

From the Foundation of Mathematics to the Birth of Computation

Fairouz Kamareddine
Heriot-Watt University, Edinburgh, Scotland

November 2012

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 function 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 (1968) were also developed.
- Also, the earlier work of Frege, Russell and Whitehead, Hilbert, etc., on the formalisation 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, 1910¹, 1927²] 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* $\lambda \rightarrow$ [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)

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*.
- Frege 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:
$$\text{Let } f(x) = \neg\forall\varphi[(\lambda\varphi(\alpha) = x) \longrightarrow \varphi(x)]$$
$$\text{i.e. } f(x) = \exists\varphi[(\lambda\varphi(\alpha) = x) \wedge \neg\varphi(x)] \quad \text{hence } \neg\varphi(\lambda\varphi(\alpha))$$
- *Both $f(\hat{\epsilon}f(\epsilon))$ and $\neg f(\hat{\epsilon}f(\epsilon))$ 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 $\neg A$, 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 Begriffsschrift 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 $K =_{\text{def}} \{x \mid x \notin x\}$ and deriving $K \in K \iff K \notin K$.
- 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, 1910¹, 1927²] (1910-1912).

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)** $x =_L y \stackrel{\text{def}}{\leftrightarrow} \forall z[z(x) \leftrightarrow z(y)]$.
In order to express this general notion in RTT, we have to incorporate *all* pfs $\forall z : (0^0)^n [z(x) \leftrightarrow z(y)]$ 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)

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_α for each type α , and \exists_α 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) $(\lambda x:\alpha. A)B \rightarrow_\beta A[x := B]$.

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

- $\Gamma \vdash \neg : o \rightarrow o$;
 $\Gamma \vdash \wedge : o \rightarrow o \rightarrow o$;
 $\Gamma \vdash \forall_\alpha : (\alpha \rightarrow o) \rightarrow o$;
 $\Gamma \vdash \iota_\alpha : (\alpha \rightarrow o) \rightarrow \alpha$;
- $\Gamma \vdash x : \alpha$ 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 $\Gamma \vdash (AB) : \beta$.

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*).

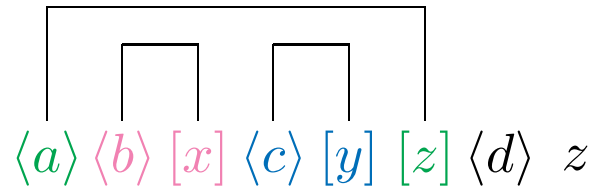
Theme 2: Lambda Calculus à la de Bruijn

- $\mathcal{I}(x) = x$, $\mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B)$, $\mathcal{I}(AB) = \langle \mathcal{I}(B) \rangle \mathcal{I}(A)$
- $(\lambda x.\lambda y.xy)z$ translates to $\langle z \rangle [x][y] \langle y \rangle x$.
- The *applicator wagon* $\langle z \rangle$ and *abstractor wagon* $[x]$ occur NEXT to each other.
- $(\lambda x.A)B \rightarrow_{\beta} A[x := B]$ becomes

$$\langle B \rangle [x] A \rightarrow_{\beta} [x := B] A$$
- The “bracketing structure” of $((\lambda_x.(\lambda_y.\lambda_z. _ _)c)b)a$, is ‘ $[_1 \ [_2 \ [_3 \]_2 \]_1 \]_3$ ’, where ‘ $[_i$ ’ and ‘ $]_i$ ’ match.
- The bracketing structure of $\langle a \rangle \langle b \rangle [x] \langle c \rangle [y] [z] \langle d \rangle$ is simpler: $[[[]][[]]]$.
- $\langle b \rangle [x]$ and $\langle c \rangle [y]$ are AT-pairs whereas $\langle a \rangle [z]$ is an AT-couple.

Redexes in de Bruijn's notation

Classical Notation	de Bruijn's Notation
$\frac{((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a}{\downarrow\beta}$	$\langle a \rangle \langle b \rangle [x] \langle c \rangle [y] [z] \langle d \rangle z$
$\frac{((\lambda_y.\lambda_z.zd)c)a}{\downarrow\beta}$	$\langle a \rangle \langle c \rangle [y] [z] \langle d \rangle z$
$\frac{(\lambda_z.zd)a}{\downarrow\beta}$	$\langle a \rangle [z] \langle d \rangle z$
ad	$\langle d \rangle a$



- This makes it easy to introduce local/global/mini reductions into the λ -calculus.
- Further study of de Bruijn's notation can be found in [Kamareddine and Nederpelt, 1995, 1996]

Some notions of reduction studied in the literature

Name	In Classical Notation	In de Bruijn's notation
(θ)	$((\lambda_x.N)P)Q$ \downarrow $(\lambda_x.NQ)P$	$\langle Q \rangle \langle P \rangle [x] N$ \downarrow $\langle P \rangle [x] \langle Q \rangle N$
(γ)	$(\lambda_x.\lambda_y.N)P$ \downarrow $\lambda_y.(\lambda_x.N)P$	$\langle P \rangle [x] [y] N$ \downarrow $[y] \langle P \rangle [x] N$
(γ_c)	$((\lambda_x.\lambda_y.N)P)Q$ \downarrow $(\lambda_y.(\lambda_x.N)P)Q$	$\langle Q \rangle \langle P \rangle [x] [y] N$ \downarrow $\langle Q \rangle [y] \langle P \rangle [x] N$
(g)	$((\lambda_x.\lambda_y.N)P)Q$ \downarrow $(\lambda_x.N[y := Q])P$	$\langle Q \rangle \langle P \rangle [x] [y] N$ \downarrow $\langle P \rangle [x] [y := Q] N$
(β_e)	$?$ \downarrow $?$	$\langle Q \rangle \bar{s}[y] N$ \downarrow $\bar{s}[y := Q] N$

A Few Uses of these reductions/term reshuffling

- Regnier [1992] uses θ and γ in analyzing perpetual reduction strategies.
- Term reshuffling is used in [Kfoury et al., 1994; Kfoury and Wells, 1994] in analyzing typability problems.
- [Nederpelt, 1973; de Groote, 1993; Kfoury and Wells, 1995; Kamareddine, 2000] use generalised reduction and/or term reshuffling in relating SN to WN.
- [Ariola et al., 1995] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- [Kamareddine and Nederpelt, 1995; Kamareddine et al., 1999a, 1998; Bloo et al., 1996] shows that they could reduce space/time needs.
- All these works have been heavily influenced by de Bruijn's Automath whose λ -notation facilitated the manipulation of redexes.
- All can be represented clearer in de Bruijn's notation.

Even more: de Bruijn's generalised reduction has better properties

$$\begin{aligned}(\beta) \quad & (\lambda_x.M)N \rightarrow M[x := N] \\(\beta_I) \quad & (\lambda_x.M)N \rightarrow M[x := N] \quad \text{if } x \in FV(M) \\(\beta_K) \quad & (\lambda_x.M)N \rightarrow M \quad \text{if } x \notin FV(M) \\(\theta) \quad & (\lambda_x.N)PQ \rightarrow (\lambda_x.NQ)P \\(\beta_e) \quad & (M)\bar{s}[x]N \rightarrow \bar{s}\{N[x := M]\} \quad \text{for } \bar{s} \text{ well-balanced.}\end{aligned}$$

- [Kamareddine, 2000] shows that β_e satisfies PSN, postponement of K -contraction and conservation (latter 2 properties fail for β -reduction).
- Conservation of β_e : If A is $\beta_e I$ -normalisable then A is β_e -strongly normalisable.
- Postponement of K -contraction : Hence, discard arguments of K -redexes after I -reduction. This gives flexibility in implementation: unnecessary work can be delayed, or even completely avoided.

- Attempts have been made at establishing some reduction relations for which postponement of K -contractions and conservation hold.
- The picture is as follows (-N stands for normalising and $r \in \{\beta_I, \theta_K\}$).

$(\beta_K\text{-postponement for } r)$ If $M \rightarrow_{\beta_K} N \rightarrow_r O$ then $\exists P$ such that $M \twoheadrightarrow_{\beta_I \theta_K}^+ P \twoheadrightarrow_{\beta_K}$
 $(\text{Conservation for } \beta_I)$ If M is $\beta_I\text{-N}$ then M is $\beta_I\text{-SN}$ Barendregt's book
 $(\text{Conservation for } \beta + \theta)$ If M is $\beta_I \theta_K\text{-N}$ then M is $\beta\text{-SN}$ [de Groote, 1993]

- De Groote does not produce these results for a single reduction relation, but for $\beta + \theta$ (this is more restrictive than β_e).
- β_e is the first single relation to satisfy β_K -postponement and conservation.
- [Kamareddine, 2000] shows that:

$(\beta_e K\text{-postponement for } \beta_e)$ If $M \rightarrow_{\beta_e K} N \rightarrow_{\beta_e I} O$ then $\exists P$ such that $M \rightarrow_{\beta_e I} P \twoheadrightarrow_{\beta_e}^+$
 $(\text{Conservation for } \beta_e)$ If M is $\beta_e I\text{-N}$ then M is $\beta_e\text{-SN}$

Theme 3: Types and Functions à la de Bruijn

- *General definition of function* [Frege, 1879] is key to Frege's *formalisation of logic*.
- *Self-application of functions* was at the heart of *Russell's paradox* [Russell, 1902].
- To *avoid paradox* Russell controlled function application via *type theory*.
- [Russell, 1903] gives the first type theory: the *Ramified Type Theory* (RTT).
- RTT is used in *Principia Mathematica* [Whitehead and Russell, 1910¹, 1927²] 1910–1912.
- *Simple theory of types* (STT): [Ramsey, 1926], [Hilbert and Ackermann, 1928].
- Church's *simply typed λ -calculus* $\lambda \rightarrow$ 1940 = λ -calculus + STT.

- The hierarchies of types/orders of **RTT** and **STT** are *unsatisfactory*.
- The *notion of function adopted in the λ -calculus* is *unsatisfactory* (cf. [Kamareddine et al., 2003]).
- Hence, birth of *different systems of functions and types*, each with *different functional power*.
- Frege's functions \neq Principia's functions \neq *λ -calculus* functions (1932).
- Not all functions need to be *fully abstracted* as in the λ -calculus. For some functions, their values are enough.
- *Non-first-class functions* allow us to stay at a lower order (keeping decidability, typability, etc.) without losing the flexibility of the higher-order aspects.
- Non-first-class functions allow placing the type systems of modern theorem provers/programming languages like ML, LF and Automath more accurately in the modern hierarchy of types.

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.

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* $A \rightarrow A$
 - $\lambda_{A:*.}\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*.}A \rightarrow A$
- 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* $(A \rightarrow A)[A := \mathbb{N}]$ or $\mathbb{N} \rightarrow \mathbb{N}$.
 - $(\lambda_{A:*.}\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*.}A \rightarrow A)\mathbb{N} = (A \rightarrow A)[A := \mathbb{N}]$ or $\mathbb{N} \rightarrow \mathbb{N}$.
- $(\lambda x:\alpha.A)B \rightarrow_{\beta} A[x := B]$ $(\Pi x:\alpha.A)B \rightarrow_{\Pi} A[x := B]$
- Write $A \rightarrow A$ as $\Pi_{y:A}.A$ when y not free in A .

