From the Foundation of Mathematics to the Birth of Computation

Fairouz Kamareddine
Heriot-Watt University, Edinburgh, Scotland

November 2011

Logic/Mathematics/Computation: A word of warning

- Logic is *OLD*. Mathematics is *OLD*. But, *SO IS* computation.
- Just like in the times of Euclides or of AlHambra/Andalucia or of Frege/Russell, deduction/Logic was taken as a foundation for Mathematics, computation was also taken throughout as an essential tool in mathematics.
- Our ancestors used sandy beaches to compute the circumference of a circle, and to work out approximations/values of numbers like π.
- The word algorithm dates back centuries? Algorithms existed in anciant Egypt at the time of Hypatia. The word is named after Al-Khawarizmi.
- But even more impressively, the following important 20th century (un)computability result was known to Aristotle.
- Assume a problem Π,
  - If you *give* me an algorithm to solve Π, I can check whether this algorithm really solves Π.
  - But, if you ask me to *find* an algorithm to solve Π, I may go on forever trying but without success.

- But, this result was already known to Aristotle:
- Assume a proposition Φ.
  - If you *give* me a proof of Φ, I can check whether this proof really proves Φ.
  - But, if you ask me to *find* a proof of Φ, I may go on forever trying but without success.
- In fact, *programs* are *proofs*:
  - *program* $=$ *algorithm* $=$ *computable function* $=$ *λ-term*.
  - By the PAT principle: *Proofs* are *λ-terms*.
- Although computation is old, the science, art and foundation of computation was developed in the 20th century.
- Just like types are old, but type theories were only developed since 1903.
- Even more, computation comes alive with a general powerful physical body.
- This talk will not address any aspect of the physical computer.

Why did computer science kick off in the 20th century? Formal systems in the 19th century

After the ancient egyptians, ancient Greeks and the Arab empire, logic was dormant until the 17th century when Leibniz wanted to prove things like the existence of god in a mechanical manner.

But the biggest kick off was in the 19th century, when the *need for a more precise* style in mathematics arose, *because controversial results* had appeared in *analysis*.

- 1821: Many controversies in analysis were solved by Cauchy. E.g., he gave *a precise definition of convergence* in his *Cours d'Analyse* [Cauchy, 1821].
- 1872: Due to the more *exact definition of real numbers* given by Dedekind [Dedekind, 1872], the rules for reasoning with real numbers became even more precise.
- 1895-1897: Cantor began formalizing *set theory* [Cantor, 1895, 1897] and made contributions to *number theory*.

Formal systems in the 19th century
symbols (not natural language) define logical concepts

- 1889: *Peano* formalized *arithmetic* [Peano, 1889], but did not treat logic or quantification.

- 1879: *Frege* was not satisfied with the use of *natural language in mathematics*:

  > ". . . I found *the inadequacy of language to be an obstacle*; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required."
  >
  > (*Begriffsschrift*, Preface)

  Frege therefore presented *Begriffsschrift* [Frege, 1879], the first formalisation of logic giving logical concepts via symbols rather than natural language.

Formal systems in the 19th century
A general definition of functions

"[Begriffsschrift's] first purpose is to *provide us with the most reliable test of the validity of a chain of inferences* and to point out *every presupposition* that tries to sneak in unnoticed, so that its origin *can be investigated*."

(*Begriffsschrift*, Preface)

- The introduction of a *very general definition of function* was the key to the formalisation of logic. Frege defined the Abstraction Principle.

Abstraction Principle 1.

"If in an expression, [. . .] a simple or a compound sign has one or more occurrences *and if we regard that sign as* replaceable *in all or some of these occurrences* by something else *(but everywhere by the same thing), then we call* the part that remains invariant in the expression *a* function, *and* the replaceable part *the* argument *of the function."*

(Begriffsschrift, Section 9*)*

Russell's paradox due to self-application of functions Hilbert's program

- 1892-1903 Frege's *Grundgesetze der Arithmetik* [Frege, 1892a, 1903], could handle elementary arithmetic, set theory, logic, and quantification.

- *Self-application of functions* (not in Begriffsschrift) was at the heart of *Russell's paradox 1902* [Russell, 1902].

- Also in the early 1900s, Hilbert, a master in posing difficult problems wanted to believe that any logical statement can either have a proof or be disproved.

- More than 30 years later, Hilbert's wish was negatively answered by Turing (Turing machines), Goedel (incompleteness results) and Church (λ-calculus).

Can we solve/compute everything?

- Turing answered the question via a *machine for running/computing programs*. *a function f is computable iff f can be computed on a Turing machine.*

- Church invented the λ-calculus, a *language for describing programs*. *a function f is computable iff f can be described in the λ-calculus.*

- Note that Church's λ-calculus was initially intended as a language of programs and logic, but it turned out to be inconsistent (Kleene and Rosser) and Church restricted the λ-calculus to programs.

- Goedel's result meant that *no absolute guarantee can be given that many significant branches of mathematics are entirely free of contradictions.*

- This means: we can compute a very small (∞ly countable, size of $\mathbb{N}$) amount compared to what we will never be able to compute (uncountable, size of $\mathbb{R}$).

- Hilbert's dream was shattered. According to the great historian of Mathematics Ivor Grattan-Guinness, Hilbert behaved coldly towards Goedel.

How did this foundational work influence programming?

- By the 1950s we had the computer, we knew what a computable functon is, and programming languages started in earnest.
- For example, untyped λ-calculus was adopted by John McCarthy in 1958 in the language LISP.
- Algol 60 (1958) and Algol 68 (1958) were also developed.
- Also, the earlier work of Frege, Russell and Whitehead, Hilbert, etc., on the formalisaton of mathematics, were now being complemented/replaced in the 1960s by the computerisation of mathematics.
- De Bruijn's Automath and Trybulec's Mizar were conceived around 1967.
- But before we can talk more about programming languages, theorem provers or the computerisation of mathematics, we need to go back and look at the Paradoxes and their solution and how this influenced on expressivity.
- I will only discuss the solution to the paradoxes using type theory (I will not discuss set theory or category theory).

Why Type Theory?

- To *avoid paradox* Russell controlled function application via *type theory*.

- Russell [Russell, 1903] *1903* gives the first type theory: the *Ramified Type Theory* (RTT). But, *types* existed since the time of *Euclid* (325 B.C.). And Frege did use typing to avoid paradoxes (still the paradoxes sneaked from the backdoor).

- RTT is used in Russell and Whitehead's Principia Mathematica [Whitehead and Russell, 1910^1, 1927^2] 1910–1912.

- *Simple theory of types* (STT): Ramsey [Ramsey, 1926] *1926*, Hilbert and Ackermann [Hilbert and Ackermann, 1928] *1928*.

- Church's *simply typed λ-calculus* $\lambda{\to}$ [Church, 1940] 1940 $=$ λ-calculus $+$ STT.

- Simply typed λ-calculus was adopted in theorem provers like HOL and was used to make sense of other programming languages (e.g., pascal).
- Then, simple types were independently extended to *polymorphic* (logic [Girard, 1972]) and (programming language [Reynolds, 1974]).
- *Dependent* types (necessary for reasoning about proofs inside the system) were also introduced in Automath by de Bruijn.
- Polymorphic types are used in programming languages like ML although not the full 2nd order λ-calculus since type Checking and typability in the 2nd order λ-calculus is undecidable (this was an open problem for over 25 years and was shown in 1995 by Joe Wells).
- And the search continues for better and better programming languages.
- *Types* continue to play an influential role in the design and implementation of programming languages and theorem provers.

Prehistory of Types (Euclid)
The class to which an object belongs

- Euclid's *Elements* (circa 325 B.C.) begins with:

  1. A *point* is that which has no part;
  2. A *line* is breadthless length.

     ⋮

  15. A *circle* is a plane figure contained by one line such that all the straight lines falling upon it from one point among those lying within the figure are equal to one another.

- 1..15 *define* points, lines, and circles which Euclid *distinguished* between.

- Euclid always mentioned to which *class* (points, lines, etc.) an object belonged.

Prehistory of Types (Euclid)
Intuition forced Euclid to think of the type of objects

- By distinguishing classes of objects, Euclid prevented *undesired/impossible* situations. E.g., whether two points (instead of two lines) are parallel.
- *Intuition* implicitly forced Euclid to think about the *type* of the objects.
- As intuition does not support the notion of parallel points, he did not even *try* to undertake such a construction.
- In this manner, types have always been present in mathematics, although they were not noticed explicitly until the late 1800s.
- If you studied geometry, then you have an (implicit) understanding of types.

Prehistory of Types (Paradox Threats) [Kamareddine et al., 2002, 2004]

- From 1800, mathematical systems became less intuitive, for several reasons:
  - Very *complex* or abstract systems.
  - *Formal* systems.
  - Something with *less intuition* than a human using the systems: a *computer* or an *algorithm*.
- These situations are *paradox threats*. An example is Frege's Naive Set Theory.
- *Not enough intuition* to activate the (implicit) type theory *to warn against an impossible situation*.

Prehistory of Types (Begriffsschrift's functions)
Paradox threats

- Frege put *no restrictions* on what could play the role of *an argument*.
- An argument could be a *number* (as was the situation in analysis), but also a *proposition*, or a *function*.
- Similarly, the *result of applying* a function to an argument did not necessarily have to be a number.
- Functions of more than one argument were constructed by a method that is very close to the method presented by Schönfinkel [Schönfinkel, 1924] in 1924.

Prehistory of Types (Begriffsschrift's functions))
Paradox threats

With this definition of function, two of the three possible *paradox threats occurred*:

1. The generalisation of the concept of function made the system more abstract and *less intuitive*.
2. Frege introduced a *formal* system instead of the informal systems that were used up till then.

Type theory, that would be helpful in distinguishing between the different types of arguments that a function might take, *was left informal*.

So, *Frege had to proceed with caution*. And so he did, at this stage.

Prehistory of Types (Begriffsschrift's functions)
Typing functions

Frege was *aware* of some typing rule that does *not* allow to *substitute functions for object variables or objects for function variables*:

> "if the [. . .] letter [sign] occurs as a function sign, this circumstance [should] be taken into account."
>
> (*Begriffsschrift*, Section 11)

> " Now just as functions are fundamentally different from objects, so also *functions whose arguments are and must be functions* are fundamentally different from *functions whose arguments are objects and cannot be anything else*. I call the latter *first-level*, the former *second-level*."
>
> (*Function and Concept*, pp. 26–27)

Prehistory of Types (Begriffsschrift's functions)
First level versus second level and avoiding paradox in Begriffsschrift

In *Function and Concept* he was aware of the fact that making a *difference between first-level and second-level objects is essential to prevent paradoxes*:

> "The ontological proof of God's existence suffers from the fallacy of treating existence as a first-level concept."
>
> (*Function and Concept*, p. 27, footnote)

The above discussion on functions and arguments shows that *Frege did indeed avoid the paradox in his Begriffsschrift*.

Prehistory of Types (Grundgesetze's functions) Self application

The *Begriffsschrift*, however, was only a prelude to Frege's writings.

- In Grundlagen der Arithmetik [Frege, 1884] he argued that mathematics can be seen as a branch of logic.

- In Grundgesetze der Arithmetik [Frege, 1892a, 1903] he described the elementary parts of arithmetic within an extension of the logical framework of *Begriffsschrift*.

- Frege approached the *paradox threats for a second time* at the end of Section 2 of his *Grundgesetze*.

- He did *not* want to *apply a function to itself*, but to its course-of-values.

Prehistory of Types (Grundgesetze's functions)

"the function $\Phi(x)$ has the same *course-of-values* as the function $\Psi(x)$" if:

> "the functions $\Phi(x)$ and $\Psi(x)$ always have the same value for the same argument."
>
> (*Grundgesetze*, p. 7)

- Note that functions $\Phi(x)$ and $\Psi(x)$ may have equal courses-of-values even if they have different definitions. E.g., $x \wedge \neg x$, and $x \leftrightarrow \neg x$.

- Frege denoted the course-of-values of a function $\Phi(x)$ by $\grave{\varepsilon}\Phi(\varepsilon)$. The definition of equal courses-of-values could therefore be expressed as

$$\grave{\varepsilon}f(\varepsilon) = \grave{\varepsilon}g(\varepsilon) \longleftrightarrow \forall a[f(a) = g(a)]. \tag{1}$$

  In modern terminology, we could say that the functions $\Phi(x)$ and $\Psi(x)$ have the same course-of-values if they have the same *graph*.

Prehistory of Types (Grundgesetze's functions)

- The notation $\grave{\varepsilon}\Phi(\varepsilon)$ may be the *origin* of Russell's notation $\hat{x}\Phi(x)$ for the class of objects that have the property Φ.
- According to a paper by Rosser [Rosser, 1984], the notation $\hat{x}\Phi(x)$ has been at the *basis* of the current notation $\lambda x.\Phi(x)$.
- Church is supposed to have written $\wedge x\Phi(x)$ for the function $x \mapsto \Phi(x)$: the hat $\wedge$ in front of the x distinguishes this function from the class $\hat{x}\Phi(x)$.

Prehistory of Types (Grundgesetze's functions)
Applying a function to its course-of-values

- Frege treated *courses-of-values* as *ordinary objects*.
- As a consequence, *a function that takes objects as arguments could have its own course-of-values as an argument*.
- In modern terminology: a function that takes objects as arguments can have its own graph as an argument.
- *BUT,* all essential information of a function is contained in its graph.
- A system in which a function can be applied to its own graph should have similar possibilities as a system in which a function can be applied to itself.
- Frege *excluded the paradox threats* by *forbidding self-application*
- but due to his *treatment of courses-of-values* these threats were able to *enter his system through a back door*.

Prehistory of Types (Russell's paradox in $Grundgesetze$)

- In 1902, Russell wrote a letter to Frege [Russell, 1902], informing him that he had *discovered a paradox* in his *Begriffsschrift.*
- *WRONG: Begriffsschrift does not suffer from a paradox*.
- Russell gave his well-known argument, defining the propositional function

  $$f(x) \text{ by } \neg x(x).$$

  In Russell's words: "*to be a predicate that cannot be predicated of itself.*"
- Russell assumed $f(f)$. Then by definition of f, $\neg f(f)$, *a contradiction*. Therefore: $\neg f(f)$ holds. But then (again by definition of f), $f(f)$ holds. Russell concluded that *both* $f(f)$ *and* $\neg f(f)$ hold, a *contradiction*.

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- *6 days later*, Frege wrote [Frege, 1902] that *Russell's derivation of paradox is incorrect*.

- Ferge explained that *self-application $f(f)$ is not possible in* Begriffsschrift.

- *$f(x)$ is a function, which requires an* object *as an argument.*
  A function cannot be an object in the Begriffsschrift.

- Frege explained that *Russell's argument could be amended to a paradox* in Grundgesetze, using the *course-of-values* of functions:
  Let $f(x) = \neg\forall\varphi[(\grave{\alpha}\varphi(\alpha) = x) \longrightarrow \varphi(x)]$
  I.e. $f(x) = \exists\varphi[(\grave{\alpha}\varphi(\alpha) = x) \wedge \neg\varphi(x)]$ hence $\neg\varphi(\grave{\alpha}\varphi(\alpha))$

- *Both* $f(\grave{\varepsilon}f(\varepsilon))$ and $\neg f(\grave{\varepsilon}f(\varepsilon))$ hold.

- Frege added an appendix of 11 pages to the 2nd volume of *Grundgesetze* in which he gave a very detailed description of the paradox.

Prehistory of Types (How wrong was Frege?)

- Due to Russell's Paradox, Frege is often depicted as the pitiful person whose system was inconsistent.
- This suggests that Frege's system was the only one that was inconsistent, and that Frege was very inaccurate in his writings.
- On these points, history does Frege an injustice.
- Frege's system was much more accurate than other systems of those days.
- Peano's work, for instance, was *less precise* on several points:
- Peano *hardly paid attention to logic* especially quantification theory;
- Peano *did not make a strict distinction* between his *symbolism* and the *objects underlying this symbolism*. Frege was much more accurate on this point (see Frege's paper *Über Sinn und Bedeutung* [Frege, 1892b]);

Prehistory of Types (How wrong was Frege?)

- Frege *made a strict distinction* between a *proposition* (as an object) and the assertion *of a proposition*. Frege denoted a *proposition*, by $-A$, and *its assertion* by $\vdash A$. Peano did not make this distinction and simply wrote A.

Nevertheless, Peano's work was very popular, for several reasons:

- Peano had *able collaborators*, and a *better eye for presentation and publicity*.
- Peano bought *his own press* to supervise the printing of his own journals Rivista di Matematica and Formulaire [Peano, 1894–1908]

Prehistory of Types (How wrong was Frege?)

- Peano used a *familiar symbolism* to the notations used in those days.
- Many of *Peano's notations*, like $\in$ for "is an element of", and $\supset$ for logical implication, are used in *Principia Mathematica*, and are actually still in use.
- *Frege's work* did not have these advantages and *was hardly read before 1902*
- When *Peano* published his formalisation of mathematics in 1889 [Peano, 1889] he clearly *did not know* Frege's *Begriffsschrift* as he did not mention the work, and *was not aware* of Frege's formalisation of quantification theory.

Prehistory of Types (How wrong was Frege?)

- Peano considered quantification theory to be "abstruse" in [Peano, 1894–1908]:

  > "In this respect my *[Frege] conceptual notion of 1879 is superior to the Peano one*. Already, at that time, I specified all the laws necessary for my designation of generality, so that nothing fundamental remains to be examined. These laws are few in number, and *I do not know why they should be said to be abstruse*. If it is otherwise with the *Peano conceptual notation*, then this is due to the *unsuitable* notation."
  >
  > ([Frege, 1896], p. 376)

Prehistory of Types (How wrong was Frege?)

- In the last paragraph of [Frege, 1896], Frege concluded:

  "...I observe merely that the *Peano notation* is unquestionably *more convenient for the typesetter*, and in many cases *takes up less room* than mine, but that these advantages seem to me, due to the inferior perspicuity and *logical defectiveness*, to have been paid for too dearly — at any rate for the purposes I want to pursue."
  (*Ueber die Begriffschrift des Herrn Peano und meine eigene*, p. 378)

Prehistory of Types (paradox in Peano and Cantor's systems)

- Frege's system was *not the only paradoxical* one.
- The Russell Paradox can be derived in *Peano's system* as well, by defining the class $K =_{\mathtt{def}} \{x \mid x \notin x\}$ and deriving $K \in K \longleftrightarrow K \notin K$.
- In *Cantor's Set Theory* one can derive the paradox via the same class (or *set*, in Cantor's terminology).

Prehistory of Types (paradoxes)

- Paradoxes were already widely known in *antiquity*.
- The oldest logical paradox: the *Liar's Paradox* "This sentence is not true", also known as the Paradox of Epimenides. It is referred to in the Bible (Titus $1{:}12$) and is based on the confusion between language and meta-language.
- The *Burali-Forti paradox* ([Burali-Forti, 1897], 1897) is a paradox within Cantor's theory on ordinal numbers.
- Cantor was *aware* of the Burali-Forti paradox but *did not think* it would render his system incoherent.
- *Cantor's paradox* on the largest cardinal number occurs in the same field. It was discovered by Cantor around 1895, but was not published before 1932.

Prehistory of Types (paradoxes)

- Logicians considered these paradoxes to be *out of the scope of logic*:
  - The *Liar's Paradox* can be regarded as a problem of *linguistics*.
  - The *paradoxes of Cantor and Burali-Forti* occurred in what was considered in those days a *highly questionable* part of mathematics: *Cantor's Set Theory*.
- The Russell Paradox, however, was *a paradox that could be formulated in all* the systems of the end of the 19th century (except for Frege's *Begriffsschrift*).
- Russell's Paradox was at the very basics of logic.
- It could not be disregarded, and a solution to it had to be found.
- In 1903-1908, Russell suggested the use of *types* to solve the problem [Russell, 1908].

Prehistory of Types (vicious circle principle)

When Russell proved Frege's *Grundgesetze* to be inconsistent, Frege was not the only person in *trouble*. In Russell's letter to Frege (1902), we read:

> "I am on the point of finishing a book on the principles of mathematics"
>
> (*Letter to Frege*, [Russell, 1902])

Russell *had to find a solution* to the paradoxes, before finishing his book.

His paper Mathematical logic as based on the theory of types [Russell, 1908] *(1908)*, in which a first step is made towards the Ramified Theory of Types, started with a description of the most important contradictions that were known up till then, including Russell's own paradox. He then concluded:

Prehistory of Types (vicious circle principle)

"In all the above contradictions there is a common characteristic, which we may describe as *self-reference* or *reflexiveness*. [...] In each contradiction something is said about all cases of some kind, and from what is said a new case seems to be *generated*, which both *is and is not* of the same kind as the cases of which *all* were concerned in what was said."

(*Ibid.*)

Russell's plan was, *to avoid the paradoxes* by *avoiding all possible self-references*. He postulated the *"vicious circle principle"*:

Ramified Type Theory

"*Whatever involves all of a collection* must not be one *of the collection.*"

(Mathematical logic as based on the theory of types)

- Russell applies this principle *very strictly*.
- He implemented it using *types*, in particular the so-called ramified *types*.
- The type theory of 1908 was elaborated in Chapter II of the Introduction to the famous *Principia Mathematica* [Whitehead and Russell, 1910^1, 1927^2] (1910-1912).

Ramified Type Theory and Principia

- In the *Principia*, *mathematics was founded on logic*, as far as possible.
- A *formal and accurate* build-up of mathematics, avoiding the logical paradoxes.
- The *logical part* of *Principia* was *based* on the works of *Frege* (acknowledged by Whitehead and Russell in the preface, and can be seen throughout the description of Type Theory).
- The notion of *function is based on Frege's Abstraction Principles*.
- The *Principia* *notation* $\hat{x}f(x)$ for a class looks very *similar to Frege's* $\grave{\varepsilon}f(\varepsilon)$ for course-of-values.
- An important difference is that Whitehead and Russell treated functions as first-class citizens. Frege used courses-of-values when speaking about functions.
- In the *Principia* a direct approach was possible.

Ramified Type Theory and Principia

- Type Theory had *not yet* become an *independent subject*. The theory

  > "only recommended itself to us in the first instance by its ability to solve certain contradictions. .... it has also a certain consonance with common sense which makes it inherently credible"
  >
  > (*Principia Mathematica*, p. 37)

- Type Theory was not introduced because it was interesting on its own, but because it had to serve as a *tool* for logic and mathematics.

- A *formalisation* of Type Theory, therefore, was *not considered* in those days.

- Though the *description* of the ramified type theory in the *Principia* was still informal, it was *clearly present* throughout the work.

- Types in the *Principia* have a double hierarchy: *(simple) types* and *orders*.

- It was *not mentioned very often*, but when necessary, Russell made a remark on the ramified type theory.

Ramified Type Theory and Principia

- There is *no definition of "type"* in the *Principia*, only a definition of *"being of the same type"*:

  "*Definition of* being of the same type*.* The following is a step-by-step definition, the definition for higher types presupposing that for lower types. We say that u and v *are of the same type* if

  1. both are individuals,
  2. both are elementary [propositional] functions (in Principia, they only take elementary propositions as value) taking arguments of the same type,
  3. u is a pf and v is its negation,
  4. u is $\varphi\hat{x}$ *$\varphi\hat{x}$ is a pf that has x as a free variable* or $\psi\hat{x}$, and v is $\varphi\hat{x} \vee \psi\hat{x}$, where $\varphi\hat{x}$ and $\psi\hat{x}$ are elementary pfs,
  5. u is $(y).\varphi(\hat{x}, y)$ *forall* and v is $(z).\psi(\hat{x}, z)$, where $\varphi(\hat{x}, \hat{y})$, $\psi(\hat{x}, \hat{y})$ are of the same type,

6. both are elementary propositions,
7. u is a proposition and v is $\sim u$*negation* or
8. u is $(x).\varphi x$ and v is $(y).\psi y$, where $\varphi\hat{x}$ and $\psi\hat{x}$ are of the same type."

(*Principia Mathematica*, $*9{\cdot}131$, p. 133)

- There are some omissions in Russell and Whitehead's definition.

Ramsey's Simple Types

- The ideas behind simple types was already explained by Frege (see earlier quotes from *Function and Concept*).

- *Ramsey's Simple types*:
  1. 0 is a simple type, the type of *individuals*.
  2. If $t_1, \ldots, t_n$ are simple types, then also $(t_1, \ldots, t_n)$ is a simple type.[1]
     $n = 0$ is allowed: then we obtain the simple type $()$ of *propositions*.
  3. All simple types can be constructed using the rules 1 and 2.

[1] $(t_1, \ldots, t_n)$ is the type of pfs that should take n arguments, the ith argument having type t_i.

Ramsey's Simple Types

- The propositional function $\mathtt{R}(\mathtt{x})$ should have type $(\mathtt{0})$, as it takes one individual as argument.
- The proposition $S(a)$ has type $()$.
- We conclude that in $\mathtt{z}(\mathtt{R}(\mathtt{x}), \mathtt{S}(\mathtt{a}))$, we must substitute pfs of type $((\mathtt{0}), ())$ for $\mathtt{z}$. Therefore, $\mathtt{z}(\mathtt{R}(\mathtt{x}), \mathtt{S}(\mathtt{a}))$ has type $(((\mathtt{0}), ()))$.

Whitehead and Russell's Ramified Types

- With *simple types*, the type of a pf only depends on the *types of the arguments* that it can take.

- In the *Principia*, a *second hierarchy* is introduced by regarding also *the types of the variables that are bound by a quantifier* (see *Principia*, pp. 51–55).

- Whitehead and Russell consider, for instance, the propositions $\mathtt{R}(\mathtt{a})$ and $\forall \mathtt{z}{:}()[\mathtt{z}() \vee \neg\mathtt{z}()]$ to be of a *different level*.

- The first is an *atomic proposition*, while the latter is based on the pf $\mathtt{z}() \vee \neg\mathtt{z}()$.

Whitehead and Russell's Ramified Types

- The pf $\mathrm{z}() \vee \neg \mathrm{z}()$ involves an arbitrary proposition z, therefore $\forall \mathrm{z}:()[\mathrm{z}() \vee \neg \mathrm{z}()]$ *quantifies over all* propositions z.

- According to the *vicious circle principle*, $\forall \mathrm{z}:()[\mathrm{z}() \vee \neg \mathrm{z}()]$ cannot belong to this collection of propositions.

- This problem is solved by dividing types into *orders* which are natural numbers.

- *Basic* propositions are of order 0. In $\forall \mathrm{z}:()[\mathrm{z}() \vee \neg \mathrm{z}()]$ we must mention the *order of the propositions over which is quantified*. The pf $\forall \mathrm{z}:()^n[\mathrm{z}() \vee \neg \mathrm{z}()]$ quantifies over *all propositions of order* n, and has order $n+1$.

Whitehead and Russell's Ramified Types

1. 0^0 is a ramified type of order 0;

2. If $t_1^{a_1}, \ldots, t_n^{a_n}$ are ramified types, and $a \in \mathbb{N}$, $a > \max(a_1, \ldots, a_n)$, then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a ramified type of order a (if $n = 0$ then take $a \geq 0$);

3. All ramified types can be constructed using the rules 1 and 2.

0^0; $(0^0)^1$; $\left((0^0)^1, (0^0)^4\right)^5$; and $\left(0^0, ()^2, \left(0^0, (0^0)^1\right)^2\right)^7$ are *all ramified types*.

