Small Scale Reflection, a guided tour

TYPES 2010 - Warsaw 13-16 October 2010

Assia Mahboubi

INRIA Microsoft Research Joint Centre (France)
INRIA Saclay – Île-de-France
École Polytechnique, Palaiseau

October 14th 2010

Finite group theory

In all what follows we mainly focus on the study of finite groups.
In all what follows we work in the Coq system.

Finite group theory

- Elements of a group combine thanks to the group law:

$$g \quad g^{-1} \quad e \quad g * h \quad g * h * g^{-1}$$

Finite group theory

- Elements of a group combine thanks to the group law:

$$g \quad g^{-1} \quad e \quad g * h \quad g * h * g^{-1}$$

- But groups themselves combine through various kinds of operators:

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Finite group theory

- Elements of a group combine thanks to the group law:

$$g \quad g^{-1} \quad e \quad g * h \quad g * h * g^{-1}$$

- But groups themselves combine through various kinds of operators:

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

- And most of the theory is developed forgetting about the points.

Decomposition

Theorem (Existence of prime decomposition)

A number always admit a decomposition into a product of prime numbers.

-

Theorem (Existence of composition series)

For any finite group G, there exists a sequence of subgroups:

$$\{1\} = G_0 \triangleleft G_1 \triangleleft \cdots \triangleleft G = G_n$$

such that for all k, G_{k+1}/G_k is simple.

-

Decomposition

Theorem (Existence of prime decomposition)

A number always admit a decomposition into a product of prime numbers.

-

Theorem (Existence of composition series)

For any finite group G, there exists a sequence of subgroups:

$$\{1\} = G_0 \lhd G_1 \lhd \cdots \lhd G = G_n$$

such that for all k, G_{k+1}/G_k is simple.

-

Such a sequence is called a composition series for G.
Such a quotient G_{k+1}/G_k is called a factor.

Uniqueness

Theorem (Prime decomposition uniqueness)

The decomposition of any number into a product of primes is unique up to permutations.

-

Theorem (Jordan-Hölder uniqueness)

For any group G, two composition series for G have the same length, and the same factors up to isomorphism and permutation.

-

Proving Jordan Hölder Theorem

By induction on the cardinal of G:

Proving Jordan Hölder Theorem

By induction on the cardinal of G:

- Base case: $\sharp G = 0$.
  - trivial (a group has at least one element: the neutral)

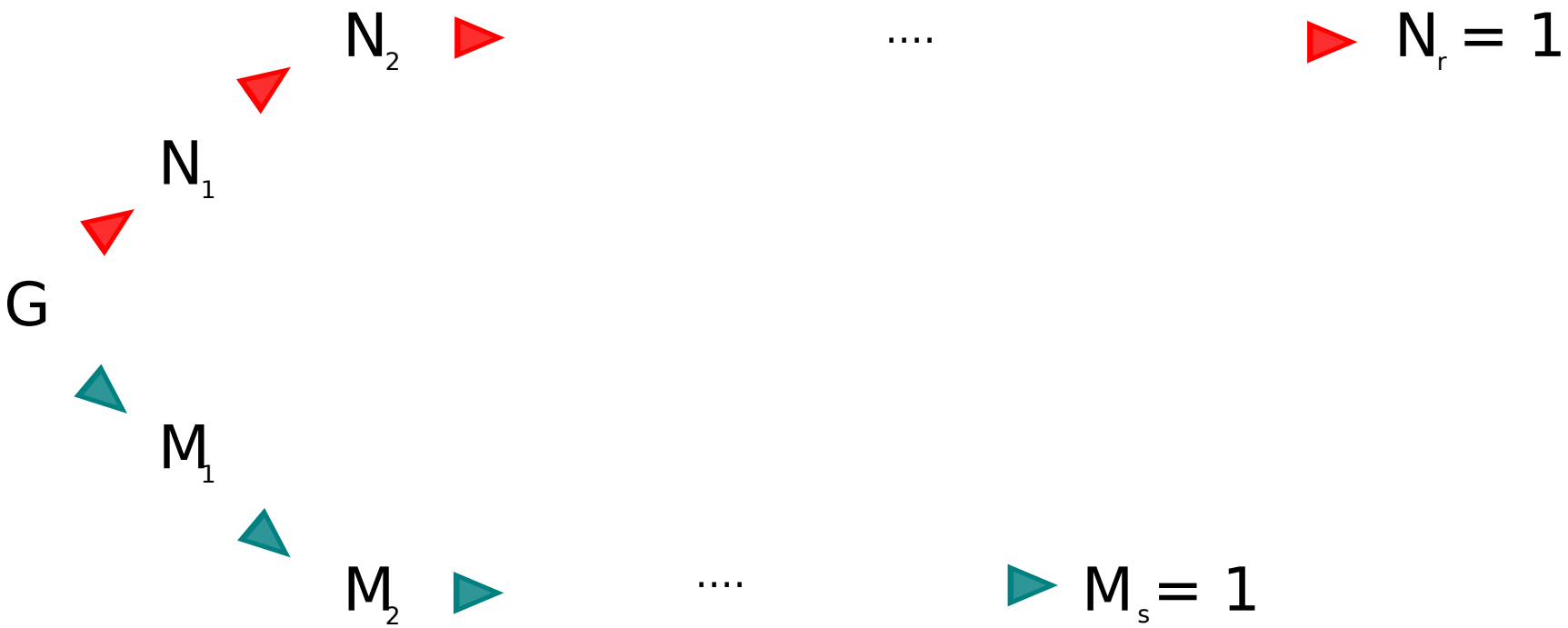
Proving Jordan Hölder Theorem

By induction on the cardinal of G:

- Base case: $\sharp G = 0$.
  - trivial (a group has at least one element: the neutral)
- Inductive case: $\sharp G > 0$:
  - If G has an empty series: then it is trivial and all its series are empty.
  - If G is simple: then all its series are trivial.
  - Else let (N_i) and (M_j) be two (non empty) composition series of the form.