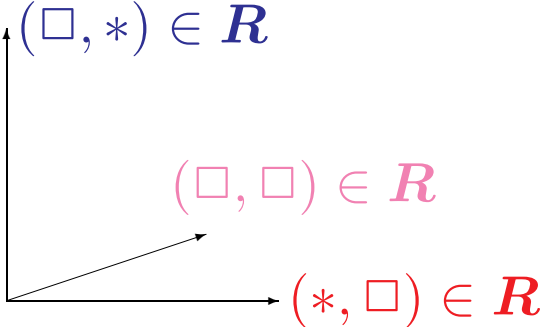
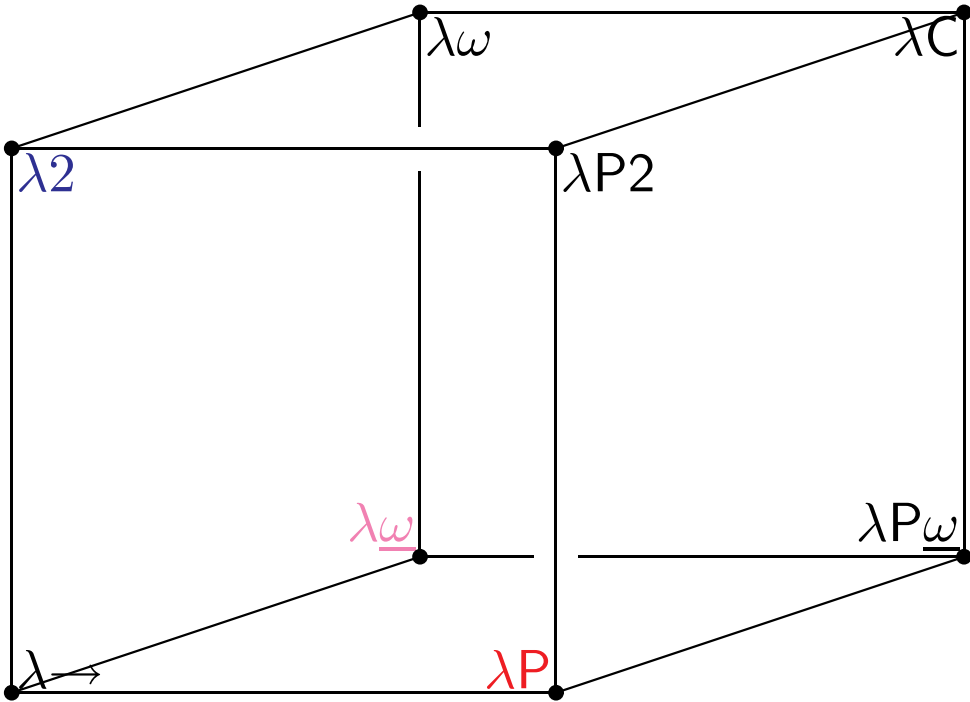
The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \square \mid AB \mid \lambda x:A.B \mid \Pi x:A.B$

- Formation rule:
$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A.B : s_2} \quad \text{if } (s_1, s_2) \in \mathbf{R}$$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda \rightarrow$	$(*, *)$				λ^τ	[Church, 1940; B
$\lambda 2$	$(*, *)$	$(\square, *)$			F	[Girard, 1972; Re
λP	$(*, *)$		$(*, \square)$		AUT-QE, LF	[Bruijn, 1968; Ha
$\lambda \underline{\omega}$	$(*, *)$			(\square, \square)	POLYREC	[Renardel de Lava
$\lambda P2$	$(*, *)$	$(\square, *)$	$(*, \square)$			[Longo and Mogg
$\lambda \omega$	$(*, *)$	$(\square, *)$		(\square, \square)	$F\omega$	[Girard, 1972]
$\lambda P \underline{\omega}$	$(*, *)$		$(*, \square)$	(\square, \square)		
λC	$(*, *)$	$(\square, *)$	$(*, \square)$	(\square, \square)	CC	[Coquand and Hu

The Barendregt Cube



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

- $$\frac{y : * \vdash y : * \quad y : *, x : y \vdash y : *}{y : * \vdash \Pi x : y . y : *}$$
 by $(\Pi) (*, *)$
- $$\frac{y : *, x : y \vdash x : y \quad y : * \vdash \Pi x : y . y : *}{y : * \vdash \lambda x : y . x : \Pi x : y . y}$$
 by (λ)
- $$\frac{\vdash * : \square \quad y : * \vdash \Pi x : y . y : *}{\vdash \Pi y : * . \Pi x : y . y : *}$$
 by $(\Pi) (\square, *)$
- $$\frac{y : * \vdash \lambda x : y . x : \Pi x : y . y \quad \vdash \Pi y : * . \Pi x : y . y : *}{\vdash \lambda y : * . \lambda x : y . x : \Pi y : * . \Pi x : y . y}$$
 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 view of ML

- When Robin Milner designed the language ML, he wanted 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 => x) in (id id) end` as this Cube term
 $(\lambda \text{id} : (\prod \alpha : *. \alpha \rightarrow \alpha). \text{id}(\beta \rightarrow \beta)(\text{id } \beta))(\lambda \alpha : *. \lambda x : \alpha. x)$
- To type this in the Cube, the $(\square, *)$ rule is needed (i.e., $\lambda 2$).
- ML's typing rules forbid this expression:
`let val id = (fn x => x) in (fn y => y y)(id id) end`
 Its equivalent Cube term is this well-formed typable term of $\lambda 2$:
 $(\lambda \text{id} : (\prod \alpha : *. \alpha \rightarrow \alpha). (\lambda y : (\prod \alpha : *. \alpha \rightarrow \alpha). y(\beta \rightarrow \beta)(y \beta)) (\lambda \alpha : *. \text{id}(\alpha \rightarrow \alpha)(\text{id } \alpha))) (\lambda \alpha : *. \lambda x : \alpha. x)$
- Therefore, ML should not have the full Π -formation rule $(\square, *)$.
- ML has limited access to the rule $(\square, *)$.
- 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 $(*, \square)$ is very restricted in the practical use of LF [Geuvers, 1993].
- The only need for a type $\Pi x:A.B : \square$ is when the Propositions-As-Types principle PAT is applied during the construction of the type $\Pi \alpha:\text{prop}.*$ of the operator Prf where for a proposition Σ , $\text{Prf}(\Sigma)$ is the type of proofs of Σ .

$$\frac{\text{prop}:* \vdash \text{prop}:* \quad \text{prop}:*, \alpha:\text{prop} \vdash *: \square}{\text{prop}:* \vdash \Pi \alpha:\text{prop}.* : \square}.$$

- In LF, this is the only point where the Π -formation rule $(*, \square)$ is used.
- But, Prf is only used when applied $\Sigma:\text{prop}$. 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 $(*, \square)$ as a parameter instead of a Π -formation rule.
- [Kamareddine et al., 2001] finds a more precise position of LF on the Cube (**between $\lambda \rightarrow$ 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:
$$x_1 : A_1, \dots, x_n : A_n \vdash g(x_1, \dots, x_n) = t : T.$$
Here g is a new name, an abbreviation for the expression t of type T and x_1, \dots, x_n are the parameters of g , with respective types A_1, \dots, 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** [Bentham 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 $c(b_1, \dots, b_n)$ with b_1, \dots, b_n terms of certain types and $c \in \mathcal{C}$.
- b_1, \dots, b_n are called the *parameters* of $c(b_1, \dots, b_n)$.
- **R allows** several kinds of **Π -constructs**. We also use a set **P** of (s_1, s_2) where $s_1, s_2 \in \{*, \square\}$ to **allow** several kinds of **parametric constants**.
- $(s_1, s_2) \in \mathbf{P}$ means that we **allow** parametric constants $c(b_1, \dots, b_n) : A$ where b_1, \dots, b_n have types B_1, \dots, B_n of sort s_1 , and A is of type s_2 .
- If both $(*, s_2) \in \mathbf{P}$ and $(\square, s_2) \in \mathbf{P}$ then **combinations of parameters allowed**. For example, it is allowed that B_1 has type $*$, whilst B_2 has type \square .

The Cube with parametric constants

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

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

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

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

$$\Delta \equiv x_1 : B_1, \dots, x_n : B_n.$$

$$\Delta_i \equiv x_1 : B_1, \dots, x_{i-1} : B_{i-1}$$

Properties of the Refined Cube

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

Example

- $R = \{(*, *), (*, \square)\}$

$$P_1 = \emptyset \quad P_2 = \{(*, *)\} \quad P_3 = \{(*, \square)\} \quad P_4 = \{(*, *), (*, \square)\}$$

All $\lambda R P_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

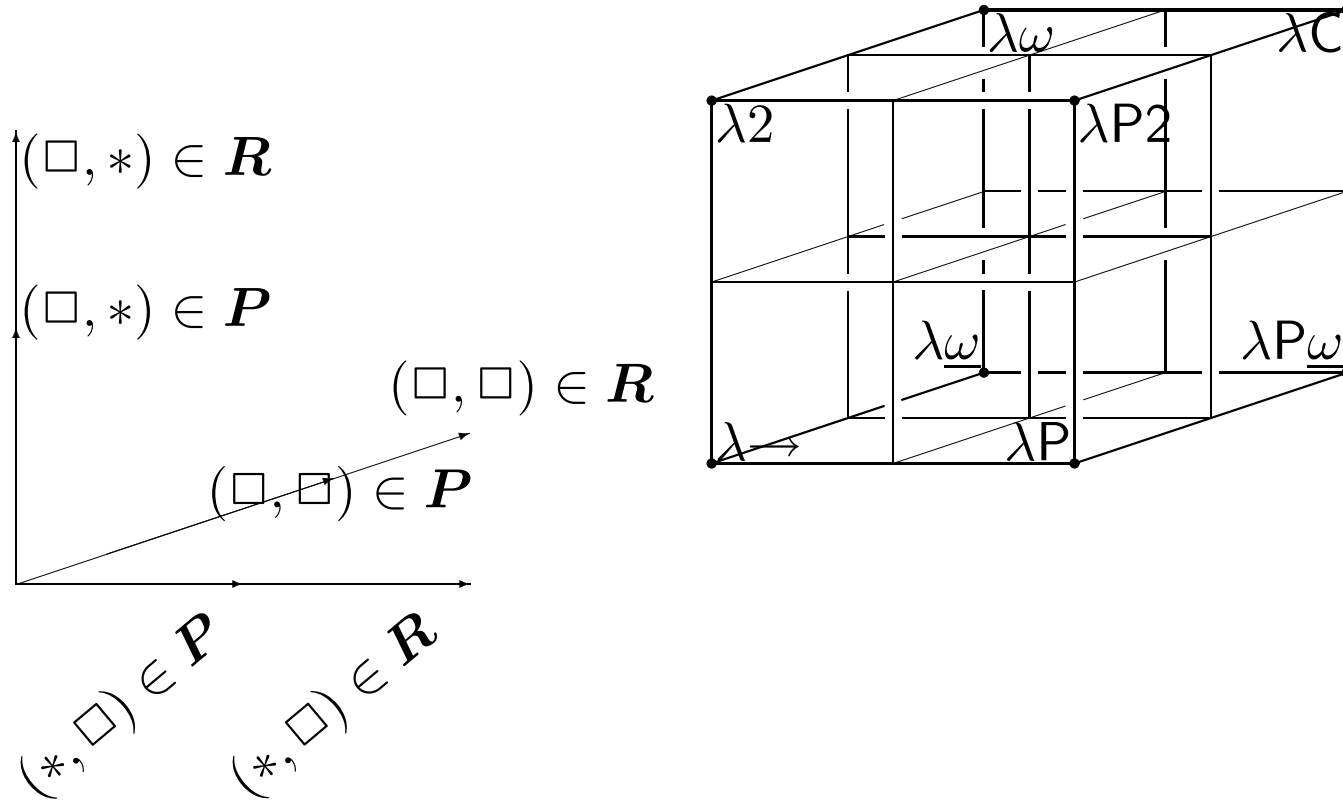
- $R_5 = \{(*, *)\} \quad P_5 = \{(*, *), (*, \square)\}.$

$\lambda \rightarrow < \lambda R_5 P_5 < \lambda P$: we can talk about predicates:

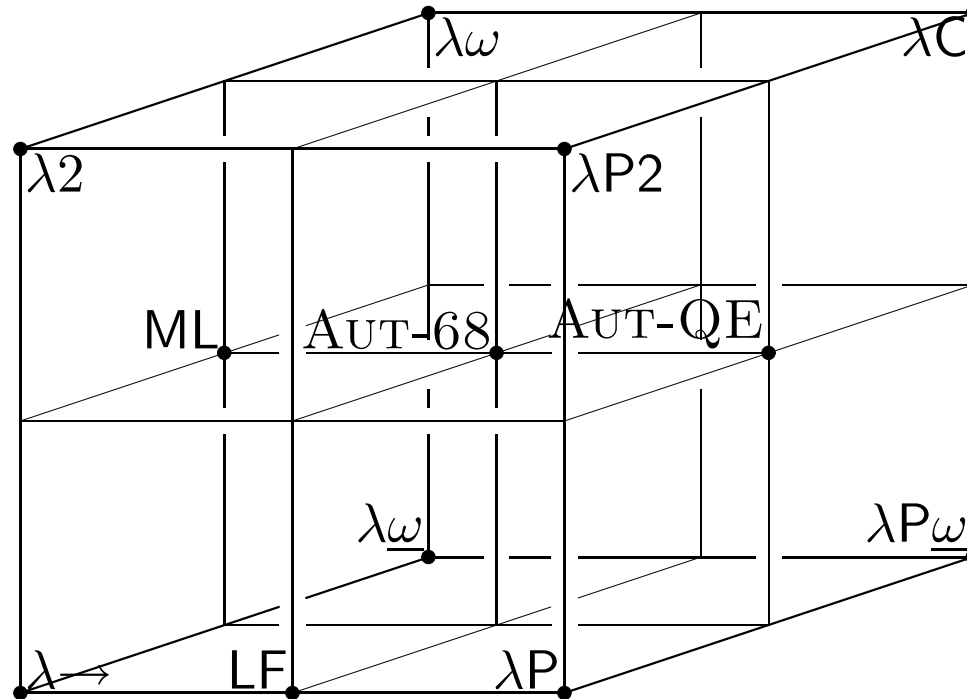
$$\begin{aligned} \alpha & : * , \\ \text{eq}(x:\alpha, y:\alpha) & : * , \\ \text{refl}(x:\alpha) & : \text{eq}(x, x), \\ \text{symm}(x:\alpha, y:\alpha, p:\text{eq}(x, y)) & : \text{eq}(y, x), \\ \text{trans}(x:\alpha, y:\alpha, z:\alpha, p:\text{eq}(x, y), q:\text{eq}(y, z)) & : \text{eq}(x, z). \end{aligned}$$

eq not possible in $\lambda \rightarrow$.

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 $\lambda\mathbf{R}$ where $\mathbf{R} = \{(*, *), (*, \square), (\square, *)\}$:

$$\mathbf{Ind} = \Pi p:(\mathbb{N} \rightarrow *) . p0 \rightarrow (\Pi n:\mathbb{N} . \Pi m:\mathbb{N} . pn \rightarrow Snm \rightarrow pm) \rightarrow \Pi n:\mathbb{N} . pn \quad (1)$$

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

- 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):

$$\mathbf{ind}(p:\mathbb{N} \rightarrow *, q:p0, r:(\Pi n:\mathbb{N} . \Pi m:\mathbb{N} . pn \rightarrow Snm \rightarrow pm)) : \Pi n:\mathbb{N} . pn \quad (2)$$

- The logician's type **Ind** is not needed by the mathematician and the types that occur in 2 can all be constructed in $\lambda\mathbf{R}$ with $\mathbf{R} = \{(*, *)(*, \square)\}$.

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

$$\mathcal{T}_b ::= \mathcal{V} \mid \mathbf{S} \mid \mathcal{T}_b \mathcal{T}_b \mid b\mathcal{V}:\mathcal{T}_b.\mathcal{T}_b$$

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

- The β -reduction rule becomes $(b) \quad (b_{x:A}.B)C \rightarrow_b B[x := C]$.

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

- The type formation rule becomes

$$(b_1) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash (b_{x:A}.B) : s_2} \quad (s_1, s_2) \in \mathbf{R}$$

$$\begin{array}{l}
\text{(axiom)} \quad \langle \rangle \vdash * : \square \\
\\
\text{(start)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x:A \vdash x : A} \quad x \notin \text{DOM}(\Gamma) \\
\\
\text{(weak)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x:C \vdash A : B} \quad x \notin \text{DOM}(\Gamma) \\
\\
\text{(b}_2\text{)} \quad \frac{\Gamma, x:A \vdash b : B \quad \Gamma \vdash (bx:A.B) : s}{\Gamma \vdash (bx:A.b) : (bx:A.B)} \\
\\
\text{(appb)} \quad \frac{\Gamma \vdash F : (bx:A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x:=a]} \\
\\
\text{(conv)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta} B'}{\Gamma \vdash A : B'}
\end{array}$$

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

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

Isomorphism of the cube and the \flat -cube

- If $\Gamma \vdash A : B$ then $\bar{\Gamma} \vdash_{\flat} \bar{A} : \bar{B}$.
- 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_b and \rightarrow_b [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 $\Gamma \vdash_b A : B_1$ and $\Gamma \vdash_b A : B_2$, then $B_1 \overset{\diamond}{=} B_2$.
2. If $\Gamma \vdash_b A_1 : B_1$ and $\Gamma \vdash_b A_2 : B_2$ and $A_1 =_b A_2$, then $B_1 \overset{\diamond}{=} B_2$.
3. If $\Gamma \vdash_b B_1 : s_1$, $B_1 =_b B_2$ and $\Gamma \vdash_b A : B_2$ then $\Gamma \vdash_b B_2 : s_1$.
4. Assume $\Gamma \vdash_b A : B_1$ and $(\Gamma \vdash_b A : B_1)^{-1} = (\Gamma', A', B'_1)$. Then $B_1 =_b B_2$ if:
 - (a) either $\Gamma \vdash_b A : B_2$, $(\Gamma \vdash_b A : B_2)^{-1} = (\Gamma', A'', B'_2)$ and $B'_1 =_\beta B'_2$,
 - (b) or $\Gamma \vdash_b C : B_2$, $(\Gamma \vdash_b C : B_2)^{-1} = (\Gamma', C', B'_2)$ and $A' =_\beta C'$.

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

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

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

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

[Kamareddine et al., 1999b] 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 λ -cube

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

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

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

Failure of correctness of types and subject reduction

- **Correctness of types no longer holds.** With (appl_b) one can have $\Gamma \vdash A : B$ without $B \equiv \square$ or $\exists S . \Gamma \vdash B : S$.
- For example, $z : *, x : z \vdash (\lambda_{y:z}.y)x : (\lambda_{y:z}.z)x$ yet $(\lambda_{y:z}.z)x \not\equiv \square$ and $\forall s . z : *, x : z \not\vdash (\lambda_{y:z}.z)x : s$.
- **Subject Reduction no longer holds.** That is, with (appl_b): $\Gamma \vdash A : B$ and $A \rightarrow A'$ may not imply $\Gamma \vdash A' : B$.
- For example, $z : *, x : z \vdash (\lambda_{y:z}.y)x : (\lambda_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \rightarrow_b x$, but one can't show $z : *, x : z \vdash x : (\lambda_{y:z}.z)x$.

Solving the problem

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

$$\begin{array}{l}
 \text{(start-def)} \quad \frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B:A \vdash x : A} \quad x \notin \text{DOM}(\Gamma) \\
 \\
 \text{(weak-def)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D:C \vdash A : B} \quad x \notin \text{DOM}(\Gamma) \\
 \\
 \text{(def)} \quad \frac{\Gamma, x = B:A \vdash C : D}{\Gamma \vdash (\lambda x:A.C)B : D[x := B]} \\
 \\
 \text{(new-conv)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad \Gamma \vdash B =_{def} B'}{\Gamma \vdash A : B'} \\
 \\
 \text{(appl}\lambda) \quad \frac{\Gamma \vdash F : \lambda_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\lambda_{x:A}.B)a}
 \end{array}$$

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

- If $B =_b B'$ 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, 1910¹, 1927²].

Correctness of types holds.

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

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

Subject Reduction holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\lambda_{y:z}.y)x : (\lambda_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \rightarrow_{\beta} x$ and we need to show that $z : *, x : z \vdash x : (\lambda_{y:z}.z)x$.
- Here is how the latter formula now holds:
 - $z : *, x : z \vdash x : z$ (start and weakening)
 - $z : *, x : z \vdash (\lambda_{y:z}.z)x : *$ (from 1 above)
 - $z : *, x : z \vdash x : (\lambda_{y:z}.z)x$ (conversion, a , b , and $z =_{\beta} (\lambda_{y:z}.z)x$)

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 . \square

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.

L^AT_EX example

draft documents		✓
public documents		✓
computations and proofs		X

```
\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.

	draft documents	X
Coq example	public documents	X
	computations and proofs	✓

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

Lemma `plus_sym`: $(n,m:\text{nat}) (n+m)=(m+n)$.

Proof.

```
Intros n m ; Elim n ; Simpl_rewrite ; Auto with arith.
```

```
Intros y H ; Elim (plus_n_Sm m y) ; Simpl_rewrite ; 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 *freedom of movement* between mathematics, logic and computation.
- At every stage, we must have *the choice* of the level of formality and the depth of computation.

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

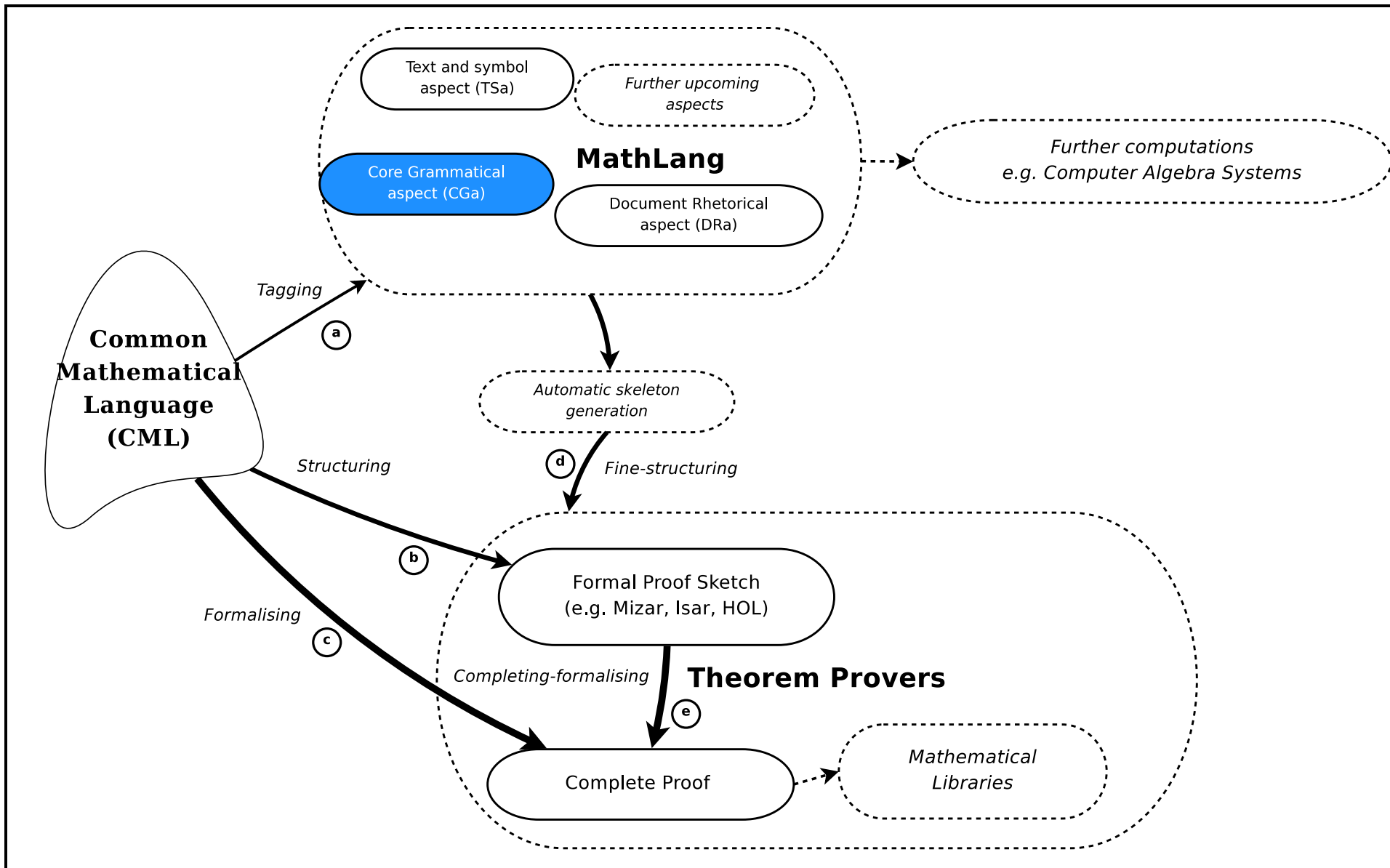
Can we formalise a mathematical text, avoiding as much as possible the ambiguities of natural language, while still guaranteeing the following four goals?