$\left(0^0, \left(0^0, (0^0)^2\right)^2\right)^7$ is *not a ramified type*.

Predicative Types

- In the type $(0^0)^1$, all *orders are "minimal"*, i.e., not higher than strictly necessary. Unlike $(0^0)^2$ where orders are not minimal.

- Types in which all orders are minimal are called predicative and play a special role in the Ramified Theory of Types.

  1. 0^0 is a predicative type;
  2. If ${t_1}^{a_1}, \ldots, {t_n}^{a_n}$ are predicative types, and $a = 1 + \max(a_1, \ldots, a_n)$ (take $a = 0$ if $n = 0$), then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a predicative type;
  3. All predicative types can be constructed using the rules 1 and 2 above.

Problems of Ramified Type Theory

- The main part of the *Principia* is devoted to the development of logic and mathematics using the legal pfs of the ramified type theory.

- *ramification*/division of simple types into orders make RTT not easy to use.

- **(Equality)** $\mathtt{x} =_{\mathrm{L}} \mathtt{y} \stackrel{\mathrm{def}}{\leftrightarrow} \forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$.
  In order to express this general notion in RTT, we have to incorporate *all* pfs $\forall \mathtt{z} : (0^0)^n[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$ for $n > 1$, and this cannot be expressed in one pf.

- Not possible to give a constructive proof of the theorem of the least upper bound within a ramified type theory.

Axiom of Reducibility

- It is not possible in RTT to give a definition of an object that refers to the class to which this object belongs (because of the Vicious Circle Principle). Such a definition is called an *impredicative definition*.
- An object defined by an impredicative definition is of a higher order than the order of the elements of the class to which this object should belong. This means that the defined object has an *impredicative type*.
- But impredicativity is not allowed by the vicious circle principle.
- Russell and Whitehead tried to solve these problems with the so-called axiom of reducibility.

Axiom of Reducibility

- *(Axiom of Reducibility)* For each formula f, there is a formula g with a *predicative* type such that f and g are (logically) equivalent.

- The validity of the Axiom of Reducibility has been questioned from the moment it was introduced.

- In the 2nd edition of the *Principia*, Whitehead and Russell admit:

  > "This axiom has a purely pragmatic justification: it leads to the desired results, and to no others. But clearly it is not the sort of axiom with which we can rest content."
  >
  > (*Principia Mathematica*, p. xiv)

Axiom of Reducibility

- Though Weyl [Weyl, 1918] made an effort to develop analysis within the Ramified Theory of Types (without the Axiom of Reducibility),
- and various parts of mathematics can be developed within RTT and without the Axiom,
- the general attitude towards RTT (without the axiom) was that the system was too restrictive, and that a *better solution* had to be found.

Deramification

- Ramsey considers it essential to divide the paradoxes into two parts:
- One group of paradoxes is removed

  > "by pointing out that a propositional function cannot significantly take itself as argument, and by dividing functions and classes into a hierarchy of types according to their possible arguments."
  >
  > (The Foundations of Mathematics, p. 356)

  This means that a class can never be a member of itself. The paradoxes solved by introducing the hierarchy of types (but not orders), like the Russell paradox, and the Burali-Forti paradox, are logical or syntactical paradoxes;

Deramification

- The second group of paradoxes is excluded by the hierarchy of orders. These paradoxes (like the Liar's paradox, and the Richard Paradox) are based on the confusion of language and meta-language. These paradoxes are, therefore, not of a purely mathematical or logical nature. When a proper distinction between object language and meta-language is made, these so-called semantical paradoxes disappear immediately.

- Ramsey agrees with the part of the theory that eliminates the syntactic paradoxes. I.e., RTT without the orders of the types.

- The second part, the hierarchy of orders, does not gain Ramsey's support.

Deramification

- By accepting the hierarchy in its full extent one either has to accept the Axiom of Reducibility or reject ordinary real analysis.
- Ramsey is supported in his view by Hilbert and Ackermann [Hilbert and Ackermann, 1928].
- They all suggest a deramification of the theory, i.e. leaving out the orders of the types.
- When making a proper distinction between language and meta-language, the deramification will not lead to a re-introduction of the (semantic) paradoxes.

Deramification

- *Deramification* and the *Axiom of Reducibility* are both *violations of the Vicious Circle Principle*. Gödel [Gödel, 1944] fills the gap why they can be *harmlessly made*

  > "it seems that the vicious circle principle [. . .] *applies* only if the entities involved are *constructed by ourselves*. In this case there must clearly exist a *definition* (namely the description of the construction) which does *not refer to a totality* to which the object defined belongs, because the construction of a thing can certainly not be based on a totality of things to which the thing to be constructed itself belongs. If, however, it is a question of objects that *exist independently of our constructions*, there is *nothing in the least absurd in the existence of totalities* containing members, which can be described only by reference to this totality."
  >
  > (Russell's mathematical logic)

Deramification

- This turns the Vicious Circle Principle into a *philosophical principle* that will be easily *accepted by intuitionists* but that will be *rejected*, at least in its full strength, by mathematicians with a more *platonic* point of view.

- Gödel is supported in his ideas by *Quine* [Quine, 1963], sections 34 and 35.

- Quine's *criticism on impredicative* definitions (for instance, the definition of the least upper bound of a nonempty subset of the real numbers with an upper bound) is *not on the definition* of a special symbol, but rather on the very assumption of the existence of such an object at all.

Deramification

- Quine states that even for *Poincaré*, who was an *opponent of impredicative definitions and deramification*, one of the doctrines of *classes is that they are there "from the beginning"*. So, even for Poincaré *there should be no evident fallacy in impredicative definitions*.

- The *deramification* has played an *important role* in the development of *type theory*. In *1932 and 1933*, Church presented his (untyped) *λ-calculus* [Church, 1932, 1933]. *In 1940* he combined this theory with a deramified version of Russell's theory of types to the system that is known as the simply typed λ-calculus

The Simple Theory of Types

- Ramsey [Ramsey, 1926], and Hilbert and Ackermann [Hilbert and Ackermann, 1928], *simplified* the Ramified Theory of Types RTT by removing the orders. The result is known as the Simple Theory of Types *(*STT*)*.

- Nowadays, STT is known via Church's formalisation in λ-calculus. However, STT *already existed (1926) before λ-calculus did (1932)*, and is therefore not inextricably bound up with λ-calculus.

- How to obtain STT from RTT? Just *leave out all the orders* and the references to orders (including the notions of predicative and impredicative types).

Church's Simply Typed λ-calculus $\lambda\!\!\rightarrow$

- The *types* of $\lambda\!\!\rightarrow$ are defined as follows:
  - ι *individuals* and o *propositions* are types;
  - If α and β are types, then so is $\alpha \rightarrow \beta$.

- The *terms* of $\lambda\!\!\rightarrow$ are the following:
  - $\neg$, $\wedge$, $\forall_\alpha$ for each type α, and $\imath_\alpha$ for each type α, are terms;
  - A *variable* is a term;
  - If A, B are terms, then so is AB;
  - If A is a term, and x a variable, then $\lambda x{:}\alpha.A$ is a term.

- *(β)* $\qquad (\lambda x{:}\alpha.A)B \rightarrow_\beta A[x := B]$.

Typing rules in Church's Simply Typed λ-calculus $\lambda\!\rightarrow$

- $\Gamma \vdash \neg : o \rightarrow o$;

  $\Gamma \vdash \wedge : o \rightarrow o \rightarrow o$;

  $\Gamma \vdash \forall_\alpha : (\alpha \rightarrow o) \rightarrow o$;

  $\Gamma \vdash \imath_\alpha : (\alpha \rightarrow o) \rightarrow \alpha$;

- $\Gamma \vdash x : \alpha$ if $x{:}\alpha \in \Gamma$;

- If $\Gamma, x{:}\alpha \vdash A : \beta$ then $\Gamma \vdash (\lambda x{:}\alpha.A) : \alpha \rightarrow \beta$;

- If $\Gamma \vdash A : \alpha \rightarrow \beta$ and $\Gamma \vdash B : \alpha$ then $\Gamma \vdash (AB) : \beta$.

Comparing $\lambda\!\rightarrow$ with STT and RTT

- Apart from the orders, RTT *is a subsystem of* $\lambda\!\rightarrow$.

- The rules of RTT, and the method of deriving the types of pfs have a *bottom-up character*: one can only introduce a variable of a certain type in a context Γ, if there is a pf that has that type in Γ. In $\lambda\!\rightarrow$, one can introduce variables of any type without wondering whether such a type is inhabited or not.

- Church's $\lambda\!\rightarrow$ is more general than RTT in the sense that Church does not only describe (typable) *propositional* functions. In $\lambda\!\rightarrow$, also functions of type $\tau \rightarrow \iota$ (where ι is the type of individuals) can be described, and functions that take such functions as arguments, etc..

Limitation of the simply typed λ-calculus

- $\lambda\!\rightarrow$ is very restrictive.
- Numbers, booleans, the identity function have to be defined at every level.
- We can represent (and type) terms like $\lambda x : o.x$ and $\lambda x : \iota.x$.
- We cannot type $\lambda x : \alpha.x$, where α can be instantiated to any type.
- This led to new (modern) type theories that allow more general notions of functions (e.g, *polymorphic*).

The evolution of functions with Frege, Russell and Church

- Historically, functions have long been treated as a kind of meta-objects.
- Function *values* were the important part, not abstract functions.

- In the *low level/operational approach* there are only function values.
- The sine-function, is always expressed with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- In many mathematics courses, one calls $f(x)$—and not f—the function.
- Frege, Russell and Church wrote $x \mapsto x + 3$ resp. as $x + 3$, $\hat{x} + 3$ and $\lambda x.x + 3$.
- Principia's *functions are based on Frege's Abstraction Principles* but can be first-class citizens. Frege used courses-of-values to speak about functions.
- Church made every function a first-class citizen. This is rigid and does not represent the development of logic in 20th century.
- In *Principia Mathematica* [Whitehead and Russell, 1910^1, 1927^2]: If, for some a, there is a proposition ϕa, then there is a function $\phi\hat{x}$, and vice versa.
- The function ϕ is not a separate entity but always has an argument.

Functionalisation and Instantiation

[Kamareddine et al., 2003] assessed evolution of the function concept from two points of vue:

- *Functionalisation*: *the construction of a function out of an expression,* as in constructing the function $\lambda_x.x \times 3 + x$ from the expression $2 \times 3 + 2$.
- Functionalisation is
  - *Abstraction from a subexpression* e.g., moving from $2 \times 3 + 2$ to $x \times 3 + x$
  - *Function construction* e.g., turning $x \times 3 + x$ into $\lambda_x.x \times 3 + x$.
- *Instantiation*: *the calculation of a function value when a suitable argument is assigned to the function,* as in the construction of $2 \times 3 + 2$ by applying the function $\lambda_x.x \times 3 + x$ *to* 2.
- Instantiation is:
  - *Application construction* e.g., $(\lambda_x.x \times 3 + x)2$ the application of $\lambda_x.x \times 3 + x$ *to* 2
  - *Concretisation to a subexpression* e.g., calculating $(\lambda_x.x \times 3 + x)2$ to $2 \times 3 + 2$.

Functionalisation and Instantiation for Frege, Russell and Church

- Frege [Frege, 1879] focuses on abstraction from a subexpression and does not employ function construction. He does not distinguish the function $x \times 3 + x$ from the expression $x \times 3 + x$ and uses the notation $\grave{x}(x \times 3 + x)$ for what he calls the *course-of-value* of the function.

- Principia allows both parts of functionalisation and writes $\hat{x} \times 3 + \hat{x}$ for the function (see $*$9·14 and $*$9·15 of [Whitehead and Russell, 1910^1, 1927^2]).

- The λ-calculus focuses on function construction and does not employ abstraction from a subexpression. This means that we have to go all the way to obtain $\lambda_x.x \times 3 + x$. The abstraction from $2 \times 3 + 2$ to $x \times 3 + x$ is not included in the syntax.

λ-calculus does not fully represent functionalisation

1. Abstraction from a subexpression $2 + 3 \mapsto x + 3$
2. Function construction $x + 3 \mapsto \lambda x.x + 3$
3. Application construction $(\lambda x.x + 3)2$
4. Concretisation to a subexpression $(\lambda x.(x + 3))2 \rightarrow 2 + 3$

- cannot abstract only half way: $x + 3$ is not a function, $\lambda x.x + 3$ is.
- cannot apply $x + 3$ to an argument: $(x + 3)2$ does not evaluate to 2+3.

Common features of modern types and functions

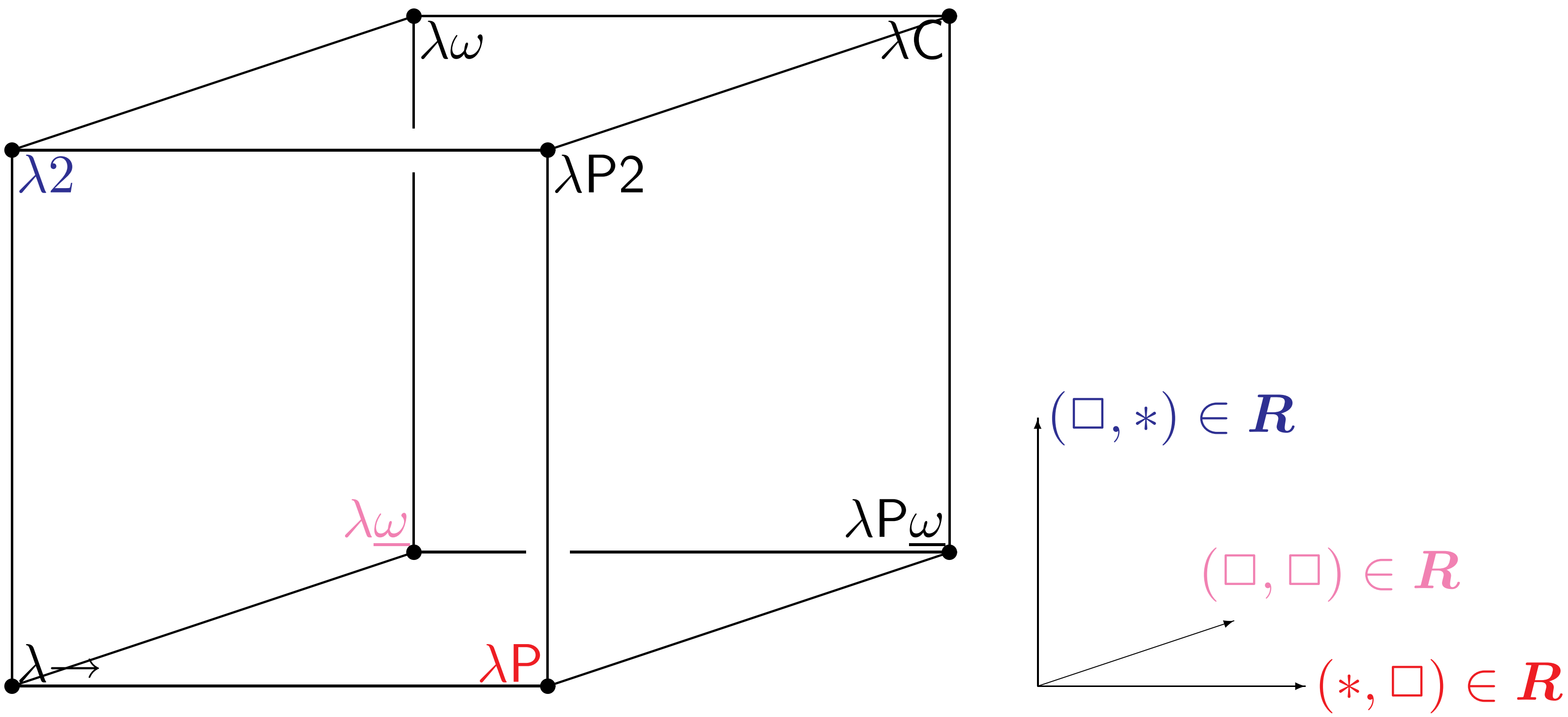
- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{y:A}.y)[A := \mathbb{N}]$ *has type* $(A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \to A)\mathbb{N} = (A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$ $\qquad\qquad$ $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$
- Write $A \to A$ as $\Pi_{y:A}.A$ when y not free in A.

The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- Formation rule: $\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2}$ if $(s_1, s_2) \in \boldsymbol{R}$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\rightarrow$	$(*,*)$				λ^τ	[Church, 1940; B
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[Girard, 1972; Re
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[Bruijn, 1968; Ha
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[Renardel de Lava
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[Longo and Mogg
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	Fω	[Girard, 1972]
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[Coquand and Hu

The Barendregt Cube



Typing Polymorphic identity needs $(\square, *)$

- $$\frac{y:*\vdash y:* \qquad y:*,x{:}y\vdash y:*}{y:*\vdash \Pi x{:}y.y:*}$$ by (Π) $(*,*)$

- $$\frac{y:*,x:y\vdash x:y \qquad y:*\vdash \Pi x{:}y.y:*}{y:*\vdash \lambda x:y.x:\Pi x{:}y.y}$$ by (λ)

- $$\frac{\vdash *:\square \qquad y:*\vdash \Pi x{:}y.y:*}{\vdash \Pi y:*.\Pi x{:}y.y:*}$$ by (Π) $(\square,*)$

- $$\frac{y:*\vdash \lambda x:y.x:\Pi x{:}y.y \qquad \vdash \Pi y:*.\Pi x{:}y.y:*}{\vdash \lambda y:*.\lambda x:y.x:\Pi y:*.\Pi x{:}y.y}$$ by (λ)

The story so far of the evolution of functions and types

- Functions have gone through a long process of evolution involving various degrees of abstraction/construction/instantiation/concretisation/evaluation.
- Types too have gone through a long process of evolution involving various degrees of abstraction/construction/instantiation/concretisation/evaluation.
- During their progress, some aspects have been added or removed.
- In this talk we argue that their progresses have been interlinked and that their abstraction/construction/instantiation/concretisation/evaluation have much in common.
- We also argue that some of the aspects that have been dismissed during their evolution need to be re-incorporated.

From the point of vue of ML

- When Robin Milner designed the language ML, he wanted to to use all of system F (the second order polymorphic λ-calculus).
- He could not do so because it was not known then whether type checking and type finding are decidable.
- So, Milner used a fragment of system F for which it was known that type checking and type finding are decidable.
- Just as well since 23 years later Wells showed that type checking and type finding in system F are undecidable.
- This meant that ML has polymorphism but not all the polymorphic power of system F.
- The question is, what system of functions and types does ML use?
- A clean answer can be given when we re-incorporate the low-level function notion used by Frege and Russell and dismissed by Church.

- ML treats let val id = (fn $x \Rightarrow x$) in (id id) end as this Cube term $(\lambda \mathsf{id}{:}(\Pi\alpha{:}*.\, \alpha \to \alpha).\, \mathsf{id}(\beta \to \beta)(\mathsf{id}\, \beta))(\lambda\alpha{:}*.\, \lambda x{:}\alpha.\, x)$

- To type this in the Cube, the $(\Box, *)$ rule is needed (i.e., $\lambda 2$).

- ML's typing rules forbid this expression:
  let val id = (fn $x \Rightarrow x$) in (fn $y \Rightarrow y\, y$)(id id) end
  Its equivalent Cube term is this well-formed typable term of $\lambda 2$:
  $(\lambda \mathsf{id} : (\Pi\alpha{:}*.\, \alpha \to \alpha).$
  $\quad (\lambda y{:}(\Pi\alpha{:}*.\, \alpha \to \alpha).\, y(\beta \to \beta)(y\, \beta))$
  $\quad (\lambda\alpha{:}*.\, \mathsf{id}(\alpha \to \alpha)(\mathsf{id}\, \alpha)))$
  $(\lambda\alpha{:}*.\, \lambda x{:}\alpha.\, x)$

- Therefore, ML should not have the full Π-formation rule $(\Box, *)$.

- ML has limited access to the rule $(\Box, *)$.

- ML's type system is none of those of the eight systems of the Cube.

- [Kamareddine et al., 2001] places the type system of ML on our refined Cube (between $\lambda 2$ and $\lambda\underline{\omega}$).

LF

- LF [Harper et al., 1987] is often described as λP of the Barendregt Cube.
- Use of Π-formation rule $(*, \Box)$ is very restricted in the practical use of LF [Geuvers, 1993].
- The only need for a type $\Pi x{:}A.B : \Box$ is when the Propositions-As-Types principle PAT is applied during the construction of the type $\Pi\alpha{:}\texttt{prop}.*$ of the operator Prf where for a proposition Σ, Prf(Σ) is the type of proofs of Σ.

$$\frac{\texttt{prop}{:}* \vdash \texttt{prop}: * \qquad \texttt{prop}{:}*, \alpha{:}\texttt{prop} \vdash *{:}\Box}{\texttt{prop}{:}* \vdash \Pi\alpha{:}\texttt{prop}.* : \Box}.$$

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used.
- But, Prf is only used when applied $\Sigma{:}\texttt{prop}$. We never use Prf on its own.
- This use is in fact based on a parametric constant rather than on Π-formation.
- Hence, the practical use of LF would not be restricted if we present Prf in a parametric form, and use $(*, \Box)$ as a parameter instead of a Π-formation rule.
- [Kamareddine et al., 2001] finds a more precise position of LF on the Cube (between $\lambda{\to}$ and λP).

Parameters: What and Why

- We speak about *functions with parameters* when referring to functions with variable values in the *low-level* approach. The x in $f(x)$ is a parameter.

- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. first-order with parameters versus second-order without [Laan and Franssen, 2001].

- Desirable properties of the lower order theory (decidability, easiness of calculations, typability) can be maintained, without losing the flexibility of the higher-order aspects.

- This low-level approach is still worthwhile for many exact disciplines. In fact, both in logic and in computer science it has certainly not been wiped out, and for good reasons.

- Parameters describe the difference between *developers* and *users* of systems.

Automath

- The first tool for mechanical representation and verification of mathematical proofs, AUTOMATH, has a parameter mechanism.

- Mathematical text in AUTOMATH written as a finite list of *lines* of the form:

  $x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T.$

  Here g is a new name, an abbreviation for the expression t of type T and $x_1, \ldots, x_n$ are the parameters of g, with respective types $A_1, \ldots, A_n$.

- Each line introduces a new definition which is inherently parametrised by the variables occurring in the context needed for it.

- Developments of ordinary mathematical theory in AUTOMATH [Benthem Jutting, 1977] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

Extending the Cube with parametric constants, see [Kamareddine et al., 2001]

- We add parametric constants of the form $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types and $c \in \mathcal{C}$.

- $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.

- $\mathbf{R}$ allows several kinds of Π-constructs. We also use a set $\boldsymbol{P}$ of (s_1, s_2) where $s_1, s_2 \in \{*, \Box\}$ to allow several kinds of parametric constants.

- $(s_1, s_2) \in \boldsymbol{P}$ means that we allow parametric constants $c(b_1, \ldots, b_n) : A$ where $b_1, \ldots, b_n$ have types $B_1, \ldots, B_n$ of sort s_1, and A is of type s_2.

- If both $(*, s_2) \in \boldsymbol{P}$ and $(\Box, s_2) \in \boldsymbol{P}$ then combinations of parameters allowed. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\Box$.

The Cube with parametric constants

- Let $(*,*) \subseteq \mathbf{R}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.

- $\lambda\mathbf{R}\boldsymbol{P} = \lambda\mathbf{R}$ and the two rules **($\vec{\mathsf{C}}$-weak)** and **($\vec{\mathsf{C}}$-app)**:

$$\frac{\Gamma \vdash b : B \qquad \Gamma, \Delta_i \vdash B_i : s_i \qquad \Gamma, \Delta \vdash A : s}{\Gamma, c(\Delta) : A \vdash b : B} \ (s_i, s) \in \boldsymbol{P}, c \text{ is } \Gamma\text{-fresh}$$

$$\frac{\begin{array}{lclr} \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n) \\ \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & A : s & (\text{if } n = 0) \end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

$\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$.
$\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$

Properties of the Refined Cube

- (Correctness of types) If $\Gamma \vdash A : B$ then $(B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort $S)$.
- (Subject Reduction SR) If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$
- (Strong Normalisation) For all $\vdash$-legal terms M, we have $\mathsf{SN}_{\twoheadrightarrow_\beta}(M)$.
- Other properties such as Uniqueness of types and typability of subterms hold.
- $\lambda\mathbf{R}\boldsymbol{P}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.
- Let $\lambda\mathbf{R}\boldsymbol{P}$ parametrically conservative (i.e., $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$).
  - The parameter-free system $\lambda\mathbf{R}$ is at least as powerful as $\lambda\mathbf{R}\boldsymbol{P}$.
  - If $\Gamma \vdash_{\mathbf{R}\boldsymbol{P}} a : A$ then $\{\Gamma\} \vdash_{\boldsymbol{R}} \{a\} : \{A\}$.