Proving Jordan Hölder Theorem

By induction on the cardinal of G:



Proving Jordan Hölder Theorem

By induction on the cardinal of G:

Proving Jordan Hölder Theorem

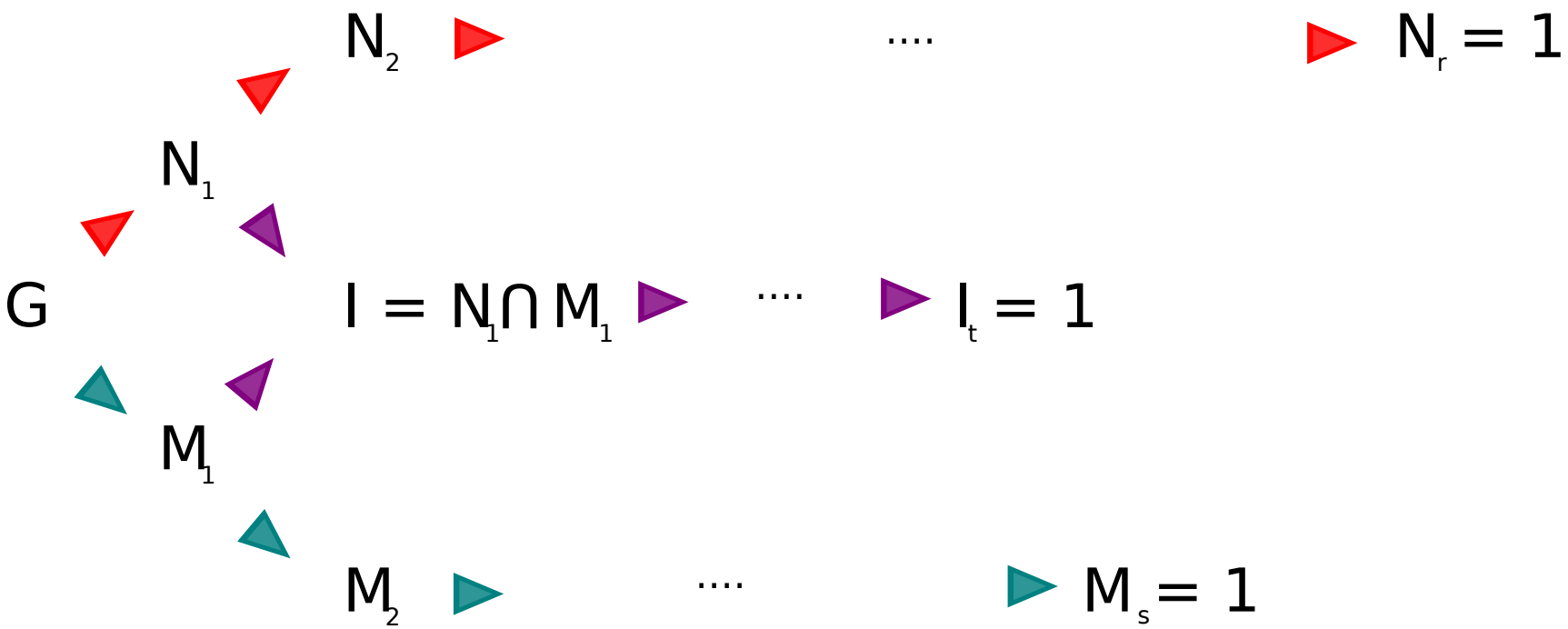
By induction on the cardinal of G:

Proving Jordan Hölder Theorem

By induction on the cardinal of G:

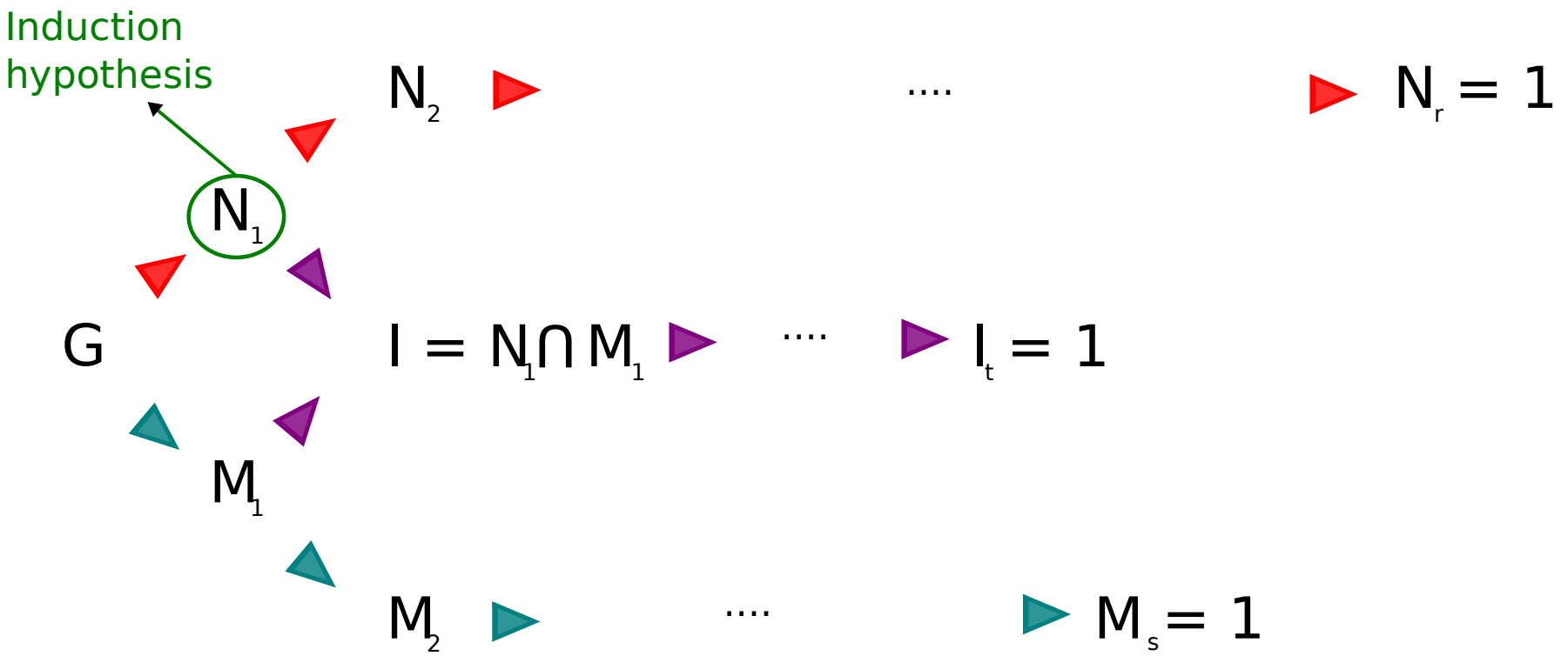
Proving Jordan Hölder Theorem

By induction on the cardinal of G:



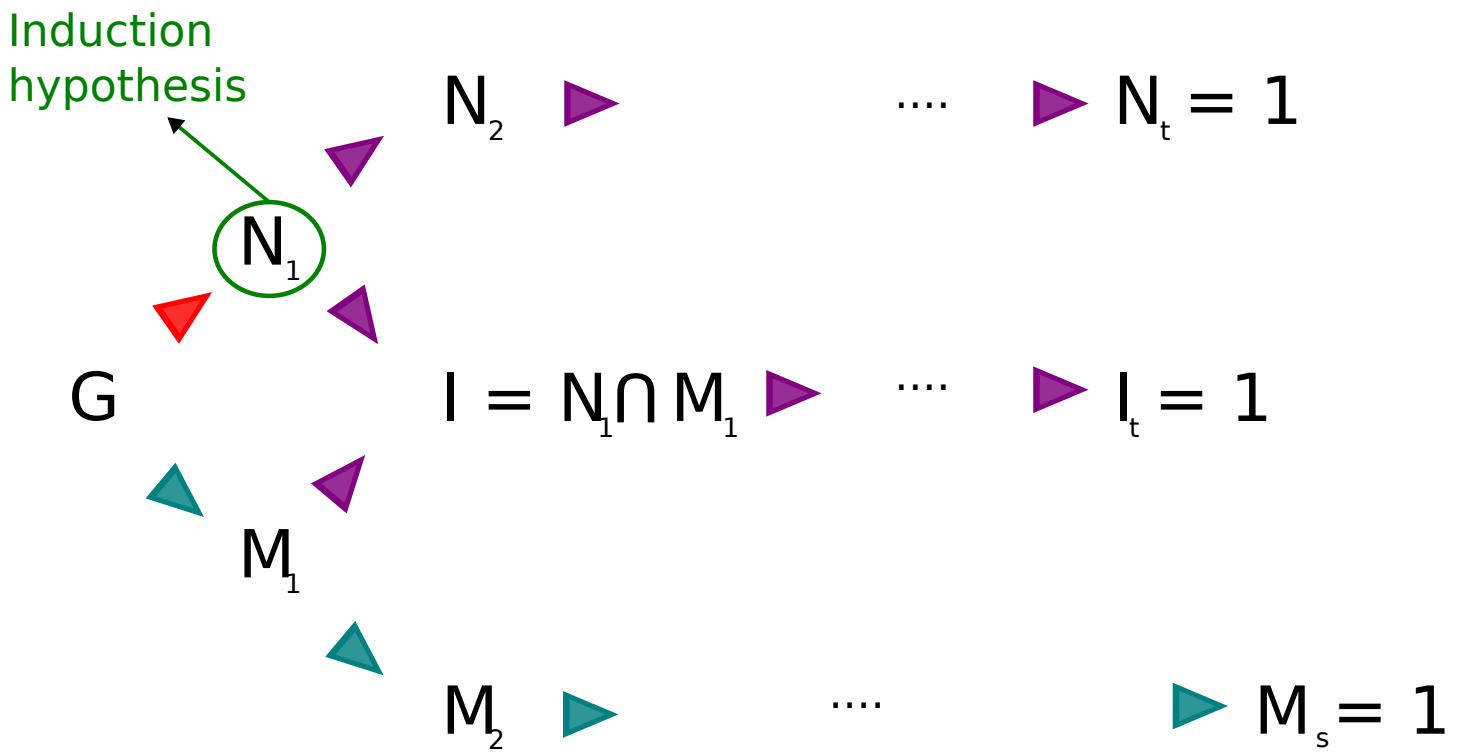
Proving Jordan Hölder Theorem

By induction on the cardinal of G:



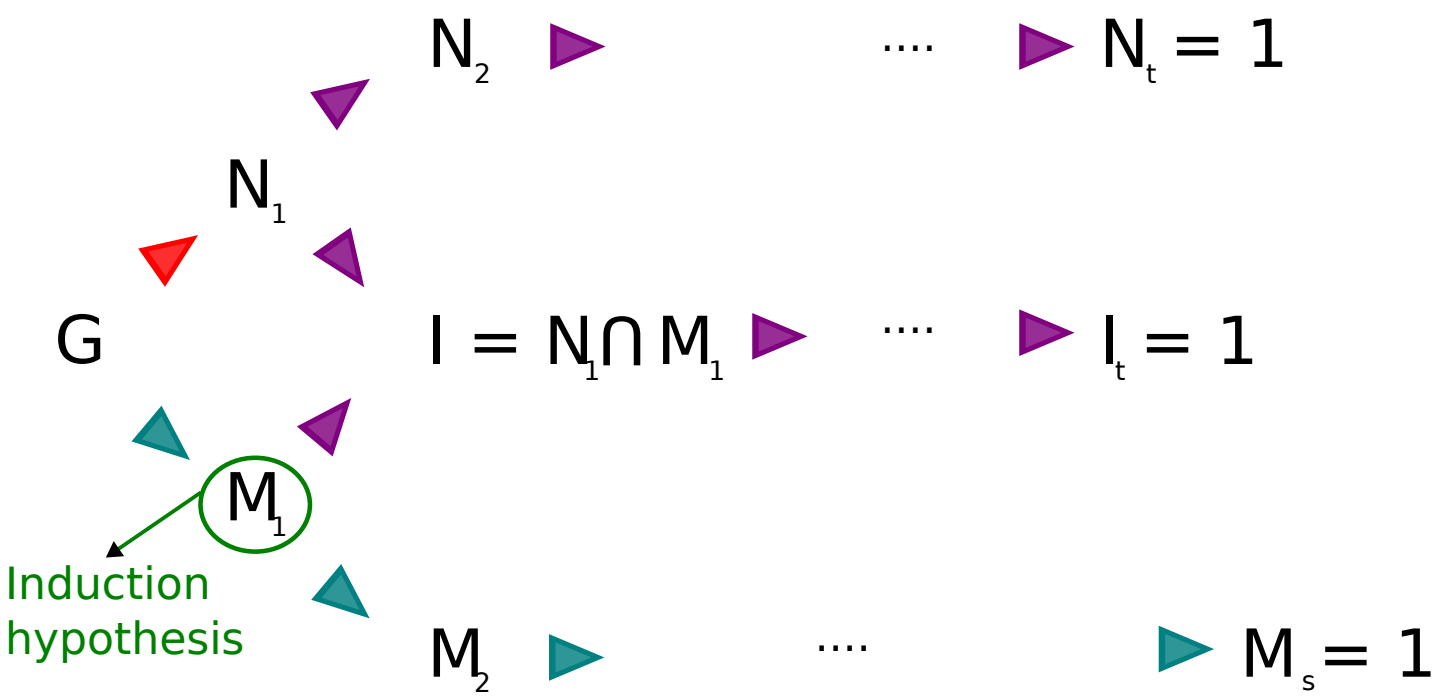
Proving Jordan Hölder Theorem

By induction on the cardinal of G:



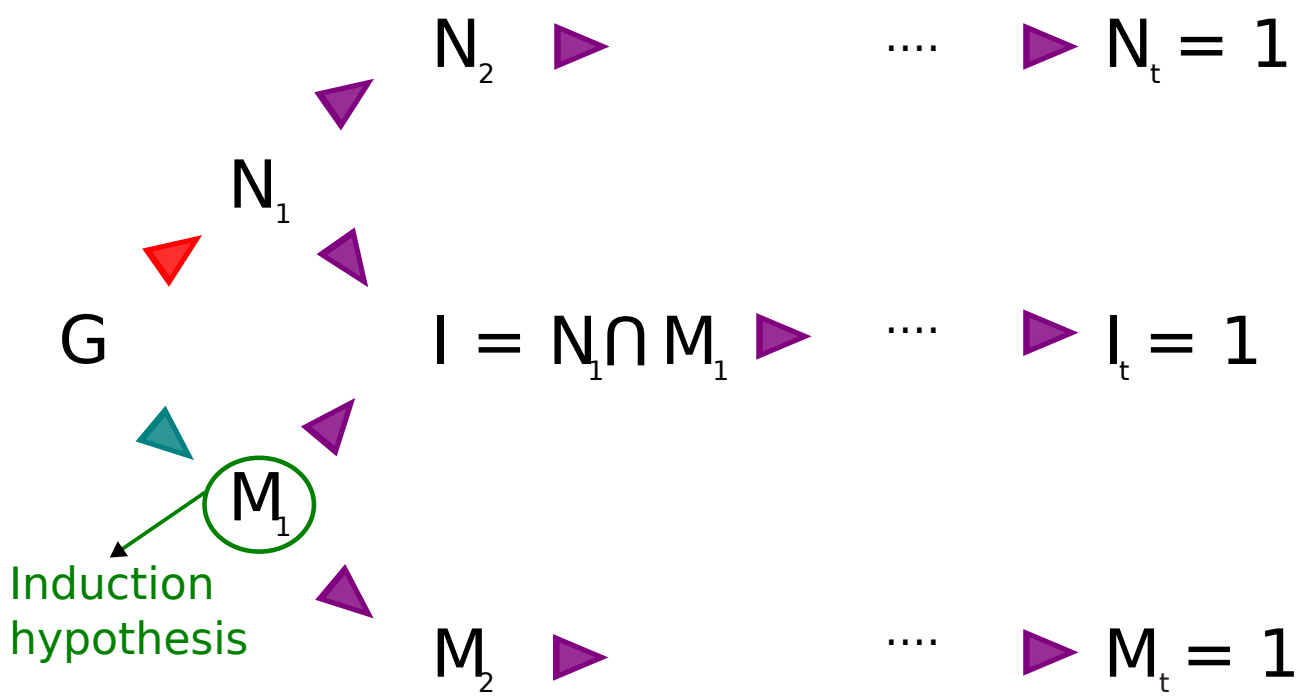
Proving Jordan Hölder Theorem

By induction on the cardinal of G:



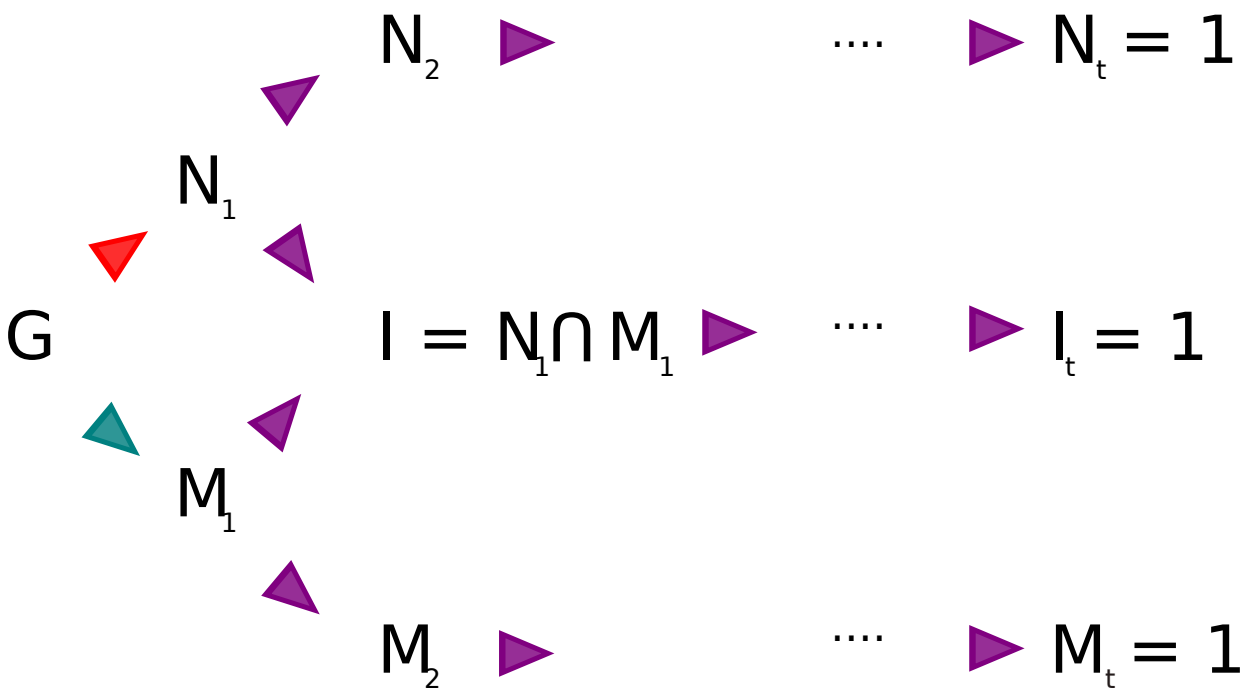
Proving Jordan Hölder Theorem

By induction on the cardinal of G:



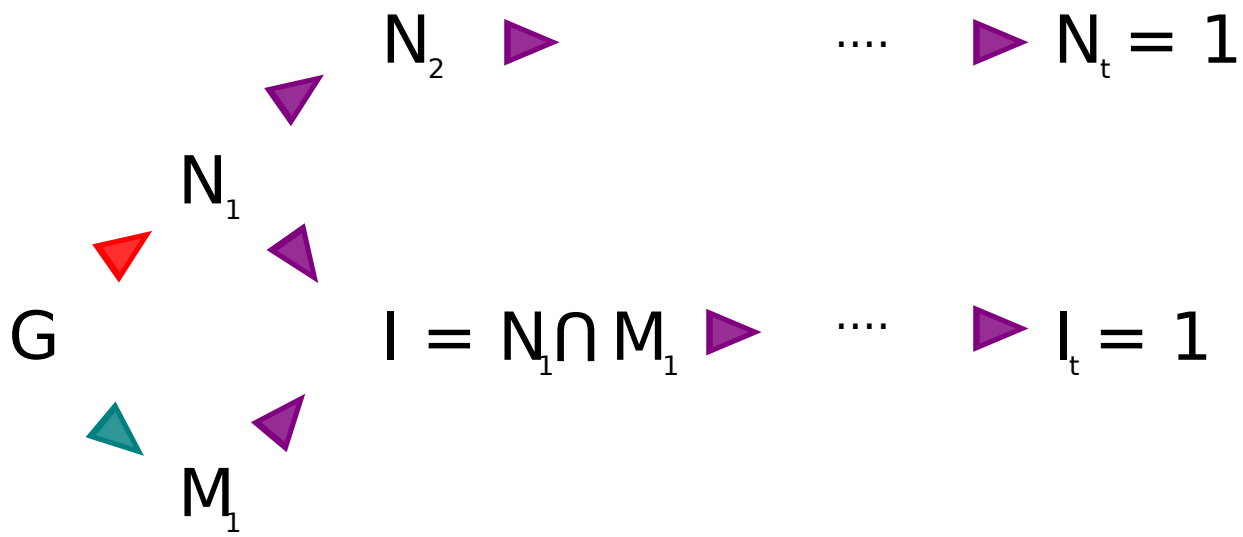
Proving Jordan Hölder Theorem

We have obtained:



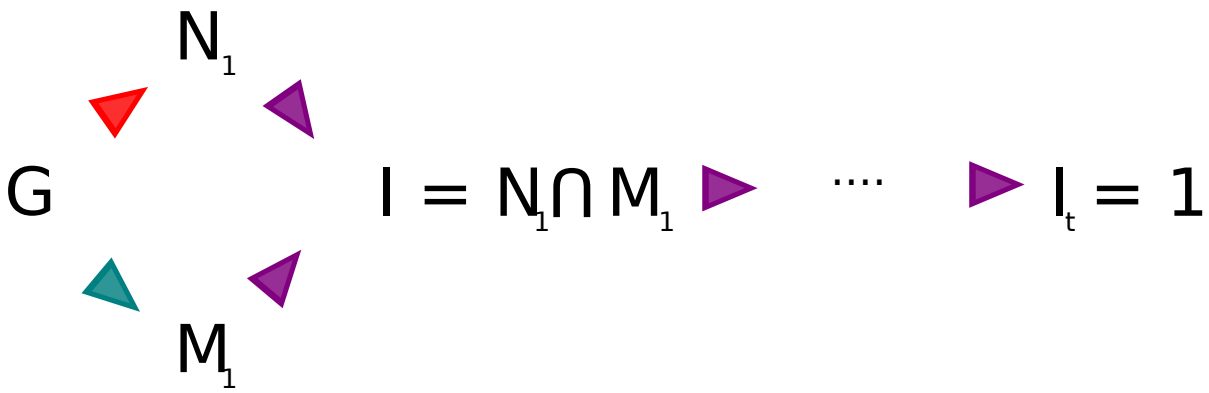
Proving Jordan Hölder Theorem

We have obtained:



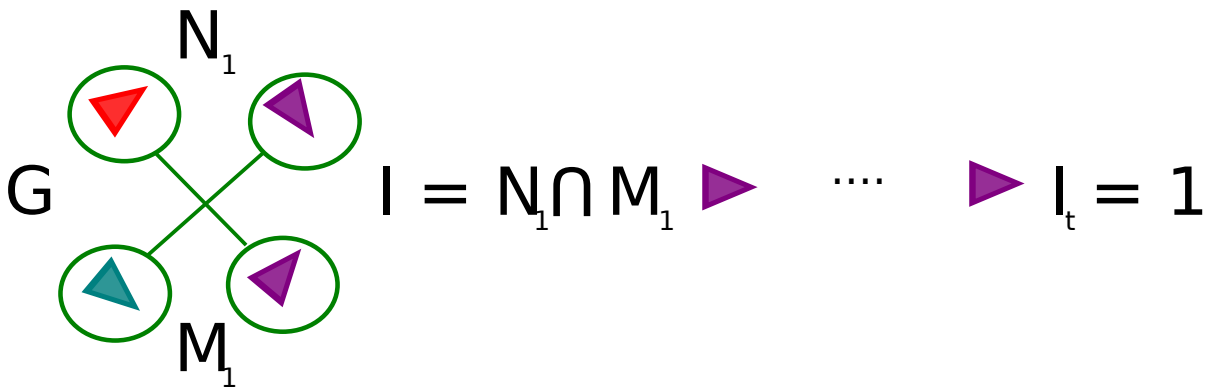
Proving Jordan Hölder Theorem

We have obtained:



Proving Jordan Hölder Theorem

We conclude by a "butterfly lemma":



Formalization issues: sets vs. types

Not all distinct sets should be modelled as distinct types

- A type fixes common requirements for its inhabitants.
- Type inhabitants are objects we want to observe and combine.