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.)

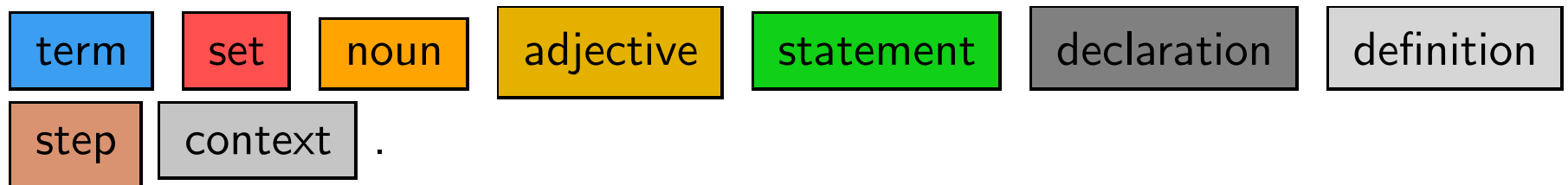
Mathlang	draft documents	✓
	public documents	✓
	computations and proofs	✓

- 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.



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".



- 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.

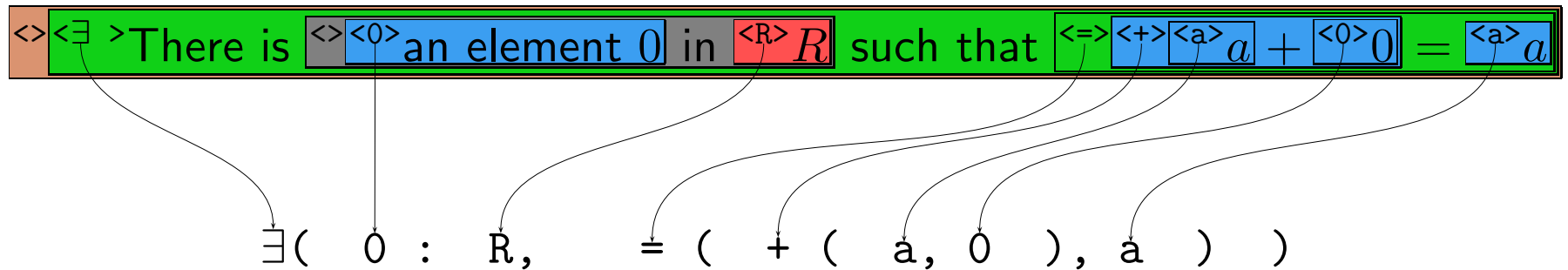


Figure 1: Example of CGa encoding of CML text

Weak Type Theory

In Weak Type Theory (or W_{TT}) we have the following linguistic categories:

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

Categories of syntax of WTT

Other category	abstract syntax	symbol
<i>expressions</i>	$\mathcal{E} = T \mathcal{S} \mathcal{N} P$	E
<i>parameters</i>	$\mathcal{P} = T \mathcal{S} P$ (note: $\vec{\mathcal{P}}$ is a list of \mathcal{P} s)	P
<i>typings</i>	$\mathbf{T} = \mathcal{S} : \text{SET} \mathcal{S} : \text{STAT} T : \mathcal{S} T : \mathcal{N} T : \mathcal{A}$	T
<i>declarations</i>	$\mathcal{Z} = V^S : \text{SET} V^P : \text{STAT} V^T : \mathcal{S} V^T : \mathcal{N}$	Z

level	category	abstract syntax	symbol
atomic	<i>variables</i>	$V = V^T V^S V^P$	x
	<i>constants</i>	$C = C^T C^S C^N C^A C^P$	c
	<i>binders</i>	$B = B^T B^S B^N B^A B^P$	b
phrase	<i>terms</i>	$T = C^T(\vec{\mathcal{P}}) B_Z^T(\mathcal{E}) V^T$	t
	<i>sets</i>	$S = C^S(\vec{\mathcal{P}}) B_Z^S(\mathcal{E}) V^S$	s
	<i>nouns</i>	$\mathcal{N} = C^N(\vec{\mathcal{P}}) B_Z^N(\mathcal{E}) \mathcal{AN}$	n
	<i>adjectives</i>	$\mathcal{A} = C^A(\vec{\mathcal{P}}) B_Z^A(\mathcal{E})$	a
sentence	<i>statements</i>	$P = C^P(\vec{\mathcal{P}}) B_Z^P(\mathcal{E}) V^P$	S
	<i>definitions</i>	$\mathcal{D} = \mathcal{D}^\varphi \mathcal{D}^P$ $\mathcal{D}^\varphi = C^T(\vec{V}) := T C^S(\vec{V}) := S $ $C^N(\vec{V}) := \mathcal{N} C^A(\vec{V}) := \mathcal{A}$ $\mathcal{D}^P = C^P(\vec{V}) := P$	D
discourse	<i>contexts</i>	$\mathbf{\Gamma} = \emptyset \mathbf{\Gamma}, \mathcal{Z} \mathbf{\Gamma}, P$	Γ
	<i>lines</i>	$\mathbf{l} = \mathbf{\Gamma} \triangleright P \mathbf{\Gamma} \triangleright \mathcal{D}$	l
	<i>books</i>	$\mathbf{B} = \emptyset \mathbf{B} \circ \mathbf{l}$	B

Derivation rules

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

Examples of derivation rules

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

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

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

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

$$\frac{B; \Gamma \vdash p :: P}{\vdash B \circ \Gamma \triangleright p :: \text{book}}$$

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

Properties of WTT

- *Every variable is declared* If $B; \Gamma \vdash \Phi :: \mathbf{W}$ then $FV(\Phi) \subseteq \text{dvar}(\Gamma)$.
- *Correct subcontexts* If $B \vdash \Gamma :: \text{cont}$ and $\Gamma' \subseteq \Gamma$ then $B \vdash \Gamma' :: \text{cont}$.
- *Correct subbooks* If $\vdash B :: \text{book}$ and $B' \subseteq B$ then $\vdash B' :: \text{book}$.
- *Free constants are either declared in book or in contexts* If $B; \Gamma \vdash \Phi :: \mathbf{W}$, then $FC(\Phi) \subseteq \text{prefcons}(B) \cup \text{defcons}(B)$.
- *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 $\vdash B :: \text{book}$ and $\Gamma \triangleright c(x_1, \dots, x_n) := \Phi$ a line in B .
- We write $B \vdash c(P_1, \dots, P_n) \xrightarrow{\delta} \Phi[x_i := P_i]$.
- *Church-Rosser* If $B \vdash \Phi \xrightarrow{\delta} \Phi_1$ and $B \vdash \Phi \xrightarrow{\delta} \Phi_2$ then there exists Φ_3 such that $B \vdash \Phi_1 \xrightarrow{\delta} \Phi_3$ and $B \vdash \Phi_2 \xrightarrow{\delta} \Phi_3$.
- *Strong Normalisation* Let $\vdash B :: \text{book}$. 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 \mathcal{M} be a set ,

y and x are natural numbers ,

if x belongs to \mathcal{M}

then $x + y = y + x$

CGa Weak Type checking detects grammatical errors

Let \mathcal{M} be a set ,

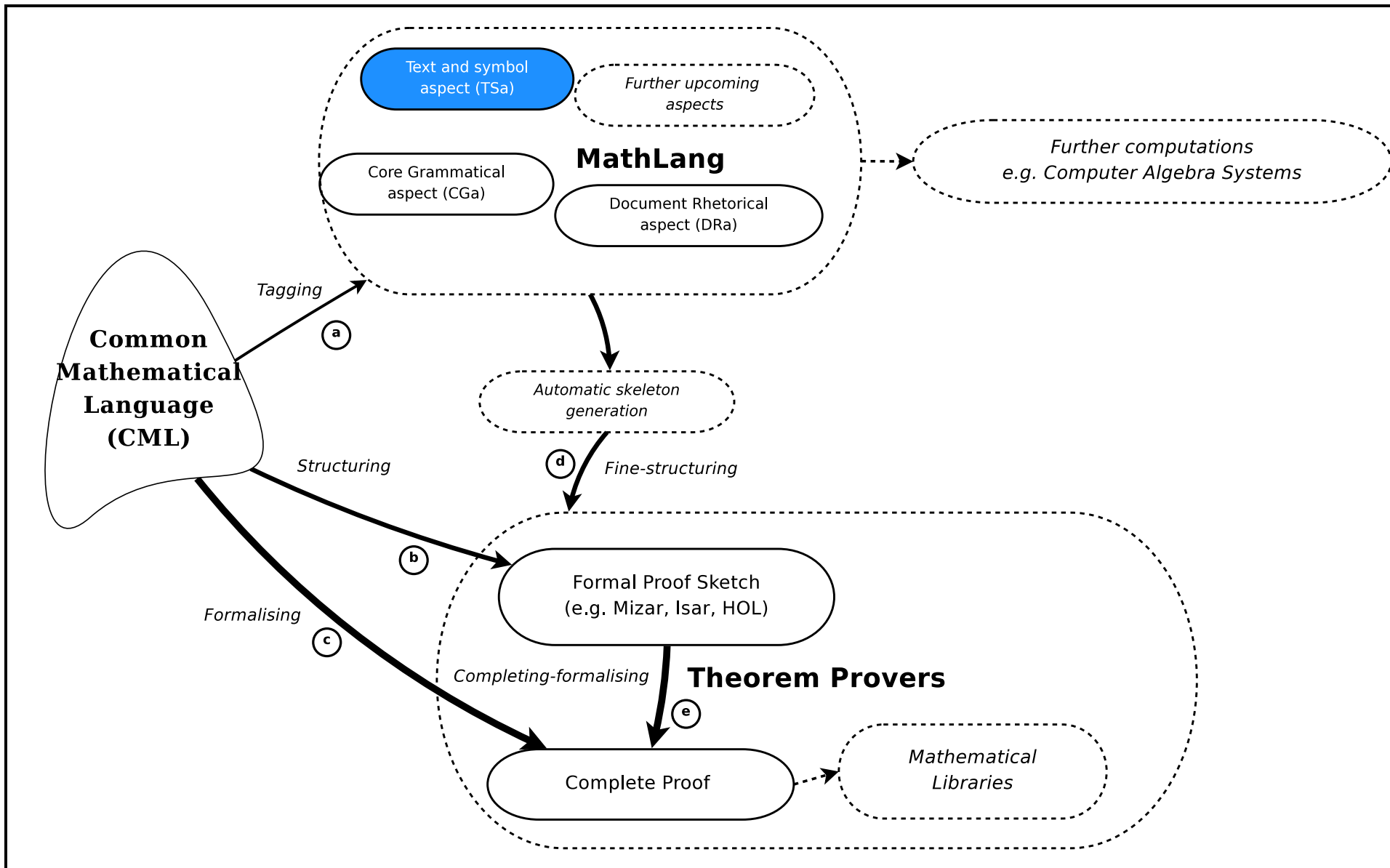
y and x are natural numbers ,

if x belongs to \mathcal{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.