Example

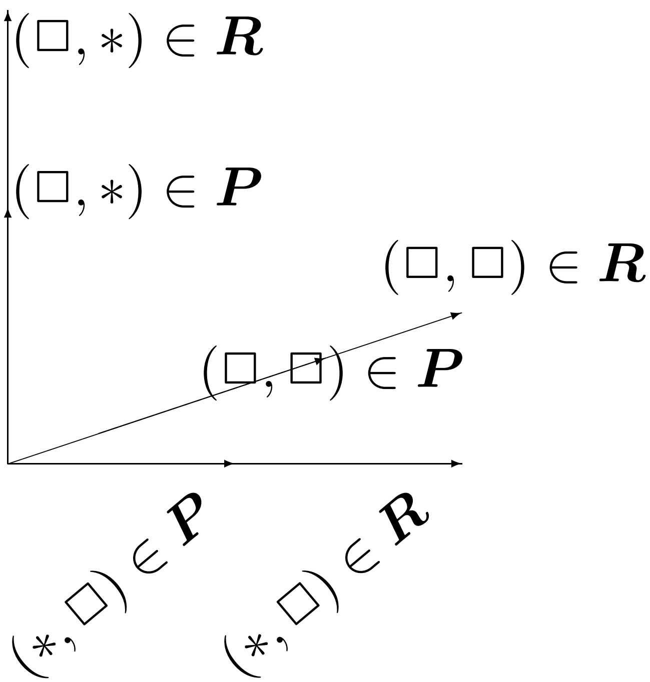
- $\boldsymbol{R} = \{(*,*),(*,\Box)\}$
  $\boldsymbol{P}_1 = \emptyset \qquad \boldsymbol{P}_2 = \{(*,*)\} \qquad \boldsymbol{P}_3 = \{(*,\Box)\} \qquad \boldsymbol{P}_4 = \{(*,*),(*,\Box)\}$
  All $\lambda\boldsymbol{R}\boldsymbol{P}_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

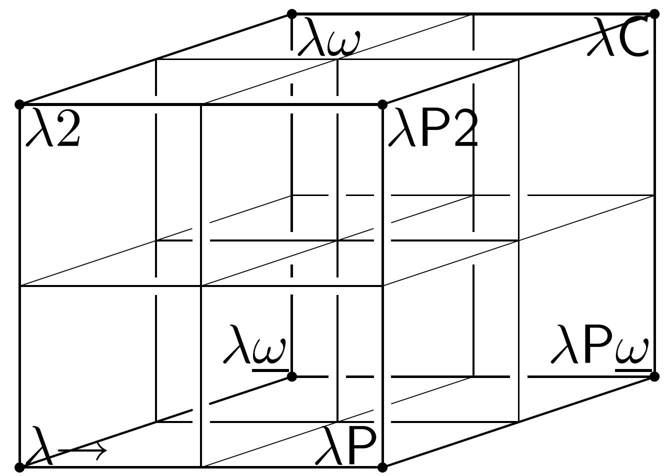
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*),(*,\Box)\}$.
  $\lambda{\to} \;<\; \lambda\boldsymbol{R}_5\boldsymbol{P}_5 \;<\; \lambda$P: we can to talk about predicates:

$$\begin{array}{rcl}
\alpha & : & *, \\
\mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\
\mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \\
\mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & : & \mathtt{eq}(\mathtt{y},\mathtt{x}), \\
\mathtt{trans}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{z}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y}), \mathtt{q}{:}\mathtt{eq}(\mathtt{y},\mathtt{z})) & : & \mathtt{eq}(\mathtt{x},\mathtt{z}).
\end{array}$$

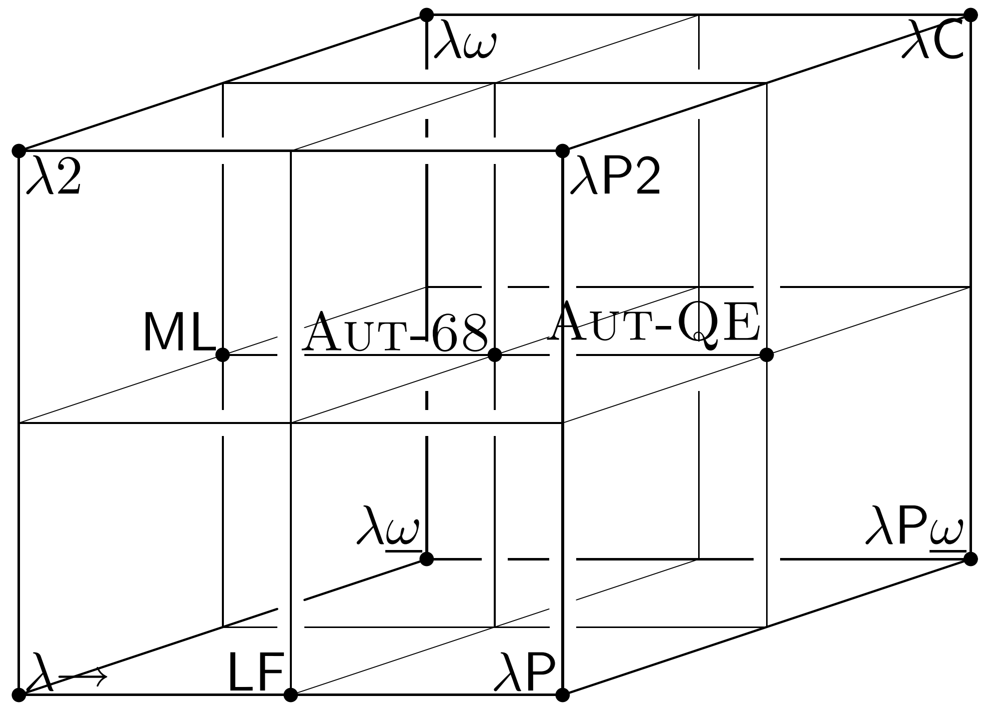
eq not possible in $\lambda{\to}$.

The refined Barendregt Cube





LF, ML, AUT-68, and AUT-QE in the refined Cube



Logicians versus mathematicians and induction over numbers

- Logician uses `ind: Ind` as proof term for an application of the induction axiom. The type `Ind` can only be described in $\lambda\boldsymbol{R}$ where $\boldsymbol{R} = \{(*,*), (*,\Box), (\Box,*)\}$:

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}{\to}*).p0{\to}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm){\to}\Pi n{:}\mathbb{N}.pn \tag{2}$$

- Mathematician uses `ind` only with $P : \mathbb{N}{\to}*$, $Q : P0$ and $R : (\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.Pn{\to}Snm{\to}Pm)$ to form a term $(\texttt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.

- The use of the induction axiom by the mathematician is better described by the parametric scheme (p, q and r are the *parameters* of the scheme):

$$\texttt{ind}(p{:}\mathbb{N}{\to}*, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm)) : \Pi n{:}\mathbb{N}.pn \tag{3}$$

- The logician's type `Ind` is not needed by the mathematician and the types that occur in 3 can all be constructed in $\lambda\boldsymbol{R}$ with $\boldsymbol{R} = \{(*,*)(*,\Box)\}$.

Logicians versus mathematicians and induction over numbers

- Mathematician: only *applies* the induction axiom and doesn't need to know the proof-theoretical backgrounds.

- A logician develops the induction axiom (or studies its properties).

- $(\Box, *)$ is not needed by the mathematician. It is needed in logician's approach in order to form the Π-abstraction $\Pi p{:}(\mathbb{N} \to *). \cdots)$.

- Consequently, the type system that is used to describe the mathematician's use of the induction axiom can be weaker than the one for the logician.

- Nevertheless, the parameter mechanism gives the mathematician limited (but for his purposes sufficient) access to the induction scheme.

Identifying λ and Π (see [Kamareddine, 2005])

- In the cube of the generalised framework of type systems, we saw that the syntax for terms (functions) and types was intermixed with the only distinction being λ- versus Π-abstraction.

- We unify the two abstractions into one.
  $\mathcal{T}_\flat \ ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{T}_\flat\mathcal{T}_\flat \mid \flat\mathcal{V}{:}\mathcal{T}_\flat.\mathcal{T}_\flat$

- $\mathcal{V}$ is a set of variables and $\boldsymbol{S} = \{*, \Box\}$.

- The β-reduction rule becomes $(\flat)$ $\qquad (\flat_{x:A}.B)C \to_\flat B[x := C]$.

- Now we also have the old Π-reduction $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$ which treats type instantiation like function instantiation.

- The type formation rule becomes

$$(\flat_1) \quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\flat x{:}A.B) : s_2} \ (s_1, s_2) \in \boldsymbol{R}$$

(axiom) $$\langle\rangle \vdash * : \Box$$

(start) $$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \; x \notin \textsc{dom}\,(\Gamma)$$

(weak) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B} \; x \notin \textsc{dom}\,(\Gamma)$$

($\flat_2$) $$\frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash (\flat x{:}A.B) : s}{\Gamma \vdash (\flat x{:}A.b) : (\flat x{:}A.B)}$$

(app$\flat$) $$\frac{\Gamma \vdash F : (\flat x{:}A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

(conv) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$$

Translations between the systems with 2 binders and those with one binder

- For $A \in \mathcal{T}$, we define $\overline{A} \in \mathcal{T}_\flat$ as follows:
  - $\overline{s} \equiv s \qquad \overline{x} \equiv x \qquad \overline{AB} \equiv \overline{A}\,\overline{B}$
  - $\overline{\lambda_{x:A}.B} \equiv \overline{\Pi_{x:A}.B} \equiv \flat_{x:\overline{A}}.\overline{B}$.
- For contexts we define: $\overline{\langle\rangle} \equiv \langle\rangle \quad \overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : \overline{A}$.
- For $A \in \mathcal{T}_\flat$, we define $[A]$ to be $\{A' \in \mathcal{T}$ such that $\overline{A'} \equiv A\}$.
- For context, obviously: $[\Gamma] \equiv \{\Gamma'$ such that $\overline{\Gamma'} \equiv \Gamma\}$.

Isomorphism of the cube and the $\flat$-cube

- If $\Gamma \vdash A : B$ then $\overline{\Gamma} \vdash_\flat \overline{A} : \overline{B}$.
- If $\Gamma \vdash_\flat A : B$ then there are unique $\Gamma' \in [\Gamma]$, $A' \in [A]$ and $B' \in [B]$ such that $\Gamma' \vdash_\pi A' : B'$.
- The $\flat$-cube enjoys all the properties of the cube except the unicity of types.

Organised multiplicity of Types for $\vdash_\flat$ and $\rightarrow_\flat$ [Kamareddine, 2005]

For many type systems, unicity of types is not necessary (e.g. Nuprl).

We have however an organised multiplicity of types.

1. If $\Gamma \vdash_\flat A : B_1$ and $\Gamma \vdash_\flat A : B_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.

2. If $\Gamma \vdash_\flat A_1 : B_1$ and $\Gamma \vdash_\flat A_2 : B_2$ and $A_1 =_\flat A_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.

3. If $\Gamma \vdash_\flat B_1 : s_1$, $B_1 =_\flat B_2$ and $\Gamma \vdash_\flat A : B_2$ then $\Gamma \vdash_\flat B_2 : s_1$.

4. Assume $\Gamma \vdash_\flat A : B_1$ and $(\Gamma \vdash_\flat A : B_1)^{-1} = (\Gamma', A', B_1')$. Then $B_1 =_\flat B_2$ if:
   (a) either $\Gamma \vdash_\flat A : B_2$, $(\Gamma \vdash_\flat A : B_2)^{-1} = (\Gamma', A'', B_2')$ and $B_1' =_\beta B_2'$,
   (b) or $\Gamma \vdash_\flat C : B_2$, $(\Gamma \vdash_\flat C : B_2)^{-1} = (\Gamma', C', B_2')$ and $A' =_\beta C'$.

Extending the cube with Π-reduction loses subject reduction [Kamareddine et al., 1999]

If we change (appl) by (new appl) in the cube we lose subject reduction.

$$\text{(appl)} \quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

$$\text{(new appl)} \quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

[Kamareddine et al., 1999] solved the problem by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function).

The same problem and solution can be repeated in our $\flat$-cube.

Adding type instantiation to the typing rules of the $\flat$-cube

If we change (app$\flat$) by (new app$\flat$) in the $\flat$-cube we lose subject reduction.

$$\text{(app}\flat\text{)} \quad \frac{\Gamma \vdash_\flat F : (\Pi_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : B[x := a]}$$

$$\text{(app}\flat\flat\text{)} \quad \frac{\Gamma \vdash_\flat F : (\flat_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : (\flat_{x:A}.B)a}$$

Failure of correctness of types and subject reduction

- **Correctness of types no longer holds.** With (appl$\flat\flat$) one can have $\Gamma \vdash A : B$ without $B \equiv \Box$ or $\exists S \,.\, \Gamma \vdash B : S$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ yet $(\flat_{y:z}.z)x \not\equiv \Box$ and $\forall s \,.\, z : *, x : z \not\vdash (\flat_{y:z}.z)x : s$.

- **Subject Reduction no longer holds.** That is, with (appl$\flat$): $\Gamma \vdash A : B$ and $A \twoheadrightarrow A'$ may not imply $\Gamma \vdash A' : B$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\flat_{y:z}.y)x \rightarrow_\flat x$, but one can't show $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.

Solving the problem

Keep all the typing rules of the $\flat$-cube the same except: replace (conv) by (new-conv), (appl$\flat$) by (appl$\flat\flat$) and add three new rules as follows:

(start-def) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A} \qquad x \notin \mathrm{DOM}\,(\Gamma)$$

(weak-def) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B} \qquad x \notin \mathrm{DOM}\,(\Gamma)$$

(def) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\flat x{:}A.C)B : D[x := B]}$$

(new-conv) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad \Gamma \vdash B =_{def} B'}{\Gamma \vdash A : B'}$$

(appl$\flat\flat$) $$\frac{\Gamma \vdash F : \flat_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\flat_{x:A}.B)a}$$

In the conversion rule, $\Gamma \vdash B =_{def} B'$ is defined as:

- If $B =_\flat B'$ then $\Gamma \vdash B =_{def} B'$
- If $x = D : C \in \Gamma$ and B' arises from B by substituting one particular free occurrence of x in B by D then $\Gamma \vdash B =_{def} B'$.
- Our 3 new rules and the definition of $\Gamma \vdash B =_{def} B'$ are trying to re-incorporate low-level aspects of functions that are not present in Church's λ-calculus.
- In fact, our new framework is closer to Frege's abstraction principle and the principles $*$9·14 and $*$9·15 of [Whitehead and Russell, 1910^1, 1927^2].

Correctness of types holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and want that for some s, $z : *, x : z \vdash (\flat_{y:z}.z)x : s$.
- Here is how the latter formula now holds:

$$
\begin{array}{ll}
z : *, x : z \vdash z : * & \text{(start and weakening)} \\
z : *, x : z.y : z\rangle x \vdash z : * & \text{(weakening)} \\
z : *, x : z \vdash (\flat_{y:z}.z)x : *[y := x] \equiv * & \text{(def rule)}
\end{array}
$$

Subject Reduction holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \to_\beta x$ and we need to show that $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.
- Here is how the latter formula now holds:

$a.$	$z : *, x : z \vdash x : z$	(start and weakening)
$b.$	$z : *, x : z \vdash (\flat_{y:z}.z)x : *$	(from 1 above)
	$z : *, x : z \vdash x : (\flat_{y:z}.z)x$	(conversion, a, b, and $z =_\beta (\flat_{y:z}.z)x$)

Common Mathematical Language of mathematicians: CML

+ CML is *expressive*: it has linguistic categories like *proofs* and *theorems*.
+ CML has been refined by intensive use and is rooted in *long traditions*.
+ CML is *approved* by most mathematicians as a communication medium.
+ CML *accommodates many branches* of mathematics, and is adaptable to new ones.

– Since CML is based on natural language, it is *informal* and *ambiguous*.

– CML is *incomplete:* Much is left implicit, appealing to the reader's intuition.

– CML is *poorly organised:* In a CML text, many structural aspects are omitted.

– CML is *automation-unfriendly:* A CML text is a plain text and cannot be easily automated.

A CML-text

From chapter 1, § 2 of E. Landau's *Foundations of Analysis* (Landau 1930, 1951).

Theorem 6. [Commutative Law of Addition]

$$x + y = y + x.$$

Proof Fix y, and let $\mathfrak{M}$ be the set of all x for which the assertion holds.
I) We have

$$y + 1 = y',$$

and furthermore, by the construction in the proof of Theorem 4,

$$1 + y = y',$$

so that

$$1 + y = y + 1$$

and 1 belongs to $\mathfrak{M}$.
II) If x belongs to $\mathfrak{M}$, then

$$x + y = y + x,$$

Therefore

$$(x + y)' = (y + x)' = y + x'.$$

By the construction in the proof of Theorem 4, we have

$$x' + y = (x + y)',$$

hence

$$x' + y = y + x',$$

so that x' belongs to $\mathfrak{M}$. The assertion therefore holds for all x. □

The problem with formal logic

- No logical language is an alternative to CML
  - A logical language does not have *mathematico-linguistic* categories, is *not universal* to all mathematicians, and is *not a good communication medium*.
  - Logical languages make fixed choices (*first versus higher order, predicative versus impredicative, constructive versus classical, types or sets*, etc.). But different parts of mathematics need different choices and there is no universal agreement as to which is the best formalism.
  - A logician reformulates in logic their *formalization* of a mathematical-text as a formal, complete text which is structured considerably *unlike* the original, and is of little use to the *ordinary* mathematician.
  - Mathematicians do not want to use formal logic and have *for centuries* done mathematics without it.

- *So, mathematicians kept to CML.*

- We would like to find an alternative to CML which avoids some of the features of the logical languages which made them unattractive to mathematicians.

What are the options for computerization?

Computers can handle mathematical text at various levels:

- Images of pages may be stored. While useful, this is not a good representation of *language* or *knowledge*.
- Typesetting systems like LaTeX, TeXmacs, can be used.
- Document representations like OpenMath, OMDoc, MathML, can be used.
- Formal logics used by theorem provers (Coq, Isabelle, HOL, Mizar, Isar, etc.) can be used.

We are gradually developing a system named Mathlang which we hope will eventually allow building a bridge between the latter 3 levels.

This talk aims at discussing the motivations rather than the details.

The issues with typesetting systems

+ A system like LaTeX, TeXmacs, provides good defaults for visual appearance, while allowing fine control when needed.

+ LaTeX and TeXmacs support commonly needed document structures, while allowing custom structures to be created.

– Unless the mathematician is amazingly disciplined, the *logical structure of symbolic formulas is not represented* at all.

– The *logical structure of mathematics as embedded in natural language text is not represented.* Automated discovery of the semantics of natural language text is still too primitive and requires human oversight.

LaTeX example

draft documents	✓
public documents	✓
computations and proofs	✗

```
\begin{theorem}[Commutative Law of Addition]\label{theorem:6}

  $$x+y=y+x.$$

\end {theorem}

\begin{proof}

  Fix $y$, and $\mathfrak{M}$ be the set of all $x$ for which
  the assertion holds.

  \begin{enumerate}

  \item We have $$y+1=y',$$
    and furthermore, by the construction in
    the proof of Theorem~\ref{theorem:4}, $$1+y=y',$$

    so that $$1+y=y+1$$
    and $1$ belongs to $\mathfrak{M}$.
```

```
  \item If $x$ belongs to $\mathfrak{M}$, then $$x+y=y+x,$$

    Therefore
    $$(x+y)'=(y+x)'=y+x'.$$

    By the construction in the proof of
    Theorem~\ref{theorem:4}, we have $$x'+y=(x+y)',$$

    hence
    $$x'+y=y+x',$$
    so that $x'$ belongs to $\mathfrak{M}$.

  \end{enumerate}

  The assertion therefore holds for all $x$.

\end{proof}
```

Full formalization difficulties: choices

A CML-text is structured differently from a fully formalized text proving the same facts. *Making the latter involves extensive knowledge and many choices:*

- The choice of the *underlying logical system.*
- The choice of *how concepts are implemented* (equational reasoning, equivalences and classes, partial functions, induction, etc.).
- The choice of the *formal foundation*: a type theory (dependent?), a set theory (ZF? FM?), a category theory? etc.
- The choice of the *proof checker*: Automath, Isabelle, Coq, PVS, Mizar, HOL, ...

An issue is that one must in general commit to one set of choices.

Full formalization difficulties: informality

Any informal reasoning in a CML-text will cause various problems when fully formalizing it:

- A single (big) step may need to expand into a (series of) syntactic proof expressions. *Very long expressions can replace a clear CML-text.*

- The entire CML-text may need *reformulation* in a fully *complete* syntactic formalism where every detail is spelled out. New details may need to be woven throughout the entire text. The text may need to be *turned inside out.*

- Reasoning may be obscured by *proof tactics*, whose meaning is often *ad hoc* and implementation-dependent.

Regardless, ordinary mathematicians do not find the new text useful.

Coq example

draft documents	✗
public documents	✗
computations and proofs	✓

From Module `Arith.Plus` of Coq standard library (`http://coq.inria.fr/`).

```
Lemma plus_sym: (n,m:nat)(n+m)=(m+n).

Proof.

Intros n m ; Elim n ; Simpl_rew ; Auto with arith.

Intros y H ; Elim (plus_n_-Sm m y) ; Simpl_rew ; Auto with arith.

Qed.
```

Mathlang's Goal: Open borders between mathematics, logic and computation

- Ordinary mathematicians *avoid* formal mathematical logic.
- Ordinary mathematicians *avoid* proof checking (via a computer).
- Ordinary mathematicians *may use* a computer for computation: there are over 1 million people who use Mathematica (including linguists, engineers, etc.).
- Mathematicians may also use other computer forms like Maple, LaTeX, etc.
- But we are not interested in only *libraries* or *computation* or *text editing*.
- We want *freedeom of movement* between mathematics, logic and computation.
- At every stage, we must have *the choice* of the level of formalilty and the depth of computation.

Aim for Mathlang? (Kamareddine and Wells 2001, 2002)

Can we formalise a mathematical text, avoiding as much as possible the ambiguities of natural language, while still guaranteeing the following four goals?

1. The formalised text looks very much like the original mathematical text (and hence the content of the original mathematical text is respected).

2. The formalised text can be fully manipulated and searched in ways that respect its mathematical structure and meaning.

3. Steps can be made to do computation (via computer algebra systems) and proof checking (via proof checkers) on the formalised text.

4. This formalisation of text is not much harder for the ordinary mathematician than LaTeX. *Full formalization down to a foundation of mathematics is not required*, although allowing and supporting this is one goal.

(No theorem prover's language satisfies these goals.)

	draft documents	✓
Mathlang	**public documents**	✓
	computations and proofs	✓

- A Mathlang text captures the grammatical and reasoning aspects of mathematical structure for further computer manipulation.

- A *weak type system* checks Mathlang documents at a grammatical level.

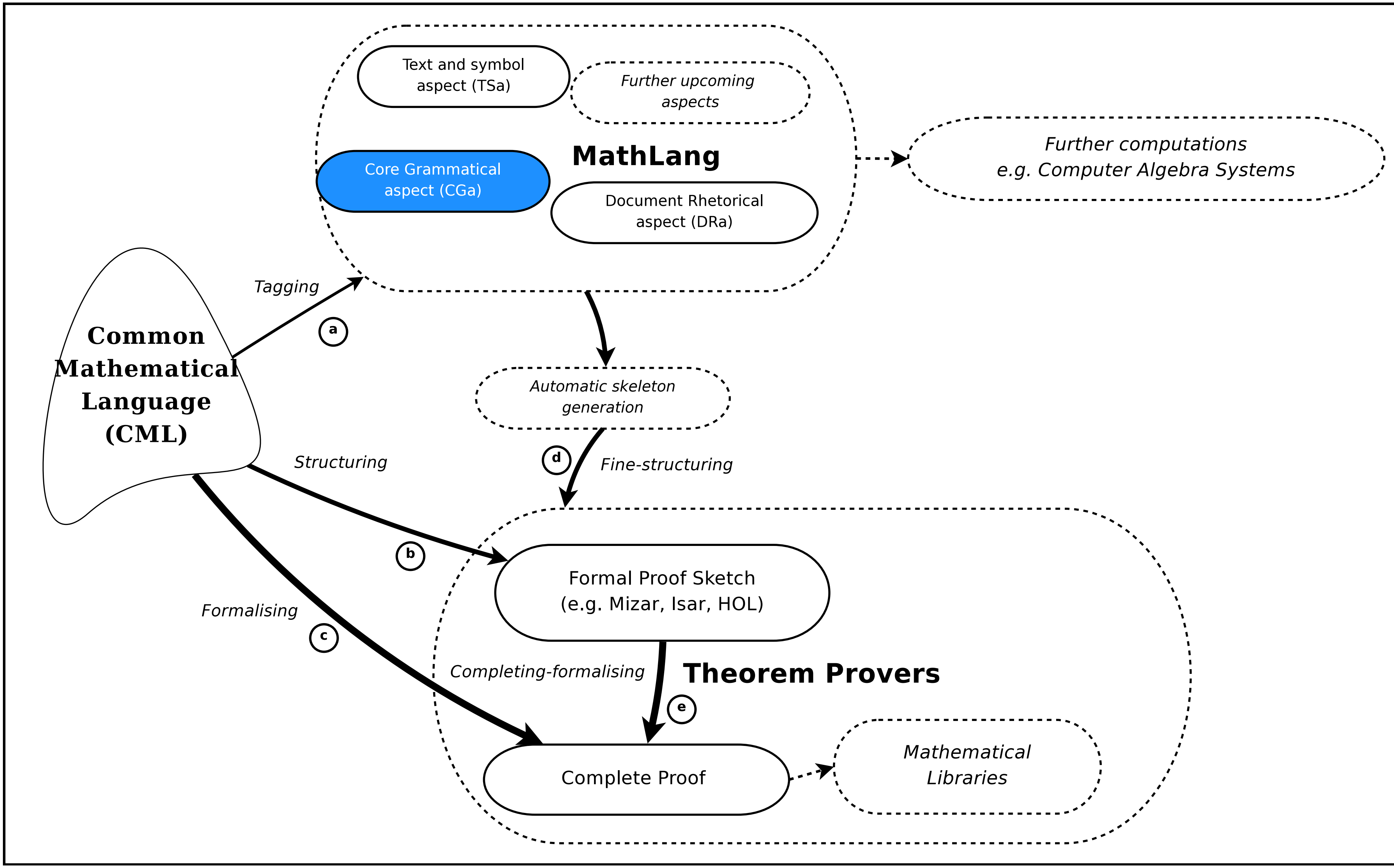
- A Mathlang text remains *close* to its CML original, allowing confidence that the CML has been captured correctly.

- We have been developing ways to weave natural language text into Mathlang.

- Mathlang aims to eventually support *all encoding uses*.

- The CML view of a Mathlang text should match the mathematician's intentions.

- The formal structure should be suitable for various automated uses.


Text and symbol aspect (TSa)
Further upcoming aspects
MathLang
Core Grammatical aspect (CGa)
Document Rhetorical aspect (DRa)
Further computations
e.g. Computer Algebra Systems
Tagging
a
Common Mathematical Language (CML)
Automatic skeleton generation
d
Fine-structuring
Structuring
b
Formal Proof Sketch
(e.g. Mizar, Isar, HOL)
Formalising
c
Completing-formalising
Theorem Provers
e
Complete Proof
Mathematical Libraries

What is CGa? (Maarek's PhD thesis)

- CGa is a formal language derived from MV (N.G. de Bruijn 1987) and WTT (Kamareddine and Nederpelt 2004) which aims at expliciting the grammatical role played by the elements of a CML text.

- The structures and common concepts used in CML are captured by CGa with a finite set of grammatical/linguistic/syntactic categories: *Term* "$\sqrt{2}$", *set* "$\mathbb{Q}$", *noun* "number", *adjective* "even", *statement* "$a = b$", *declaration* "Let a be a number", *definition* "An even number is..", *step* "a is odd, hence $a \neq 0$", *context* "Assume a is even".
  term | set | noun | adjective | statement | declaration | definition | step | context .

- Generally, each syntactic category has a corresponding *weak type*.

- CGa's type system derives typing judgments to check whether the reasoning parts of a document are coherently built.