Formalization issues: sets vs. types

Types should have the appropriate granularity.

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

- Work within a finite group domain type which fixes the law.

```
gT : finGroupType
```

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

- Work within a finite group domain type which fixes the law.

```
gT : finGroupType
```

- Groups are collections (subsets) of inhabitants of this big group.

```
G H : {group gT}
```

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

- Work within a finite group domain type which fixes the law.

```
gT : finGroupType
```

- Groups are collections (subsets) of inhabitants of this big group.

```
G H : {group gT}
```

- Operations on groups remain homogeneous.

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

- Work within a finite group domain type which fixes the law.

```
gT : finGroupType
```

- Groups are collections (subsets) of inhabitants of this big group.

```
G H : {group gT}
```

- Operations on groups remain homogeneous.
- Only change type when a new group law is really needed

Formalization issues: sets vs. types

Types should have the appropriate granularity.

$$G \times H \quad G * H \quad G \cap H \quad G \rtimes H$$

Remember the main objects we manipulate most are groups, not elements:

- Work within a finite group domain type which fixes the law.
  `gT : finGroupType`
- Groups are collections (subsets) of inhabitants of this big group.
  `G H : {group gT}`
- Operations on groups remain homogeneous.
- Only change type when a new group law is really needed
  quotient groups...

Formalization issues: comprehension style

- In set theory: comprehension rule forges:
  the set $\{x \mid P\ x\}$
- In type theory: Sigma types (dependent pairs) forge:
  the types $\{x \mid P\ x\}$
- Is there more to say?

Formalization issues: comprehension style

- In set theory: comprehension rule forges:
  the set $\{x \mid P\ x\}$
- In type theory: Sigma types (dependent pairs) forge:
  the types $\{x \mid P\ x\}$
- Is there more to say?

Yes, about the status of equality.

Formalization issues: comprehension style

The sigma type of duplicate free lists on type T is:

$$\{l \ : \ \mathit{list}\, T \mid \text{duplicate\_free } l\}$$

- An inhabitant t_l of this type is a pair (l, p_l)
- Comparing two inhabitants t_1 and t_2 means comparing them component-wise:

$$t_1 = t_2 \quad \text{iff} \quad (l_1 = l_2) \wedge p_{l_1} = p_{l_2}$$

- The proof component should be irrelevant here.

But in general Coq is not a proof irrelevant system...

Formalization issues: comprehension style

A taste of proof-irrelevance:

$$\forall\ (x\ y\ :\ bool)\ (p_1\ p_2\ :\ x = y),\ p_1 = p_2$$

- Suppose that duplicate_free : list T $\rightarrow$ bool

Formalization issues: comprehension style

A taste of proof-irrelevance:

$$\forall\ (x\ y\ :\ bool)\ (p_1\ p_2\ :\ x = y),\ p_1 = p_2$$

- Suppose that duplicate_free : list T $\rightarrow$ bool
- Now the sigma type is: $\{l\ :\ list\,T \mid \text{duplicate\_free}\ l = \text{true}\}$

Formalization issues: comprehension style

A taste of proof-irrelevance:

$$\forall\ (x\ y\ :\ bool)\ (p_1\ p_2\ :\ x = y),\ p_1 = p_2$$

- Suppose that duplicate_free : list T $\rightarrow$ bool
- Now the sigma type is: $\{l\ :\ list\,T \mid \text{duplicate\_free}\ l = \text{true}\}$
- Compare (l_1, p_1) with (l_2, p_2) when $l_1 = l_2$.
  - p_1 : duplicate_free l_1 = *true*
  - p_2 : duplicate_free l_2 = *true*.

Formalization issues: comprehension style

A taste of proof-irrelevance:

$$\forall\ (x\ y\ :\ bool)\ (p_1\ p_2\ :\ x = y),\ p_1 = p_2$$

- Suppose that duplicate_free : list T $\rightarrow$ bool
- Now the sigma type is: $\{l\ :\ list\,T \mid \text{duplicate\_free}\ l = \text{true}\}$
- Compare (l_1, p_1) with (l_2, p_2) when $l_1 = l_2$.
  - p_1 : duplicate_free l_1 = *true*
  - p_2 : duplicate_free l_2 = *true*.

Using the theorem, we prove that $p_1 = p_2$.

Formalization issues: comprehension style

A taste of proof-irrelevance:

$$\forall\ (x\ y\ :\ bool)\ (p_1\ p_2\ :\ x = y),\ p_1 = p_2$$

- Suppose that duplicate_free : list T $\rightarrow$ bool
- Now the sigma type is: $\{l\ :\ list\,T \mid \text{duplicate\_free}\ l = \text{true}\}$
- Compare (l_1, p_1) with (l_2, p_2) when $l_1 = l_2$.
  - p_1 : duplicate_free l_1 = *true*
  - p_2 : duplicate_free l_2 = *true*.

Using the theorem, we prove that $p_1 = p_2$.

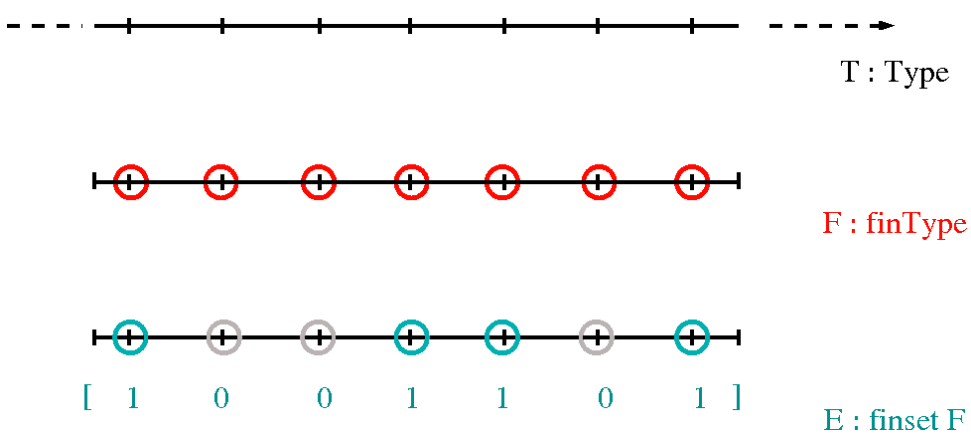
Comparing inhabitants of boolean sigma types is comparing values.