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 $^R R$ such that $\text{eq plus } ^a a + ^0 0 = ^a a$.

$\{ 0 : R; \text{ eq (plus (a, 0), a) }; \}$

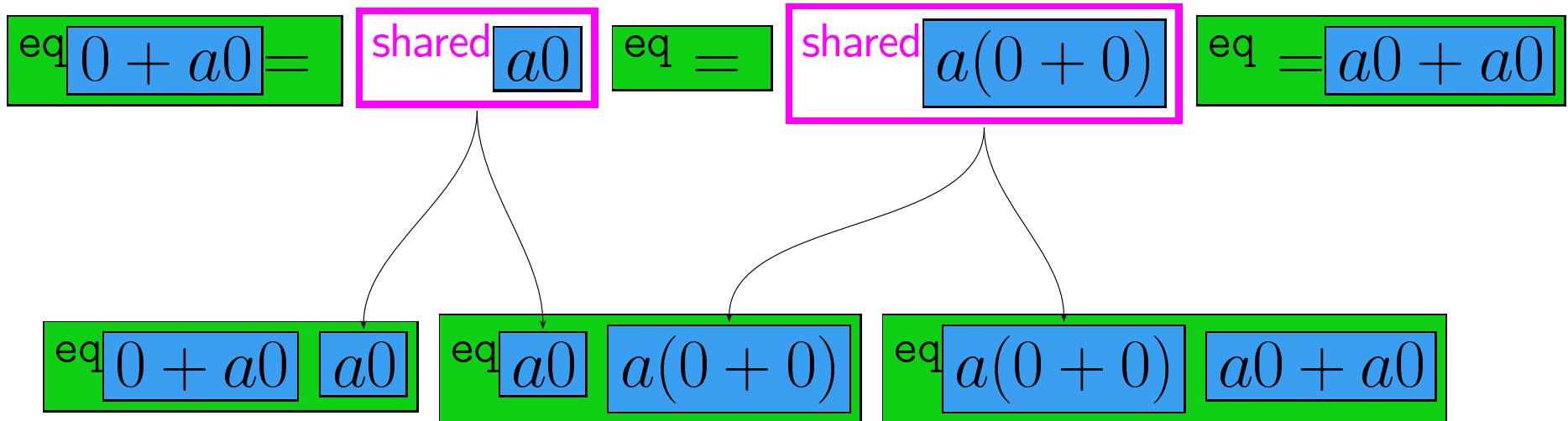
There is 0 an element 0 in $^R R$ such that $\text{eq plus } ^a a + ^0 0 = ^a a$.

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

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

Rewrite rules enable natural language representation

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



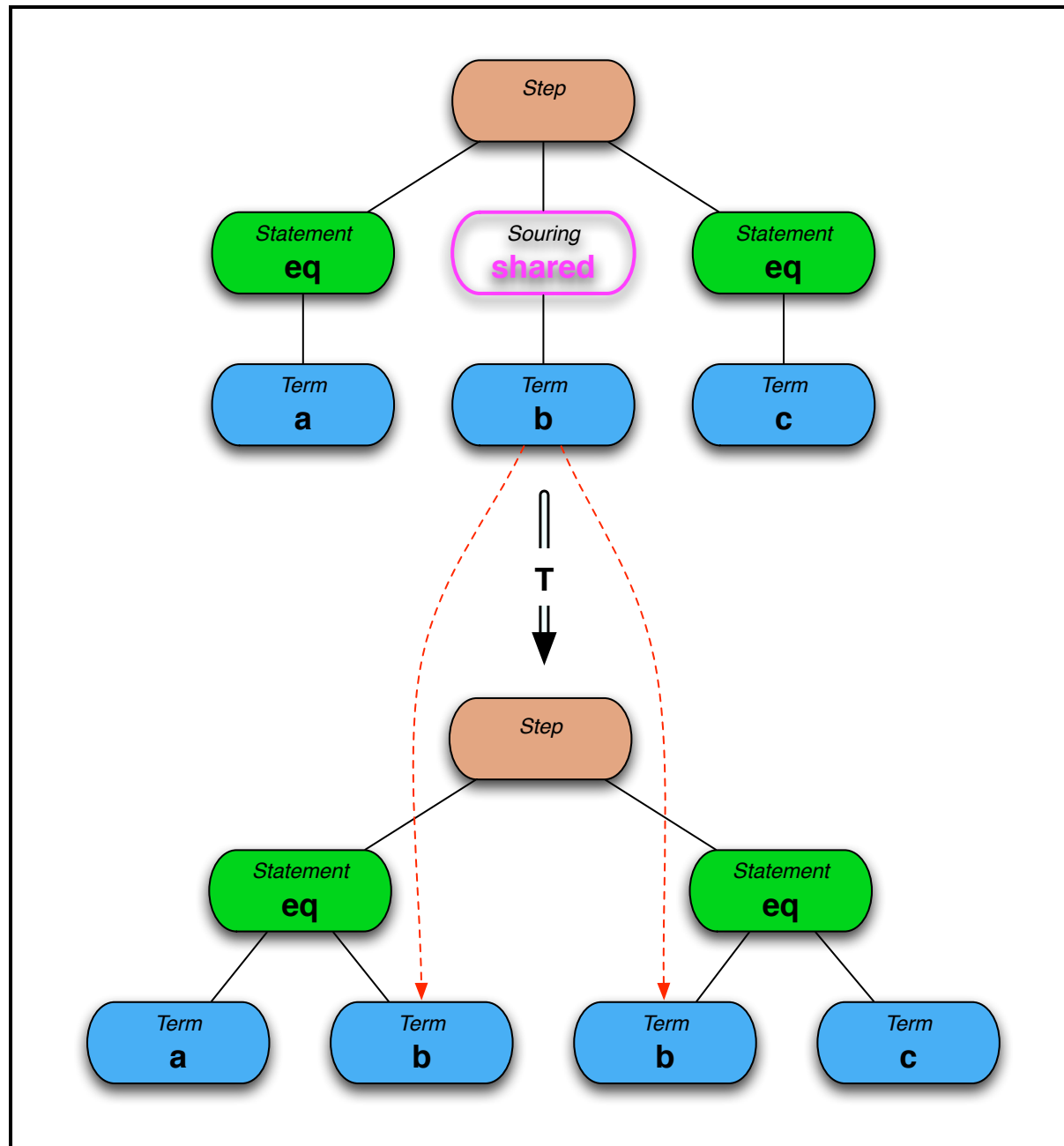
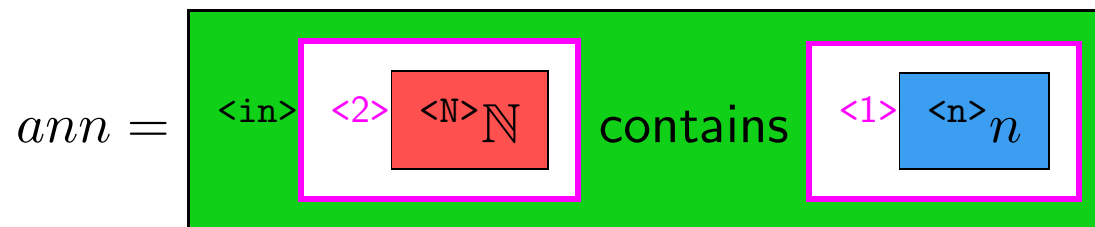
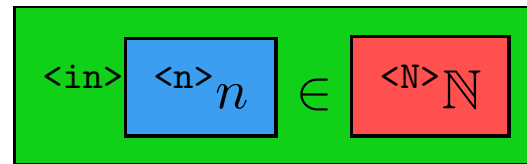


Figure 2: Example for a simple shared souring

reordering/position Sourcing



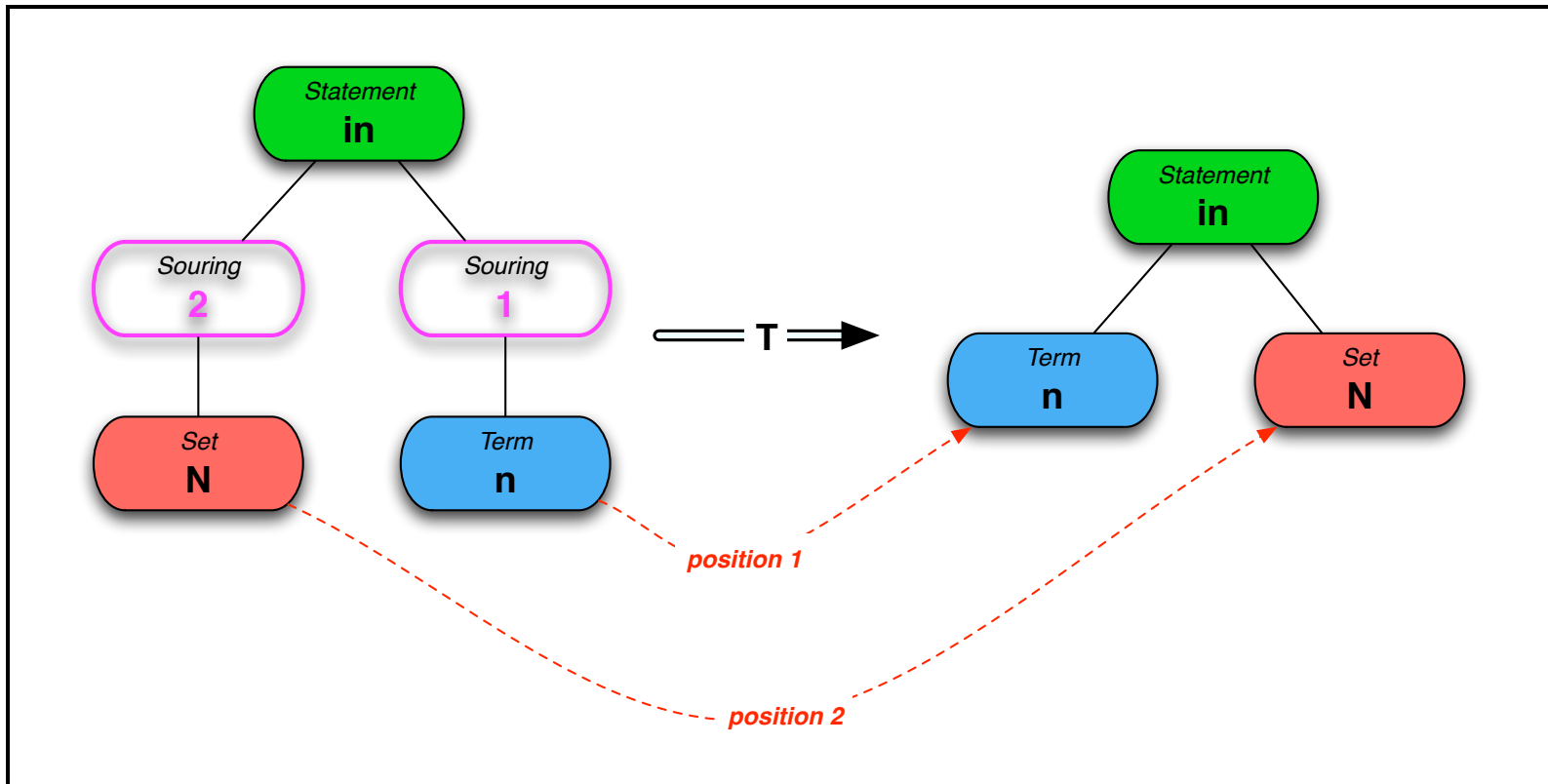
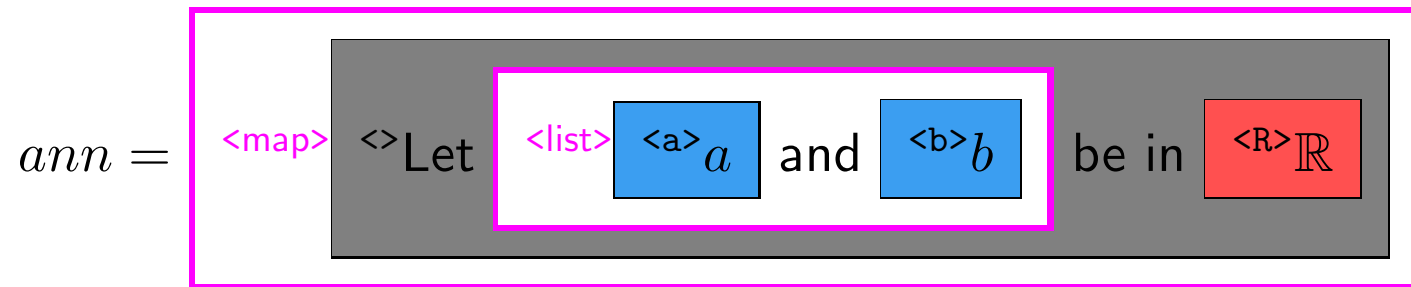
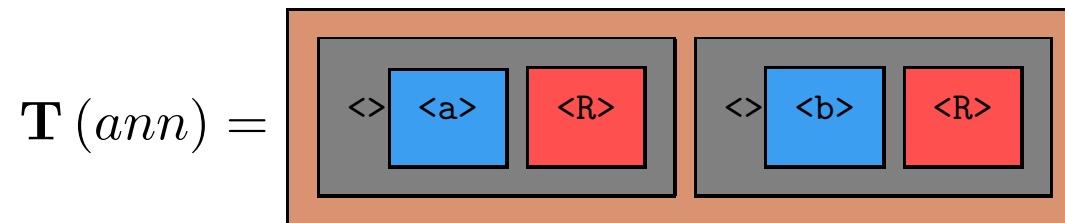


