledgements

Before everything else, I would like to thank Johan Commelin who welcomed me for this internship, guided me throughout my research and who supported me during my timee in Utrecht.

I am grateful to Yannis Monbru without who this internship would never have happened.

I am also thankful to Jiedong Jiang for bringing life to the office and for the time spent outside of the office.

I am also grateful to Joseph Lenormand for joining in the Netherlands for a little bike trip, to Micol Giacomin for our various visits, to Félix Yvonnet for its LaTeXadvises and to all of them together for helping me during the hard times of this internship.

1 The Lean theorem prover

This section is an introduction the Lean theorem prover, its aim is to give to the readers unfamiliar with L∃∀N the minimal amount of knowledge necessary to understand this report. Resources for learning how to use L∃∀N are available at https://lean-lang.org/documentation/. . One can find the code used in this section by clicking here [2]. We strongly encourage the reader to interact with the code while reading this section for a better understanding of the notions manipulated.

In this section we begin present the notion of type which is of capital importance in L∃∀N . We then describe ways to construct new types which will be used in this internship. Finally, we briefly present the types `Prop` and `Subgroup` .

1.1 General introduction to Lean

L∃∀N is based on the *calculus of construction* a type theory first created by Thierry Coquand and that notably serves as foundation for the Coq proof assistant. As such the notion of type is of

[2] If the link is not working, the example file can be found here and pasted in the L∃∀N 4 web editor.

capital importance in L∃∀N , and this subsection aims to give the reader basic knowledges about this concept.

1.1.1 Typing

Every L∃∀N expression, that is every variable, number, string, etc., is associated with a property called a type. This type will has many usage, for example infer which operations can and should be used on the expression. For instance, most programming languages have `Int` (or `int`) and `String` (or `string` or `str`) types to represent integers and strings of characters. While the addition of two integers is perfectly defined as an operation, adding an integer with a string is a bit more mystical. Thus, the compiler or executer of the programming language can become pretty unpleased when asked to add terms of type `Int` and `String` .

We will denote as `expr : t` the fact that the expression `expr` has type `t` . For instance, `4 : ℕ` , `true : Bool` and `∃ n : ℕ, 3 ≤ n : Prop` . Here, the type `ℕ` represent natural numbers, `Bool` booleans and `Prop` mathematical properties. The last type will be discussed in more details in 1.2. One can access, to the type of a term using the `#check` command, see here .

To define an object and even types in L∃∀N , one can to use the `def` keyword :

```
def one : ℕ := 1
def vrai : Bool := true

#check one --ℕ

def  α : Type := ℕ
def one' : α := one

#check one' --α
```

On some occasion the `abbrev` keyword will be used instead of `def` to define objects, the reader unfamiliar with L∃∀N can simply consider it as equivalent to `def` .

It is also possible to evaluate expression in L∃∀N by using the `#eval` command :

```
#eval 2 + 3 --5
#eval one --1
#eval one' --1
#eval vrai && true --false
```

1.1.2 Function type

We will now discuss some ways to create new types in L∃∀N , starting with the types of functions.

If `α` and `β` are two types then the type of function from `α` to `β` is denoted `α → β` . For instance :

```
#check Bool.not -- Bool → Bool
#check Nat.add -- ℕ → ℕ → ℕ
```

One way to create functions in L∃∀N is to use the `fun` keyword. If from a variable `x : α` it is possible to construct a term `t : β` then the expression `fun (x : α) => t : α → β` represent the function mapping `x` to `t` . Here are some examples :

```
#check fun n : ℕ => n + 2 -- ℕ → ℕ
#check fun (b : Bool) => ¬ b -- Bool → Bool
```

```
#check fun (x : ℕ) (b : Bool) => if not b then x else x + 1    -- ℕ → Bool → ℕ
#check fun x b => if not b then x else x + 1    -- ℕ → Bool → ℕ
```

The last two expressions are interpreted as the same expression by L∃∀N . That is an example of a very powerful feature of L∃∀N : type inference. L∃∀N is indeed able to to infer the type of the variables `x` and `b` because it has enough information to do so. In the exemple above, L∃∀N knows that an argument of the function `not : Bool → Bool` has type `Bool` so `b : Bool` , the same goes for `n : ℕ` . It is however often good practice to precise the type of variable when introducing them.

To evaluate the output of a function one simply needs to write its arguments separated by whitespaces, without parenthesis :

```
#eval Nat.add 2 3 --5
#eval (fun n => n + 2) 3 -- 5
```

An import detail about function types is that the arrow `\to` is right associative : for every three types `α β γ : Type*` the types `α → β → γ` and `α → (β → γ)` are the same (but not `(α → β) → γ` .

A second way to create functions is to use the **`def`** keyword by putting the arguments of the function between parenthesis before the colon.

```
def add_two (n : ℕ) : ℕ := n + 2
def add (n m : ℕ) : ℕ := n + m

#eval add_two 3 -- 5
#eval add 4 3-- 7
```

To create an abstract undetermined type `α` , one can define then as being of type **`Type*`** which will ”create” a new type for each iteration of **`Type*`** . Any type can then replace them afterwards. This notation can of course be used to take types as arguments and create polymorphic functions, as we can see below for the `identity` function :

```
def identity (α : Type*) (a : α) : α := a
#check identity ℕ -- ℕ → ℕ

#eval identity ℕ 2 --2
#eval identity Bool true --true
```

However, in the example above having to precise the type `α` in the `identity` function can become tedious and L∃∀N allows us to avoid that by using *implict arguments*. If the arguments of a function are written between curly braces instead of parenthesis, L∃∀N will try to infer it from the context. This can be especially useful for function which takes types as arguments and can greatly improve readability, see below :

```
def identity' {α : Type*}  (a : α) := a
#check identity' 2 -- ℕ
#eval identity' 2 --2

#check identity' true-- Bool
#eval identity' true-- true
```

Finally, readability of L∃∀N code can also be improved by using variables. The **`variable`** keyword allows to declare variables for a whole section of a file [3], and every definition can have access to them without having to declare them.

[3] See chapter 6 of [9] for definition of sections of a L∃∀N file

```
variable (n : ℕ)
def add_two' := n + 2
def add' (m : ℕ) := n + m

#eval add_two 3 -- 5
#eval add 4 3-- 7
```

Implicit arguments can also by declared thanks to the **variable** keyword. For instance, the `indentity''` function declared below is equivalent to the above `identity'` .

```
variable {α : Type*} (a : α)

def identity'' := a
#eval identity'' 2 --2
#eval identity'' true-- structures
```

1.1.3 Structure

A second way to create new types in L∃∀N is to use structures. A structure is defined as "a specification of a collection of data, possibly with constraints that the data is required to satisfy" in [8]. For instance, we can define a `Point` structure, made of two coordinates as follows :

```
structure Point where
  x : ℕ
  y : ℕ
```

Then, an instance of the structure can be created by providing values for the different fields of the structure. For example, to create an instance `point1` of `Point` we proceed as follow :

```
def point1 : Point where
  x := 5
  y := 3
```

Structure can also take arguments, for instance the type `Prod` of products of two elements is defined as :

```
structure Prod' (α β: Type*) where
  fst : α
  snd : β

def point1' : Prod' ℕ ℕ where
  fst := 5
  snd := 3
```

Finally, it is possible to access to values of the different fields of instances of structures *via* `<instance>.<field>` , for instance :

```
#eval point1.x --5
#eval point1'.fst --5
```

1.1.4 Type classes

Another very powerful feature of L∃∀N are the type classes. This concept allows to associate "canonical" structures to a type, without having to carry them.

For instance, we can declare a `Group G` instance over a type `G : Type*` as below :

```
variable {G : Type*} [Group G]
```

This automatically endows the type `G` with a multiplication function `* : G → G → G`, an inverse function `⁻¹ : G → G` and a special element `1 : G` [4] which verifies the axioms of a group :

```
#check g*g' -- G
#check g⁻¹ -- G
#check (1 : G) --G

example : g * g⁻¹ = 1 := mul_inv_cancel g
example : (g * g') * g'' = g * (g' * g'' ):= mul_assoc g g' g''
```

When L∃∀N sees a multiplication between elements of type `G` it will try to find a type class instance over `G` that provides a multiplication (here `Group G`) and use the one supplied by that instance. Thus, the user does not have to provide the definition of the multiplication each time they use it.

Concept other than algebraic structures can be formalized through typeclass instances such as topologies, non emptiness or order as we will see next section.

Some types available in Mathlib comes with their own instances, for example, the type `ℕ` comes with `Semiring ℕ` instance[5].

1.2 Some types

We here present two types that will be used throughout this report : the **`Prop`** type used to formalize mathematical propositions and the `Subgroup` type, used to formalize the notion of subgroup.

1.2.1 The `Prop` type

In this section we will briefly describe the **`Prop`** type of L∃∀N .

This type is used to formalize mathematical propositions and its syntax should be intuitively understood by any person with reasonable knowledge of mathematics : the author makes here the hypothesis that the interpretation of terms such as `∀ n : ℕ, n = 0 ∨ ∃ k : ℕ, n = k + 1 : Prop` or `∀ n : ℕ, n > 1 → n ≠ 1 : Prop` should appear clearly to the reader.

There are however a few subtleties the author wants to highlight. The first one can already be seen in the last example : the symbol used in L∃∀N to denote implication is not a double arrow `⇒` but a simple arrow `→`. It actually the same symbol that is used denotes type of functions. That is for a very good reason, both operation are indeed exactly the same in L∃∀N for reasons which will not explain here but that the interested reader can find in [9]. Therefore, implies symbols also associate on the right.

The second point the author wants to address concerns difference in usage between formalized mathematics and mathematics in natural languages. When a mathematician states that a property P holds as long as two premisses h_1 and h_2 holds, they will express it in natural language as "if h_1 **and** h_2 are true then P is true". It would then be natural to translate the last phrase as $(h_1 \wedge h_2) \Rightarrow P$ in mathematical language. However, when working with formalized mathematics it is more natural and usual to formulate it as $h_1 \Rightarrow h_2 \Rightarrow P$. Therefore, when seeing propositions of the form `h₁→h₂→...→h□→P` one should interpret them as `h₁` and `h₂`, and ... and `h□` gives `P`.

[4] The `Group G` instance provides other features, but those will not be used here.