Finite sets are intentional

Ingredients:

- A proof irrelevant definition of lists of fixed length (tuples)
- Types with a finite number of inhabitants

A finite set `(s : set F)` is a mask on a finite type `F`:

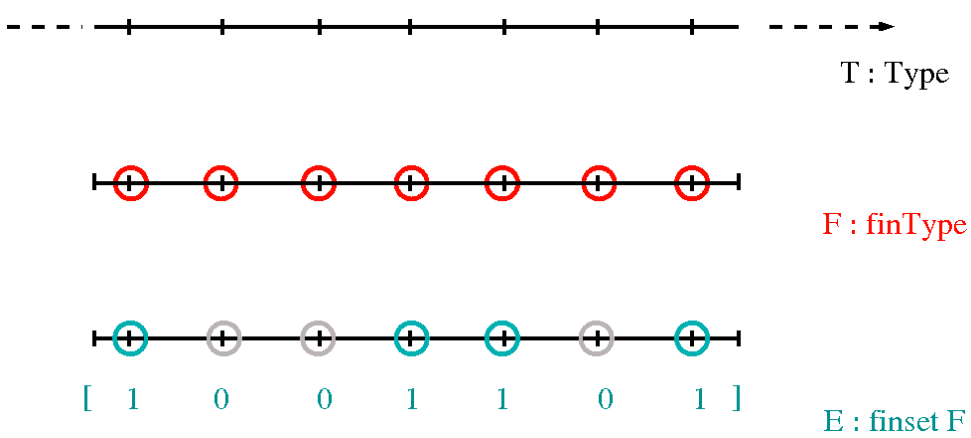


Finite sets are intentional

Ingredients:

- A proof irrelevant definition of lists of fixed length (tuples)
- Types with a finite number of inhabitants

A finite set `(s : set F)` is a mask on a finite type `F`:



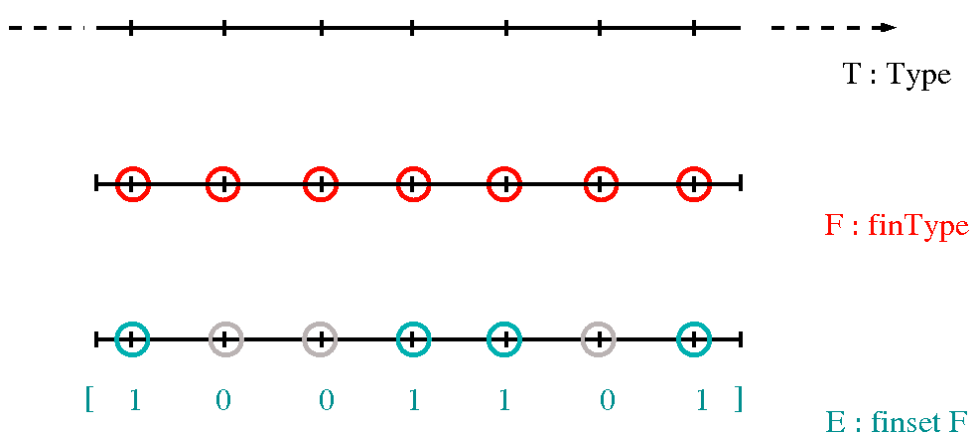
It is a boolean list of fixed length $\sharp$ `F`.

Finite sets are intentional

Ingredients:

- A proof irrelevant definition of lists of fixed length (tuples)
- Types with a finite number of inhabitants

A finite set `(s : set F)` is a mask on a finite type `F`:



A mask coerces to a characteristic function `(s : F -> bool)`, such that

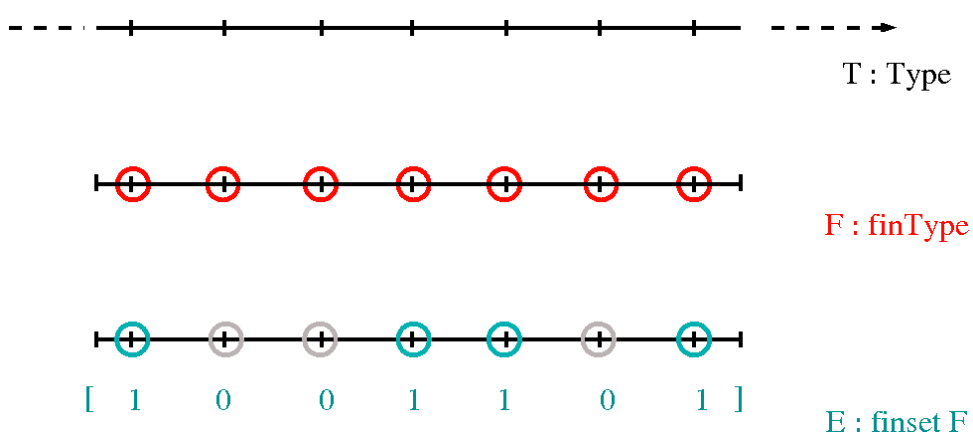
$$s_1 = s_2 \Leftrightarrow (\forall x, s_1\ x = s_2\ x)$$

Finite sets are intentional

Ingredients:

- A proof irrelevant definition of lists of fixed length (tuples)
- Types with a finite number of inhabitants

A finite set `(s : set F)` is a mask on a finite type `F`:



A mask coerces to a characteristic function `(s : F -> bool)`, such that

$$s_1 = s_2 \Leftrightarrow (\forall x, s_1\ x = s_2\ x)$$

$\Rightarrow$ Quotient types on finite types are easy.

Finite group decomposition (continued)

Unfortunately the analogy with arithmetics soon becomes irrelevant:

Finite group decomposition (continued)

Unfortunately the analogy with arithmetics soon becomes irrelevant:

- Two prime numbers with the same multiset of prime factors are equal.

Finite group decomposition (continued)

Unfortunately the analogy with arithmetics soon becomes irrelevant:

- Two prime numbers with the same multiset of prime factors are equal.
- Two groups with the same (up to isomorphism) multiset of prime factors are not necessarily equal (not even isomorphic).