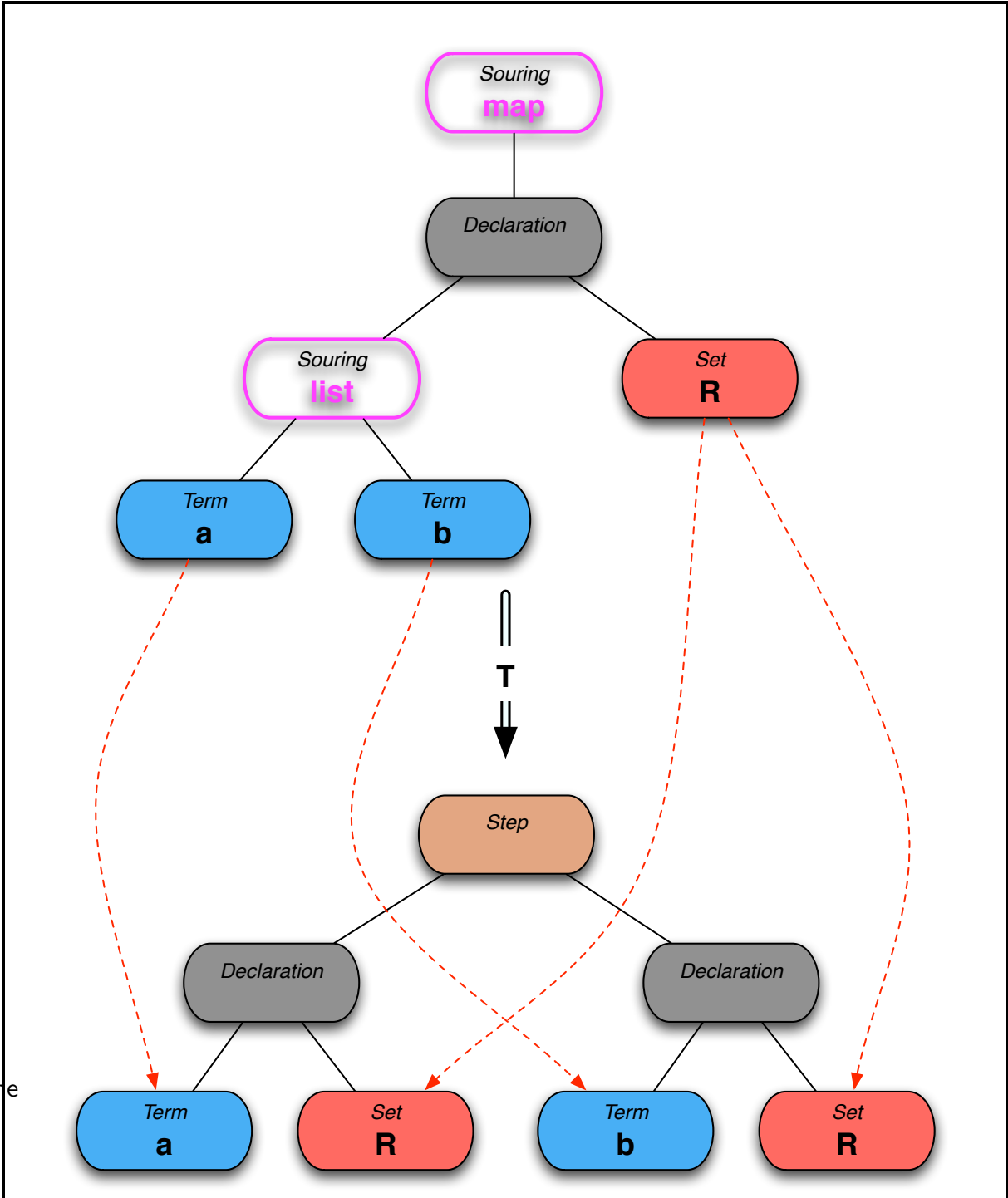
Figure 3: Example for a position souring

map souring



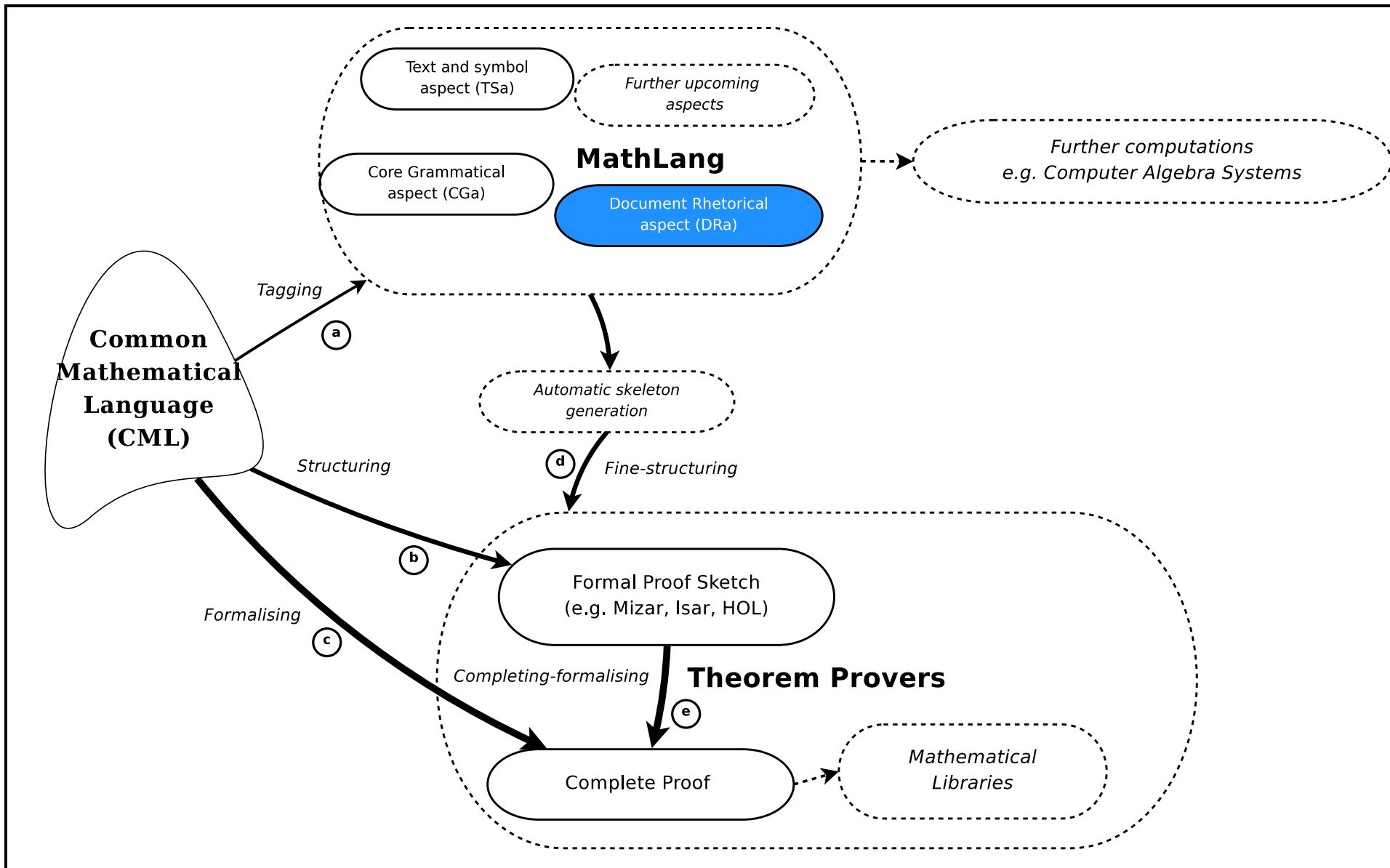
This is expanded to





How complete is TSa?

- TSa provides useful interface facilities but it is still under development.
- So far, only simple rewrite (sourcing) rules are used and they are not comprehensive. E.g., unable to cope with things like $\overbrace{x = \dots = x}^{n \text{ times}}$.
- The TSa theory and metatheory need development.



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
<i>Instances of the StructuralRhetoricalRole class:</i> preamble, part, chapter, section, paragraph, <i>etc.</i>
<i>Instances of the MathematicalRhetoricalRole class:</i> lemma, corollary, theorem, conjecture, definition, axiom, claim, proposition, assertion, proof, exercise, example, problem, solution, <i>etc.</i>
Relation
<i>Types of relations:</i> 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 $m > 0$, 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. 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: $m^2 = 2n^2 \iff \boxed{A} n = n = 0$

Proof.

Define on \mathbb{N} the predicate:

$$P(m) \iff \exists n. m^2 = 2n^2 \ \& \ m > 0. \quad \boxed{E}$$

Claim. $P(m) \implies \exists \boxed{F} < m. P(m').$

Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds squared are 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 $m' = n$. \boxed{B}

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 $\boxed{H}(m)$. Contradiction.

Therefore $m = 0$. But then also $n = \boxed{I}$. \square

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

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 $\sqrt{2} \notin \mathbb{Q}$. \square \boxed{D}

(*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:

$$P(m) \text{ uses } \exists n. m^2 = 2n^2 \ \& \ m > 0.$$

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, n^2 is odd, so n is odd. 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 $m' = n$.