[5] A semi ring is a triplet $(N, +, \times)$ such that $(N, +)$ is a commutative monoid, $(N, \times)$ is monoid and $\times$ distributes over $+$.

The main objective of L∃∀N as a theorem prover is namely to *prove* these propositions. However, no formalized proof will figure in this report (even though most of the work performed during the internship consisted in formalizing proofs). Indeed, in practice, formalizing proofs makes use of a large diversity of tools, which would need to be explained to the reader. That exercise would require a large amount of efforts for little to no interest for the reader. Moreover, a complex L∃∀N proof would hardly be readable on paper without an interactive editor. That is why, when referring to a result proven in L∃∀N we will only state its name and the property it formalizes preceded by the keyword **lemma** or **theorem** . These two keyword are the ones used when formalizing proof. You can find examples below :

```
/-This is how the results appear in the code-/
theorem eq_zero_or_succ: ∀ n : ℕ, n = 0 ∨ ∃ k : ℕ, n = k + 1 :=
  fun n => Or.elim Nat.eq_zero_or_eq_sub_one_add_one
    (fun h => Or.inl h) fun h => Or.inr ⟨n-1, h⟩
lemma ne_one_of_le_one : ∀ n : ℕ, n > 1 → n ≠ 1:= fun _ h => ne_of_gt h

/-This is how the results are written in the report-/
theorem eq_zero_or_succ: ∀ n : ℕ, n = 0 ∨ ∃ k : ℕ, n = k + 1
lemma ne_one_of_le_one : ∀ n : ℕ, n > 1 → n ≠ 1
```

Moreover, if a structure has fields of type **Prop** then when defining instances of such structures, the fields of type **Prop** will be replaced by ellipsis as below :

```
structure EvenNumber where
  n : ℕ
  even_n : ∃ k, n = 2 * k

/-How it appears in the code-/
def two_even : EvenNumber where
 n := 2
 even_n := ⟨1,mul_one 2⟩

/-How it is written in the report-/
def two_even : EvenNumber where
  n := 2
  ...
```

We finish this subsection by some more syntax. The four **lemma** below are all equivalent to the **lemma** , `ne_one_of_le_one` above and should be interpreted as follows provided with a natural number `n : ℕ` and proof `h` of `n>1` (denoted `h : n > 1`), `ne_one_of_le_one n h` outputs a proof of `n \neq 1` .

```
lemma ne_one_of_le_one' (n : ℕ) : n > 1 → n ≠ 1
lemma ne_one_of_le_one'' (n : ℕ) (h : n > 1) : n ≠ 1
variable (n : ℕ)
lemma ne_one_of_le_one''' :  n > 1 → n ≠ 1
lemma ne_one_of_le_one''''  (h : n > 1) : n ≠ 1
```

Finally, properties pasted as arguments can be used to infer types :

```
lemma ne_one_of_le_one''''' {n : ℕ} (h : n > 1) : n ≠ 1
variable (h : n > 1)
#check ne_one_of_le_one n h --n ≠ 1
#check ne_one_of_le_one''''' h --n ≠ 1
```

1.2.2 The `Subgroup` type

In this section we will present the type `Subgroup` defined in Mathlib.Algebra.Group.Subgroup.Basic. The language presented here, in particular the structure of `CompleteLattice` of `Subgroup` will be used throughout the report and the reader should pay great attention to the following lines and read them again if necessary.

The `Subgroup` type can be seen[6] as a structure taking a type `G : Type*` with `[Group G]` instance as argument and with four fields : a set of element of `G` named `carrier : Set G`, and proofs that this set is closed under product, contain the element `1 : G` and is closed under the inverse operation of `G` :

```
structure Subgroup' (G : Type*) [Group G] where
    carrier : Set G
    mul_mem' : ∀ {a b : G}, a ∈ carrier → b ∈ carrier → a * b ∈ carrier
    one_mem' : 1 ∈ carrier
    inv_mem' : ∀ {x : G}, x ∈ carrier → x⁻¹ ∈ carrier
```

Let us consider a type `G : Type*` with a `[Group G]` instance. The type `Subgroup G` is naturally endowed with a `[CompleteLattice (Subgroup G)]` instance. This instance endows `Subgroup G` with a order `≤ : Subgroup G → Subgroup G → Prop`, a supremum function `⊔ :Subgroup G → Subgroup G → Subgroup G` and a infimum function `⊓ : Subgroup G → Subgroup G → Subgroup G`, a set supremum `⨆ : Set (Subgroup G) → Subgroup G` and a set infimum `⨅ : Set (Subgroup G) → Subgroup G` functions, as well a unique maximal `⊤ : Subgroup G` and a unique minimal `⊥ : Subgroup G` element.

These different all have a very simple mathematical interpretation. The order is the one defined by $H \leq K$ if and only if H is a subgroup of K for any two subgroups H and K of `G`.

Therefore, the infimum of two subgroups is nothing more than the intersection of these two subgroups and infimum of a set of subgroup is the intersection of all the subgroups of the set. The supremum of two subgroups is the subgroup generated by the union of these two subgroups and the supremum of a set of subgroup is the group generated by the union of all the groups in the set. Finally, `⊤` and `⊥` the maximal and minimal elements of `Subgroup G` represents the whole group `G` seen as subgroup of itself and the trivial subgroup, respectively.

Remark. When working with relations between subgroups `⊤` will be used to denote the whole group `G`. For instance, if `H` has type `Subgroup G` then `H = G : Prop` states that `H` and `G` are equal as *types* which is false, whereas `H = ⊤ : Prop` states that `H` and `G` are equal as subgroups of `G` which can be true.

2 Groups of Lie type

If the lack of a consensual definition mentioned in the introduction doe not appear to be a real issue when doing pen-and-paper mathematics, it becomes a problem when it comes to formalization. Fortunately, the theory of groups of Lie type provides a powerful tool *BN-pairs*. A BN-pair over a group G is pair of subgroups of G verifying certain conditions and can be used to state very important results about the structure of the group G. Moreover, *every finite group of Lie type* is endowed with

[6] This is not actually how the `Subgroup` structure is coded in Mathlib. It instead inherits from the `Submonoid` structure but explaining inheritance here would only add an unnecessary layer of complexity.

such structure and as our work mainly aims at formalizing the CFSG , this notion makes for an almost perfect foundation for the formalization of groups of Lie type. However, working BN-pairs in L∃∀N is not exactly the best solution as it relies heavily on the theory of quotient groups, which is formalized in the `mathlib` library of L∃∀N but is not pleasant to work with. That is why we introduced an equivalent by more formalization-friendly notion called a $BN\pi$-triplet which makes use of surjection instead of quotient, and which is the one we will use throughout this report.

The goal of this section is to define BN-pairs and present some results about the notion of *double cosets*, a basic concept of groups theory bearing strong links with groups of Lie type. This section starts by defining double cosets, BN-pairs and $BN\pi$-triplets and present their formalization in L∃∀N . Then, in 2.2 some results about double cosets and especially doubles cosets of groups with $BN\pi$-triplet are exposed. Finally, these results are put to good use in 2.3 to build a $BN\pi$-triplet over the general linear group over a field.

2.1 Definition

For this all, report we we fix two groups G and W and in L∃∀N we declare two variables `G W: Type*` endowed with a `[Group G]` and `[Group W]` instance. We additionally fix two subgroups B and N of G and π a group homomorphism from N to W.

First and foremost, we need to define the notion of double coset. A concept that, although elementary, is of capital importance to discuss effectively the structure of groups of Lie type. In this subsection, we only give their definition and they will further discussed in 2.2.

Definition 2.1.1 (Double coset)**.** A *double coset* of a subgroup H of G is a set of the form HgH for $g \in G$. We will denote as $C(g)$ the double coset BgB, for $w \in G$.

Double cosets are defined in `DoubleCoset.lean` file of the project as

```
abbrev DoubleCoset (B : Subgroup G) (w : G) : Set G :=
    {x | ∃ b  b' : B, b * w * b' = x}
```

However, for the sake of readability the notation below was added to the code [7] :

```
notation "C"  => DoubleCoset
```

The definition of double coset allow us to introduce the main object of study of this report : BN-pairs.

Definition 2.1.2 (BN-pair)**.** The couple (B, N) of subgroups of G forms a *BN-pair* over G if it that satisfies the following axioms :

1. G is generated by B and N.
2. the subgroup $T := B \cap N$ is normal in N.
3. The group N/T is generated by a set S of elements of order 2.
4. For all $n_i \in N$ that maps to an element of S under the natural group homomorphism $N \to N/T$, $n_i Bni \neq B$.
5. For all n_i as above and all $n \in N$, $n_i Bn \subset C(n_i n) \cup C(n)$

where $C(g)$ denotes the double coset BgB for $g \in G$.

Notation. If (B, N) is a $BN-$pair over G, we also say that (G, B, N, S) is a *Tits system*, see for instance [4].

However, the definition of a BN-pair presented above can be impractical to manipulate in L∃∀N . Indeed, if quotient groups are fully implemented in the `mathlib` library and working with this

[7] The code presented here allows to write `C` instead `DoubleCoset` in the code.

definition would still be be possible, it is simpler to avoid their use and instead have recourse to surjective group homomorphism. This is why we introduced the new and equivalent (see 2.1.4) structure of a $BN\pi-$triplet.

Definition 2.1.3 ($BN\pi$-Triplet)**.** The triplet (B, N, π) is a *$BN\pi$-Triplet* over the G if it satisfies the following axioms :

1. G is generated by B and N.
2. the subgroup $T := B \cap N$ is the kernel of π.
3. π is surjective
4. The group W is generated by a set S of elements of order 2.
5. For all $n_i \in N$ that maps to an element of S through π, $n_i B n i \neq B$.
6. For all n_i as above and all $n \in N$, $n_i B n \subset C(n_i n) \cup C(n)$

First of all let us show that the two structures are equivalent. This is the point of the following theorem :

Theorem 2.1.4. The group G is endowed with a BN-pair structure if and only if it is endowed with a $BN\pi$-structure.

Proof. If the group G is endowed with a BN-pair structure, it immediately has a $BN\pi$-Triplet with $W = N/(B \cap N)$ and π the natural projection $N \to W$.