Finite group decomposition (continued)

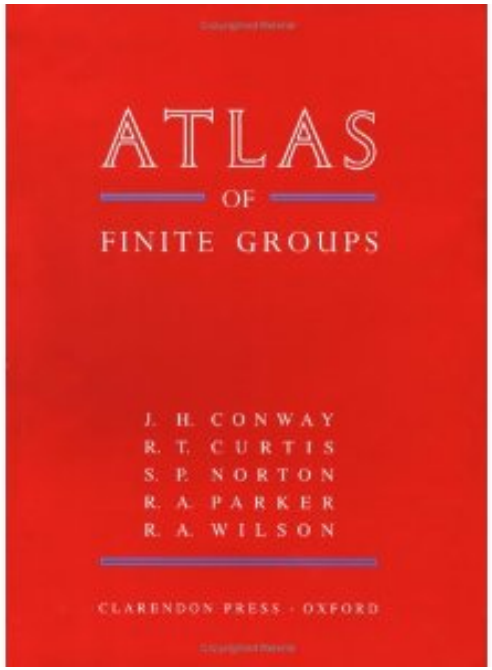
Unfortunately the analogy with arithmetics soon becomes irrelevant:

- Two prime numbers with the same multiset of prime factors are equal.
- Two groups with the same (up to isomorphism) multiset of prime factors are not necessarily equal (not even isomorphic).

$\Rightarrow$ Classifying finite groups is much more difficult than "classifying" numbers.

The Atlas of Finite Groups

The classification of all simple finite groups
aka. The Enormous Theorem



has been considered achieved in 1983 and revised in 2005.

The Odd Order Theorem

Theorem (Feit - Thompson (1963)) :

Every simple group of odd order is solvable.

otherwise said

Every simple group of odd order is cyclic.

The Odd Order Theorem

Theorem (Feit - Thompson (1963)) :

Every simple group of odd order is solvable.

otherwise said

Every simple group of odd order is cyclic.

- Original published proof: one entire volume of the Pacific Journal of Mathematics

The Odd Order Theorem

Theorem (Feit - Thompson (1963)) :

Every simple group of odd order is solvable.

otherwise said

Every simple group of odd order is cyclic.

- Original published proof: one entire volume of the Pacific Journal of Mathematics
- A collective simplification work: two entire volumes of London Math. Society Lecture Notes.

The Odd Order Theorem

Theorem (Feit - Thompson (1963)) :

Every simple group of odd order is solvable.
otherwise said
Every simple group of odd order is cyclic.

- Original published proof: one entire volume of the Pacific Journal of Mathematics
- A collective simplification work: two entire volumes of London Math. Society Lecture Notes.
- (Wikipedia 12/10/2010) "It takes a professional group theorist about a year of hard work to understand the proof completely."

The Odd Order Theorem

Theorem (Feit - Thompson (1963)) :

Every simple group of odd order is solvable.
otherwise said
Every simple group of odd order is cyclic.

- Original published proof: one entire volume of the Pacific Journal of Mathematics
- A collective simplification work: two entire volumes of London Math. Society Lecture Notes.
- (Wikipedia 12/10/2010) "It takes a professional group theorist about a year of hard work to understand the proof completely."
- The proof mixes many theories, non only combinatorics but also linear algebra, Galois theory, characters,...

Everyday mathematics syntax, implicit semantics

For any matrix $M \in M_n(F)$,

$$det(M) := \sum_{s \in S_n} (-1)^{\epsilon_s} \prod_i M_{i,s(i)}$$

Everyday mathematics syntax, implicit semantics

For any matrix $M \in M_n(F)$,

$$det(M) := \sum_{s \in S_n} (-1)^{\epsilon_s} \prod_i M_{i,s(i)}$$

In LaTeX we write:

```
det(M) :=
  \sum_{s \in S_n}(-1)^{\epsilon_s}\prod_i M_{i, s(i)}
```

Everyday mathematics syntax, implicit semantics

For any matrix $M \in M_n(F)$,

$$det(M) := \sum_{s \in S_n} (-1)^{\epsilon_s} \prod_i M_{i,s(i)}$$

In LaTeX we write:

```
det(M) :=
  \sum_{s \in S_n}(-1)^{\epsilon_s}\prod_i M_{i, s(i)}
```

In our proof assistant we would like to write:

```
Definition determinant n (A : 'M[R]_n) : R :=
  \sum_(s : 'S_n) (-1) ^+ s * \prod_i A i (s i).
```

Everyday mathematics syntax, implicit semantics

Type inference mechanism play a crucial role here.

In particular, higher order type inference:

- is triggered by higher order unification
- is used as a type classes mechanism, but with proof inference
- even allows quoting (reification by type inference)

The Mathematical Components project

- Who: G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, A. M., S. Ould Biha, I. Pasca, L. Rideau, S. Le Roux, S. McLaughlin, R. O'Connor, E. Tassi, L. Théry

The Mathematical Components project

- Who: G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, A. M., S. Ould Biha, I. Pasca, L. Rideau, S. Le Roux, S. McLaughlin, R. O'Connor, E. Tassi, L. Théry
- What is covered as for today:
  - Combinatorics: lists, finite sets, relations, paths, permutations, basic arithmetic, divisibility...
  - Finite group theory: local analysis, fair amount of representation theory
  - Algebra: hierarchy, matrices, linear algebra basics, module theory basics
  - Almost the first volume of the proof

The Mathematical Components project

- Who: G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, A. M., S. Ould Biha, I. Pasca, L. Rideau, S. Le Roux, S. McLaughlin, R. O'Connor, E. Tassi, L. Théry

- What isn't required by this proof: Topology, real, complex analysis, integration, number theory,...

Conclusion

Craft your types with care and benefit from:

- Boolean reflection:
  - relies on the status of computation in CIC
  - requires some formalized infrastructure and support in the tactic language
  - is constructive-compliant
  - can be combined with a classical monad for the possibly classical chunks of a theory
- Higher order type inference:
  - lets implicit in types what is implicit in the textbook
  - makes statements human readable
  - automatizes side conditions inference
- Infrastructure libraries

`http://www.msr-inria.inria.fr/Projects/math-components/`