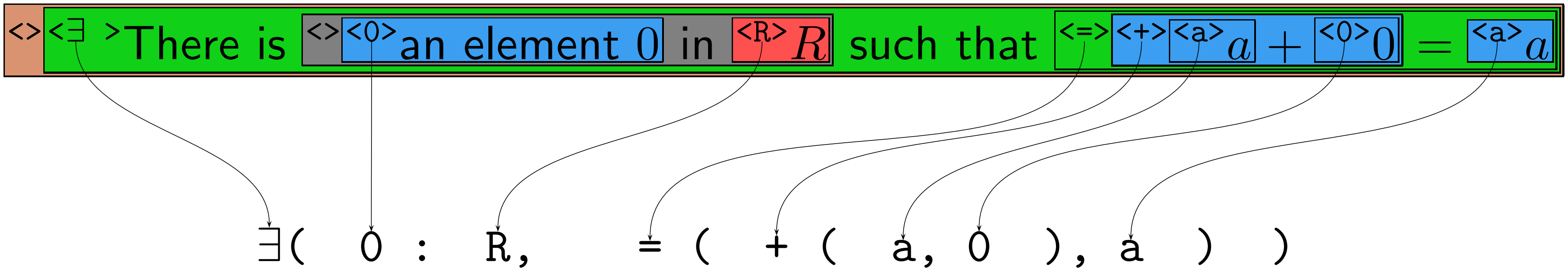


Figure 1: Example of CGa encoding of CML text

Weak Type Theory

In Weak Type Theory (or $\mathrm{W}\textsc{tt}$) we have the following linguistic categories:

- On the *atomic* level: *variables*, *constants* and *binders*,
- On the *phrase* level: *terms* $\mathcal{T}$, *sets* $\mathbb{S}$, *nouns* $\mathcal{N}$ and *adjectives* $\mathcal{A}$,
- On the *sentence* level: *statements* P and *definitions* $\mathcal{D}$,
- On the *discourse* level: *contexts* $\mathbb{\Gamma}$, *lines* $\mathbf{l}$ and *books* $\mathbf{B}$.

Categories of syntax of WTT

Other category	abstract syntax	symbol
expressions	$\mathcal{E} = T \vert \mathbb{S} \vert \mathcal{N} \vert P$	E
parameters	$\mathcal{P} = T \vert \mathbb{S} \vert P$ (note: $\overrightarrow{\mathcal{P}}$ is a list of $\mathcal{P}$s)	P
typings	$\mathbf{T} = \mathbb{S} :$ SET $\vert \mathcal{S} :$ STAT $\vert T : \mathbb{S} \vert T : \mathcal{N} \vert T : \mathcal{A}$	T
declarations	$\mathcal{Z} = \mathtt{V}^S :$ SET $\vert \mathtt{V}^P :$ STAT $\vert \mathtt{V}^T : \mathbb{S} \vert \mathtt{V}^T : \mathcal{N}$	Z

level	category	abstract syntax	symbol
atomic	*variables*	$\mathtt{V} = \mathtt{V}^T \mid \mathtt{V}^S \mid \mathtt{V}^P$	x
	constants	$\mathtt{C} = \mathtt{C}^T \mid \mathtt{C}^S \mid \mathtt{C}^N \mid \mathtt{C}^A \mid \mathtt{C}^P$	c
	binders	$\mathtt{B} = \mathtt{B}^T \mid \mathtt{B}^S \mid \mathtt{B}^N \mid \mathtt{B}^A \mid \mathtt{B}^P$	b
phrase	*terms*	$T = \mathtt{C}^T(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^T_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^T$	t
	sets	$\mathbb{S} = \mathtt{C}^S(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^S_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^S$	s
	nouns	$\mathcal{N} = \mathtt{C}^N(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^N_{\mathcal{Z}}(\mathcal{E}) \mid \mathcal{A}\mathcal{N}$	n
	adjectives	$\mathcal{A} = \mathtt{C}^A(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^A_{\mathcal{Z}}(\mathcal{E})$	a
sentence	*statements*	$P = \mathtt{C}^P(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^P_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^P$	S
	definitions	$\mathcal{D} = \mathcal{D}^\varphi \mid \mathcal{D}^P$ $\mathcal{D}^\varphi = \mathtt{C}^T(\overrightarrow{V}) := T \mid \mathtt{C}^S(\overrightarrow{V}) := \mathbb{S} \mid$ $\mathtt{C}^N(\overrightarrow{V}) := \mathcal{N} \mid \mathtt{C}^A(\overrightarrow{V}) := \mathcal{A}$ $\mathcal{D}^P = \mathtt{C}^P(\overrightarrow{V}) := P$	D
discourse	*contexts*	$\mathbb{\Gamma} = \emptyset \mid \mathbb{\Gamma}, \mathcal{Z} \mid \mathbb{\Gamma}, P$	Γ
	lines	$\mathbf{l} = \mathbb{\Gamma} \rhd P \mid \mathbb{\Gamma} \rhd \mathcal{D}$	l
	books	$\mathbf{B} = \emptyset \mid \mathbf{B} \circ \mathbf{l}$	B

Derivation rules

(1) B is a weakly well-typed book: $\vdash B :: \texttt{book}$.

(2) Γ is a weakly well-typed context relative to book B: $B \vdash \Gamma :: \texttt{cont}$.

(3) t is a weakly well-typed term, etc., relative to book B and context Γ:

$$\begin{array}{lll} B;\Gamma \vdash t :: T, & B;\Gamma \vdash s :: S, & B;\Gamma \vdash n :: N, \\ B;\Gamma \vdash a :: A, & B;\Gamma \vdash p :: P, & B;\Gamma \vdash d :: D \end{array}$$

$OK(B;\Gamma)$. stands for: $\vdash B :: \texttt{book}$, *and* $B \vdash \Gamma :: \texttt{cont}$

Examples of derivation rules

- $\mathtt{dvar}(\emptyset) = \emptyset \qquad \mathtt{dvar}(\Gamma', x : W) = \mathtt{dvar}(\Gamma'), x \qquad \mathtt{dvar}(\Gamma', P) = \mathtt{dvar}(\Gamma')$

$$\frac{OK(B;\Gamma), \quad x \in \mathtt{V}^{\mathtt{T/S/P}}, \quad x \in \mathtt{dvar}(\Gamma)}{B;\Gamma \ \vdash \ x :: T/S/P} \quad (var)$$

$$\frac{B;\Gamma \ \vdash \ n :: N, \quad B;\Gamma \ \vdash \ a :: A}{B;\Gamma \ \vdash \ an :: N} \quad (adj{-}noun)$$

$$\frac{}{\vdash \ \emptyset :: \mathsf{book}} \quad (emp{-}book)$$

$$\frac{B;\Gamma \ \vdash \ p :: P}{\vdash \ B \circ \Gamma \triangleright p :: \mathsf{book}} \qquad \frac{B;\Gamma \ \vdash \ d :: D}{\vdash \ B \circ \Gamma \triangleright d :: \mathsf{book}} \quad (book{-}ext)$$

Properties of WTT

- *Every variable is declared* If $B; \Gamma \vdash \Phi :: \mathbf{W}$ then $FV(\Phi) \subseteq \mathtt{dvar}(\Gamma)$.
- *Correct subcontexts* If $B \vdash \Gamma :: \mathtt{cont}$ and $\Gamma' \subseteq \Gamma$ then $B \vdash \Gamma' :: \mathtt{cont}$.
- *Correct subbooks* If $\vdash B :: \mathtt{book}$ and $B' \subseteq B$ then $\vdash B' :: \mathtt{book}$.
- *Free constants are either declared in book or in contexts* If $B; \Gamma \vdash \Phi :: \mathbf{W}$, then $FC(\Phi) \subseteq \mathtt{prefcons}(B) \cup \mathtt{defcons}(B)$.
- *Types are unique* If $B; \Gamma \vdash A :: \mathbf{W_1}$ and $B; \Gamma \vdash A :: \mathbf{W_2}$, then $\mathbf{W_1} \equiv \mathbf{W_2}$.
- *Weak type checking is decidable* there is a decision procedure for the question $B; \Gamma \vdash \Phi :: \mathbf{W}$?.
- *Weak typability is computable* there is a procedure deciding whether an answer exists for $B; \Gamma \vdash \Phi :: ?$ and if so, delivering the answer.

Definition unfolding

- Let $\vdash B$ :: `book` and $\Gamma \rhd c(x_1, \ldots, x_n) := \Phi$ a line in B.
- We write $B \vdash c(P_1, \ldots, P_n) \xrightarrow{\delta} \Phi[x_i := P_i]$.
- *Church-Rosser* If $B \vdash \Phi \stackrel{\delta}{\twoheadrightarrow} \Phi_1$ and $B \vdash \Phi \stackrel{\delta}{\twoheadrightarrow} \Phi_2$ then there exists Φ_3 such that $B \vdash \Phi_1 \stackrel{\delta}{\twoheadrightarrow} \Phi_3$ andf $B \vdash \Phi_2 \stackrel{\delta}{\twoheadrightarrow} \Phi_3$.
- *Strong Normalisation* Let $\vdash B$ :: `book`. For all subformulas Ψ occurring in B, relation $\xrightarrow{\delta}$ is strongly normalizing (i.e., definition unfolding inside a well-typed book is a well-founded procedure).

CGa Weak Type Checking

Let $\mathfrak{M}$ be a set ,

y and x are natural numbers ,

if x belongs to $\mathfrak{M}$

then $x + y = y + x$

CGa Weak Type checking detects grammatical errors

Let $\mathfrak{M}$ be a set,

y and x are natural numbers,

if x belongs to $\mathfrak{M}$

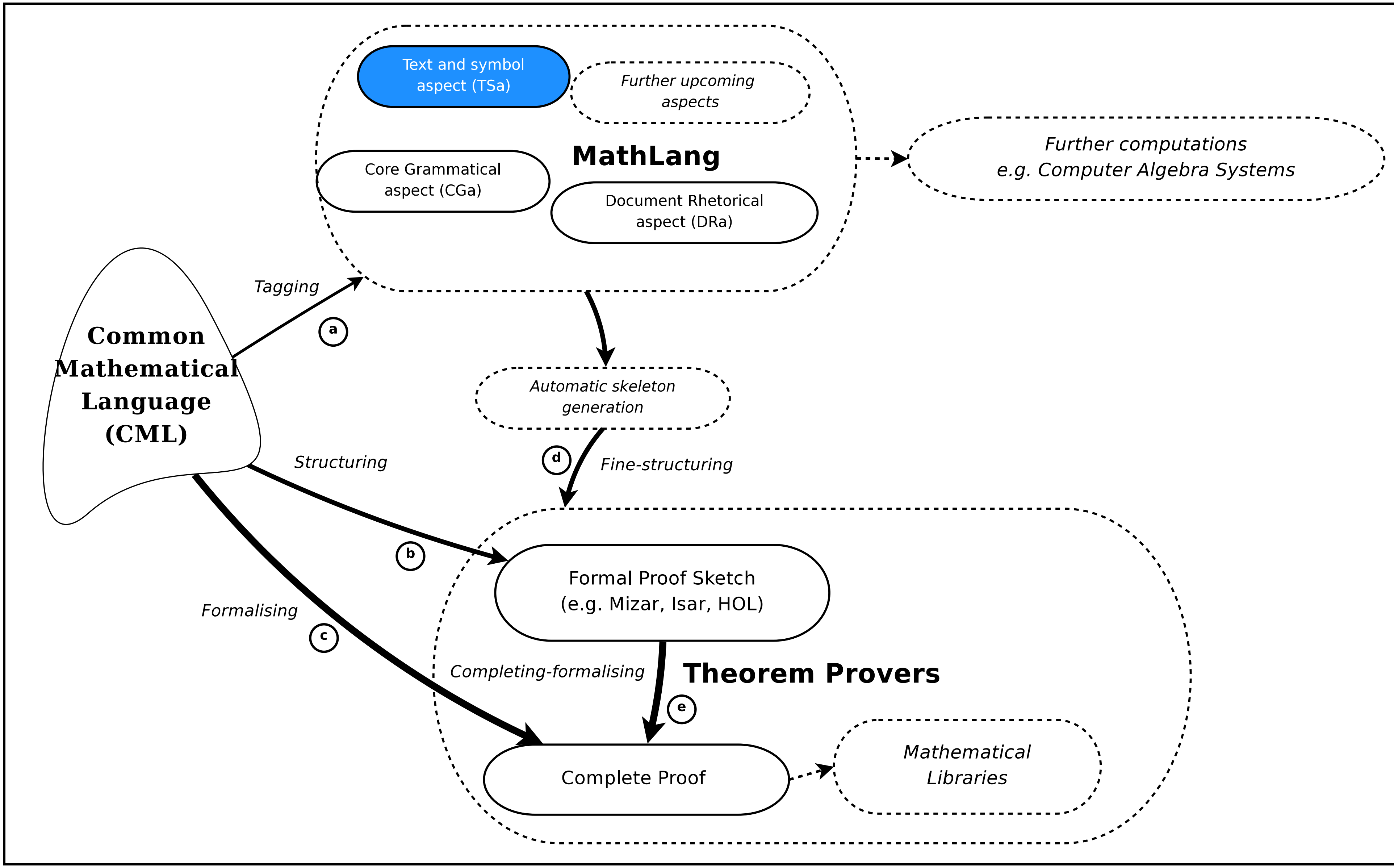
then $x + y$ $\Leftarrow$ **error**

How complete is the CGa?

- CGa is quite advanced but remains under development according to new translations of mathematical texts. Are the current CGa categories sufficient?

- The metatheory of WTT has been established in (Kamareddine and Nederepelt 2004). That of CGa remains to be established. However, since CGa is quite similar to WTT, its metatheory might be similar to that of WTT.

- The type checker for CGa works well and gives some useful error messages. Error messages should be improved.


Text and symbol aspect (TSa)
Further upcoming aspects
MathLang
Core Grammatical aspect (CGa)
Document Rhetorical aspect (DRa)
Further computations e.g. Computer Algebra Systems
Tagging
a
Common Mathematical Language (CML)
Automatic skeleton generation
d
Fine-structuring
Structuring
b
Formal Proof Sketch (e.g. Mizar, Isar, HOL)
Formalising
c
Completing-formalising
Theorem Provers
e
Complete Proof
Mathematical Libraries

What is TSa? Lamar's PhD thesis

- TSa builds the bridge between a CML text and its grammatical interpretation and adjoins to each CGa expression a string of words and/or symbols which aims to act as its CML representation.

- TSa plays the role of a user interface

- TSa can flexibly represent natural language mathematics.

- The author wraps the natural language text with boxes representing the grammatical categories (as we saw before).

- The author can also give interpretations to the parts of the text.

Interpretations

There is ^{0}an element 0 in ${}^{\mathtt{R}}R$ such that $\mathtt{eq}\ \mathtt{plus}\ {}^{\mathtt{a}}a + {}^{0}0 = {}^{\mathtt{a}}a$.

```
{  0 :  R;    eq (  plus (  a, 0  ), a  );  };
```

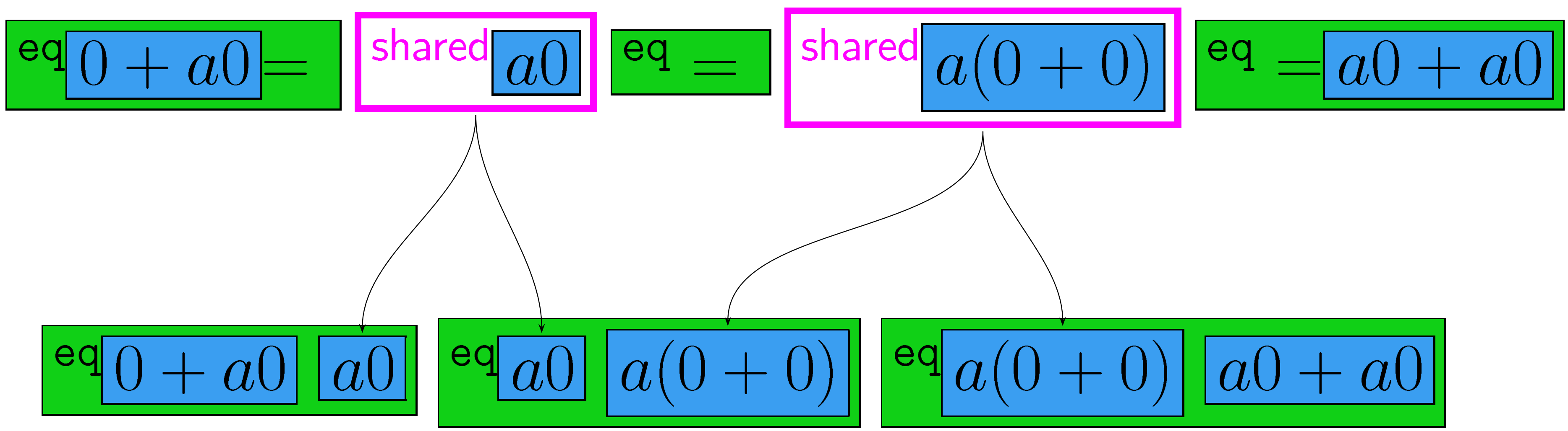
There is ^{0}an element 0 in ${}^{\mathtt{R}}R$ such that $\mathtt{eq}\ \mathtt{plus}\ {}^{\mathtt{a}}a + {}^{0}0 = {}^{\mathtt{a}}a$.

There is ^{0}an element 0 in ${}^{\mathtt{R}}R$ such that $\mathtt{eq}\ \mathtt{plus}\ {}^{\mathtt{a}}a + {}^{0}0$ equals ${}^{\mathtt{a}}a$.

${}^{0}0 \in {}^{\mathtt{R}}R$, $\mathtt{eq}\ \mathtt{plus}\ {}^{\mathtt{a}}a + {}^{0}0 = {}^{\mathtt{a}}a$.

Rewrite rules enable natural language representation

Take the example $0 + a0 = a0 = a(0 + 0) = a0 + a0$



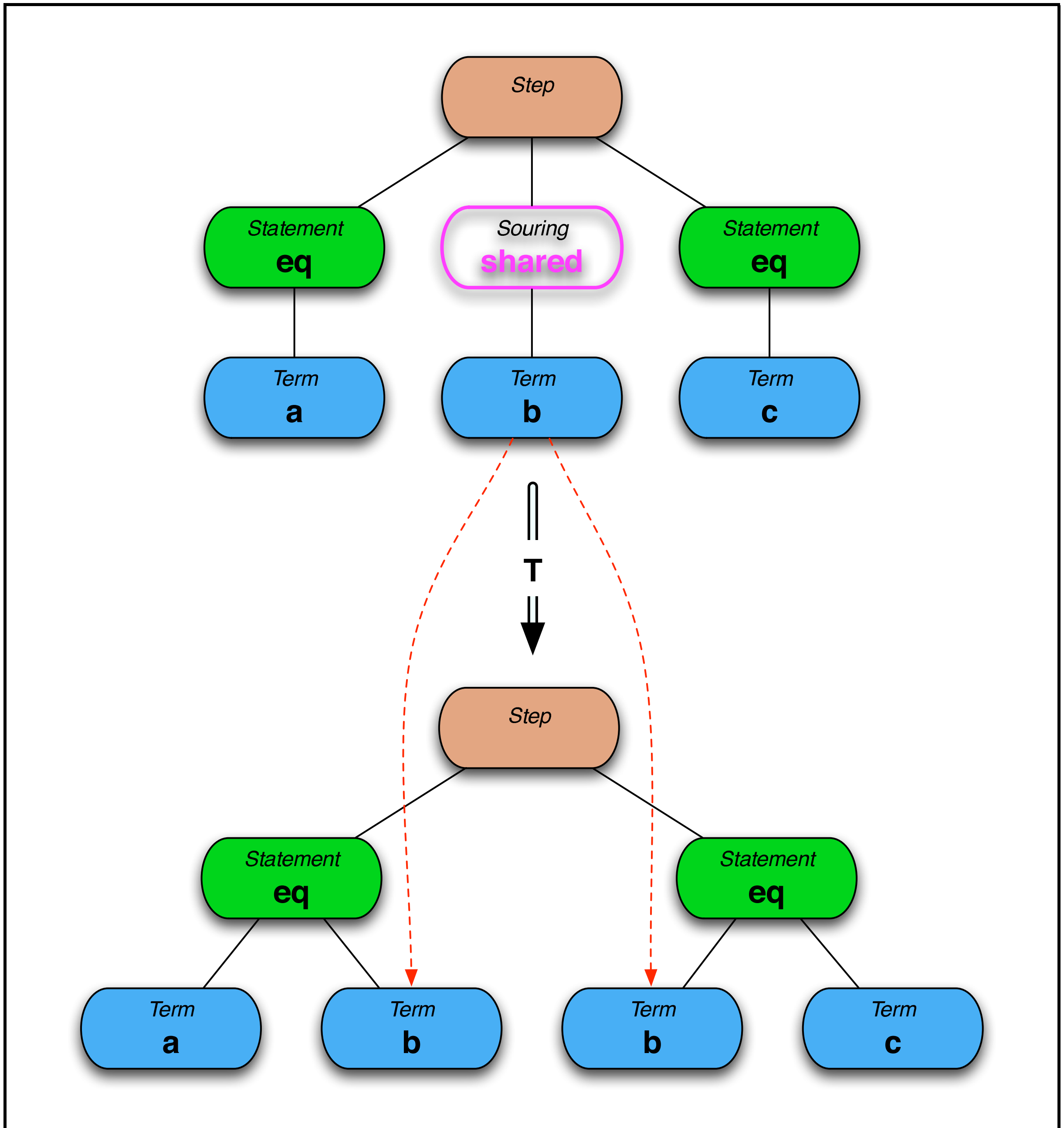


Figure 2: Example for a simple shared souring

reordering/position Souring

<in> <n> n $\in$ <N> $\mathbb{N}$

$ann =$ <in> <2> <N> $\mathbb{N}$ contains <1> <n> n

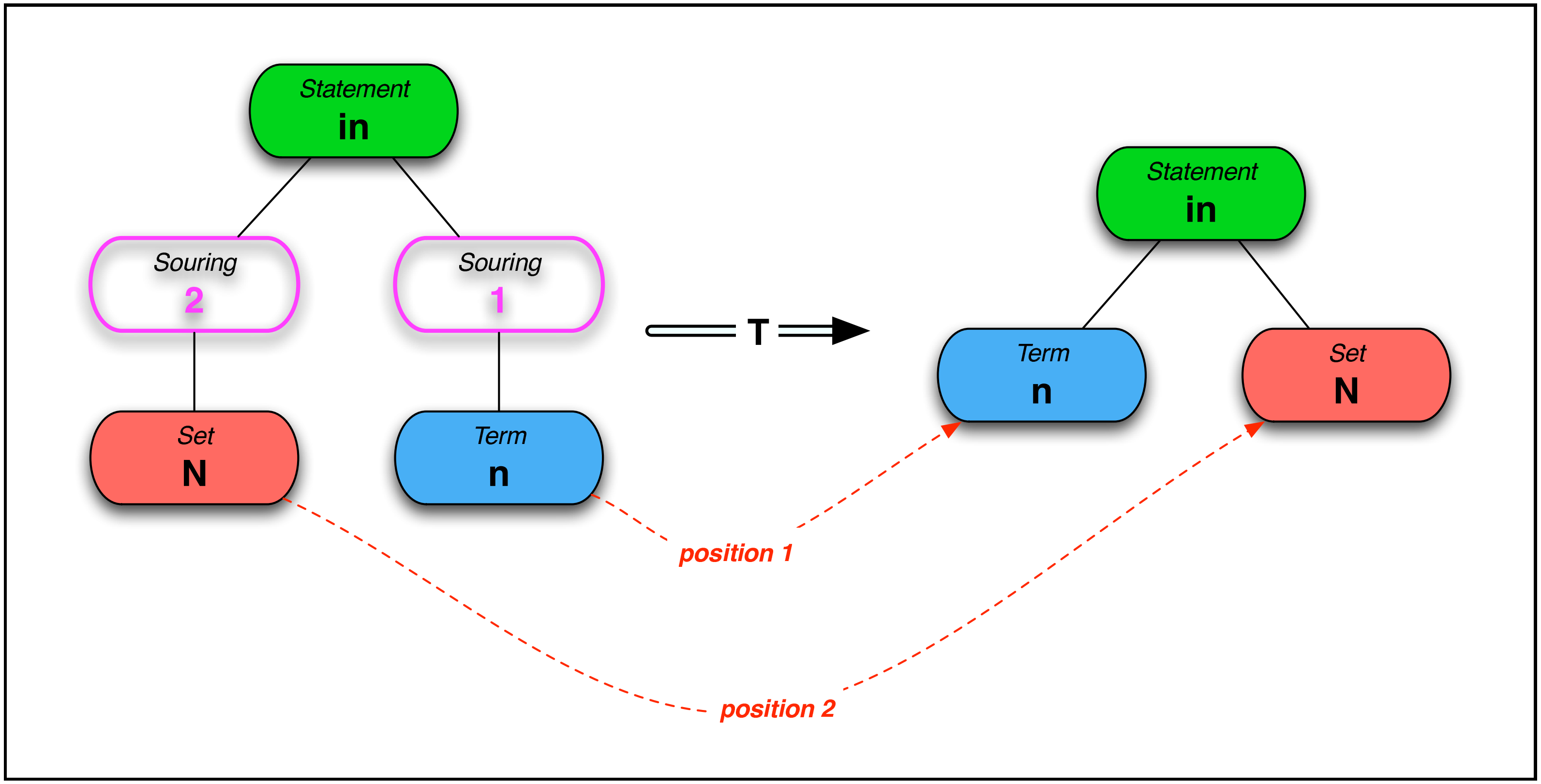
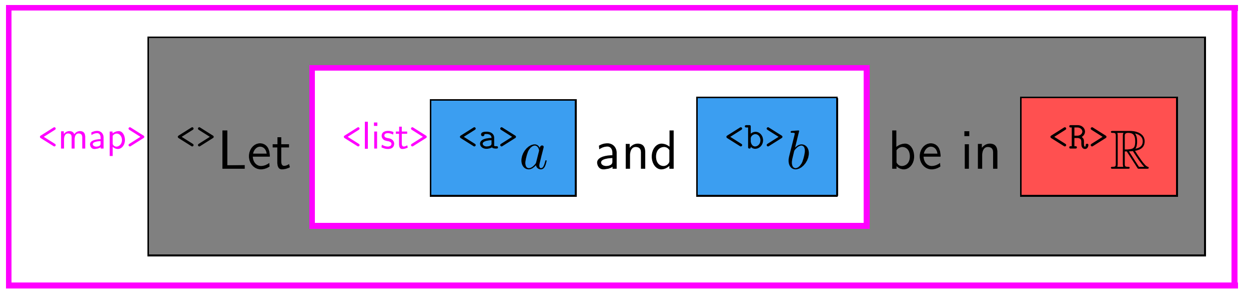