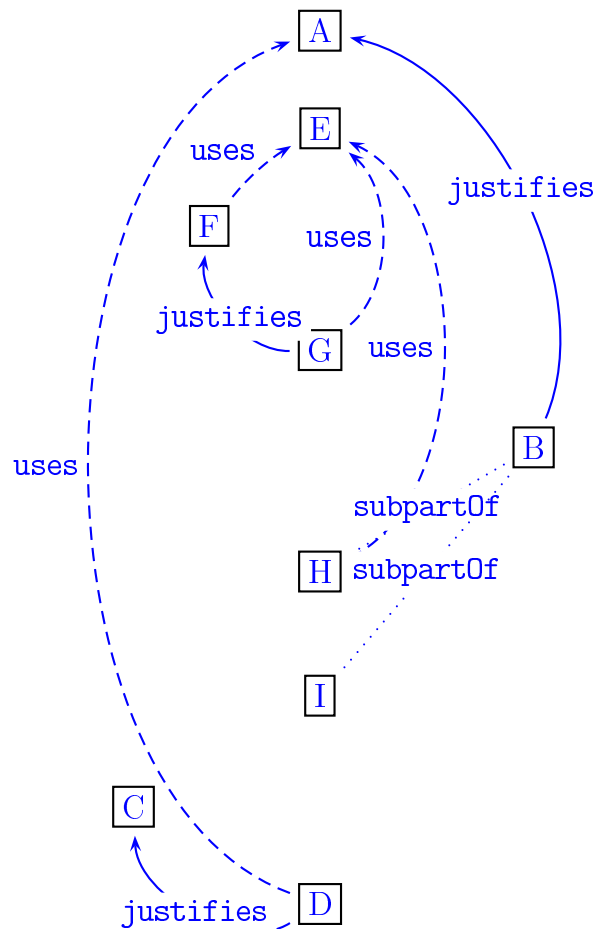
By the claim $\forall m \in \mathbb{N}. \neg P(m)$, since there are no descending sequences of natural numbers.

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

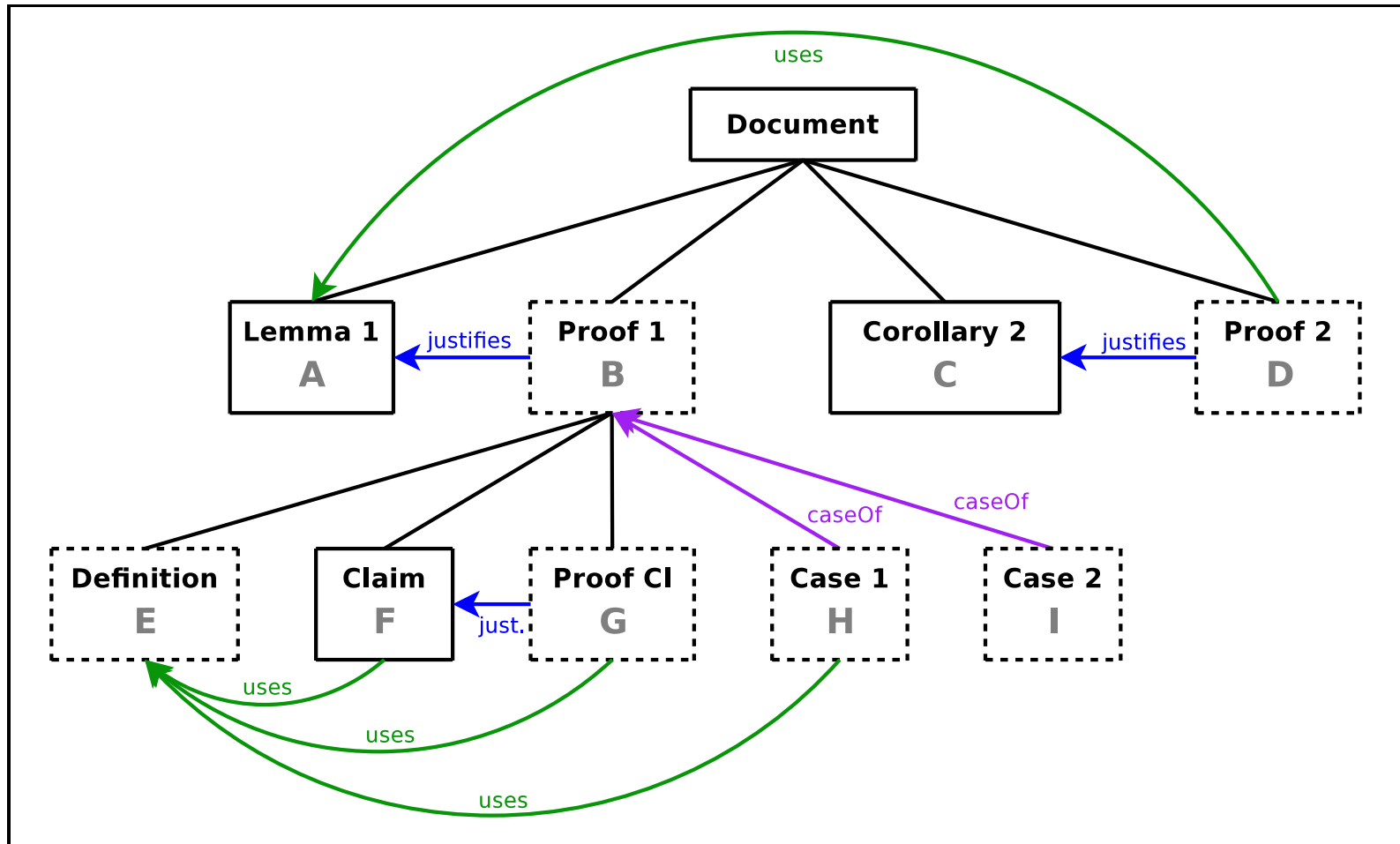
with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction.

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':

$$\mathcal{IN}v := \mathcal{DF}v \cup \mathcal{DC}v \cup \{s \mid s \in \mathcal{ST}v \wedge s \notin \mathcal{ENV}v\} \cup \bigcup_{c \text{ childOf } v} \mathcal{IN}c$$
 - Used symbol: $\mathcal{USE}v := \mathcal{T}v \cup \mathcal{S}v \cup \mathcal{N}v \cup \mathcal{A}v \cup \mathcal{ST}v \cup \bigcup_{c \text{ childOf } v} \mathcal{USE}c$
 - 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


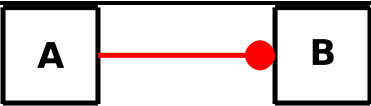
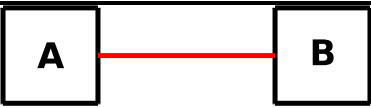
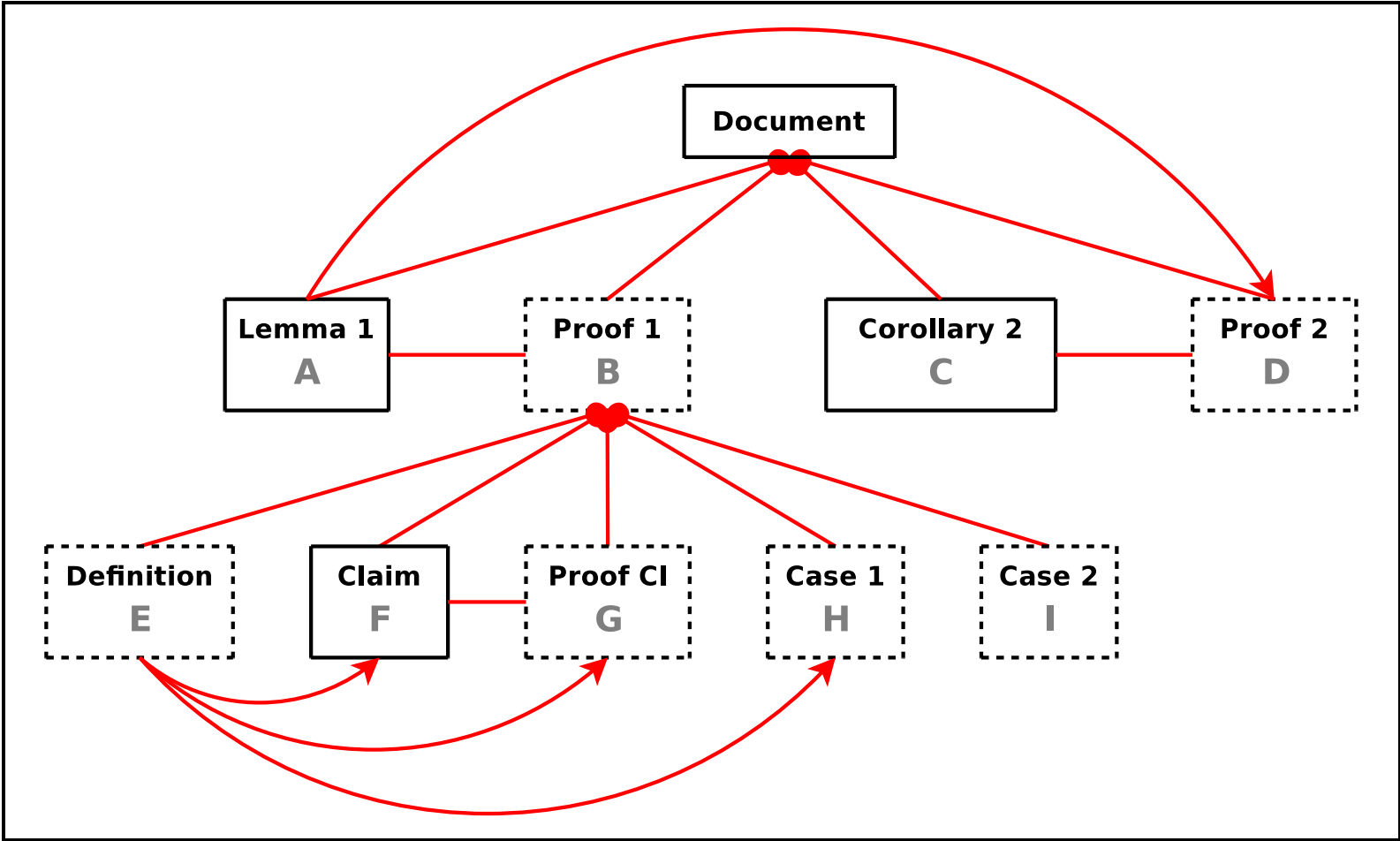
(A, uses, B)	$A \succ B$	
(A, caseOf, B)	$A \preceq B$	
$(A, \text{justifies}, B)$	$A \leftrightarrow B$	