Conversely, if G has a $BN\pi$-Triplet structure, then by lifting π through the natural projection to $N/(B \cap N)$ we obtain a group isomorphism $\tilde{\pi} : W \to N/(B \cap N)$. Then if $S \subset W$ denotes the generating set of W from the definition of a $BN\pi$-triplet, $\left(G, B, N, \tilde{\pi}^{-1}(S)\right)$ is a Tits system. □

The concept of $BN\pi$-triplet is formalized as follow in the `Basic.lean` file :

```
structure BNpiTriplet (G : Type*) [Group G] (W : Type*) [Group W] where
  B : Subgroup G

  N : Subgroup G

  S : Set W

  pi : N →* W

  BN_gen_Group : ⊤ = B ⊔ N

  pi_surj : Function.Surjective pi

  ker_pi_eq_B_subgroupOf_N : pi.ker = B.subgroupOf N

  S_auto_inverse : ∀ s ∈ S, s * s = 1

  top_eq_S_closure : ⊤ = closure S

  left_right_mul_B_neq_B_of_map_S: ∀ ni ∈ pi⁻¹' S,
    ¬ {(ni :G) * (b : G) * ni | b : B} ⊆  B

  subset_union_doubleCosets :  ∀ (n ni : N), ni ∈ pi ⁻¹' S ->
   {ni * b.val * n | b : B} ⊆ C B (ni * n) ∪ C B n
```

Remark. 1. The term `B.subgroupOf N : Subgroup ↥N` designates the groups `B ⊓ N` seen as a subgroup of `N` instead of `G` .

2. The property `S_auto_inverse : ∀ s ∈ S, s * s = 1` ensure that every element of `S : Set W` has order at most `2`, and property `left_right_mul_B_neq_B_of_map_S` ensures that they do not have order `1`.

2.2 Double Cosets

In this section we expose some properties of the double cosets and especially double cosets of groups with a BN-pair.

We do not suppose at first that there is a $BN\pi$-triplet over G and state some general facts about double cosets :

Properties 2.2.1. We have the following properties :
- $C(1) = B$
- $\forall g, g' \in G, C(gg') \subset C(g)C(g')$
- $\forall g \in G, C\left(g^{-1}\right) = C(g)^{-1}$.
- Two double cosets are either equal or disjoints.

Proof. The first three points are immediate.

Let g_1, g_2 be two elements of G such that $C(g_1) \cap C(g_2) \neq \emptyset$. There exists $b_1, b_2, b_1', b_2' \in B$ such that $b_1 g_1 b_1' = b_2 g_2 b_2'$. Hence, $g = b_1^{-1} b_2 g_2 b_2' b_1'^{-1} \in C(g_2)$ and $C(g_1) = B g_1 B \subset C(g_2)$. By symmetry, we have $C(g_1) = C(g_2)$, and hence the fourth point. □

These properties can be found in the `DoubleCoset.lean` file as :

```
variable {B : Subgroup G}

theorem doubleCoset_one : DoubleCoset B 1 = B.carrier

theorem DoubleCoset.mul_le_mul (g g': G) : C B (g*g') ⊆ C B g * C B g

theorem DoubleCoset.inv (g : G) : (C B g)⁻¹ = C B g⁻¹

theorem DoubleCoset.disjoint_of_neq (g g' :G) (h : C B g ≠ C B g') :
  Disjoint (C B g) (C B g')
```

This file also contains some results deriving from the ones above. They have little mathematical interest but make working with double cosets in L∃∀N more convenient. Some of them can be found below.

```
lemma DoubleCoset.eq_doubleProd_singleton (g :G) : C B g = B.carrier * {g} * B.carrier

theorem DoubleCoset.self_mem {g : G} : g ∈ C B g

theorem DoubleCoset.eq_of_gen_mem {g g' :G} (h : g ∈ C B g') : C B g = C B g'

theorem DoubleCoset.mul_apply (g g' :G) :
  (C B g) * (C B g') = {x :G| ∃ b1 : B,∃ b2 : B, ∃ b3 : B, b1 * g * b2 * g' * b3 =x}
```

Moreover the link between double cosets and $BN\pi$-triplet appears more clearly through the following properties:

Properties 2.2.2.

1. For all $g, g' \in G$, we have $gBg' \subset C(g) \cup C(g'g)$ if and only if $C(g).C(g') \subset C(g) \cup C(g')$
2. For all g_i, g such that $C(g_i).C(g) \subset C(g) \cup C(g_i g)$

$$C(g_i)C(g) = \begin{cases} C(g_i g) \text{ if } C(g) \not\subset C(g_i).C(g) \\ C(g) \cup C(g_i g) \text{ otherwise} \end{cases}.$$

3. If $g_i \in G$ verifies, $g_i^2 = 1$, $g_i B g_i \subset B \cup C(g_i)$ and $g_i B g_i \not\subset B$ then $B \cup C(g_i)$ is equal to $C(g_i)C(g_i)$ and is a subgroup of G.

Proof. The first point is immediate, the second follows from the hypothesis and the points 2 and 4 of 2.2.1. Concerning the third point, the identity $B \cup C(g_i) = C(g_i).C\,(g_i)$ follows from the second point in the case $g = g_i$ and $g_i^2 = 1$, and using this equality it is easy to show that $B \cup C(g_i)$ is a subgroup of G. □

Remark. The point 1 of 2.2.2 can be use to give an alternative statement to point 5 and 6 of 2.1.2 or 2.1.3. The point 6 can be seen as a very particular case of 3.3.2 and will be used in 2.3. We formalized it as :

```
def B_union_C_of_simple {s : G} (hs : s*s = 1) (h: {s*b*s | b : B} ⊆ C B s ∪ (C B (s*s)))
    (h' :¬ {s*b*s | b:B} ⊆ B.carrier) : Subgroup G where
  carrier := B.carrier ∪ (C B s)
  ...
```

We now suppose that (B, N, π) is a $BN\pi$-triplet over G. In L∃∀N we fix a variable `TS` of type `BNpiTriplet G W`.

Definition 2.2.3. The *double coset* $C'(w)$ of an element $w \in W$ is defined as the double coset $C(n)$ of B for $n \in N$ mapping to w through π.

Remark.

- The set $C'(w)$ is well defined for all $w \in W$ because using the point 4 of 2.1.3 we have $\ker \pi = N \cap B \leq B$.
- Some authors use the above definition of a double coset to define $BN\pi$-pairs (see for example [4]) as the most important results of the theory of groups with BN-pair are more conveniently expressed through double cosets of elements W than double cosets of elements of G. The theorems 2.2.6 and 3.3.2 below are good examples of this fact. However, it is more comfortable to use the definition 2.2.3 to formalize the property of a certain group having a $BN\pi$-triplet and to first prove properties of double cosets of elements of G and then transfer them to double cosets of elements of W.

In order to make the L∃∀N code more readable, we added `BNpiTriplet.C` defintion that allows to access to `C TS.B n : Set G` by typing `TS.C n : Set G` for any element `n : TS.N`. Then we formalize the above definition as `BNpiTriplet.C'`.

```
def BNpiTriplet.C (n : TS.N) : Set G := C TS.B n

def BNpiTriplet.C' (w : W): Set G :=
  TS.C (@Classical.choose TS.N (fun n => TS.pi n = w) (TS.pi_surj w))
```

Code commentary 2.2.4.
The function `Classical.choose :{α : Type*} {p : α → Prop} (h : ∃ x, p x)`

`: α` maps to an object `h : ∃ x, p x` for a certain `p : α → Prop` a element of type `α` that verifies `p` .

For a convenient use of the double cosets of elements of W, we added the following lemmas :

```
lemma C'_comm_pi (n : TS.N) : TS.C' (TS.pi n) = TS.C n

lemma C'apply (w : W) : ∃ n ∈ TS.N, TS.C' w = C TS.B n

lemma C'apply' (w : W) : ∃ n : TS.N, w = TS.pi n ∧ TS.C' w = C TS.B n
```

Most of the properties of double cosets of elements of G can be transferred over double cosets of elements of W, for instance we have :

Properties 2.2.5.
- $C'(1) = B$
- $\forall w, w' \in G, C'(ww') \subset C(w)C(w')$
- $\forall w \in G, C'\left(w^{-1}\right) = C(w)^{-1}$.
- Two double cosets are either equal or disjoints.
- $\forall s \in S, \forall w \in W, C'(s).C'(w) \subset C'(s) \cup C'(w')$
- For all $s \in S$ and $w \in W$ we have :

$$C'(s)C'(w) = \begin{cases} C'(sw) \text{ if } C'(s) \not\subset C'(s).C'(w) \\ C'(w) \cup C'(sw) \text{ otherwise} \end{cases}.$$

Proof. These properties follows from 2.2.1 and 2.2.2. □

We can now use these results to show an important result of the theory of groups with BN-pairs : the decomposition of G in double cosets.

Theorem 2.2.6 (Double coset decomposition)**.** The group G is equal to $BNB = \bigcup_{w \in W} C'(w)$.

Proof. First, let us show that BNB is a subgroup of G. It appears immediately that $1 \in BNB$ and BNB is closed by inversion. To prove that BWB is closed under multiplication we will need to prove the following lemma :

Lemma 2.2.7. Let $s_1, \ldots, s_q \in S$ and $w \in W$. We have

$$C'(s_1, \ldots s_q).C'(w) \subset \bigcup_{1 \leq i_1 < \ldots < i_p \leq q} C'(s_{i_1} \ldots \ s_{i_p} w).$$

Proof. We prove it by induction, on q, the case $q = 0$ being trivial. Suppose $q \geq 1$ and that the lemma holds for $q - 1$. Then, we have :

$$\begin{aligned}
C'(s_1, \ldots, s_q).C(w) &\subset C'(s_1).C'(s_2, \ldots, s_q).C'(w) && \text{by point 2 of 2.2.5} \\
&\subset C'(s_1) \bigcup_{2 \leq i_2 < \ldots < i_p \leq q} C'\left(s_{i_2} \ldots s_{i_p} w\right) && \text{by induction} \\
&\subset \bigcup_{2 \leq i_2 < \ldots < i_p \leq q} C'\left(s_{i_2} \ldots s_{i_p} w\right) \cup C'(s_1) C'\left(s_{i_2} \ldots s_{i_p} w\right) && \text{by point 6 of 2.2.5} \\
&= \bigcup_{1 \leq i_1 < \ldots < i_p \leq q} C'\left(s_{i_1} \ldots s_{i_p} w\right).
\end{aligned}$$

□

Let $x, y \in BNB$. There exists $v, w \in W$ such that $x \in C'(v)$ and $y \in C'(w)$. As the set S generates W we have a finite sequence $s_1, \ldots, s_q$ of elements of S such that $v = s_1 \ldots s_q$ and by the above lemma $xy \in C(v).C(w) \subset \bigcup_{1 \leq i_1 < \ldots < i_p \leq q} C(s_1 \ldots s_q w) \subset BNB$.