Figure 3: Example for a position souring

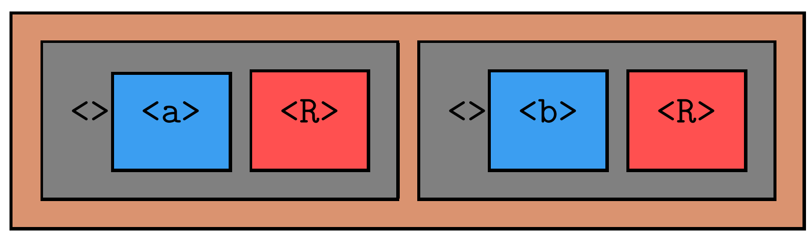
map souring

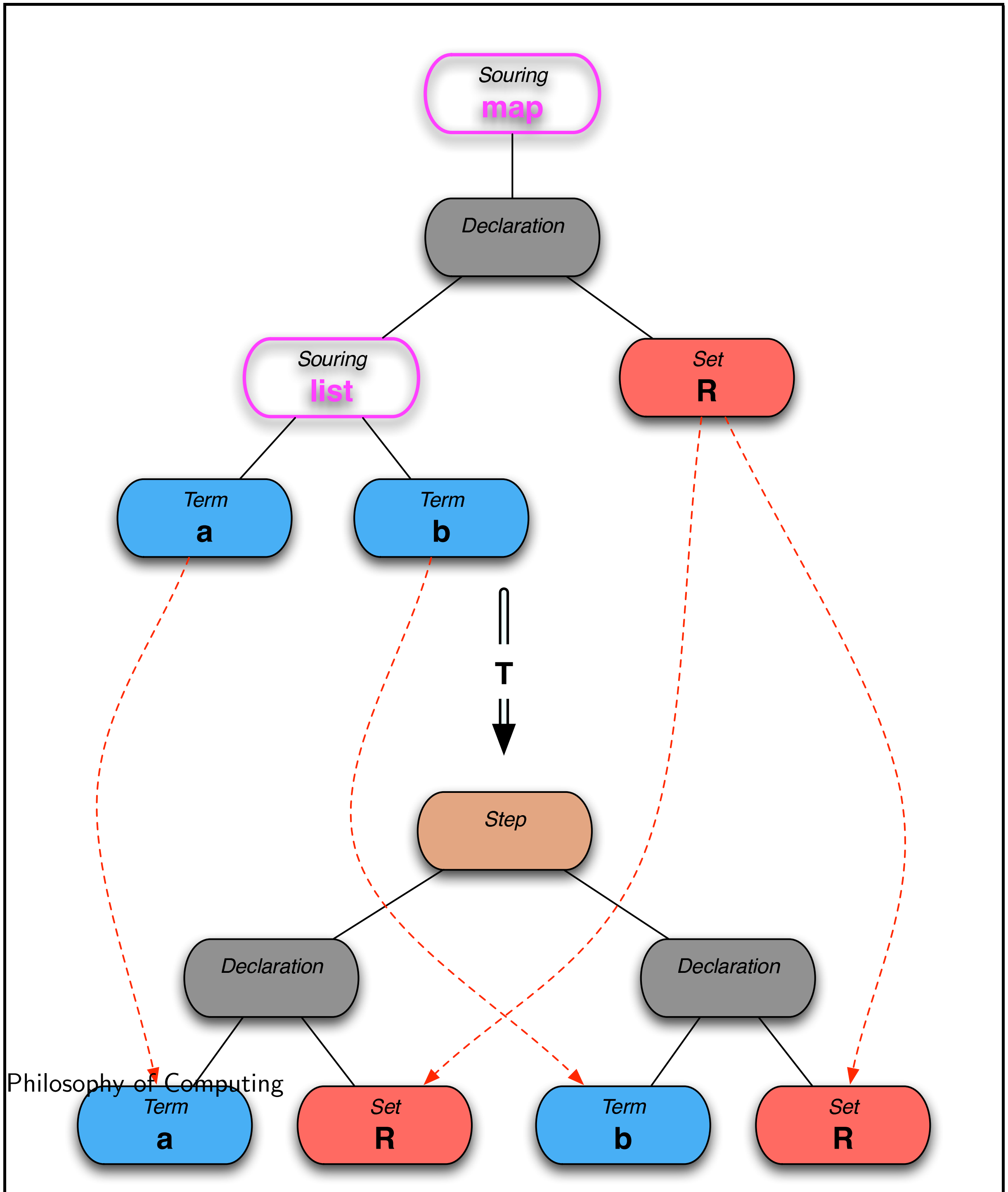
$ann =$



This is expanded to

$\mathbf{T}\,(ann) =$



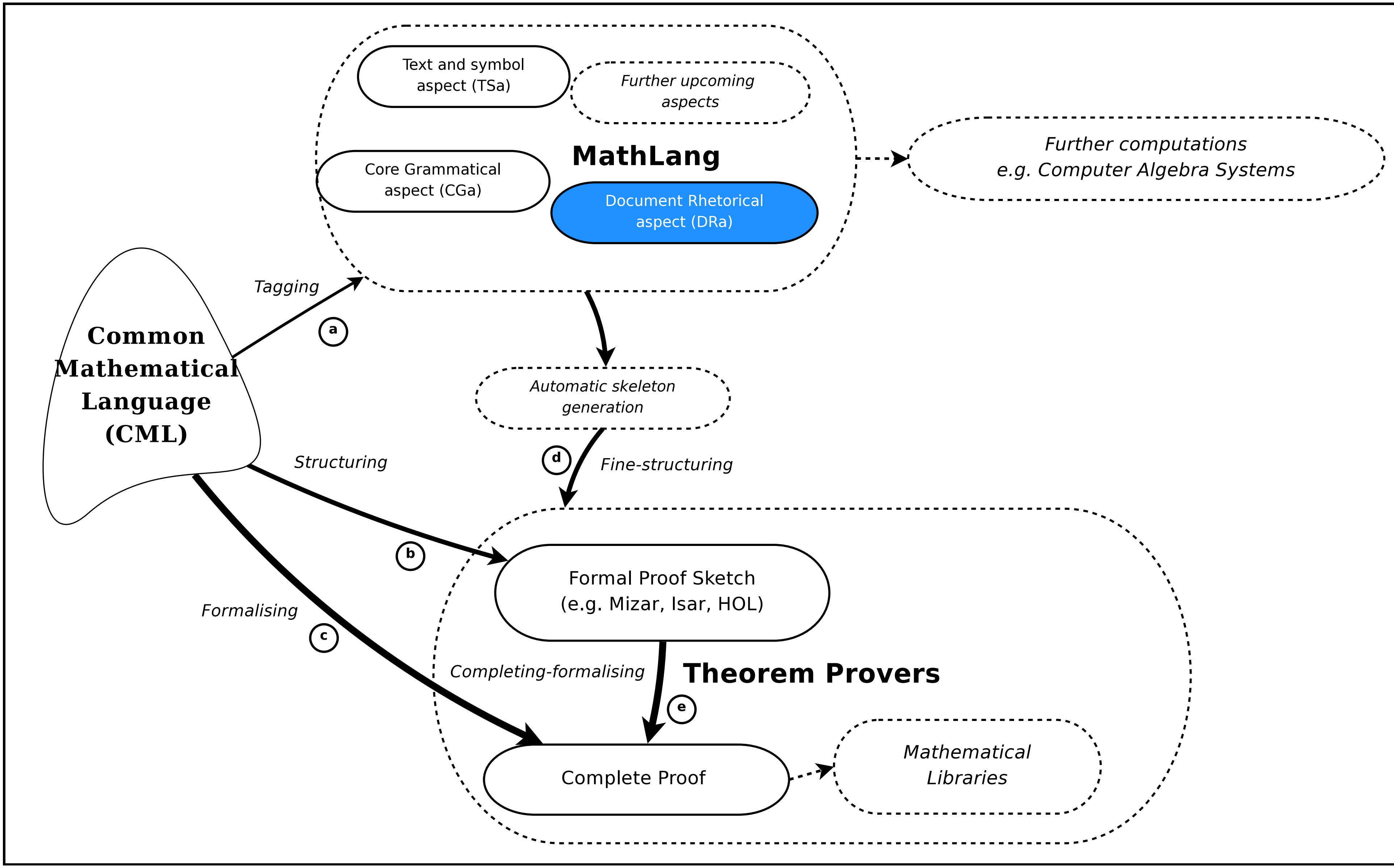

Souring
map
Declaration
Souring
list
Set
R
Term
a
Term
b
T
Step
Declaration
Declaration
Term
a
Set
R
Term
b
Set
R

How complete is TSa?

- TSa provides useful interface facilities but it is still under development.

- So far, only simple rewrite (souring) rules are used and they are not comprehensive. E.g., unable to cope with things like $\overbrace{x = \ldots = x}^{n \text{ times}}$.

- The TSa theory and metatheory need development.



Text and symbol aspect (TSa)
Further upcoming aspects
MathLang
Core Grammatical aspect (CGa)
Document Rhetorical aspect (DRa)
Further computations e.g. Computer Algebra Systems
Tagging
a
Common Mathematical Language (CML)
Automatic skeleton generation
d
Fine-structuring
Structuring
b
Formal Proof Sketch (e.g. Mizar, Isar, HOL)
Formalising
c
Completing-formalising
Theorem Provers
e
Complete Proof
Mathematical Libraries

What is DRa? Retel's PhD thesis

- DRa Document Rhetorical structure aspect.
- **Structural components of a document** like *chapter, section, subsection, etc.*
- **Mathematical components of a document** like *theorem, corollary, definition, proof, etc.*
- **Relations** between above components.
- These enhance readability, and ease the navigation of a document.
- Also, these help to go into more formal versions of the document.

Relations

Description
Instances of the StructuralRhetoricalRole *class:* preamble, part, chapter, section, paragraph, *etc.*
Instances of the MathematicalRhetoricalRole *class:* lemma, corollary, theorem, conjecture, definition, axiom, claim, proposition, assertion, proof, exercise, example, problem, solution, *etc.*

Relation
Types of relations: relatesTo, uses, justifies, subpartOf, inconsistentWith, exemplifies

What does the mathematician do?

- The mathematician wraps into boxes and uniquely names chunks of text

- The mathematician assigns to each box the structural and/or mathematical rhetorical roles

- The mathematician indicates the relations between wrapped chunks of texts

Lemma 1. For $m, n \in \mathbb{N}$ one has: $m^2 = 2n^2 \implies m = n = 0$.
Define on $\mathbb{N}$ the predicate:

$$P(m) \iff \exists n. m^2 = 2n^2 \ \& \ m > 0.$$

Claim. $P(m) \implies \exists m' < m. P(m')$. Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ and we have

$$2n^2 = m^2 = 4k^2 \implies n^2 = 2k^2$$

Since $m > 0$, if follows that $m^2 > 0, n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$.

By the claim $\forall m \in \mathbb{N}. \neg P(m)$, since there are no infinite descending sequences of natural numbers.

Now suppose $m^2 = 2n^2$ with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction. Therefore $m = 0$. But then also $n = 0$.

Corollary 1. $\sqrt{2} \notin \mathbb{Q}$.
Suppose $\sqrt{2} \in \mathbb{Q}$, i.e. $\sqrt{2} = p/q$ with $p \in \mathbb{Z}, q \in \mathbb{Z} - \{0\}$. Then $\sqrt{2} = m/n$ with $m = |p|, n = |q| \neq 0$. It follows that $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \notin \mathbb{Q}$.

Lemma 1.

For $m, n \in \mathbb{N}$ one has: $m^2 = 2n^2 \implies m = n = 0$

Proof.

Define on $\mathbb{N}$ the predicate:

$$P(m) \iff \exists n.m^2 = 2n^2 \ \& \ m > 0.$$

Claim. $P(m) \implies \exists m' < m.P(m')$.

Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ and we have $2n^2 = m^2 = 4k^2 \implies n^2 = 2k^2$ Since $m > 0$, if follows that $m^2 > 0$, $n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$.

By the claim $\forall m \in \mathbb{N}.\neg P(m)$, since there are no infinite descending sequences of natural numbers.

Now suppose $m^2 = 2n^2$

with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction.

Therefore $m = 0$. But then also $n = 0$. □

Corollary 1. $\sqrt{2} \notin \mathbb{Q}$

Proof. Suppose $\sqrt{2} \in \mathbb{Q}$, i.e. $\sqrt{2} = p/q$ with $p \in \mathbb{Z}, q \in \mathbb{Z} - \{0\}$. Then $\sqrt{2} = m/n$ with $m = |p|, n = |q| \neq 0$. It follows that $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \notin \mathbb{Q}$. □

$(A$, hasMathematicalRhetoricalRole, *lemma*$)$
$(E$, hasMathematicalRhetoricalRole, *definition*$)$
$(F$, hasMathematicalRhetoricalRole, *claim*$)$
$(G$, hasMathematicalRhetoricalRole, *proof*$)$
$(B$, hasMathematicalRhetoricalRole, *proof*$)$
$(H$, hasOtherMathematicalRhetoricalRole, *case*$)$
$(I$, hasOtherMathematicalRhetoricalRole, *case*$)$
$(C$, hasMathematicalRhetoricalRole, *corollary*$)$
$(D$, hasMathematicalRhetoricalRole, *proof*$)$

$(B$, justifies, $A)$
$(D$, justifies, $C)$
$(D$, uses, $A)$
$(G$, uses, $E)$
$(F$, uses, $E)$
$(H$, uses, $E)$
$(H$, subpartOf, $B)$
$(H$, subpartOf, $I)$

Lemma 1.

For $m, n \in \mathbb{N}$ one has: $m^2 = 2n^2 \implies m = n = 0$

Proof.

Define on $\mathbb{N}$ the predicate:

$$P(m) \iff \exists n. m^2 = 2n^2 \ \& \ m > 0.$$

Claim. $P(m) \implies \exists m' < m. P(m')$.

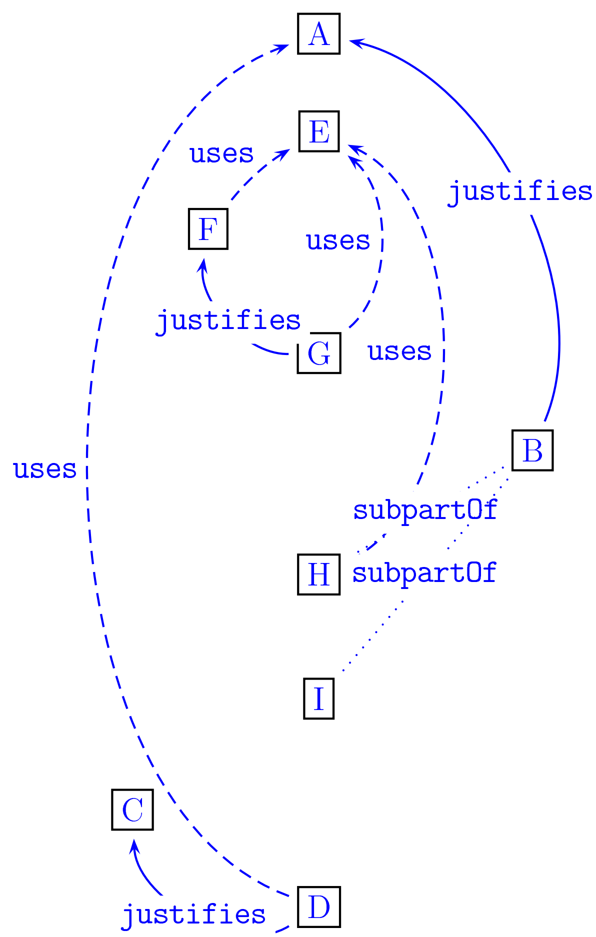
Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ and we have $2n^2 = m^2 = 4k^2 \implies n^2 = 2k^2$ Since $m > 0$, it follows that $m^2 > 0$, $n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$.

By the claim $\forall m \in \mathbb{N}. \neg P(m)$, since there are no infinite descending sequences of natural numbers.

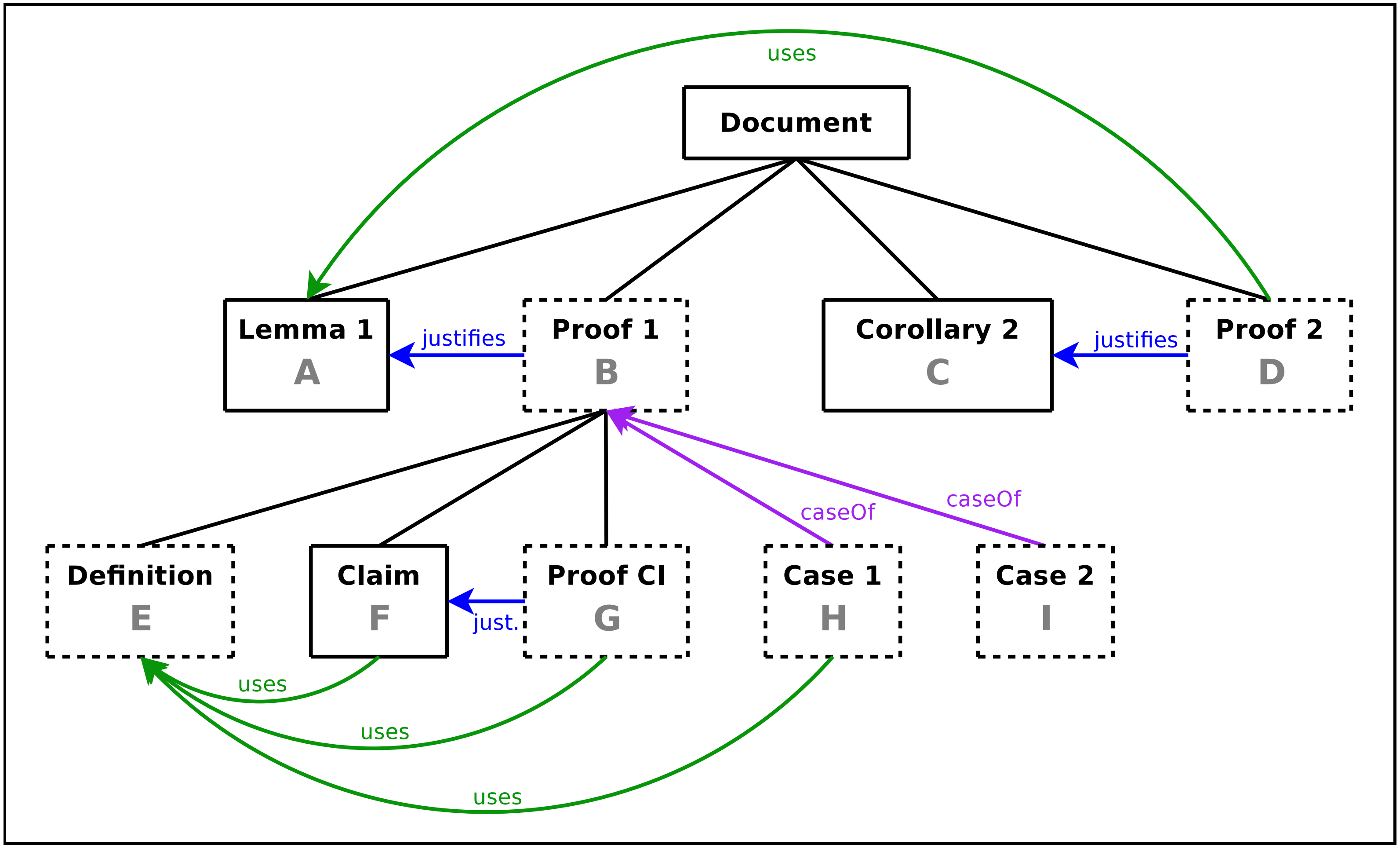
Now suppose $m^2 = 2n^2$ with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction.

The automatically generated dependency Graph

Dependency Graph (DG)



An alternative view of the DRa (Zengler's thesis)



The Graph of Textual Order: GoTO
Zengler's thesis

- To be able to examine the proper structure of a DRa tree we introduce the concept of textual order between two nodes in the tree.
- Using textual orders, we can transform the dependency graph into a GoTO by transforming each edge of the DG.
- So far there are two reasons why the GoTO is produced:
  1. Automatic Checking of the GoTO can reveal errors in the document (e.g. loops in the structure of the document).
  2. The GoTO is used to automatically produce a proof skeleton for a prover (we use a variety: Isabelle, Mizar, Coq).
- We automatically transform a DG into GoTO and automatically check the GoTO for errors in the document:

1. Loops in the GoTO (error)
2. Proof of an unproved node (error)
3. More than one proof for a proved node (warning)
4. Missing proof for a proved node (warning)

- To achieve this we define for each vertex v of the tree:
  - $\mathcal{ENV}v$ is the environment of all mathematical statements that occur before the statements of v (from the root vertex).
  - Introduced symbols':
    $\mathcal{IN}v := \mathcal{DF}v \cup \mathcal{DC}v \cup \{s|s \in \mathcal{ST}v \wedge s \notin \mathcal{ENV}v\} \cup \bigcup_{c\ childOf\ v} \mathcal{IN}c$
  - Used symbol: $\mathcal{USE}v := \mathcal{T}v \cup \mathcal{S}v \cup \mathcal{N}v \cup \mathcal{A}v \cup \mathcal{ST}v \cup \bigcup_{c\ childOf\ v} \mathcal{USE}c$

- Strong textual order $\prec$: $B \prec A := \exists x(x \in \mathcal{IN}B \wedge x \in \mathcal{USE}A)$

- Weak textual order $\preceq$: $A \preceq B := \mathcal{IN}A \subseteq \mathcal{IN}B \wedge \mathcal{USE}A \subseteq \mathcal{USE}B$

- Common textual order $\leftrightarrow$: $A \leftrightarrow B := \exists x(x \in \mathcal{USE}A \wedge x \in \mathcal{USE}B)$

Graph of Textual Order

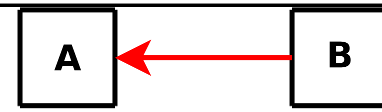
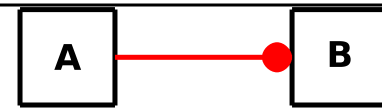
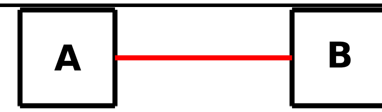
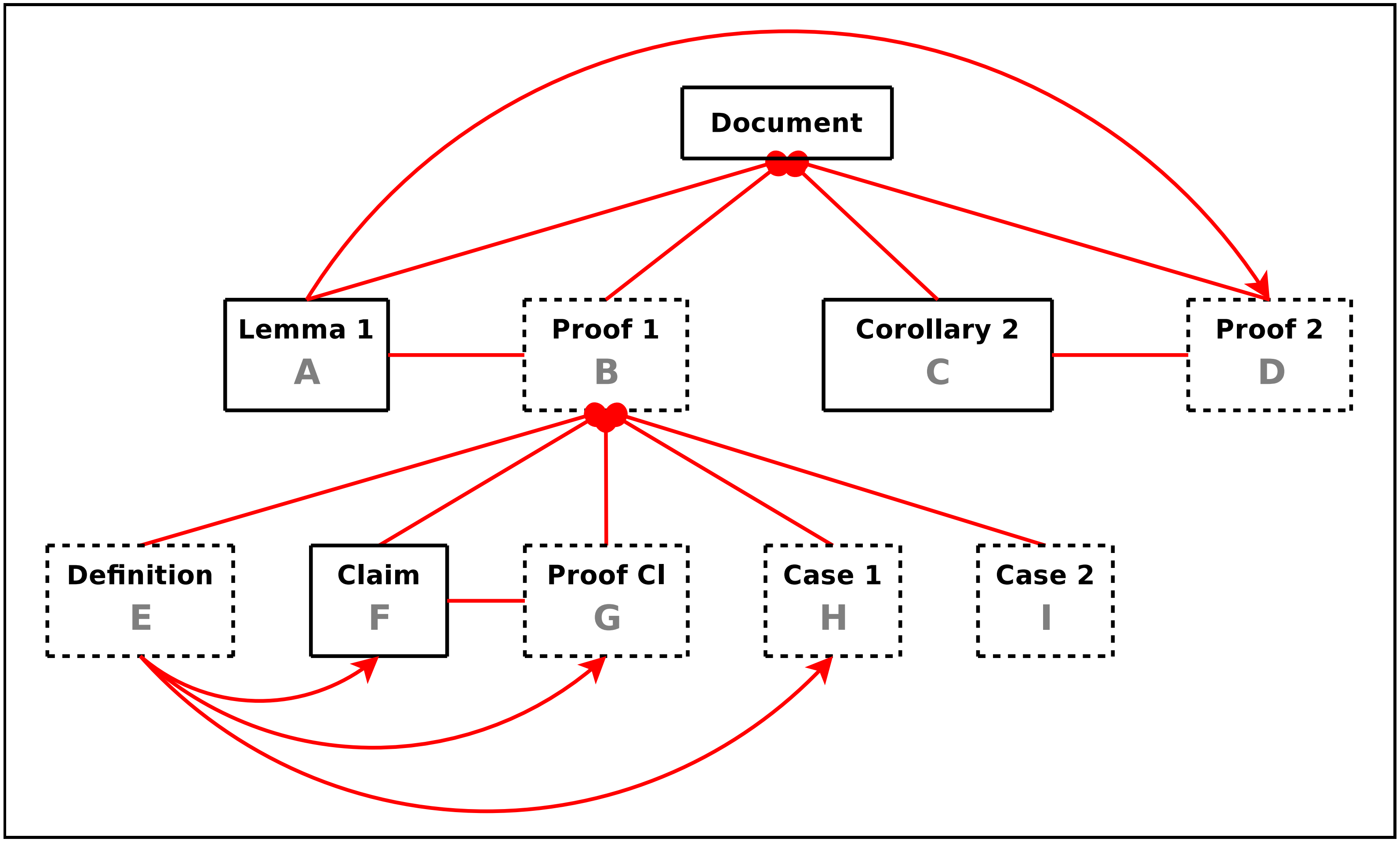
$(A$, uses, $B)$	$A \succ B$	
$(A$, caseOf, $B)$	$A \preceq B$	
$(A$, justifies, $B)$	$A \leftrightarrow B$	

Table 1: Graphical representation of edges in the GoTO

The GoTO can be generated automatically from the DG and therefore (since the DG can be produced automatically from an annotated document) automatically from an annotated document.

Graph of Textual Order for the DRa tree example



How complete is DRa?

- The dependency graph can be used to check whether the logical reasoning of the text is coherent and consistent (e.g., no loops in the reasoning).

