Conjecture
u]
u]

1 Interactive theorem provers

TODO: Explain the concept of tactic, and how it relates to traditional inference rules when read bottom-up.

2 Proof theory

3 This thesis

In this thesis, we are concerned mainly with the question of providing a *direct-manipulation* interface for building formal proofs *dynamically* and *incrementally*. To this effect, we explore new ways to represent and interact with the *proof state*, that is the mental state in which a human finds herself when searching for, or unrolling the steps of a proof. Following the LCF tradition of proof assistants [130], we assume that the underlying data modelling this mental state consists at its core of a set of *conclusions* waiting for evidence, each having their own set of *hypotheses* or *assumptions*.

Another important data is the *arguments* used for already justified conclusions. Typically one might want a *static* representation of arguments, to ease reviewing, modification or reuse of a proof at a later time. These activities are generally regrouped under the term of *proof evolution*. Until the very last chapter, we will not consider the question of having such a static representation. Our hope is that once we find an adequate representation for the proof state geared towards *dynamic* manipulation, the sole data of the *transitions* between states through a set of canonical *proof actions* will be enough to reconstruct both a machine-readable formal proof object, and a human-readable proof text.

4 Traditional approach

Summary: Instead of designing an interface that facilitates the learning and usage of a traditional proof language (natural deduction, sequent calculus), we aim for an interface that facilitates logical reasoning broadly construed.

In a sense, our methodology for designing proving interfaces goes in the opposite direction compared to traditional approaches.

Indeed, one usually starts with a canonical static representation of partial proofs[1], and then builds user-facing abstractions on top of it. These include proof languages which can be more or less declarative[2], but also extensible notation and macro systems [187] for concise, human-readable representations of mathematical objects. Ultimately, all these abstractions are compiled to low-level proof objects in the kernel language, a process called *elaboration*.

Our point is that the initial choice of kernel language can influence the design of higher-level abstractions, either unconsciously or consciously: typically, one will want to stay close to the kernel representation in order to mitigate the complexity of the elaboration process, since it could impact negatively the speed and memory usage of the system. This is especially true of abstractions used in structuring the purely logical, non-domain specific parts of proofs, since those occur in virtually every development. This can be observed at two different levels:

- the level of *statements*, where specific constructors (typically logical connectives) need a specific interface, and are not always treated uniformly**TODO:** example of various tactics for the "same" action (intro/elim) but with differing syntax (`intros` vs. `split`, `apply` vs. `destruct`...), and of their shortcomings (`apply` only works with ∀ and ⊃);
- the level of *inference rules*, which are usually modelled after standard proof formalisms such as natural deduction and sequent calculus[3], and where many pervasive commands for manipulating the proof state map directly to these rules[4].

This focus on the formal, low-level proof object is likely rooted in the historical origins of proof assistants, with projects such as Automath, Mizar and LCF**TODO:** add citations. Retrospectively, they shared a common conceptual framework which was the direct consequence of two scientific and technological developments:

- the advent of mathematical logic and proof theory at the dawn of the 20^{th} century, motivated by a foundational crisis concerning first the unity of various branches of mathematics, and then the very notion of mathematical *truth* following the discovery of Russell's paradox;
- the invention and democratization of computers, and more specifically of *programming languages* as their principal means of interaction with humans.

way mathematicians build and think about proofs. Indeed, those were designed to solve foundational questions[5] rather than model the way mathematicians effectively reason.

5 Modern interfaces

We argue that this historical influence has created a *blind spot* for a more modern approach to proving interfaces. Nowadays, the vast majority of interactions between humans and computers happen through *graphical user interfaces*, which have proved to make software systems a lot more intuitive and accessible to people without any specific training or education. This is likely one of the important roadblocks towards a wider adoption of proof assistants by researchers and students in mathematics, but also in other domains where formal logical thinking could be applied such as the modelling and verification of computer systems, or even legal systems.

6 Existing solutions to overcome poor proof languages

- Maximizing *custom automation* with a rich domain-specific language for implementing proof search procedure (Ltac). Works well in the domain of program verification, where one can rely on both the rich inductive structure of object programs, and the programming expertise of the user. But less suited to the structure of standard mathematics, which are built around the rich logical interleaving of theorems and definitions, rather than complex inductive definitions. Or more sociologically, does not resemble the actual practice of mathematicians on paper, and is unusable by non-programmers, which constitute a large portion of potential users in both research and education.
- More *declarative* proof languages, agnostic to the underlying logic (Isar). Solves most of the previously mentioned flaws, but is still very verbose and sensitive to syntax errors, and does not tackle the question of a more guided interface, which could improve both the learning curve and the speed of the proof building process.

7 Roadmap

In this thesis, we start our exploration in the design of proof-building interfaces with graphical interaction principles inspired by standard approaches to logic and proof theory: statements are represented by symbolic formulas made from the usual logical connectives, and inference rules are in direct correspondance, or simulate closely the flow of argumentation found in Gentzen's sequent calculus.

Then we gradually stray away from tradition, the first step being the adoption of a more flexible approach to argumentation as offered by the *deep inference* paradigm. This allows us to build proofs through *drag-and-drop* actions with the *subformula linking* technique of K. Chaudhuri, and even to imagine a nested and metaphorical representation of *subgoals* with our own *bubble calculus*. But there are still some defects in these systems, which confine the reasoner to weird habits that restrict their freedom.

Finally we fully embrace the idea of manipulating directly iconic subgoals instead of symbolic formulas, by reviving an old system from the genius and avant-gardist logician C. S. Peirce: the *existential graphs*. We add a layer of playfulness and intuitionism by devising the *flower calculus*, a fun and alternative way to draw and manipulate the graphs as flowers connected by sprinklers in nested gardens. But more importantly, the flower calculus exhibits the latent *analyticity* of Peirce's graphs, thus linking them with the systems of Gentzen despite a very different approach to the representation and use of statements. This allows us to design a new kind of direct manipulation interface that merges very tightly two concepts/structures, which have always been segregated ontologically: formulas and proofs. Hopefully, this foundation will enable new ways to build and maintain formal proofs, that are less buraucratic and more enjoyable to humans.

TODO: Explain the concept of iconicity somewhere

8 How to read this thesis

TODO: Introduce a special format for long digressions in sidenotes, so that the reader can safely ignore them.

[5]For instance, Gentzen invented natural deduction, and later sequent calculus in order to obtain finitist proofs of soundness for basic systems of arithmetic, through his well-known *Haupstatz* or *cut-elimination* theorem.

TODO: Advise the reader to read with a PDF reader that supports color, hyperlink jumping and hyperlink preview. We tend to cross-reference a lot ideas from different chapters.