Therefore BNB is a subgroup of G that contains both B and N, so it is equal to G. □

The group BNB appear in the document `basic.lean` as

```
def BNB : Subgroup G where
  carrier := TS.B *  TS.N * TS.B
  ...
```

We formalized the theorem 2.2.6 as lemma 2.2.7 as following :

```
theorem doubleCosetDecomp : ⊤ = BNB TS

theorem simple_prodsubset_union_of_simples' (w : W) {ls : List W}
 (lsinS :∀ s ∈ ls, s ∈ TS.S):
  TS.C' ls.prod * TS.C' w ⊆ ⋃ l ∈ { l | l.Sublist ls}, TS.C' (l.prod * w)
```

A more convenient version of 2.2.6 can be found as :

```
theorem doubleCosetDecomp' (g : G) : ∃ w, g ∈ TS.C' w
```

2.3 Example : the general linear group

For this section we fix a natural number $n \geq 2$ and a field F. In a similar fashion, in lean we fix :

```
variable {F : Type*} [Field F] [DecidableEq F]
variable {n : Type*} [Fintype n] [DecidableEq n]
```

Notation. We will denote as $GL_n(F)$ the *general linear groups i.e.* the groups of invertible $n \times n$ matrices over the field F. In L∃∀N this groups is denoted `GL n F` and is defined as the group of units of the ring `Matrix n n F` of matrices indexed by the type `n` with coefficients in `F` .

The goal of this section is to show that the general linear group $GL_n(F)$ admits a BN-pair by showing that it has a $BN\pi$-triplet structure. In order to do so, we first need to define the notion of monomial matrices.

Definition 2.3.1 (Monomial matrix)**.** A *monomial matrix* M is a square matrix with one and only one nonzero coefficient per line and column.

Property 2.3.2. The set of monomial matrices is a subgroup of $GL_n(F)$ that will be denoted as $N_n(F)$.

Proof. First, the identity matrix is monomial so $I_n \in N_n(F)$. Next, let $M = (m_{i,j})_{i,j \in \{1,\ldots,n\}}$ and $N = (n_{i,j})_{i,j \in \{1,..,n\}}$ be monomial matrices. Let $i \in \{1,..,n\}$ and k_i denote the only nonzero coefficient of the $i-$th line of M. Then, $p_{i,j}$the coefficient (i,j) of the matrix $P := M \times N$ is equal to $p_{i,j} = \sum_{k=1}^{n} m_{i,k} n_{k,j} = m_{i,k_i} n_{k_i,j}$ for all $j \in \{1, \ldots, n\}$. Hence, P has only one nonzero coefficient on the i-th line : the coefficient on the $j_{k_i}-$th line ; where j_{k_i} is the only nonzero coefficient on the k_i-th line of N. By a similar reasoning on columns we obtain that P is a monomial matrix.
Finally, the calculations above show that the matrix $M' = \left(m'_{i,j}\right)_{i,j \in \{1,\ldots,n\}}$ defined by $m'_{i,j} = 1/m_{j,i}$ if $m_{j,i} \neq 0$ and 0 otherwise is the inverse of M and is monomial. □

Implementation. In L∃∀N the property of being monomial is formalized through

```
def Matrix.Monomial (M : Matrix n n F) : Prop :=
    (∀ i, ∃! j, M i j ≠ 0) ∧ ∀ j, ∃! i, M i j ≠ 0
```

The groups of monomial matrices is defined as the subgroup of `GL n F` made of the invertible matrices who coerce to monomial matrices.

```
def MonomialGroup : Subgroup (GL n F) where
  carrier := {M : GL n F | Monomial M.val}
    ...
```

The group $N_n(F)$ will serve as the N subgroup of the $BN\pi$-triplet over $GL_n(F)$. We will now define the B subgroup.

Definition 2.3.3 (Triangular group)**.** The *(upper) triangular group* $B_n(F)$ is the subgroup of $GL_n(F)$ made of the invertible upper triangular matrices.

Remark. Unless explicitly stated otherwise a triangular matrix denotes an *upper* triangular matrix.

The implementation of this group in L∃∀N is based on the formalization of block triangular matrices, which is already coded in `mathlib` (see here) as follows :

```
variable {α : Type*} [LT α]
def Matrix.BlockTriangular (M : Matrix n n F) (b : n → α) : Prop :=
 ∀ ⦃i j : n⦄, b j < b i → M i j = 0
```

Code commentary 2.3.4.
In the code above, the type `α : Type*` is associated with a `[LT α]` instance. This type class instance endows `α` with a binary relation `< : α → α → Prop` with no particular property.

With this we define the groups of block triangular matrices (which depends on the block decomposition given by the parameter `b : n → α`) :

```
def BlockTriangularGroup (b : n → α ) : Subgroup (GL n R) where
  carrier :=  {M : GL n R | BlockTriangular M.val b }
  ...
```

Now, we endow the type `n` with a `[LinearOrder n]` instance, doing this endows `n` with a *total* order $\leq$. We then define the group of triangular matrices as :

```
variable [LinearOrder n]
def  TriangularGroup : Subgroup (GL n F):= BlockTriangularGroup id
```

where `id : n → n` is the identity map.

Finally, let us define π homomorphism that forms the $BN\pi$-triplet over $GL_n(F)$.

If $M \in GL_n(F)$ is a monomial matrix, let $f : \{1\dots,n\} \to \{1,\dots,n\}$ be the map that to $j \in \{1,\dots,n\}$ associate the index $f(j)$ such that $M_{f(j),j} \neq 0$, then

Lemma 2.3.5. 1. f belongs to the permutation group $\mathfrak{S}_n$ over $\{1,\dots,n\}$.
2. the map

$$\begin{aligned} \pi : N_n(F) &\longrightarrow \mathfrak{S}_n \\ M &\longmapsto \pi(M) = f \end{aligned}$$

is a group homomorphism.

Proof. The first point is immediate by definition of a monomial matrix and the second follows from the calculations performed in the proof 2.3.2 to show that $N_n(F)$ is closed by product. □

The above map is implemented in the `Monomial.lean` file as follows :

```
def Matrix.toPermFun {M : Matrix n n F} (hM : Monomial M) : n → n :=
  fun j ↦ Classical.choose (hM.2 j).exists_list_transvec_mul_mul_list_transvec_eq_diagonal

def Matrix.toPerm {M : Matrix n n F} (hM : Monomial M) : Perm n where
  toFun := Matrix.toPermFun hM
  ...

def toPermHom : (MonomialGroup : Subgroup (GL n F)) →* Perm n where
  toFun := fun M => toPerm M.2
  ...
```

Code commentary 2.3.6.
In the above code we first define the function f as `Matrix.toPermFun` for matrices which are monomial. Then `Matrix.toPerm` lift it to an element of `Perm n`, the type of permutation on `n`. Finally, `toPermHom` lift `Matrix.toPerm` a group homomorphism from `MonomialGroup : Subgroup (GL n F)` to `Perm n`.

We can now state the following theorem :

Theorem 2.3.7. The subgroups $B_n(F)$ and $N_n(F)$ alongside the group homomorphism $\pi : N \to \mathfrak{S}_n$ form a $BN\pi$-triplet over $GL_n(F)$.

Proof. Before proving that the first axiom of $BN\pi$-Triplet holds let us introduce one last group.

Definition 2.3.8 (Permutation matrices)**.** The *permutation matrix* associated to the permutation f is the matrix $P_f = \left(\delta_{i,f(j)}\right)_{i,j\in\{1,..,n\}}$.

Immediately we have the following property :

Property 2.3.9. The permutation matrices are monomial and the map $P : f \mapsto P_f$ is group isomorphism on its image.

Proof. It is immediately clear that the permutation matrices are monomial and that the map P is injective. Then, the multiplicativity of P can be verified through simple computations. □

In L∃∀N we first define the map P as the function `PermMatrix : Perm n → Matrix n n F`, then we endow it with a monoid homomorphism `_PermMatrix_Hom : Perm n →* Matrix n n F`. This monoid homomorphism is then lifted to a group homomorphism from `Perm n` to the group of units of `Matrix n n F`, that is `GL n F := (Matrix n n F)ˣ` called `PermMatrix_Hom : Perm n →* GL n F`. Finally, `PermMatrixGroup : Subgroup (GL n F)` the group of permutation matrices is defined as the range of that last homomorphism and by restricting this same morphism to its range we obtain the group homomorphism `PermMatrix_Hom' : Perm n →* PermMatrixGroup`.

```
/--The permutation matrix associated to a permutation `f`.-/
def PermMatrix (f : Perm n) : Matrix n n F :=
    Matrix.of (fun i j ↦ if i = f j then 1 else 0)

 /-- PermMatrix seen as a monoid homomorphism from `Perm n` to `Matrix n n R`-/
```

```
def _PermMatrix_Hom : Perm n →* Matrix n n F where
  toFun := fun f ↦ PermMatrix f
  ...

/-- The group homomorphism version of `_PermMatrix_Hom`-/
def Matrix.PermMatrix_Hom : Perm n →* GL n F :=
  (_PermMatrix_Hom : Perm n →* Matrix n n F).toHomUnits

/-- The Group of permutation matrices.-/
abbrev PermMatrixGroup : Subgroup (GL n F) := PermMatrix_Hom.range

/--The PermMatrix seen as a group homomorphism from `Perm n` to `PermMatrixGroup`-/
def PermMatrix_Hom' : Perm n →* (PermMatrixGroup :Subgroup (GL n F)) :=
  PermMatrix_Hom.rangeRestrict
```

Now let us show that $(B_n(F), N_n(F), \pi)$ verify the $BN\pi$-triplet axioms.