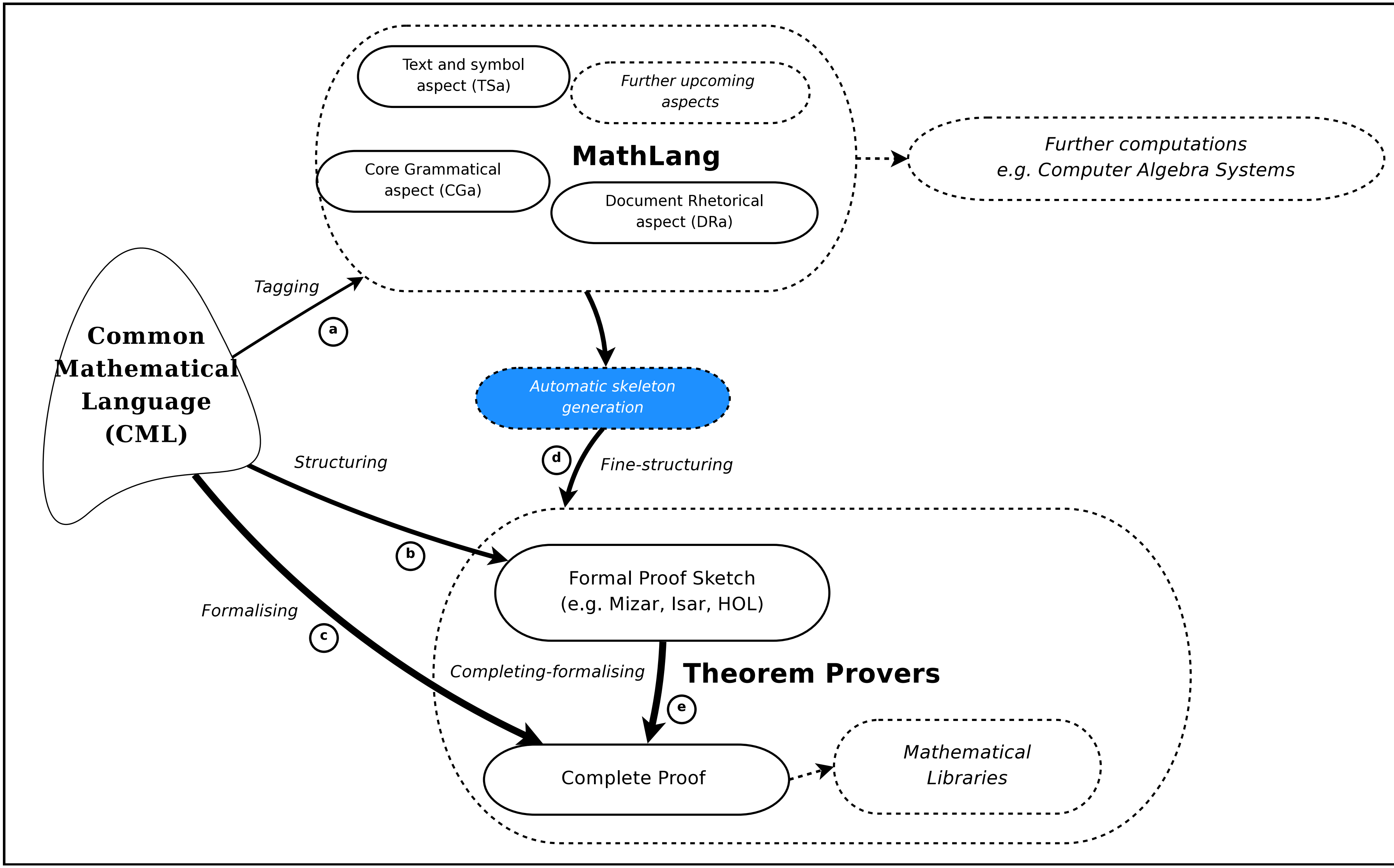
- However, both the DRa language and its implementation need more experience driven tests on natural language texts.

- Also, the DRa aspect still needs a number of implementation improvements (the automation of the analysis of the text based on its DRa features).

- Extend TSa to also cover DRa (in addition to CGa).

- Extend DRa depending on further experience driven translations.

- Establish the soundness and completeness of DRa for mathematical texts.

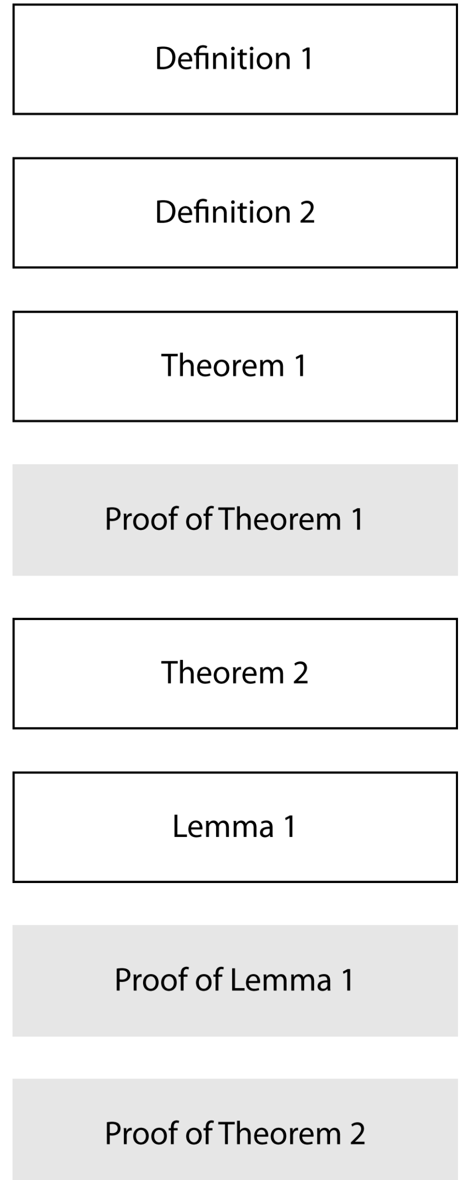


Text and symbol aspect (TSa)
Further upcoming aspects
MathLang
Core Grammatical aspect (CGa)
Document Rhetorical aspect (DRa)
Further computations e.g. Computer Algebra Systems
Common Mathematical Language (CML)
Tagging
a
Automatic skeleton generation
d
Fine-structuring
Structuring
b
Formalising
c
Formal Proof Sketch (e.g. Mizar, Isar, HOL)
Completing-formalising
Theorem Provers
e
Complete Proof
Mathematical Libraries

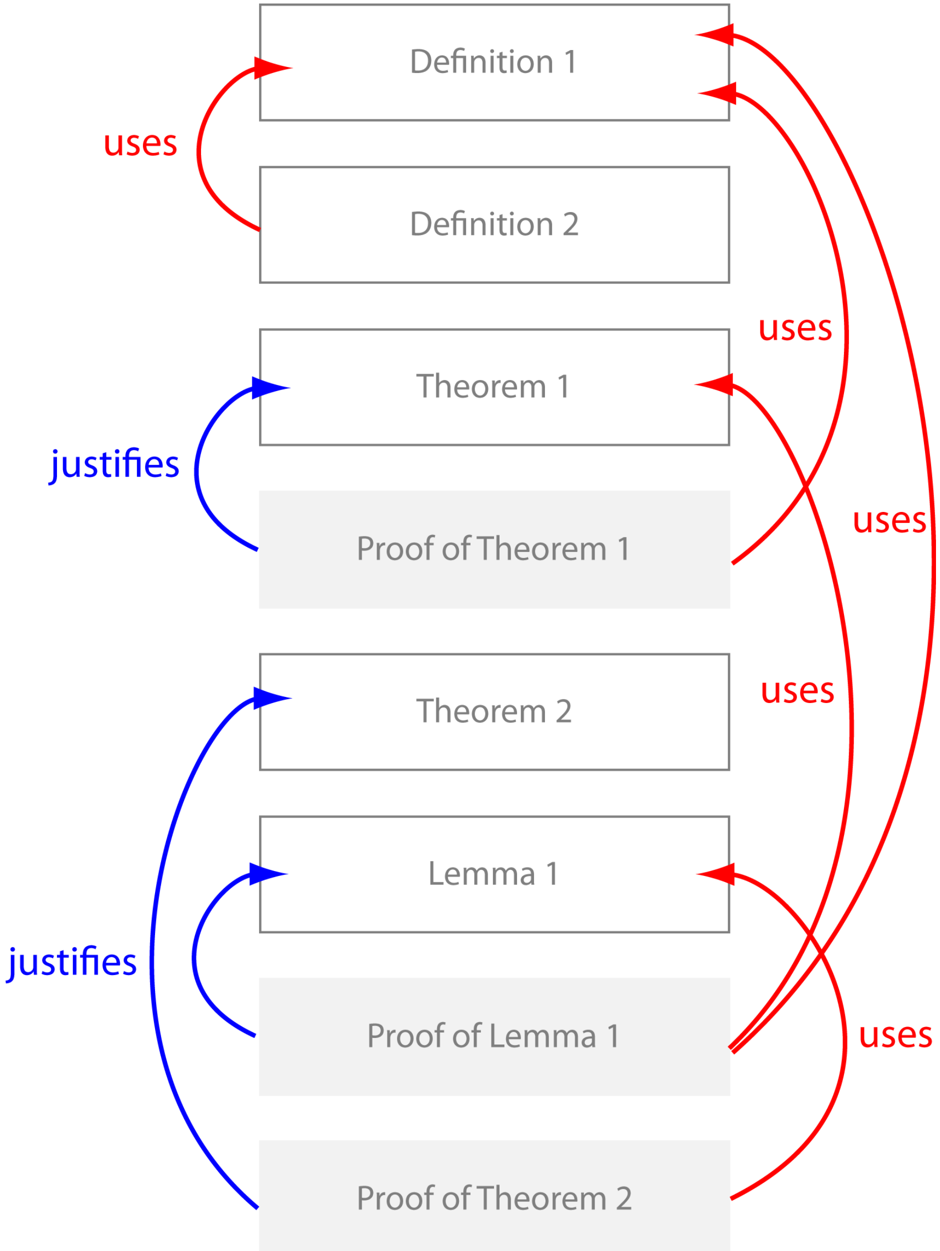
Different provers have

- different syntax
- different requirements to the structure of the text
  e.g.
  - no nested theorems/lemmas
  - only backward references
  - ...
- Aim: Skeleton should be as close as possible to the mathematician's text but with re-arrangements when necessary

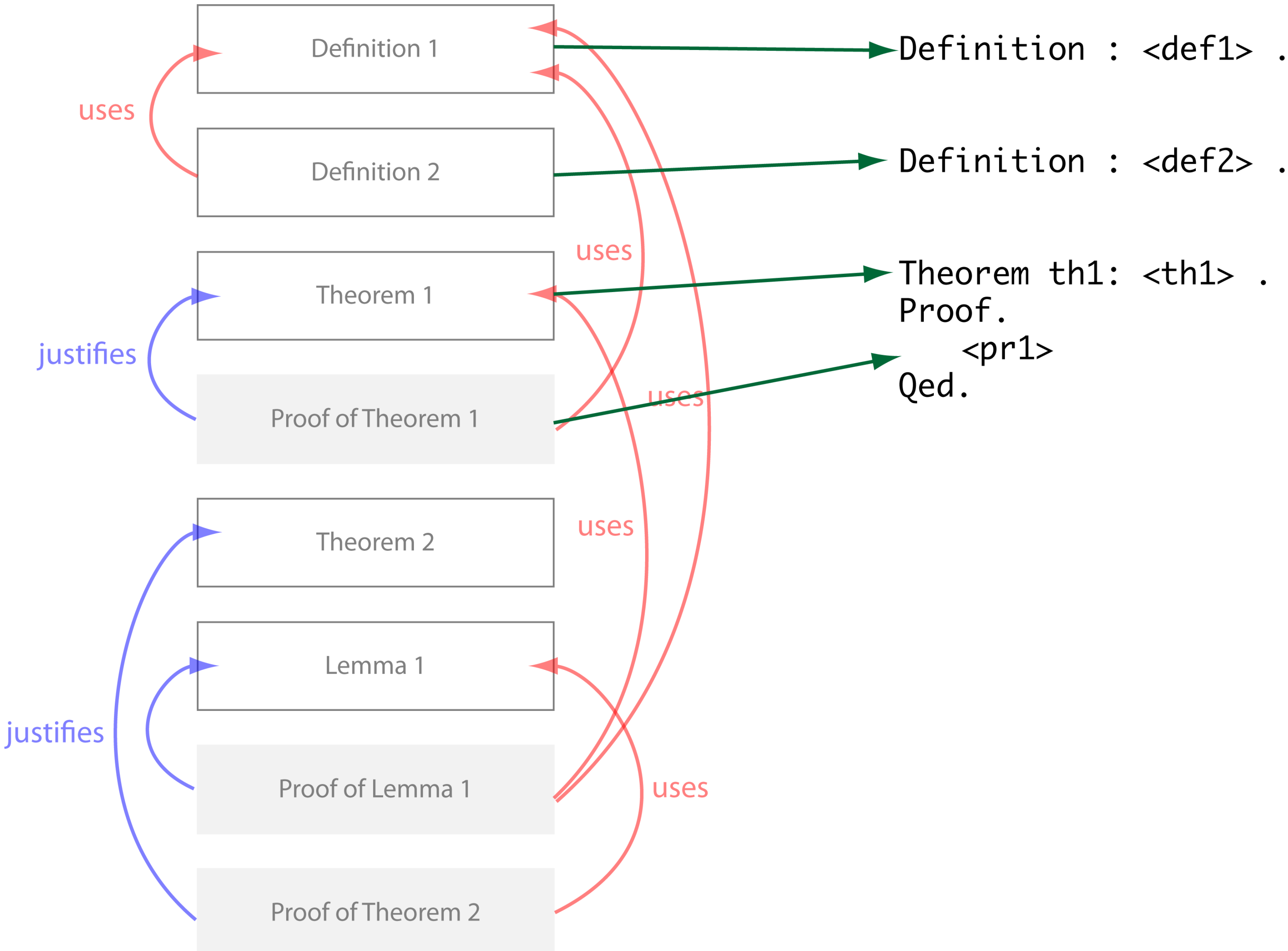
Example of nested theorems/lemmas (Moller, 03, Chapter III,2)



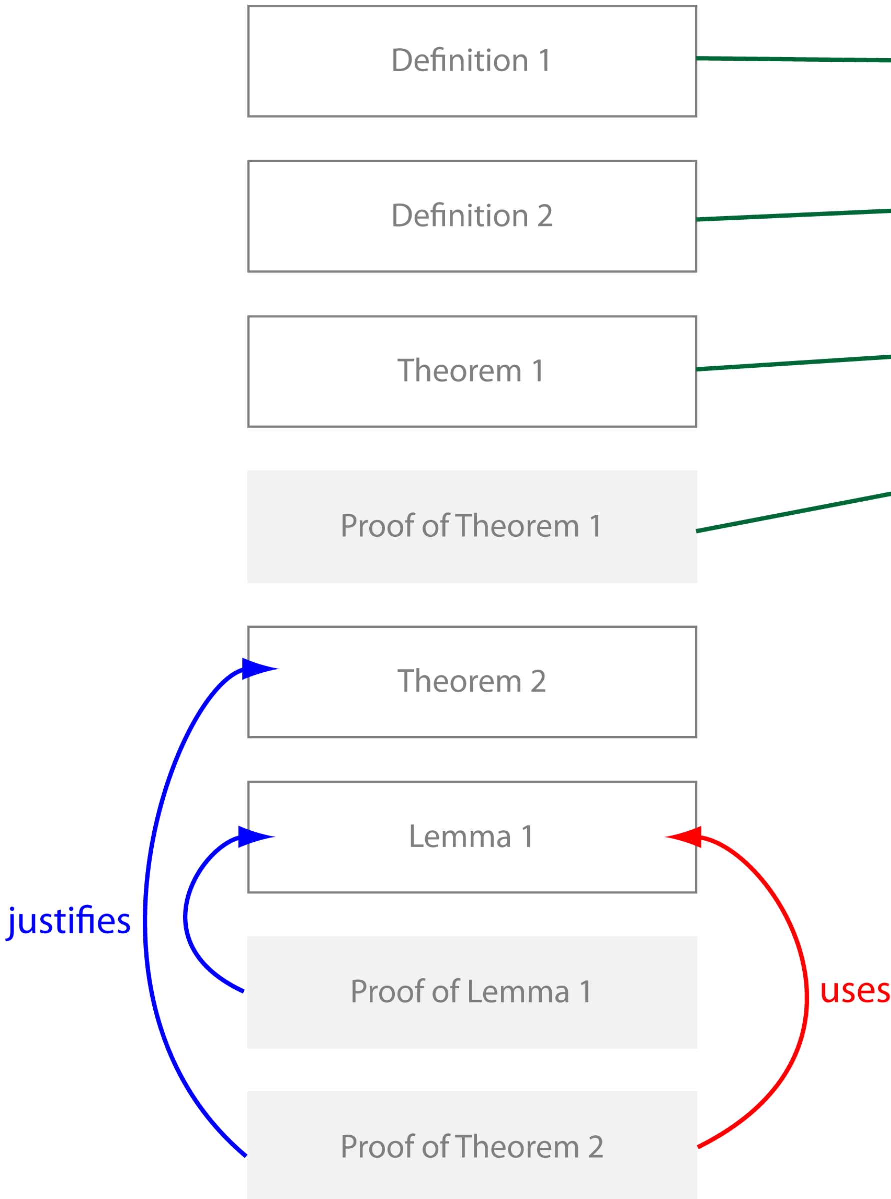
The automatic generation of a proof skeleton



The DG for the example



Straight-forward translation of the first part



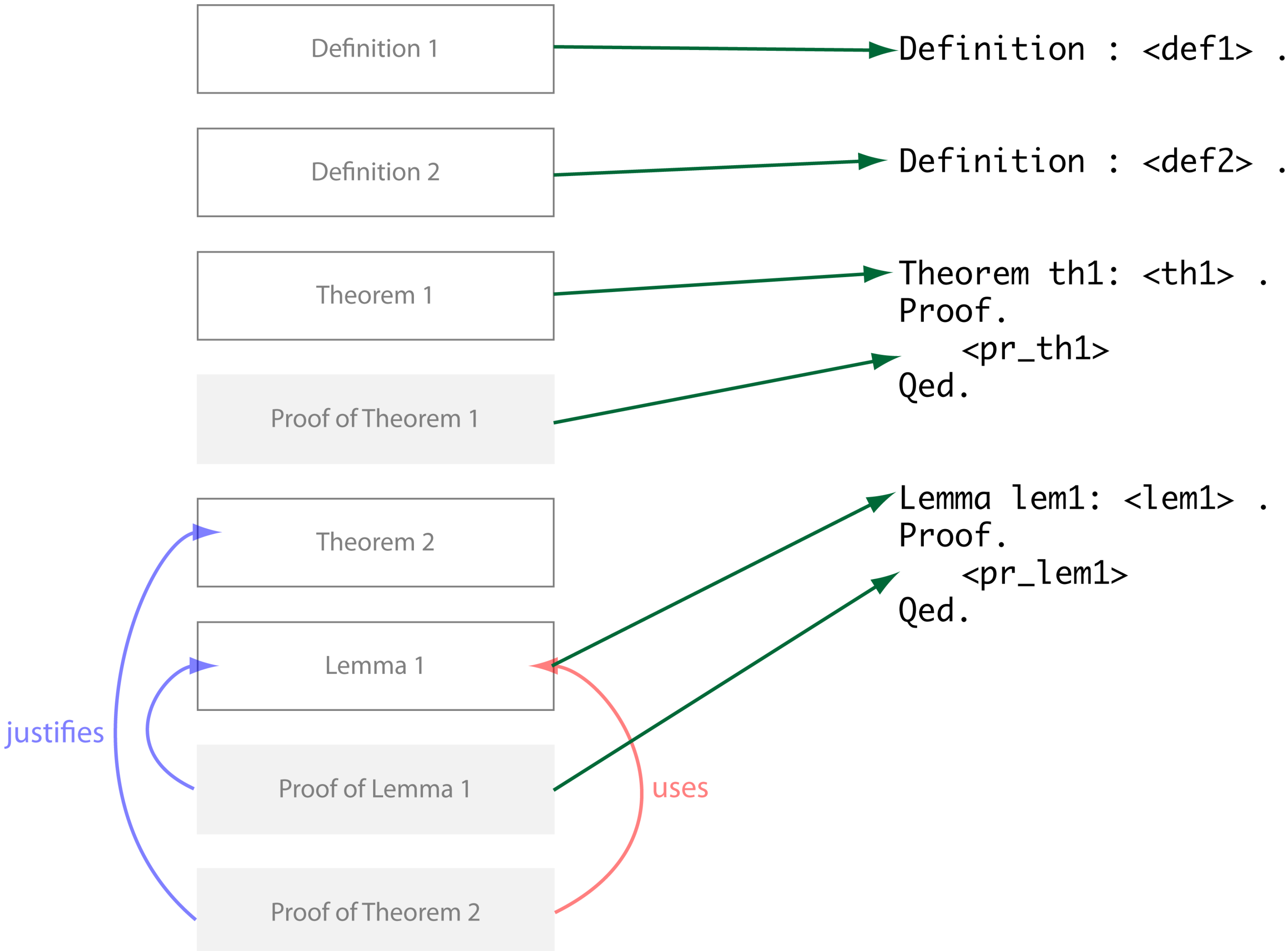

```
Definition : <def1> .
Definition : <def2> .
Theorem th1: <th1> .
Proof.
    <pr1>
Qed.
```

Problem

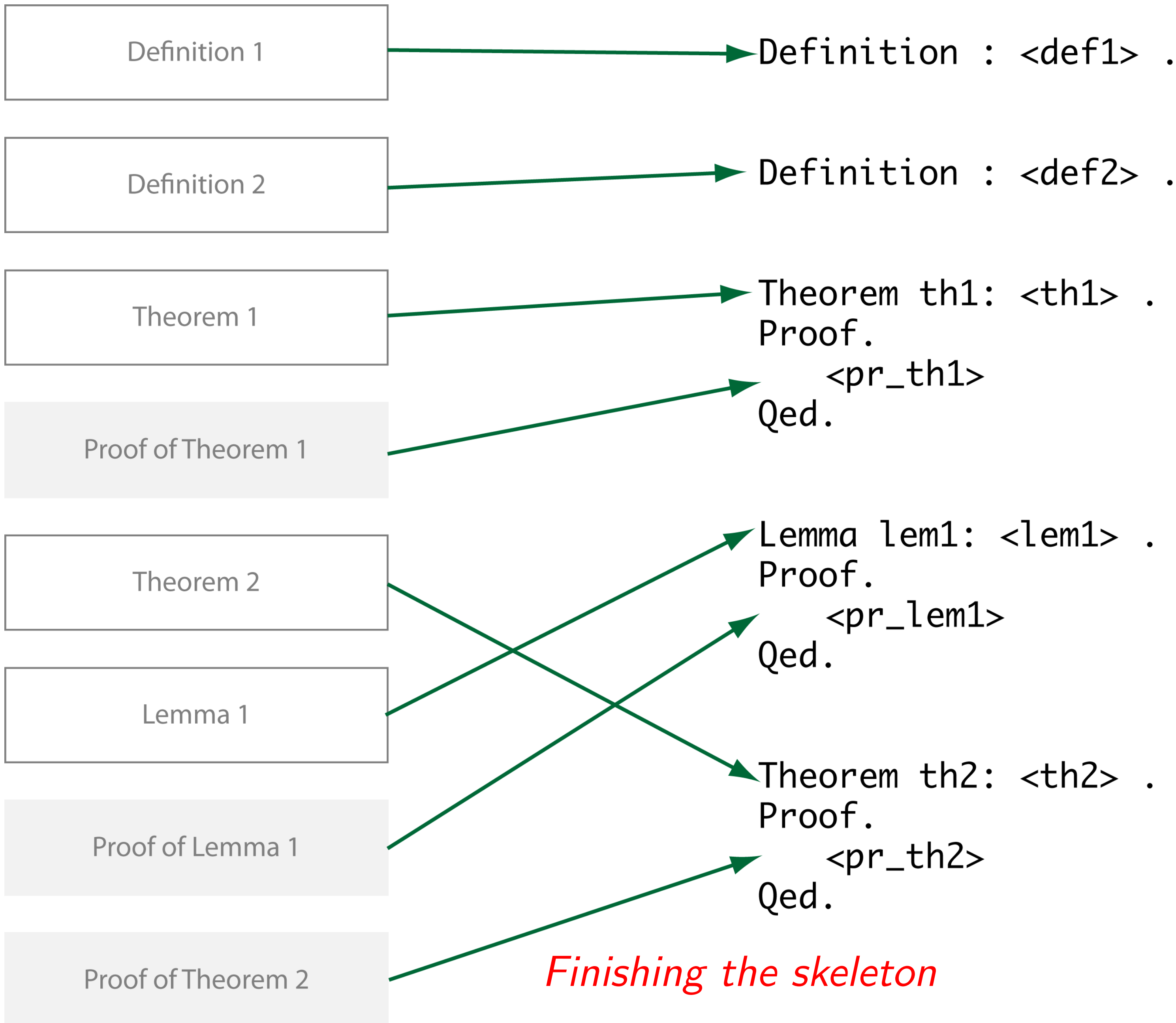
No nested theorems/lemmas in Coq e.g.