1. First we prove that $GL_n(F)$ is generated by $B_n(F)$ and $N_n(F)$. To do so, we will use the following result :

   Lemma 2.3.10. Every matrix in $GL_n(F)$ can be written as a product of transvection matrices, an invertible diagonal matrix and transvection matrices where transvection matrices are matrices with only 1 as diagonal coefficient and exactly one nonzero nondiagonal coefficient.

   Proof. By operation on the lines and columns of the matrix. □

   This lemma appears in the `mathlib` library as :

   ```
   theorem Matrix.Pivot.exists_list_transvec_mul_mul_list_transvec_eq_diagonal
     (M : Matrix n n F) : ∃ (L : List (Matrix.TransvectionStruct n F))
       (L' : List (Matrix.TransvectionStruct n F)) (D : n → F),
         (List.map Matrix.TransvectionStruct.toMatrix L).prod * M *
           (List.map Matrix.TransvectionStruct.toMatrix L').prod = Matrix.diagonal D
   ```

   Diagonal matrices being blatantly triangular, we just need to show that are in $\langle B, N\rangle$. Let $c \in F$ and i, j be two distinct elements of $\{1, \dots, n\}$ and let $T_{i,j}(c)$ denote the transvection matrix of indexes (i, j) and scalar c as defined in **??**. If $i < j$, $T_{i,j}(c)$ is an upper triangular matrix. Otherwise, if $j > i$ then by remarking that $T_{i,j}(c) = P_{(i,j)}T_{j,i}(c)P_{(i,j)}$, we have the result because as just seen $T_{j,i}(c) \in B_n(F)$ and by 2.3.9, the transposition matrix $P_{i,j}$ is monomial.
   In L∃∀N , we refer to this result as :

   ```
   theorem MonomialTriangular_gen_GL :
       (⊤ : Subgroup (GL n F)) = MonomialGroup ⊔ TriangularGroup
   ```

2. For the second point let us show that both the intersection of $B_n(F)$ and $N_n(F)$ and the kernel of π are equal to the group $T_n(F)$ of invertible diagonal matrices.
   Firstly, all invertible diagonal matrices are clearly triangular so $T_n(F) \leq B_n(F) \cap N_n(F)$. Conversely, if $M \in B_n(F) \cap N_n(F)$, as an invertible triangular matrix, its diagonal coefficients are nonzero, then as a monomial matrix they are the only nonzero coefficients, so $M \in B_n(F) \cap N_n(F)$.
   Secondly, a matrix $M \in GL_n(F)$ belongs to the kernel of π if and only if for all index $i \in \{1, .., n\}$ the only nonzero coefficient on the i-th line are the diagonal ones *i.e.* if and only if $M \in T_n(F)$. Therefore, $T_n(F) = B_n(F) \cap N_n(F)$.
   Hence, by transitivity of the equality relation we have $\ker \pi = Bn(F) \cap N_n(F)$. That statement is accessed through :

```
theorem ker_toPermHom_eq_Diagonal :
  toPermHom.ker = TriangularGroup.subgroupOf (MonomialGroup :Subgroup (GL n F))
```

3. We will now show that $\pi : N_n(F) \to \mathfrak{S}_n$ is surjective. We will do it by showing that $P : f \mapsto P_f$ is a right section of π. Indeed, for all $f \in \mathfrak{S}_n$ we saw that the only nonzero coefficient of the i-th line, for $i = 1, ..., n$, is $f(i)$. Therefore, $\pi(P_f) = f$ and π is surjective.We formalized this result in L∃∀N as:

```
theorem toPermHom_surj :
  Function.Surjective (toPermHom : (MonomialGroup : Subgroup (GL n F)) →* Perm n)
```

4. A simple induction on n shows that $\mathfrak{S}_n$ is generated by the set $S := \{(i, i+1) \,|1 \leq i \leq n-1\}$ and transposition being of order 2, this proves the fourth point.
To formalize the definition of the set S (denoted `SuccSwapSet : Set (Perm n)` in L∃∀N) we need to endow `n` with a `[SuccOrder n]` instance. We then obtain a successor function `succ : n → n` that verifies the following axioms `le_succ : ∀ (a : n), a ≤ succ a` , `succ_le_of_lt : ∀ {a b : n}, a < b → succ a ≤ b` and `le_of_lt_succ : ∀ {a b : n}, a < succ b → a ≤ b` and such that `succ i = i` if and only if `i : n` is a maximal element of `n` . We formalize the set S and the above result as :

```
def SuccSwapSet : Set (Perm n) := {f | ∃ i, i < succ i ∧ f = swap i (succ i) }
theorem top_eq_succ_swap_closure :⊤ = Subgroup.closure (SuccSwapSet : Set (Perm n))
```

5. Let $i \in \{1, .., n-1\}$ and $M_i \in N_n(F)$ maps to $(i, i+1)$ through π. By 2 and 3 there exists $D \in T_n(F)$ such that $M_i = P_{(i,i+1)}D$. Therefore, $M_iB_n(F)M_i \neq B_n(F)$ if and only if $P_iB_n(F)P_i \neq B_n(F)$, where P_i denotes the permutation matrix $P_{(i,i+1)}$. As seen in 1, the transvection matrix $T_{i+1,i}(1)$ is equal to $P_iT_{i,i+1}(1)P_i \in P_iB_n(F)P_i$. However, $T_{i+1,i}(1)$ is not upper triangular and hence $P_iB_n(F)P_i \neq B_n(F)$.
This result appears in our code as :

```
theorem left_right_mul_Triangular_neq_Triangular_of_map_swap {Ma : MonomialGroup}
 (h : Ma ∈ toPermHom⁻¹' SwapSet) :
   ¬ {Ma * (b.val: GL n F) * (Ma.val : GL n F) | b :  TriangularGroup} ⊆
    (TriangularGroup : Subgroup (GL n F)).carrier
```

6. The final point of the proof is the hardest to prove. We introduce $i \in \{1, \ldots, n-1\}$ and $M_i \in N_n(F)$ as above and we fix a monomial matrix $M \in N_n(F)$.
Let us show that $M_iB_n(F)M \subset C(M_iM) \cup C(M)$. Thanks to a reasoning similar to the one above, we only have to prove that $P_iB_n(F)M \subset C(P_iM) \cup C(M)$ or equivalently $P_iB_n(F) \subset B_n(F)P_iB'_n(F) \cup B_n(F)B'_n(F)$ where $B'_n(F) := MB_n(F)M^{-1}$.
Let us denote as G_i the subgroup of $GL_n(F)$ consisting of the block diagonal matrices of the form

$$\begin{pmatrix} I_{i-1} & & \\ & M & \\ & & I_{n-(i+1)} \end{pmatrix}$$

for $M \in GL_2(F)$.
This group appear in L∃∀N as `Gi hii : Subgroup (GL n F)` where `hii : i < succ i` is a proof that an implicit variable `i : n` is less than its successor. The condition `hii` is the natural way to translate in L∃∀N the condition $i < n$.
The group G_i is clearly isomorphic to $GL_2(F)$ and $P_i = \begin{pmatrix} I_{i-1} & & \\ & \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} & \\ & & I_{n-i-1} \end{pmatrix} \in G_i$. Additionally, some simple but very annoying to formalize computations show that $G_iB_n(F) \subset B_n(F)G_i$

and hence $P_i B_n(F) \subset B_n(F) G_i$ so we only need to prove that :

$$G_i \subset (B_n(F) \cap G_i)\ P_i\ (B'_n(F) \cap G_i) \bigcup (B_n(F) \cap G_i)\ (B'_n(F) \cap G).$$

Let ψ denote the natural isomorphism from $GL_2(F)$ to G_i. In Lean we formalized it as `GL2toG2ij hij : GL (Fin 2) F ≃* (Gi hii : Subgroup (GL n F)`. To do so we first observe that that $\psi^{-1}(B_n(F) \cap G_i)$ is the group $B_2(F)$ of 2×2 triangular matrices and that $B'_2(F) := \psi^{-1}(B'_n(F) \cap G_i)$ is either the upper or the lower triangular subgroup of $GL_2(F)$ depending on whether $\pi(M)(i) < \pi(M)(i+1)$ or the opposite.
In the first case we need to show that

$$GL_2(F) = B_2(F) \cup B_2(F) s B_2(F)$$

with $s = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. It is easy to verify using 2.2.2 that $B_2(F) \cup B_2(F) s B_2(F)$ is a subgroup of $GL_2(F)$. Additionally, it contains both $B_2(F)$ and $N_2(F) = \{1, s\}$ so as shown in 1 it is equal to $GL_2(F)$.
In the second case we have to show that $GL_2(F) = B_2(F) B_2^-(F) \cup B_2(F) s B_2^-(F)$ where $B_2^-(F)$ denotes the lower triangular group of $GL_2(F)$. This result is immediate follows from the precedent one by remarking that $B^-(F) = s B_2(F) s$.
Hence we have the following statement in Lean :

```
theorem GLsubset_union_doubleCosets : ∀ (M Ma : MonomialGroup), ∀ s ∈ SwapSet,
 toPermHom Ma = s ->
  {(Ma.val * b.val * M.val : GL n F) | b : (TriangularGroup : Subgroup (GL n F))} ⊆
      C TriangularGroup (Ma * M).val∪ C TriangularGroup M.val
```

Implementation.

Formalizing the above proof of 2.3.7 allows us to build an object `GeneralLinearGroup_BNpiTriplet` of type `BNpiTriplet (GL n F) (Perm n)` as follows :

```
/--The `BNMphiQuadruplet`  structure over `GL n F`.-/
noncomputable
def GeneralLinearGroup_BNpiTriplet : BNpiTriplet (GL n F) (Perm n) where
  B := TriangularGroup
  N := MonomialGroup
  S := SwapSet
  pi := toPermHom
  BN_gen_Group := TriangularMonomial_gen_GL
  pi_surj := toPermHom_surj
  ker_pi_eq_B_subgroupOf_N := ker_toPermHom_eq_Diagonal
  self_inverse_of_mem_S := self_inverse_of_mem_SwapSet
  top_eq_S_closure := top_eq_swap_closure
  left_right_mul_B_neq_B_of_map_S :=
    fun _ h => left_right_mul_Triangular_neq_Triangular_of_map_swap h
  subset_union_doubleCosets := GLsubset_union_doubleCosets'
```

□

3 Some results on groups with a BN-pair

This section present some of the main results that have been formalized on groups with $BN\pi$-triplet during this internship. However, as it will be discussed in 3.2.2 note all of the results necessary to prove the simplicity theorem 4.1.1 have been proven due to the need for a more advanced theory of

Coxeter groups than the one currently available in `mathlib` . Thus, most of the results presented in this part will not be proven.

This section first starts by presenting a way to endow with a $BN\pi$-triplet certain quotients of groups with $BN\pi$-triplet. Then after an introduction to the theory of Coxeter groups, we will the link of such groups with groups with $BN\pi$-triplet. Finally, a few results concerning the subgroups of G that contains B or one of its conjugates will be exposed.

3.1 Quotient $BN\pi$-triplet

Let Z be a normal subgroup of G contained in B. Let us denotes as $\hat{G}$ the group G/Z, as $\hat{B}$ the groups B/Z, as $\hat{N}$ the group $N/(N \cap Z)$ and as $\hat{\pi}$ the lift of π through the natural homomorphism $N \to (N \cap Z)$. Then we have :

Definition 3.1.1 (Quotient $BN\pi$-triplet)**.** $\left(\hat{B}, \hat{N}, \hat{\pi}\right)$ forms a $BN\pi$-triplet over $\hat{G}$ called the quotient $BN\pi$-triplet of (B, N, π) by Z.

If proving this theorem does not require much efforts[8], formalizing it is a bit more arduous. That is why we introduced the structure of image $BN\pi$-triplet :

Definition 3.1.2 (Image $BN\pi$-triplet)**.** Let G' be a group, p a surjective homomorphism from G to G' and $\pi' : p(N) \to W$ a group homomorphism. If $\ker p \leq B$ and $\pi' \circ p = \pi$ then $(p(B), p(N), pi')$ forms a $BN\pi$-triplet over G' called the image $BN\pi$-triplet of (B, N, π) by p.

It appears clearly from the definitions above that if $Z \leq B$ is normal in G then the quotient $BN\pi$-triplet of (B, N, π) if nothing more than the image $BN\pi$-triplet of (B, N, π) by the natural homomorphism $G \to G/Z$. Thus, we can use image $BN\pi$-triplet to formalize quotient $BN\pi$-triplet and that is how we proceeded :

```
variable (G' : Type*) [Group G']

def ImageBNpiTriplet {p : G →* G'} (p_surj: Surjective p) (ker_p_le_B : p.ker ≤ TS.B)
    {p' : TS.N →* TS.N.map p} (p'_rest_p : ∀ n, p n.val = (p' n).val) {pi': TS.N.map p →* W}
      (p_comm_pi' : TS.pi = pi'.comp p') : BNpiTriplet G' W where
  B := TS.B.map p
  N := TS.N.map p
  pi := pi'
  S := TS.S
  ...

noncomputable
def QuotientBNpiTriplet (Z : Subgroup G) [Normal Z] (ZleB : Z ≤ TS.B): BNpiTriplet (G ⧸ Z) W :=
    ImageBNpiTriplet TS (QuotientGroup.mk'_surjective Z) ...
```

Code commentary 3.1.3.

- The definition of `ImageBNpiTriplet` may seem verbose and ask for more argument than necessary. It is true but it makes the formalization easier.
- The term `QuotientGroup.mk'_surjective Z` of type `Function.Surjective ⇑(QuotientGroup.mk' Z)` used in the definition of `QuotientBNpiTriplet` is a proof that `QuotientGroup.mk' Z: G → G ⧸ Z` , the natural homomorphism from G to G/Z is surjective.

[8] and we incite the reader to do so

Exemple. In the case of $G = GL_n(F)$, the center Z of $GL_n(F)$ is normal in $GL_n(F)$ and is equal the group of nonzero scalar matrices and as such $Z \leq B_n(F)$. Therefore, the quotient $BN\pi$-triplet of $(B_n(F), N_n(F), \pi)$ by Z forms a $BN\pi$-triplet over $PGL_n(F) := GL_n(F)/Z$. One can find this fact formalized in `GLn.lean` [9] as :

```
noncomputable
def ProjectiveGeneralLinearGroup_BNpiTriplet : BNpiTriplet (PGL n F) (Perm n) :=
  QuotientBNpiTriplet (GeneralLinearGroup_BNpiTriplet n F) (Subgroup.center (GL n F)) ...
```

3.2 Relations with Coxeter groups

Coxeter groups are an important family of abstract groups which have been intensively studied since their introduction by H. S. M. Coxeter in 1934 in [1]. Lots of important results have been discovered concerning these groups (see §1 of [4], for instance), results which can be used to get more insights on the structure of groups with $BN\pi$-triplet. Indeed, as we will see in 3.2.2 if the group G has $BN\pi$-triplet with $\pi : G \to W$ then W is a Coxeter group and this additional property of W is especially useful to highlight even more the link between groups with $BN\pi$-triplet and double cosets. However, this link will not be really put forward in this report for reasons explained in 3.2.2 but can be found in [4]. This subsection first presents the definition and some basic properties as well as the L∃∀N implementation of Coxeter groups, and then the link with groups with $BN\pi$-triplet is discussed.

3.2.1 Introduction to Coxeter theory

We present here a few basic definition of the theory of Coxeter groups. These elements were already implemented in `mathlib` were not coded by the author during its the internship. They are available here.

Let W be a group and S a subset of elements of order 2 of W. For all two elements $s, s' \in S$, we define $m(s, s') \in \mathbb{N}$ to be the order of the product ss' if it is finite and 0 otherwise.

Definition 3.2.1 (Coxeter system)**.** We say that the couple (W, S) is a *Coxeter system* if the generating set S alongside with the relations $(ss')^{m(s,s')} = 1$ for all $s, s' \in S$, forms a presentation of W.

When (W, S) is a Coxeter system, the elements of S are called simple (reflections) and by an abuse of language, W is called a *Coxeter group*.

Exemple.

1. For all natural number n, $\mathfrak{S}_n$, the symmetric group over $\{1, \dots, n\}$ with the set of transposition $(i, i+1)$ for $1 \leq i \leq n$ forms a Coxeter system.
2. Let m be a natural number, let s denote the symmetry of the complex plane over the real line and r the rotation of angle $\frac{2i\pi}{m}$. The dihedral group $D_{2m} := \langle r, s \rangle$ alongside with $S = \{r, s\}$ is a Coxeter System.

The Lean implementation `CoxeterSystem` of a Coxeter system does not make a direct use of the above definition but is in fact based upon the following concept of a Coxeter matrix :

[9] The definition of `PGL` is not present in Mathlib and was coded for this project, you can find it defined in the `GLn.lean` file

Definition 3.2.2 (Coxeter matrix)**.** Let A be a set. A square symmetric matrix $M = (m_{a,a'})_{a,a' \in A}$ with coefficient in $\mathbb{N}$ is a *Coxeter matrix* of type A if it satisfies
- $\forall a \in A, m_{a,a} = 1$
- $\forall a, a' \in I, a \neq a' \Rightarrow m_{a,a'} \neq 1$

If (W, S) is a Coxeter system, then the matrix $M = (m(s, s'))_{s,s' \in S}$ is a Coxeter matrix of type S called the Matrix of (W, S).

Proof. Suppose that (W, S) is a Coxeter system. It is immediate that $m(s, s) = 1$, and $m(s, s') \geq 2$ for all $s, s' in S$. Additionally, $s.s' = (s'.s)^{-1}$ and therefore $(m(s, s'))_{s,s' \in S}$ is symmetric. □

The Lean implementation of Coxeter matrices is pretty straightforward and does not need any commentary (the `by decide` s can be ignored):

```
structure CoxeterMatrix (A : Type*) where
  /-- The underlying matrix of the Coxeter matrix. -/
  M : Matrix A A ℕ
  isSymm : M.IsSymm := by decide
  diagonal i : M i i = 1 := by decide
  off_diagonal i i' : i ≠ i' → M i i' ≠ 1 := by decide
```

Definition 3.2.3 (Group of a Coxeter matrix)**.** The *group* $G(M)$ *of a Coxeter matrix* M of type A is the presented group with generator A and relations $(a.a')^{m(a,a')} = 1$ for all $a, a' \in A$.

In Lean, if `M` has type `CoxeterMatrix A` for a certain type `A` its group is denoted as `M.Group`. Additionally, the generators of the group `M.Group` can be accessed through the function `M.simple : A → M.Group` mapping each element of type `A` to its image in the group of `M`. This group also verifies the following properties :

Property 3.2.4.
1. If M is a Coxeter matrix of type A, for A a set, then $(G(M), A)$ is a Coxeter system.
2. (W, S) is a Coxeter system if and only if there exists a set A and a Coxeter matrix M of type A, and an isomorphism $f : G(M) \to W$ sending bijectively A to S.

Proof. The proof of 1. follows from the definition of $G(M)$. If (W, S) is a Coxeter system then its Coxeter matrix M fits and we have $W \simeq G(M)$ by definition of a Coxeter system. The converse implication is immediate from the definition of a Coxeter matrix.

□

Let M be a matrix of type A and an isomorphism $f : G(M) \to W$ sending A to S. Then the latter proposition tells us that the couple (M, f) characterizes the Coxeter system structure on (W, S). This fact serves as the definition of a Coxeter system in Lean. Indeed, given a type `A`, an element `M` of type `CoxeterMatrix A` and a type `W` endowed with a group structure, an element of type `CoxeterSystem M W` consists in the data of a group isomorphism between `W` and `M.Group`.

```
structure CoxeterSystem {A : Type*} (M : CoxeterMatrix A) (W : Type*)
    [Group W] where
  /-- The isomorphism between `W` and the Coxeter group associated to `M`. -/
  mulEquiv : W ≃* M.Group
```

3.2.2 BN-pairs and Coxeter theory

This subsection details the relations between Coxeter groups and groups with BN-pair, they all derives from the following theorem :

Theorem 3.2.5. (W, S) is a Coxeter system.

The proof of the above theorem can be found in [4] §2 n°4 and is a consequence of the decomposition of G in double cosets (see theorem 2.2.6). However, like many of the results that will be described from now on, it makes a heavy use of parts of the theory of Coxeter groups that are not yet in implemented in L∃∀N .

As proving all the results needed might necessitate a whole other internship, it was decided to leave the task of formalizing them for future works and focus the goal of proving the simplicity theorem (theorem 4.1.1). The L∃∀N code is therefore filled with holes. However, every proof that did not make a direct use of the Coxeter group theory has been formalized, and the full list of the necessary results and where they are needed in the code is available on github.