→ **Re-Ordering !**

Problem: nested theorems

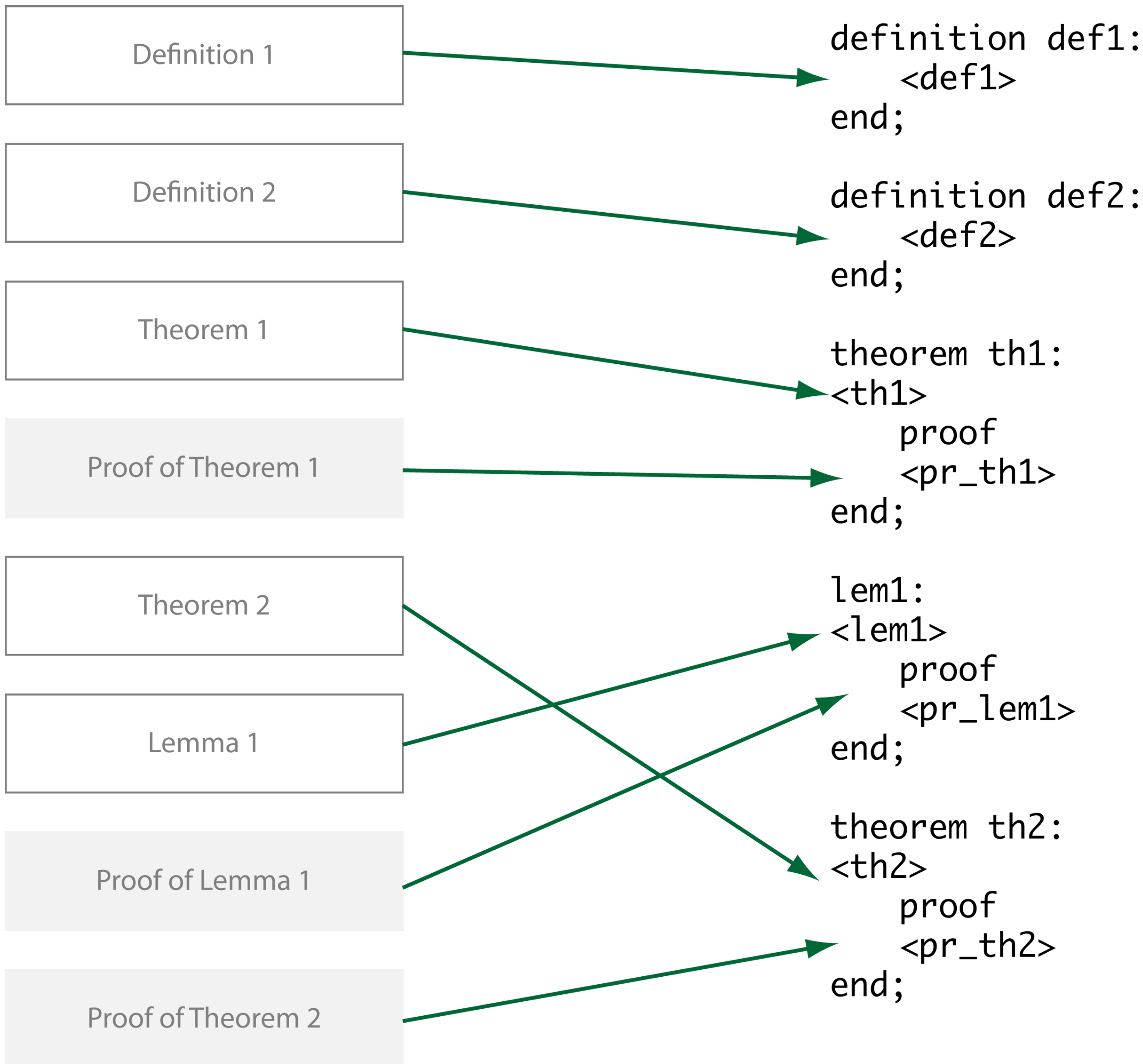


Solution: Re-ordering

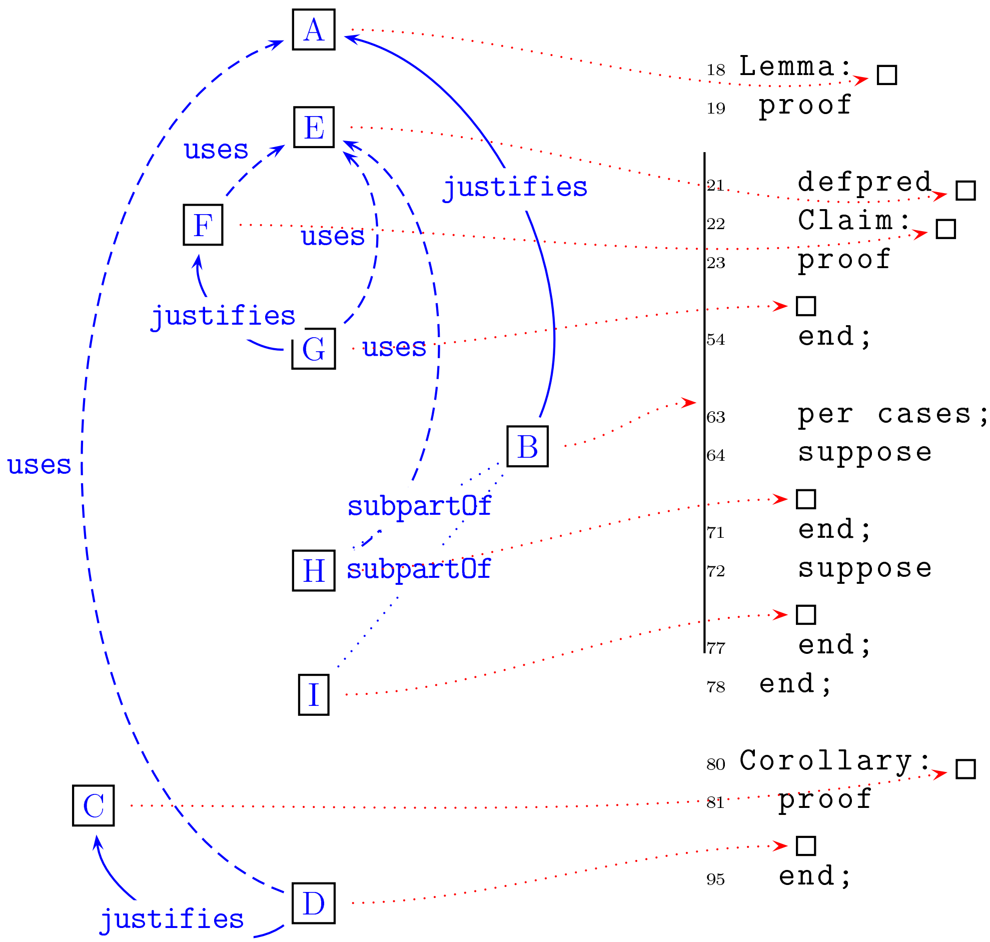


Finishing the skeleton

Skeleton for Mizar


Definition 1
Definition 2
Theorem 1
Proof of Theorem 1
Theorem 2
Lemma 1
Proof of Lemma 1
Proof of Theorem 2
definition def1:
<def1>
end;
definition def2:
<def2>
end;
theorem th1:
<th1>
proof
<pr_th1>
end;
lem1:
<lem1>
proof
<pr_lem1>
end;
theorem th2:
<th2>
proof
<pr_th2>
end;

DRa annotation into Mizar skeleton for Barendregt's example (Retel's PhD thesis)

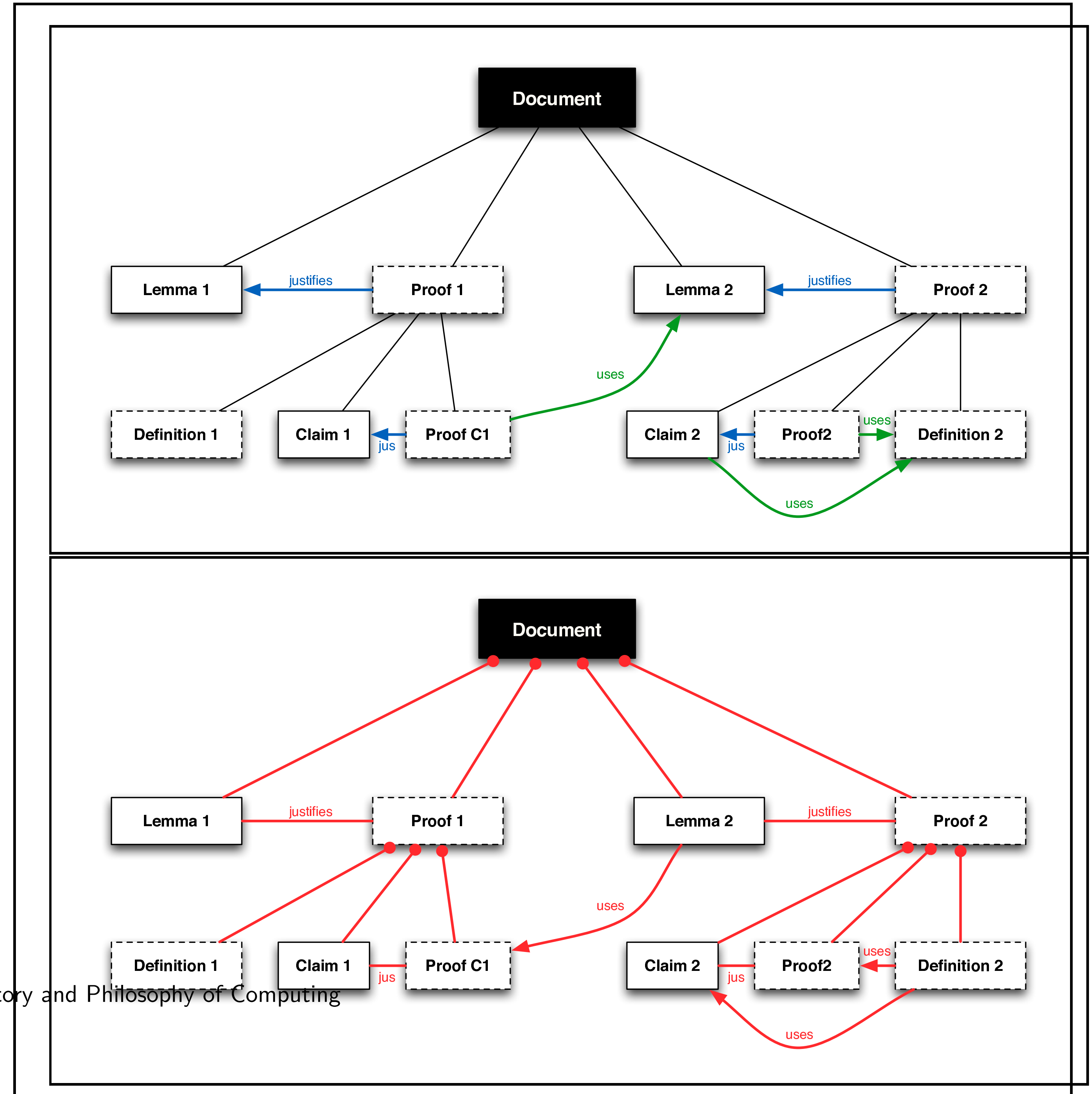


The generic algorithm for generating the proof skeleton (SGa, Zengler's thesis)

A vertex is ready to be processed iff:

- it has no incoming $\prec$ edges (in the GoTO) of unprocessed (white) vertices
- all its children are ready to be processed
- if the vertex is a proved vertex: its proof is ready to be processed

Consider the DG and GoTO of a (typical and not well structured) mathematical text:


Document
Lemma 1
justifies
Proof 1
Lemma 2
justifies
Proof 2
uses
Definition 1
Claim 1
jus
Proof C1
Claim 2
jus
Proof2
uses
Definition 2
uses
Document
Lemma 1
justifies
Proof 1
Lemma 2
justifies
Proof 2
uses
Definition 1
Claim 1
jus
Proof C1
Claim 2
jus
Proof2
uses
Definition 2
uses

The final order of the vertices is:

```
Lemma 2
Proof 2
   Definition 2
   Claim 2
   Proof C2
Lemma 1
Proof 1
   Definition 1
   Claim 1
   Proof C1
```

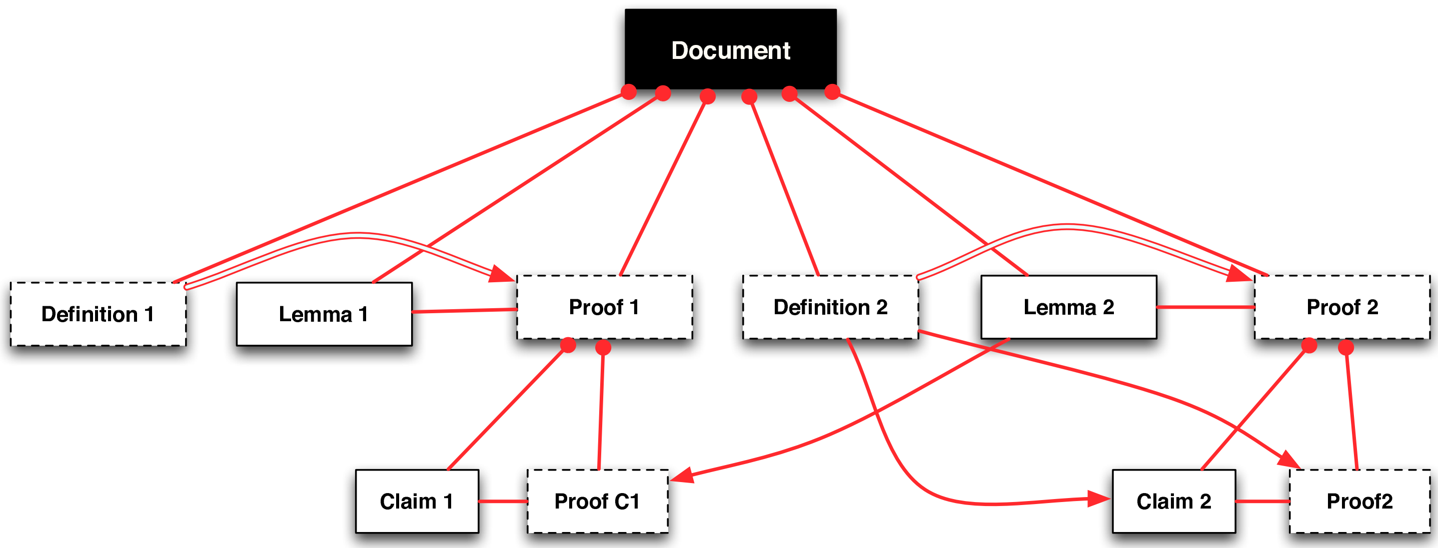


Figure 6: A flattened graph of the GoTO of figure 5 without nested definitions

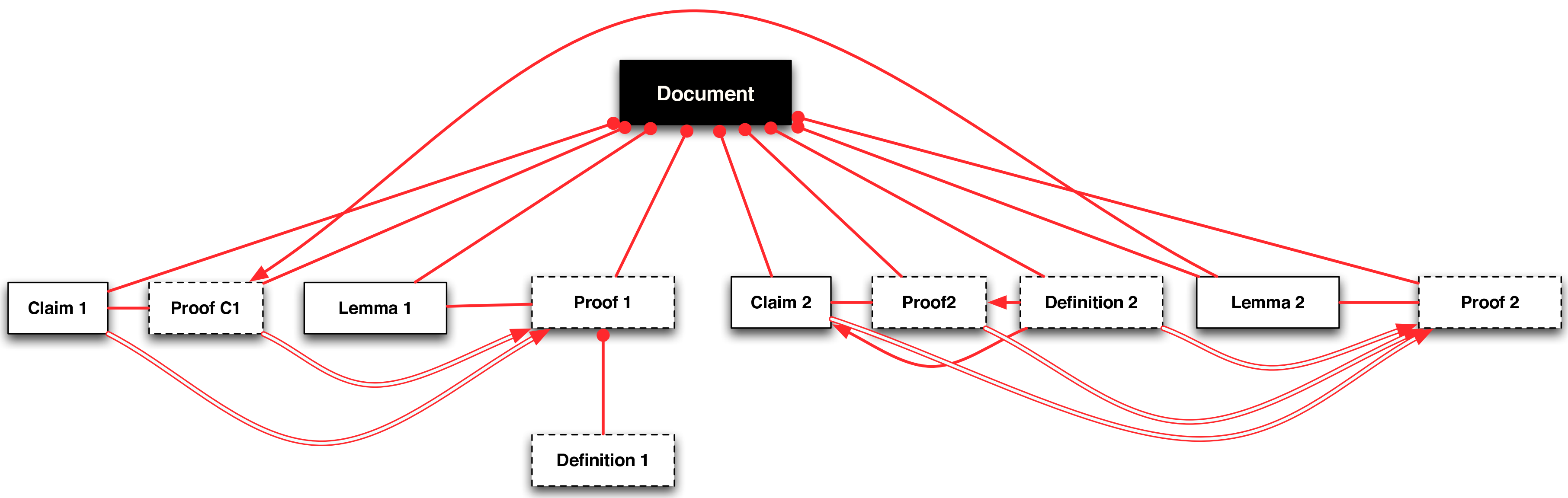


Figure 7: A flattened graph of the GoTO of figure 5 without nested claims

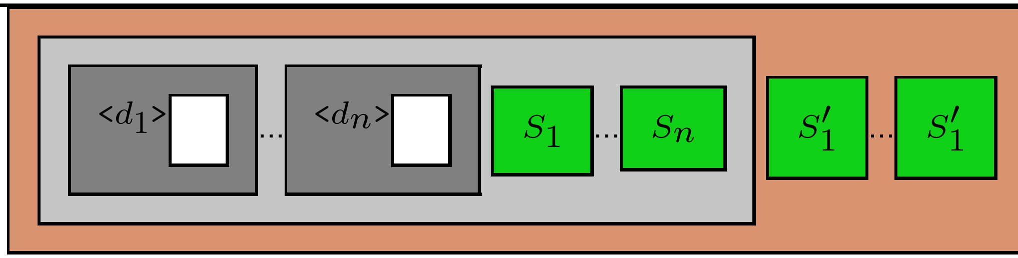
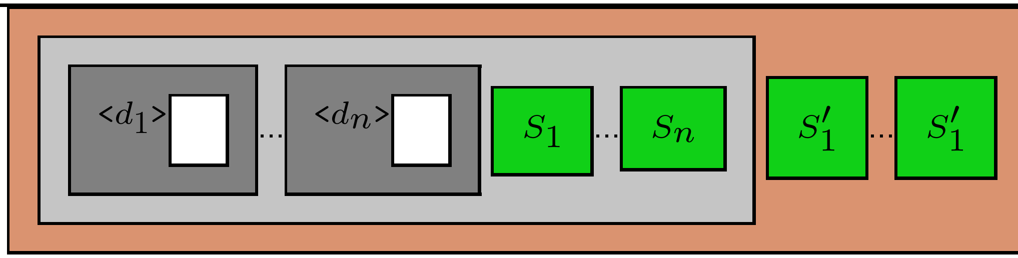
The Mizar and Coq rules for the dictionary

Role	**Mizar rule**	**Coq rule**
axiom	`%name : %body ;`	`Axiom %name : %body .`
definition	`definition %name : %nl %body %nl end;`	`Definition : %body .`
theorem	`theorem %name: %nl %body`	`Theorem %name %body .`
proof	`proof %nl %body  %nl end;`	`Proof %name : %body .`
cases	`per cases; %nl`	`%body`
case	`suppose %nl %body %nl end;`	`%body`
existencePart	`existence %nl %body`	`%body`
uniquenessPart	`uniqueness %nl %body`	`%body`

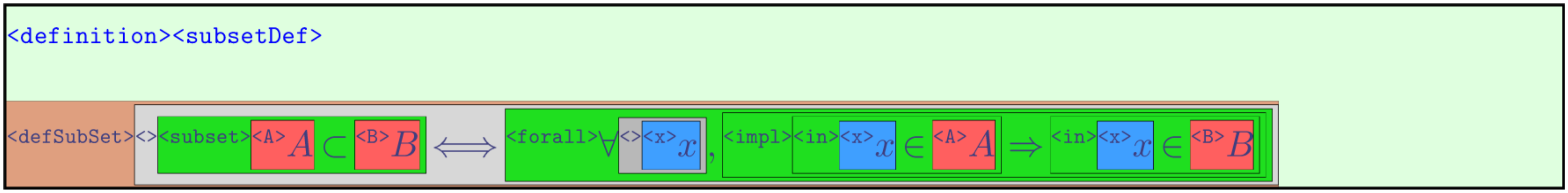
Rich skeletons for Coq

Rule N^O	Annotation ann	Coq translation $\mathcal{S}_{Coq}(ann)$
coq1)	<#>	`Set`
coq2)	<#>	`Prop`
coq3)	$<id>$ $<N>$	`id : N`
coq4)	$<id>$ $<S>$	`id : S`
coq5)	$<id>$	`id`
coq6)	$<id>$ $\boxed{p_1}$ ... $\boxed{p_n}$ $<N>$	`id :` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ `->` ... `->` $\mathcal{S}_{Coq}\left(\boxed{p_n}\right)$ `-> N`
coq7)	$<id>$ $\boxed{p_1}$ ... $\boxed{p_n}$ $<S>$	`id :` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ `->` ... `->` $\mathcal{S}_{Coq}\left(\boxed{p_n}\right)$ `-> S`

coq8)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	`id :` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ `->` ... `->` $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$ `-> Prop`
coq9)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	`id :` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ `->` ... `->` $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$ `-> Set`
coq10)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	`(id` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$ `)`
coq11)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	`(id` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$ `)`
coq12)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	`(id` $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$ `)`
coq13)	<id>	`id`
coq14)	<id> <id_1> ... <id_n> $\boxed{e}$	`id id_1 ... id_n :=` $\mathcal{S}_{Coq}\left(\boxed{e}\right)$

coq15)	 for a surrounding unproved DRa annotation	`forall` $\mathcal{S}_{Coq}\left(\boxed{<d_1>}\right)$ … $\mathcal{S}_{Coq}\left(\boxed{<d}\right.$ . . . `/\` $\mathcal{S}_{Coq}\left(\boxed{S_n}\right)$ `->` $\mathcal{S}_{Coq}\left(\boxed{S'_1}\right.$
coq16)	 for a surrounding proved DRa annotation	$\mathcal{S}_{Coq}\left(\boxed{<d_1>}\right)$ … $\mathcal{S}_{Coq}\left(\boxed{<d_n>}\right)$ `/\` $\mathcal{S}_{Coq}\left(\boxed{S_n}\right)$ `->` $\mathcal{S}_{Coq}\left(\boxed{S'_1}\right)$ /

With these rules almost every axiom, definition and theorem can be translated in a way that it is immediately usable in Coq.



the left hand side of the definition is translated according to rule (coq14)) with `subset A B`.

The right hand side is translated with the rules coq5), coq10), coq11) and coq12) and the result is

```
forall x (impl (in x A) (in x B))
```

Putting left hand and right hand side together and taking the outer DRa annotation we get the translation

```
Definition subset A B := forall x (impl (in x A) (in x B))
```

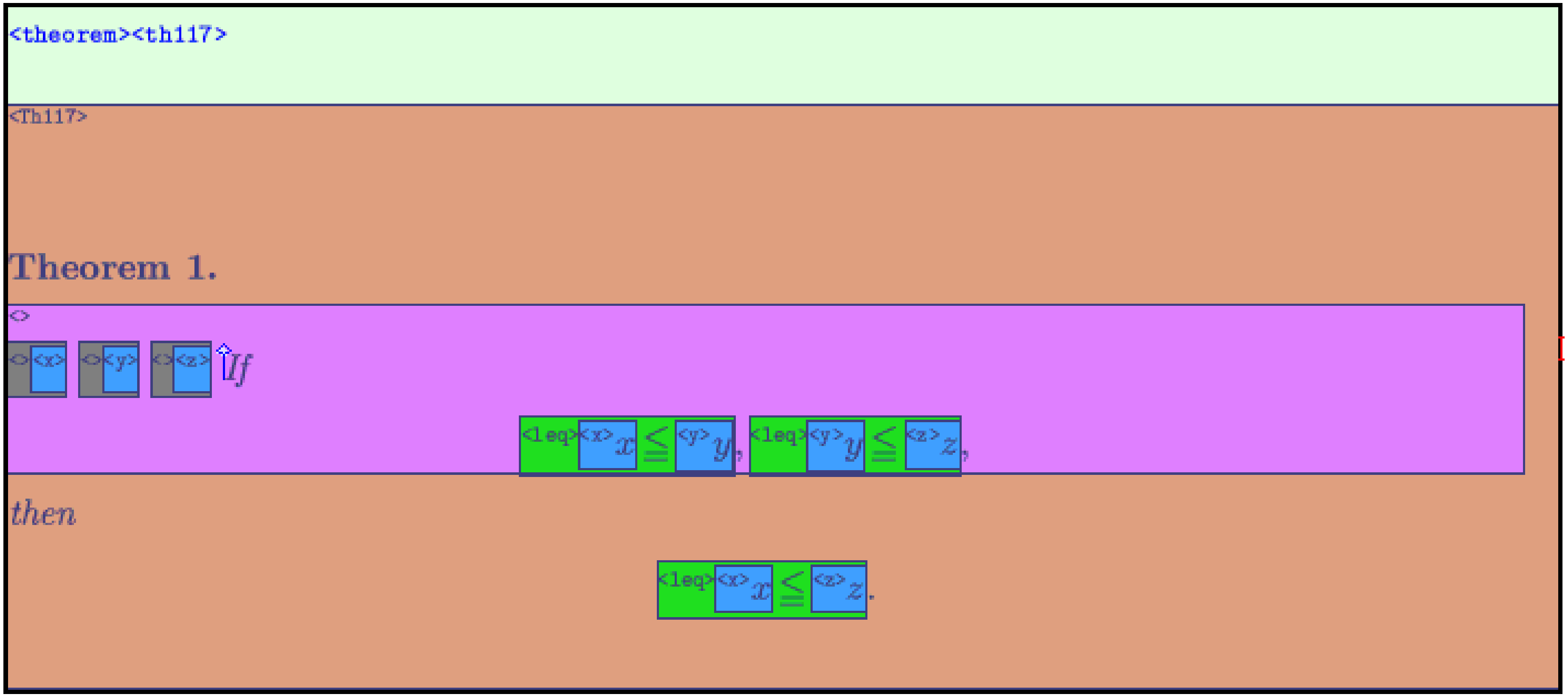
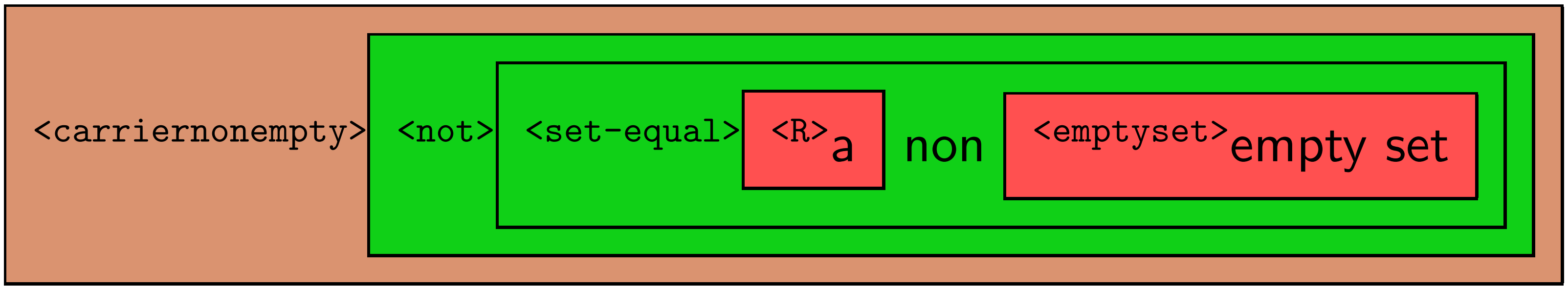


Figure 8: Theorem 17 of Landau's "Grundlagen der Analysis"

The automatic translation is:

```
Theorem th117 x y z : (leq x y /\ leq y z) -> leq x z .
```

Rich skeletons for Isabelle



The corresponding translation into Isabelle is:

```
assumes carriernonempty: "not (set-equal R emptyset)"
```