Additionally, proving the results mathematically in this report would not be of great interest, as it would either necessitate a large amount on the Coxeter group theory to be exposed -which was not the object of this internship- or that the proofs given would be filled with references to other works. That is why what follows will mostly consists in definitions and statements of theorems with the notable exception of theorem 4.1.1.

The first property that necessitate advanced Coxeter group theory to be proven is the very fact that (W, S) is a Coxeter system. We "formalized" it with the use of `sorry` [10]. The term `sorry` produce an unknown term of the expected type. It is often used as a placeholder for results waiting to be formalized.

```
def BNpiTriplet.coxeterMatrix  (TS : BNpiTriplet G W) : CoxeterMatrix TS.S where
  M := Matrix.of fun s s' => orderOf (s*s' : W)
  ...

def  BNpiTriplet.cs : CoxeterSystem TS.coxeterMatrix W where
  mulEquiv :=sorry

abbrev BNpiTriplet.length (TS : BNpiTriplet G W) := TS.cs.length

abbrev BNpiTriplet.simple (TS : BNpiTriplet G W) := TS.cs.simple

lemma simple_id_mem_S : TS.simple = fun s => s.val := sorry
```

Using the above theorem[11] we have the following:

Lemma 3.2.6. The map sending an element $w \in W$ to its double coset $C'(x)$ is bijective.

This appears in L∃∀N as `Weyl_doubleCosets_equiv : W ≃ Set.range TS.C'` and makes use of the lemma `C'_inj' : Function.Injective TS.C'` which might of interest for some readers planning to build over the code presented here.

[10] The reader unfamiliar with the implementation of Coxeter groups in L∃∀N should not pay much attention to the following lines and only keep in mind that used `sorry` to provide a Coxeter group structure over `W` .

[11] The proof only really makes use of the notion of *length* of an element, which is defined for every group generated by elements of order 2, but is only implemented in L∃∀N in the Coxeter groups. However, introducing the whole theory of length for only one proof would require important efforts for both the reader and he author for very limited interest. For more information around this subject we once again send to [4].

3.3 Subgroups of G

This subsection present a few results around subgroups of G containing B or conjugates of B.

Let X be a subset of S. We denote as W_X the group generated by X. In L∃∀N we represent X by an object `X` of type `Set TS.S` and W_X is the term `Subgroup.closure X : Subgroup W`. We have :

Theorem 3.3.1. The set $G_X := \bigcup_{x \in W_x} C'(x)$ is a subgroup of W that contains B.

In L∃∀N we the groups G_X are formalized through :

```
def DoubleCoset_of_Set ( X : Set TS.S) : Subgroup G where
  carrier := ⋃ x ∈ closure (X : Set W), TS.C' x
  ...
```

The importance of these groups appears in the following theorem :

Theorem 3.3.2. The map

$$\begin{aligned} \mathcal{P}(S) &\longrightarrow \{\text{Subgroups of } G \text{ containing } B\} \\ X &\longmapsto G_X. \end{aligned}$$

is bijective.

The above bijection can be accessed in lean *via* `DoubleCoset_of_Set_bij : Set TS.S ≃ {H :Subgroup G // TS.B ≤ H}`. This is a great example of the deep connection between the structure of the group G and the double cosets C' over B.

Now we introduce the definition of parabolic subgroups as below :

Definition 3.3.3. A subgroup P of G is said *parabolic* if it contains a conjugate of B.

The Parabolic subgroups are formalized as :

```
def Subgroup.Parabolic (P : Subgroup G) : Prop := ∃g, ConjugatedSubgroup TS.B g ≤ P
```

where `ConjugatedSubgroup TS.B g : Subgroup G` is the subgroup of `G` with `carrier := {w} * B.carrier * {w⁻¹}`.

Before, stating the only result on parabolic groups of this report we will recall the definition of the normalizer of a subgroup.

Definition 3.3.4. The normalizer of a subgroup H of G is the largest subgroup of G in which H is normal.

The normalizer of an object `H : Subtype G` is denoted `H.normalizer : Subgroup G` in L∃∀N . We can now state the following theorem :

Theorem 3.3.5. A parabolic subgroup P is its own normalizer.

This theorem has been formalized as `eq_normalizer_of_parabolic {P : Subgroup G} (h : P.Parabolic TS) : P = P.normalizer`. It will bu sed with the following lemma to prove 4.1.1. The latter is, moreover, more of a technical lemma to prove 4.1.1 than an actual result.

Lemma 3.3.6. Let $H \triangleleft G$. There exists $X \subset S$ such that $BH = G_X$ and every element of X commute with every element of $S \setminus X$.

4 The simplicity theorem

In this section we state and prove a theorem that can be used to show that groups with $BN\pi$-triplet are simple when applicable. Then we apply this theorem to case of the projective special linear group over a field.

4.1 The theorem

Let U be a normal subgroup of B. We define Z to as the group $Z := \bigcap_{g\in G} gBg^{-1}$, and we denote as G_1 the subgroup $G_1 := \bigsqcup_{g\in G} gUg^{-1}$ of G.

Theorem 4.1.1. If the following conditions are verified :
1. $B = UT$
2. The only perfect quotient group of U is the trivial group, or in other words for all strict normal subgroup V of U, $U/V \neq [U/V, U/V]$.
3. G_1 is perfect *i.e.* G_1 is equal to its derived subgroup $[G_1, G_1]$.
4. the Coxeter system (W, S) is irreducible *i.e.* there is no subset X of S such that every element of X commutes with every element of $S - X$.

then $G_1/(Z \cap G_1)$ is either trivial or simple noncommutative.

Corollary 1. If $U \cap Z = \{1\}$ then $Z \cap G_1$ is the center of G_1.

Proof. As both Z and U are normal in B, every element of Z commutes with every element of U, thus with every element of G_1. Then, the result follows from the above corollary. □

Proof of the theorem. Let us show that every subgroup H of G normalizing G_1 is either contained in Z or contains G_1.

We first show that G_1T is equal to G. To do so we need to observe that G_1T is subgroup of G that contains B so by 3.3.5 it is its own normalizer. Therefore, as N normalizes G_1 and T, and so G_1T, we have $N \leq G_1T$. Thus, as B and N generates G we have $G = G_1T$.

Lemma 4.1.2. Let us set $G' := G_1H$, $B' := B \cap G'$, $N' := N \cap G'$ and $\pi' := \pi_{|N'}$. Then (B', N', π') forms a $BN\pi$-triplet over G'.

Proof. First and foremost, remark that $G'T = G$ because $G_1 \leq G'$ and therefore, $N'T = N$.
1. The relations $B = UT = TU$ allows to rewrite theorem 2.2.6 as $G = UNU$. Then U being a subgroup G', we have $G' = UN'U$ and therefore $G' = \langle B'_n(F), N'\rangle$ because $U \leq B'$.
2. The kernel of $\pi' = \pi_{|N'}$ is equal to $\ker pi \cap N' = B \cap N \cap N' = B' \cap N'$.
3. The surjectivity of π' is direct consequence of the surjectivity of π and the relations $N = N'T$ and $\ker \pi = T$.
4. Immediate.
5. Let us denote as W' the group $W' := \pi(N) = \Im\pi'$. For all $w \in w'$, we have $Bw'B = Bw'B'$ because $B = B'T$ so $BwB \cap G' = B'w'B'$. The result is then immediate.
6. Immediate.

□

H is normal in G_1H thus, there exists $X \subset S$ such that $B'H = B'W_XB'$ and every element of X commute with every element of $S - X$ (see 3.3.5). However, the Coxeter system (W, S) is irreducible so $X = \emptyset$ or $X = S$.

First case : $X = \emptyset$ *i.e.* $B'H = B'$ so $H \leq B' \leq B$. Additionally, for all $g \in G$ we have $g_1 \in G_1$ and $t \in T$ such that $g = g_1 t$, so H being normal in G_1, $H \subset g_1 B g_1^{-1}$ and thus $H \subset gBg^{-1}$ as $T \leq B$. Therefore, $H \subset Z$.

In particular, G_1 normalizes itself so $G_1/(G_1 \cap Z)$ is trivial.

Second case $X = S$: *i.e.* $B'H = G'$ and so $G = G'T = B'HT = HB'T = HB$.

As $U \triangleleft B$ all conjugates of U are of the form hUh^{-1} with $h \in H$ which means they are all subgroups of UH. Thus $G_1 \subset UH$ and

$$U/(U \cap H) \simeq UH/H = G_1 H/H \simeq G_1/(G_1 \cap H).$$

The group G_1 being perfect, its quotient $G_1/(G_1 \cap H)$ also is. Thus, $U/(U \cap H)$ also is perfect and as such, it is by hypothesis the trivial group. Therefore, $G_1 = G_1 \cap H$ *i.e.* $G_1 \leq H$.

Thus $G_1/(G_1 \cap Z)$ is simple and as G_1 is perfect $G_1/(G_1 \cap Z)$ also is and therefore cannot be commutative. □

The above theorem appears in L∃∀N as :

```
def BNpiTriplet_SimplictyTheorem'  {U : Subgroup G} (hU : U ≤ TS.B) (hU' : (U.subgroupOf TS.B).Normal)
 (hUT : ⊤ = mul_Subgroup' hU' (TS.T.subgroupOf TS.B)) (RU : propR U) (hirred : TS.cs.irreducible)
  (perfectG1 : { (T : Subgroup (ConjSubgroup U)), T} = (⊤: Subgroup (ConjSubgroup U)))
   [Nontrivial (ConjSubgroup U/ ((⨅ g, ConjugatedSubgroup TS.B g).subgroupOf (ConjSubgroup U))) ] :
    IsSimpleGroup (ConjSubgroup U/ ((⨅ g, ConjugatedSubgroup TS.B g).subgroupOf (ConjSubgroup U)))
```

Remark. The $BN\pi$-triplet defined in the lemma 4.1.2 has been formalized in L∃∀N as :

```
variable {U : Subgroup G} (hU : U ≤ TS.B) (hU' : (U.subgroupOf TS.B).Normal)
variable (hUT : ⊤ = mul_Subgroup' hU' (TS.T.subgroupOf TS.B))
variable {H : Subgroup G} (hH : ConjSubgroup U ≤ H.normalizer)

def SubBNpiTriplet : BNpiTriplet (Subgroup_mul  hH) W where
  B := TS.B.subgroupOf (Subgroup_mul hH)
  N := TS.N.subgroupOf (Subgroup_mul hH)
  pi := TS.pi.comp (N'toN TS)
  S := TS.S
  ...
```

4.2 Application : the projective special linear group

We denote as $SL_n(F)$ the *special linear group* over F i.e. the kernel of the group homomorphism $\det : GL_n(F) \to R^*$ and as $PSL_n(F)$ the quotient of $SL_n(F)$ by its center, which is named the *projective special linear group*. These groups are (surprisingly) denoted `SL n F` and `PSL n F` in L∃∀N . We can apply the theorem 4.1.1 alongside with its corollary 1 to show :

Theorem 4.2.1. If $n \geq 3$ or $n = 2$ and $|F| \geq 4$ then the group $PSL_n(F)$ is simple.

Proof. Let U denotes the strictly upper triangular group *i.e.* the subgroup of $GL_n(F)$ made of the triangular matrices whose every diagonal entry are 1. It is a normal subgroup of the triangular matrices. We set $Z := \bigcap_{M \in GL_n(F)} MB_n(F)M^{-1}$ and $G_1 := \bigsqcup_{M \in GL_n(F)} MUM^{-1}$. First, let us show that $G_1 = SL_n(F)$. Every element of G_1 is blatantly of determinant 1 so now let us show that $SL_n(F) \leq G_1$ by using the following lemma :

Lemma 4.2.2. $SL_n(F)$ is generated by the transvection matrices.

Proof. By the lemma 2.3.10, we only need to prove that diagonal matrices of determinant 1 can be written as products of tranvection matrices. This is easily proven by operations over the lines and colums. □

The transvection matrices being either strictly upper triangular matrices or conjugates of strictly upper triangular matrices by transposition matrices as seen in the point 1 of the proof of 2.3.7, by the above lemma, $G_1 = Sl_n(F)$. Now let us prove that the groups defined above verify the hypothesis of 4.1.1.

1. A simple calculation shows that the diagonal of a product of triangular matrices is the product of their diagonal. Hence, for all triangular matrix T, if we denote as D the diagonal the matrix of its diagonal coefficient, which is invertible so in $T_n(F)$ then the matrix $T' := TD^{-1}$ is strictly upper triangular and therefore $T = T'D \in UT$.
2. By computing the iterated derived subgroups of U one can show that U is solvable, and therefore verifies the second hypothesis of theorem 4.1.1.
3. Let us show that $G_1 = SL_n(F)$ is perfect under the conditions of the theorem by proving that transvection matrices can be written as commutators and using the lemma 2.3.10.
   Let i, j be distinct elements of n and c be an element of F.
   If $n \geq 3$ then we can find an index $k \in n$ distinct from i and j and $T_{i,j}(c) = [T_{i,k}(c), T_{k,j}(1)]$.
   Otherwise, if $n = 2$ then, for all $x, y \in F$
   $$\left[\begin{pmatrix} x & 0 \\ 0 & x \end{pmatrix}, \begin{pmatrix} 1 & y \\ 0 & 1 \end{pmatrix}\right] = \begin{pmatrix} 1 & (x^2-1)y \\ 0 & 1 \end{pmatrix}.$$
   The equation $c = \left(x^2 - 1\right) y$ admits a solution if and only if there exists $x \in F$ that is not a root of the polynomial $X^2 - 1$ *i.e.* if and only if $|F| \geq 4$.
4. Let $S_n := \{(i, i+1) | 1 \leq i \leq n-1\}$ and let us show that $(\mathfrak{S}_n, S_n)$ is irreducible by induction over n. The cases $n = 0, 1$ are trivial, so suppose $n \geq 21$ and $(\mathfrak{S}_{n-1}, S_{n-1})$ irreducible.
   Let X be a subset of S_n such that every element of X commutes with every element of $S_n - X$. Set $X' := X \cap S_{n-1}$, it is a subset of S_{n-1} that commutes with every element of its complement in S_{n-1}, thus by hypothesis it is either S_{n-1} or $\emptyset$. If $X' = S_{n-1}$ then $(n-1, n)$ does not commute with $(n-2, n-1) \in X'$ so $(n-1n) \in X$ and $X = S_n$. Otherwise, if $X' = \emptyset$ then as $(n-2, n-1)$ belongs to $S_n - X$ so does $(n-1, n)$ and hence $X = \emptyset$.

Therefore, by theorem 4.1.1 the group $G_1/(Z \cap G_1) = SL_n(F)/(Z \cap SL_n(F))$ is simple or trivial. Now let us verify that Z is the center of $SL_n(F)$, by using corollary 1. We see by conjugating by permutation matrices that very matrix in Z is diagonal. The only strictly upper triangular matrix that is diagonal is the identity matrix. Hence, $Z \cap U = \{I_n\}$ and Z is the center of $SL_n(F)$. Therefore, $PSL_n(F)$ is simple because it is not trivial : for example the image of $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \in SL_n(F)$ in $PSL_n(F)$ is nontrivial. □

Remark. This property has not been fully formalized yet. Especially, formalizing the fact that the hypothesis 2 holds will be challenging.

Finally, we end this report by proving that the groups $SL_n(F)$ and $PSL_n(F)$ are groups with $BN\pi$-triplet.

Proposition 4.2.3. *$SL_n(F)$ and $PSL_n(F)$ are endowed with $BN\pi$-triplets.*

Proof. First, the lemma 4.1.2 applied in the cases of $H = \{1\}$.tells us that $\left(B_n(F) \cap SL_n(F), \tilde{N}_n(F), \pi\right)$ is $BN\pi$-triplet over $SL_n(F)$.

Then, one can verify that the center Z of $SL_n(F)$ is the group of scalar matrices of determinant 1, thus is a subgroup of $B_n(F) \cap SL_n(F)$. Therefore, we can endow $PSL_n(F)$ with the quotient $BN\pi$-triplet of $(B_n(F) \cap SL_n(F), N_n(F), \pi)$ by Z (see 3.1.1 for the definition of quotient $BN\pi$-triplet). □

Implementation.

```
def SpecialLinearGroup_BNpiTriplet : BNpiTriplet (Matrix.SpecialLinearGroup n F) (Equiv.Perm n) :=
  SubBNpiTriplet (GeneralLinearGroup_BNpiTriplet n F) ...

def ProjectiveSpecialLinearGroup_BNpiTriplet :
  BNpiTriplet (Matrix.ProjectiveSpecialLinearGroup n F) (Equiv.Perm n) :=
    QuotientBNpiTriplet SpecialLinearGroup_BNpiTriplet ...
```

5 Conclusion

During this internship we successfully formalized in L∃∀N the notion of groups of Lie type using BN-pairs. This allowed formalize quantities of results about groups with BN-pair. To do so, we introduced the new concept $BN\pi$-triplets. We also provided a concrete construction of the BN-pair over $GL_n(F)$, which allowed us to endow the groups $PGL_n(F)$, $SL_n(F)$ and $PSL_n(F)$ with such structure.

We additionally created a framework which can be used to demonstrate the simplicity of simple groups with BN-pairs and started applying it to $PSL_n(F)$. However, the above mentioned framework is currently relying on unproven theorem and additional work in the field of Coxeter groups theory will be needed to add it to `mathlib`. All the necessary results can be found in §1 of [4] and their list in available on github.

Nonetheless, once these obstacles have been removed the code produced during this internship will be able to be used a foundation to the theory of groups of Lie type. Therefore, future works on the formalization of the simplicity of such groups could happen in short amounts of time bringing the objective of formalizing the CFSG closer than ever.

This internship has been a very enriching experience which allowed me broaden my mathematical culture and confirmed my interest in fundamental mathematics. Discovering and working with L∃∀N proved to be both mathematically enriching and extremely entertaining. I will probably continue to formalize mathematics on my free time. Additionally, meeting and talking with people with very different mathematical backgrounds which was really stimulating.

References

[1] H. S. M. Coxeter. "Discrete Groups Generated by Reflections". In: *The Annals of Mathematics* 35.3 (July 1934), p. 588. ISSN: 0003486X. DOI: 10.2307/1968753.

[2] J. Tits. "Algebraic and Abstract Simple Groups". In: *Annals of Mathematics* 80.2 (1964). Publisher: [Annals of Mathematics, Trustees of Princeton University on Behalf of the Annals of Mathematics, Mathematics Department, Princeton University], pp. 313–329. ISSN: 0003-486X. DOI: 10.2307/1970394. URL: https://www.jstor.org/stable/1970394.

[3] Roger W. (Roger William) Carter. *Finite groups of Lie type : conjugacy classes and complex characters*. In collab. with Internet Archive. Chichester [West Sussex] ; New York : Wiley, 1985. 570 pp. ISBN: 978-0-471-90554-7. URL: http://archive.org/details/finitegroupsofli0000cart.

[4] N. Bourbaki. *Groupes et algèbres de Lie: Chapitres 4, 5 et 6*. Google-Books-ID: 4NpGP9BbdygC. Springer Science & Business Media, May 26, 2007. 284 pp. ISBN: 978-3-540-34491-9.

[5] Robert A. Wilson. *The Finite Simple Groups*. Vol. 251. Graduate Texts in Mathematics. London: Springer, 2009. ISBN: 978-1-84800-987-5 978-1-84800-988-2. DOI: 10.1007/978-1-84800-988-2.

[6] Kevin Buzzard, Johan Commelin, and Patrick Massot. "Formalising perfectoid spaces". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. Jan. 20, 2020, pp. 299–312. DOI: 10.1145/3372885.3373830. arXiv: 1910.12320[cs,math].

[7] *leanprover-community/lean-liquid*. original-date: 2020-11-30T11:14:21Z. Aug. 12, 2024. URL: https://github.com/leanprover-community/lean-liquid.

[8] *Mathematics in Lean — Mathematics in Lean 0.1 documentation*. URL: https://leanprover-community.github.io/mathematics_in_lean/index.html.

[9] *Theorem Proving in Lean 4 - Theorem Proving in Lean 4*. URL: https://leanprover.github.io/theorem_proving_in_lean4/title_page.html (visited on 08/17/2024).