An example of a full formalisation in Coq via MathLang

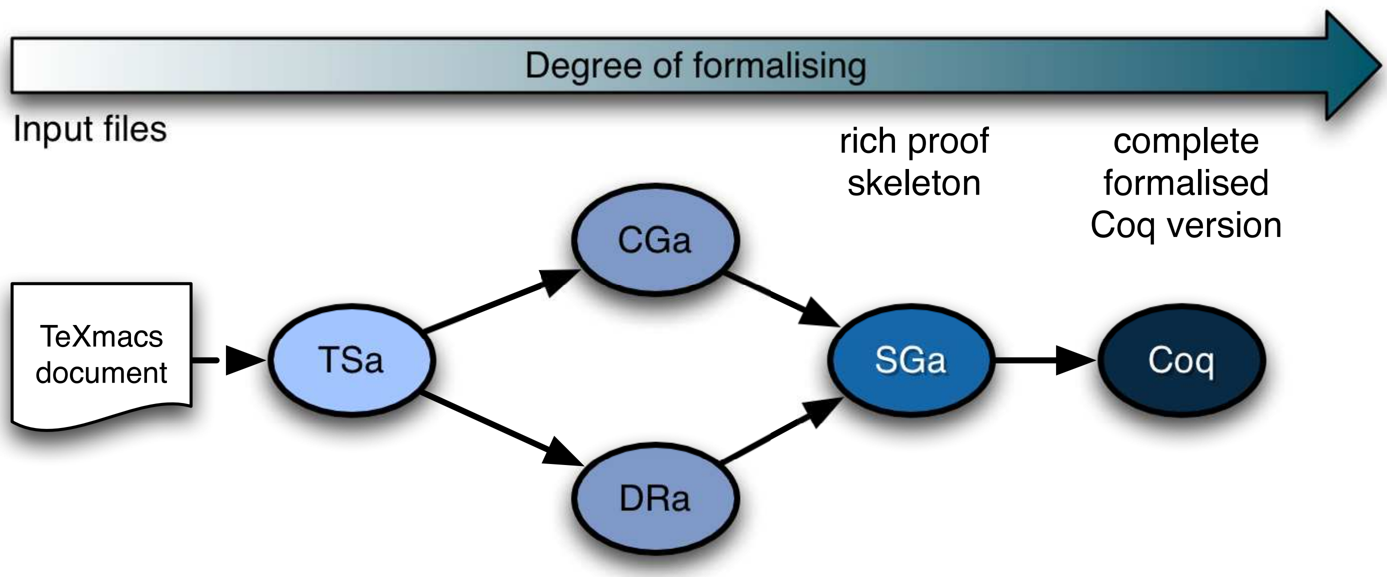


Figure 9: The path for processing the Landau chapter

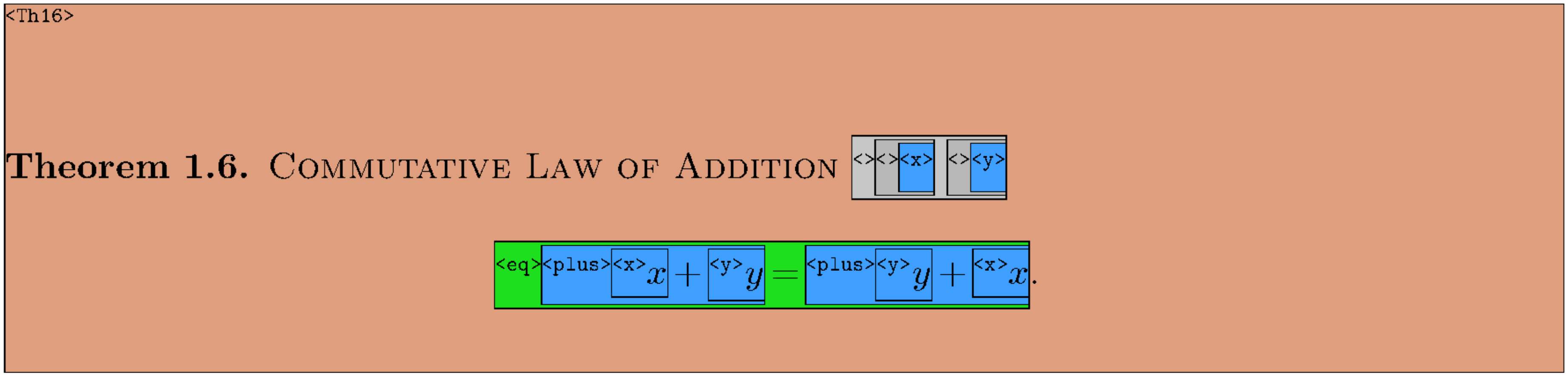


Figure 10: Simple theorem of the second section of Landau's first chapter

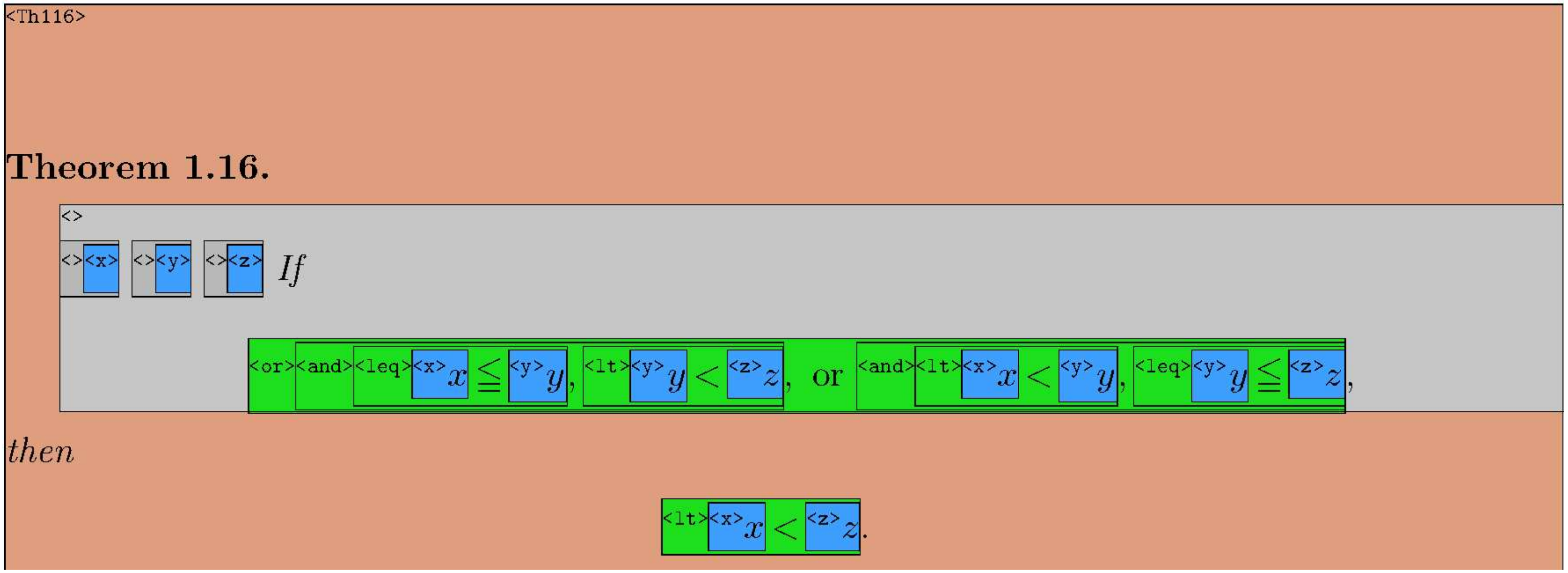
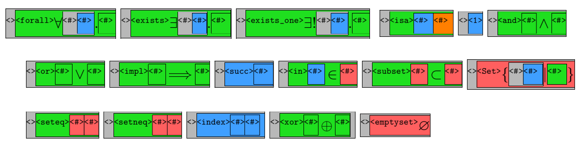


Figure 11: The annotated theorem 16 of the Landau's first chapter

Chapter 1

Natural Numbers



1.1 Axioms

We assume the following to be given:

A set (i.e. totality) of objects called natural numbers, possessing the properties - called axioms- to be listed below.

Before formulating the axioms we make some remarks about the symbols $=$ and $\neq$ which be used.

Unless otherwise specified, small italic letters will stand for natural numbers throughout this book.

If x is given and y is given, then either x and y are the same number; this may be written

$$x = y$$

($=$ to be read "equals"); or x and y are not the same number; this may be written

$$1x \neq y$$

($\neq$ to be read "is not equal to").

Accordingly, the following are true on purely logical grounds:

$x = x$ for every x

if $x = y$ then $y = x$

If $x = y$, $y = z$ then $x = z$

1

Chapter 1 of Landau:

- 5 axioms which we annotate with the mathematical role "axiom", and give them the names"ax11" - "ax15".
- 6 definitions which we annotate with the mathematical role "definition", and give them names "def11" - "def16".
- 36 nodes with the mathematical role "theorem", named "th11" - "th136" and with proofs "pr11" - "pr136".
- Some proofs are partitioned into an existential part and a uniqueness part.
- Other proofs consist of different cases which we annotate as unproved nodes with the mathematical role "case".

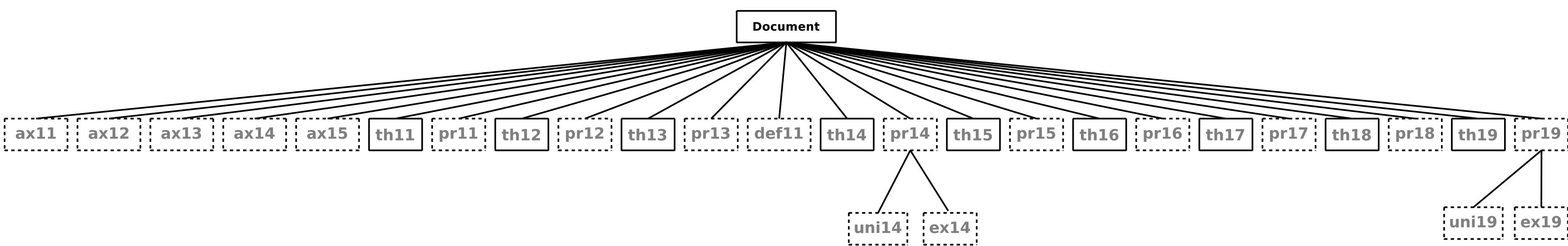


Figure 12: The DRa tree of sections 1 and 2 of chapter 1 of Landau's book

- The relations are annotated in a straightforward manner.

- Each proof $justifies$ its corresponding theorem.

- Axiom 5 ("ax15") is the axiom of induction. So every proof which uses induction, $uses$ also this axiom.

- Definition 1 ("def11") is the definition of addition. Hence every node which uses addition also $uses$ this definition.

- Some theorems use other theorems via texts like: "By Theorem ...".

- In total we have 36 $justifies$ relations, 154 uses relations, 6 caseOf, 3 existencePartOf and 3 uniquenessPartOf relations.

- The DG and GoTO are automatically generated.

- The GoTO is automatically checked and no errors result. So, we proceed to the next stage: automatically generating the SGa.

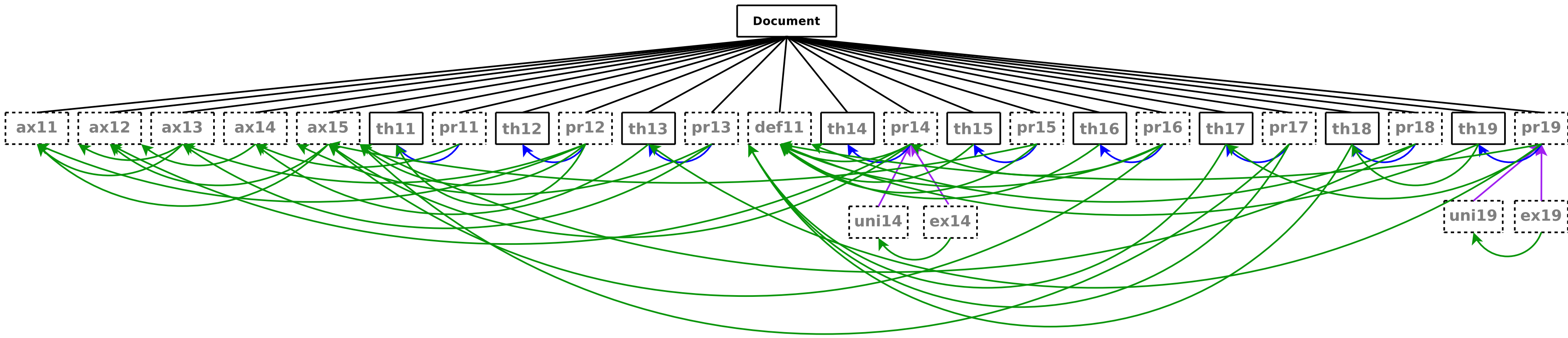
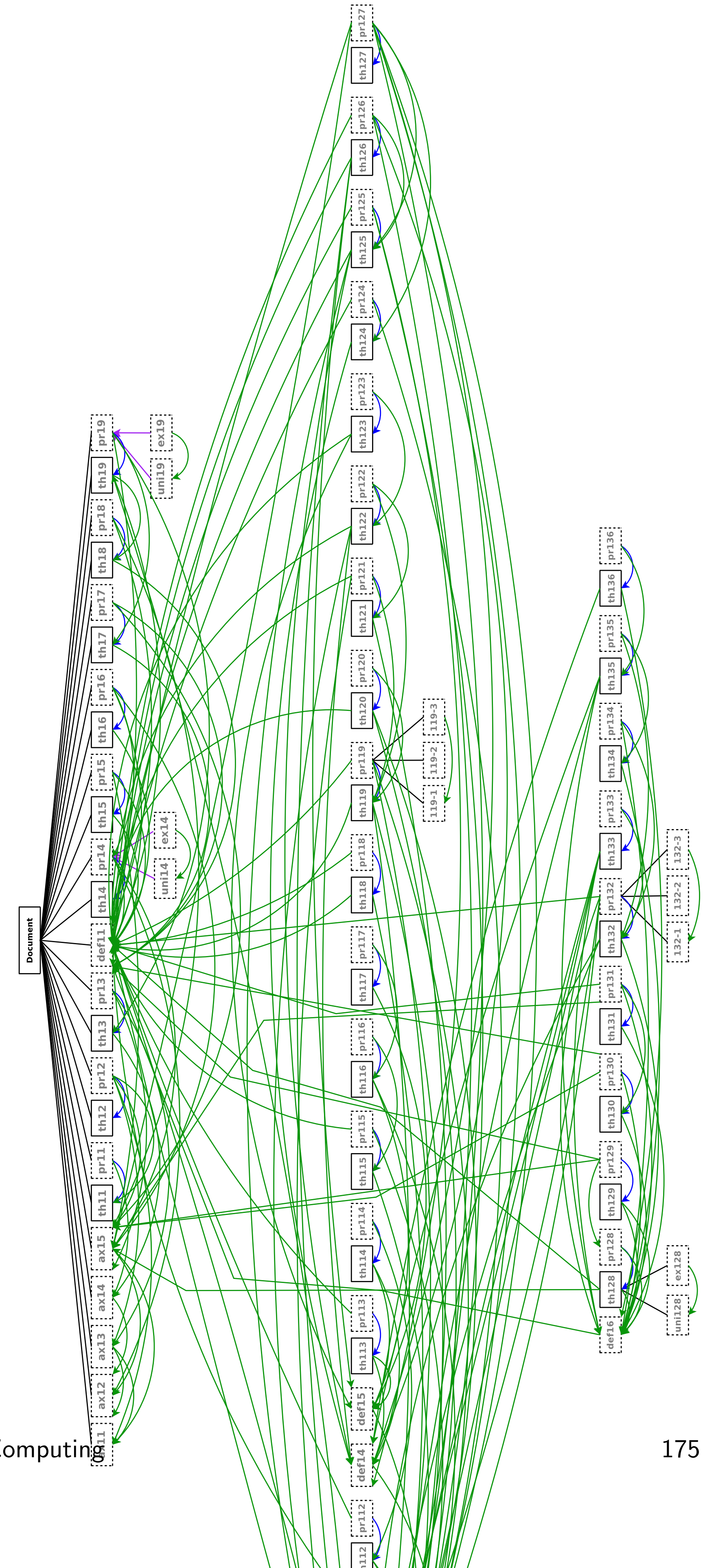
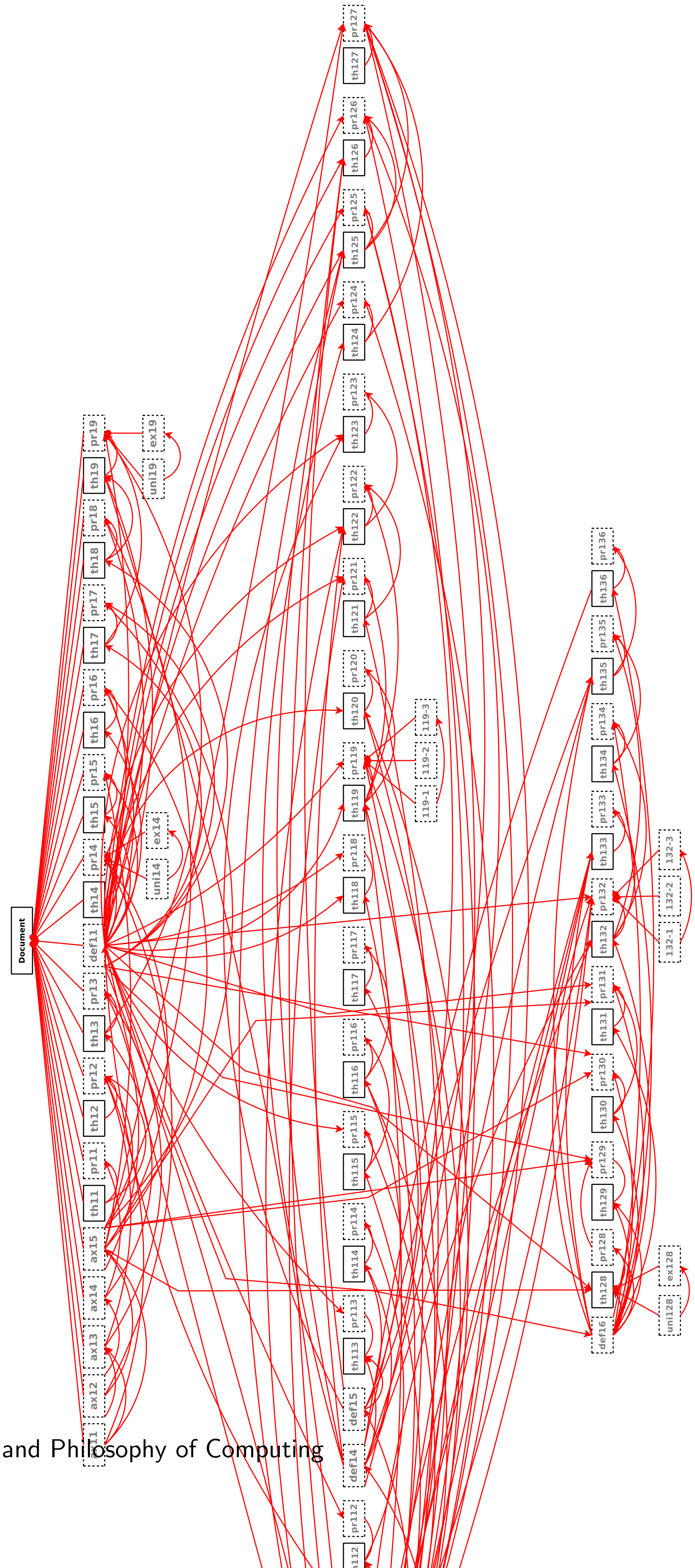


Figure 13: The DG of sections 1 and 2 of chapter 1 of Landau's book



The GoTO of section 1 - 4


Document
pr19
th19
pr18
th18
pr17
th17
pr16
th16
pr15
th15
pr14
th14
def11
pr13
th13
pr12
th12
pr11
th11
ax15
ax14
ax13
ax12
ex19
uni19
ex14
uni14
pr127
th127
pr126
th126
pr125
th125
pr124
th124
pr123
th123
pr122
th122
pr121
th121
pr120
th120
pr119
th119
pr118
th118
pr117
th117
pr116
th116
pr115
th115
pr114
th114
pr113
th113
def15
def14
pr112
119-3
119-2
119-1
pr136
th136
pr135
th135
pr134
th134
pr133
th133
pr132
th132
pr131
th131
pr130
th130
pr129
th129
pr128
th128
def16
132-3
132-2
132-1
ex128
uni128

An extract of the automatically generated rich skeleton

```
Definition geq x y := (or (gt x y) (eq x y)).
Definition leq x y := (or (lt x y) (eq x y)).

Theorem th113 x y : (impl (geq x y) (leq y x)).
Proof.
...

Qed.

Theorem th114 x y : (impl (leq x y) (geq y x)).
Proof.
...

Qed.

Theorem th115 x y z : (impl (impl (lt x y) (lt y z)) (lt x z)).
Proof.
...

Qed.
```

Completing the proofs in Coq

- We defined the natural numbers as an inductive set - just as Landau does in his book.

```
Inductive nats : Set :=
 | I : nats
 | succ : nats -> nats
```

- The encoding of theorem 2 of the first chapter in Coq is

```
theorem th12 x : neq (succ x) x .
```

- Landau proves this theorem with induction. He first shows, that $1' \neq 1$ and then that with the assumption of $x' \neq x$ it also holds that $(x')' \neq x'$.

- We do our proof in the Landau style. We introduce the variable x and eliminate it, which yields two subgoals that we need to prove. These subgoals are exactly the induction basis and the induction step.

```
Proof.
intro x. elim x.

2 subgoals
x : nats
______________________________________(1/2)
neq (succ I) I

____________________________________________________________(
forall n : nats, neq (succ n) n -> neq (succ (succ n)) (succ n)
```

Landau proved the first case with the help of Axiom 3 (for all x, $x' \neq 1$).

```
apply ax13.

1 subgoal
x : nats
____________________________________________________________(
forall n : nats, neq (succ n) n -> neq (succ (succ n)) (succ n)
```

The next step is to introduce n as natural number and to introduce the induction hypothesis:

```
intros n H.

1 subgoal
x : nats
n : nats
H : neq (succ n) n
______________________________________(1/1)
neq (succ (succ n)) (succ n)
```

We see that this is exactly the second case of Landau's proof. He proved this case with Theorem 1 - we do the same:

```
apply th11.

1 subgoal
x : nats
n : nats
```

```
H : neq (succ n) n
______________________________________(1/1)
neq (succ n) n
```

And of course this is exactly the induction hypotheses which we already have as an assumption and we can finish the proof:

```
assumption.
Proof completed.
```

The complete theorem and its proof in Coq finally look like this:

```
Theorem th12 (x:nats) : neq (succ x) x .
Proof.
intro x. elim x.
apply ax13.
intros n H.
apply th11.
assumption.
Qed.
```

With the help of the CGa annotations and the automatically generated rich proof skeleton, Zengler (who was not familiar with Coq) completed the Coq proofs of the whole of chapter one in a couple of hours.

Some points to consider

- We do not at all assume/prefer one type/logical theory instead of another.
- The formalisation of a language of mathematics should separate the questions:
  - *which type/logical theory is necessary for which part of mathematics*
  - *which language should mathematics be written in*.
- Mathematicians don't usually know or work with type/logical theories.
- Mathematicians usually *do* mathematics (manipulations, calculations, etc), but are not interested in general in reasoning *about* mathematics.
- The steps used for computerising books of mathematics written in English, as we are doing, can also be followed for books written in Arabic, French, German, or any other natural language.

- MathLang aims to support non-fully-formalized mathematics practiced by the ordinary mathematician as well as work toward full formalization.

- MathLang aims to handle mathematics as expressed in natural language as well as symbolic formulas.

- MathLang aims to do some amount of type checking even for non-fully-formalized mathematics. This corresponds roughly to grammatical conditions.

- MathLang aims for a formal representation of CML texts that closely corresponds to the CML conceived by the ordinary mathematician.

- MathLang aims to support automated processing of mathematical knowledge.

- MathLang aims to be independent of any foundation of mathematics.

- MathLang allows anyone to be involved, whether a mathematician, a computer engineer, a computer scientist, a linguist, a logician, etc.

Bibliography

H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

P. Benacerraf and H. Putnam, editors. *Philosophy of Mathematics*. Cambridge University Press, second edition, 1983.

L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*. PhD thesis, Eindhoven University of Technology, 1977. Published as Mathematical Centre Tracts nr. 83 (Amsterdam, Mathematisch Centrum, 1979).

N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [Nederpelt et al., 1994], pages 73–100.

C. Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo*, 11:154–164, 1897. English translation in [Heijenoort, 1967], pages 104–112.

G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Erster Artikel). *Mathematische Annalen*, 46: 481–512, 1895.

G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Zweiter Artikel). *Mathematische Annalen*, 49: 207–246, 1897.

A.-L. Cauchy. *Cours d'Analyse de l'Ecole Royale Polytechnique*. Debure, Paris, 1821. Also as *Œuvres Complètes* (2), volume III, Gauthier-Villars, Paris, 1897.

A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33:346–366, 1932.

A. Church. A set of postulates for the foundation of logic (2). *Annals of Mathematics*, 34:839–864, 1933.

A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

R. Dedekind. *Stetigkeit und irrationale Zahlen*. Vieweg & Sohn, Braunschweig, 1872.

G. Frege. Letter to Russell. English translation in [Heijenoort, 1967], pages 127–128, 1902.

G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume II. Pohle, Jena, 1903. Reprinted 1962 (Olms, Hildesheim).

G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [Heijenoort, 1967], pages 1–82.

G. Frege. *Grundlagen der Arithmetik, eine logisch-mathematische Untersuchung über den Begriff der Zahl*. , Breslau, 1884.

G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume I. Pohle, Jena, 1892a. Reprinted 1962 (Olms, Hildesheim).

G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik,* new series, 100:25–50, 1892b. English translation in [McGuinness, 1984], pages 157–177.

G. Frege. Ueber die Begriffschrift des Herrn Peano und meine eigene. *Berichte über die Verhandlungen der Königlich Sächsischen Gesellschaft der Wissenschaften zu Leipzig, Mathematisch-physikalische Klasse 48*, pages 361–378, 1896. English translation in [McGuinness, 1984], pages 234–248.

J.H. Geuvers. *Logics and Type Systems*. PhD thesis, Catholic University of Nijmegen, 1993.

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

K. Gödel. Russell's mathematical logic. In P.A. Schlipp, editor, *The Philosophy of Bertrand Russell*. Evanston & Chicago, Northwestern University, 1944. Also in [Benacerraf and Putnam, 1983], pages 447–469.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts, 1967.

D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the Barendregt cube using parameters. In *Proceedings of the Fifth International Symposium on Functional and Logic Programming, FLOPS 2001*, pages 375–389, 2001.

F. Kamareddine, T. Laan, and R. Nederpelt. Types in logic and mathematics before 1940. *Bulletin of Symbolic Logic*, 8(2):185–245, 2002.

F. Kamareddine, T. Laan, and R. Nederpelt. *A Modern Perspective on Type Theory*. Kluwer Academic Publishers, 2004.

Fairouz Kamareddine. Typed lambda-calculi with one binder. *J. Funct. Program.*, 15(5):771–796, 2005.

Fairouz Kamareddine, Roel Bloo, and Rob Nederpelt. On pi-conversion in the lambda-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1-3):27–45, 1999.

Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. Revisiting the notion of function. *J. Log. Algebr. Program.*, 54(1-2):65–107, 2003.

Twan Laan and Michael Franssen. Parameters for first order logic. *Logic and Computation*, 2001.

G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.

B. McGuinness, editor. *Gottlob Frege: Collected Papers on Mathematics, Logic, and Philosophy*. Basil Blackwell, Oxford, 1984.

R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.

G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [Heijenoort, 1967], pages 83–97.

G. Peano. *Formulaire de Mathématique*. Bocca, Turin, 1894–1908. 5 successive versions; the final edition issued as *Formulario Mathematico*.

W. Van Orman Quine. *Set Theory and its Logic*. Harvard University Press, Cambridge, Massachusetts, 1963.

F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society,* 2nd series, 25: 338–384, 1926.

G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.

J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.

J.B. Rosser. Highlights of the history of the lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

B. Russell. Letter to Frege. English translation in [Heijenoort, 1967], pages 124–125, 1902.

B. Russell. *The Principles of Mathematics*. Allen & Unwin, London, 1903.

B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [Heijenoort, 1967], pages 150–182.

M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. Also in [Heijenoort, 1967], pages 355–366.

H. Weyl. *Das Kontinuum*. Veit, Leipzig, 1918. German; also in: Das Kontinuum und andere Monographien, Chelsea Pub.Comp., New York, 1960.

A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910^1, 1927^2. All references are to the first volume, unless otherwise stated.