Table 1: Graphical representation of edges in the GoTO

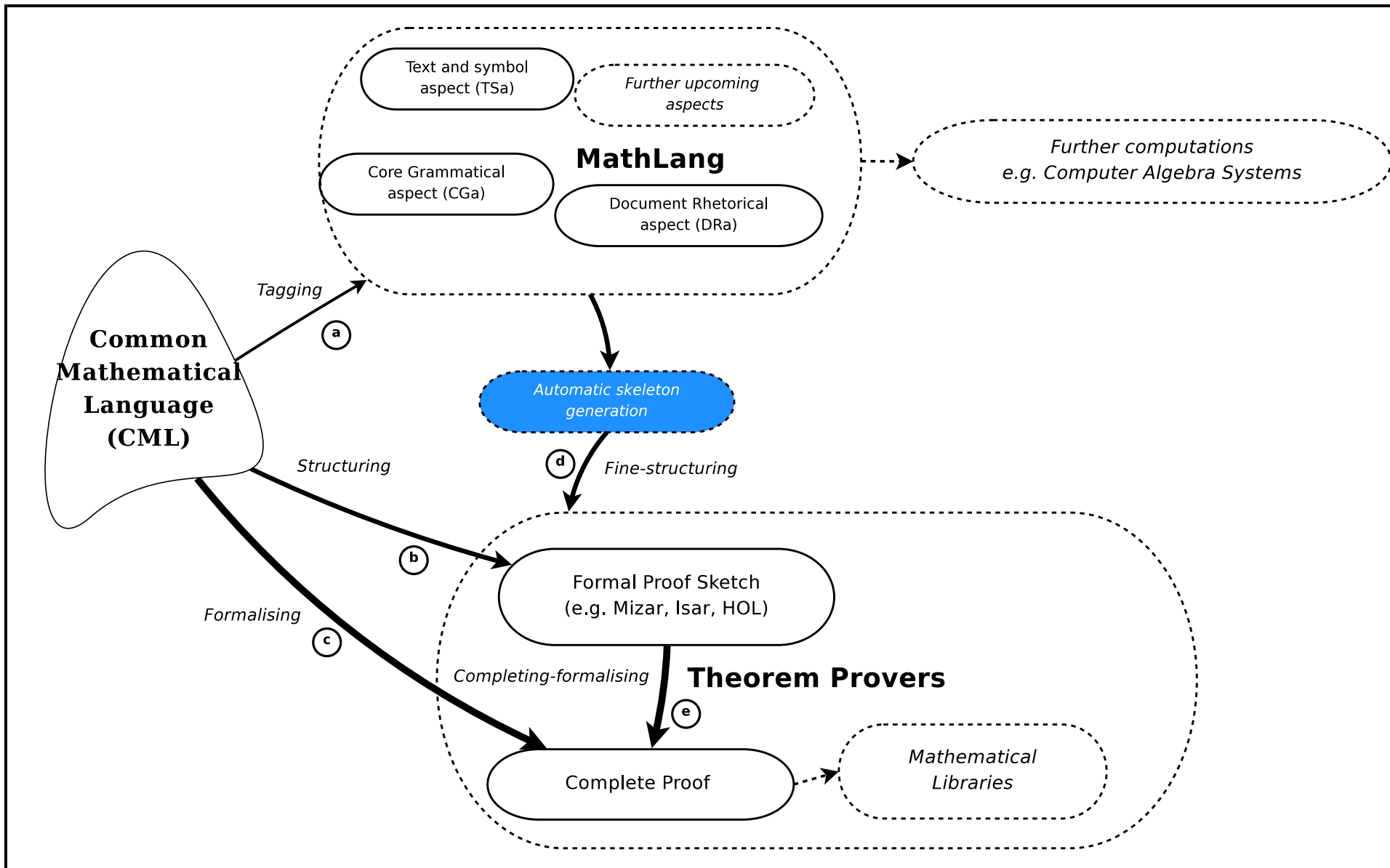
The GoTO can be generated automatically from the DG and therefore (since the DG can be produced automatically from an annotated document) automatically from an annotated document.

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.



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)

Definition 1

Definition 2

Theorem 1

Proof of Theorem 1

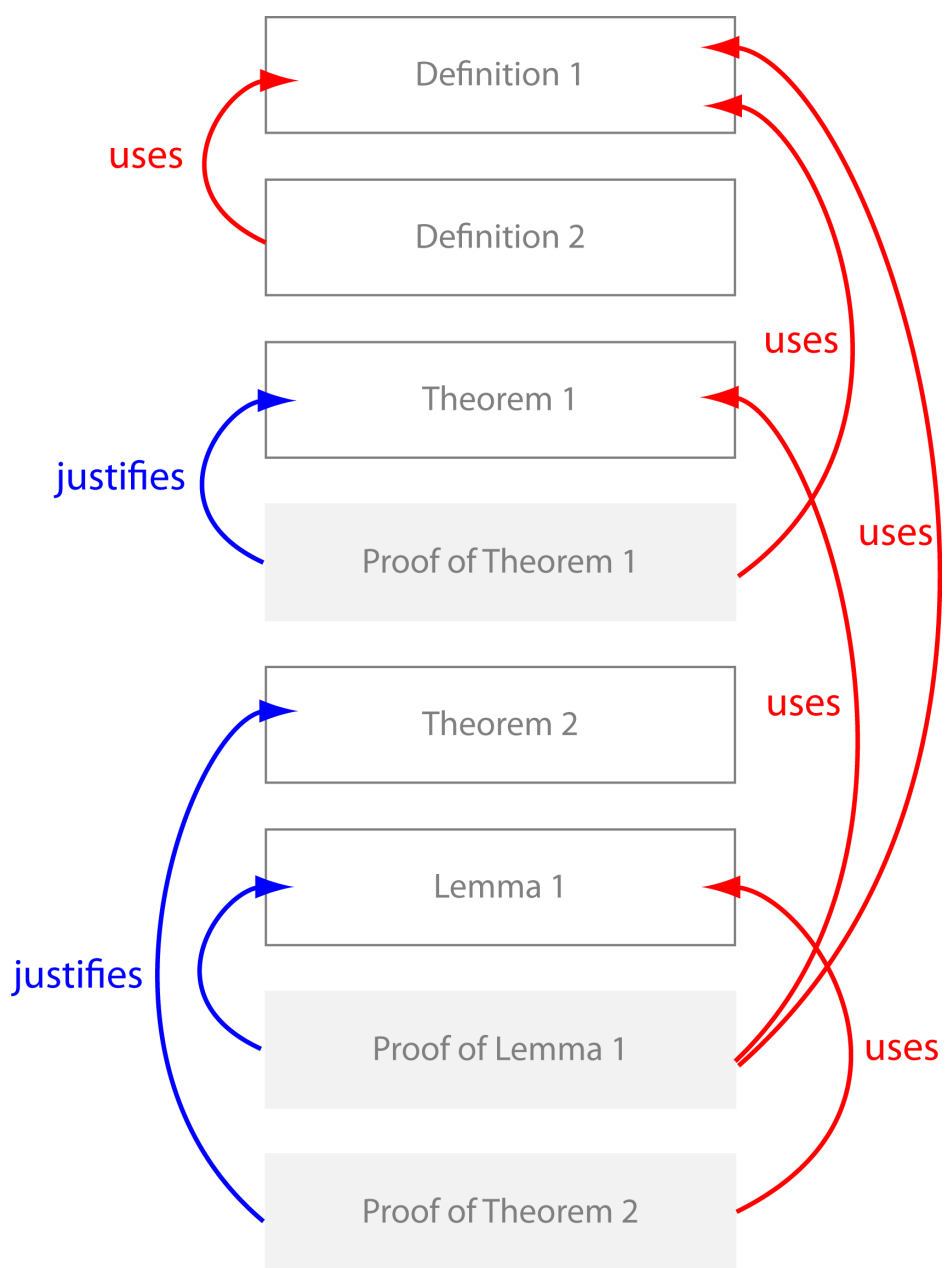
Theorem 2

Lemma 1

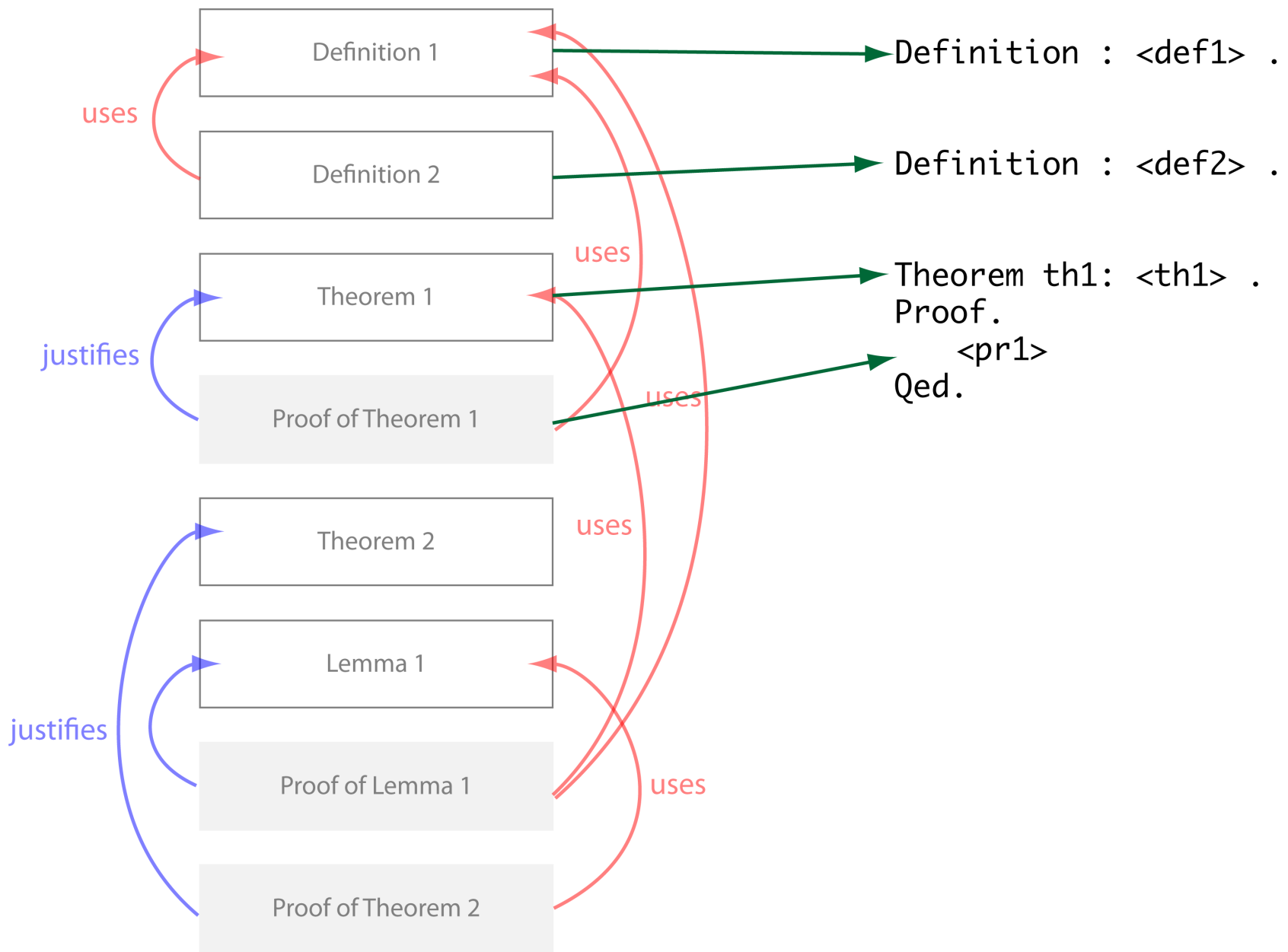
Proof of Lemma 1

Proof of Theorem 2

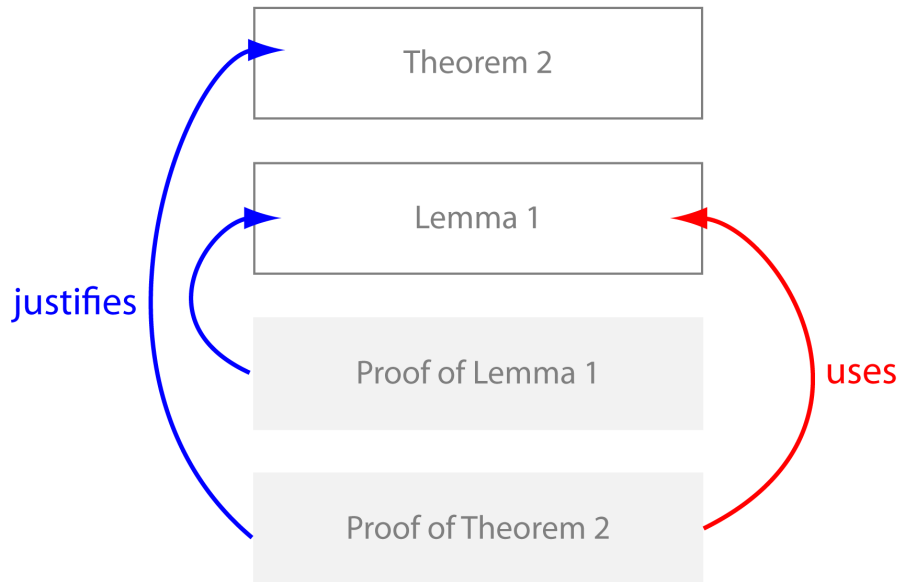
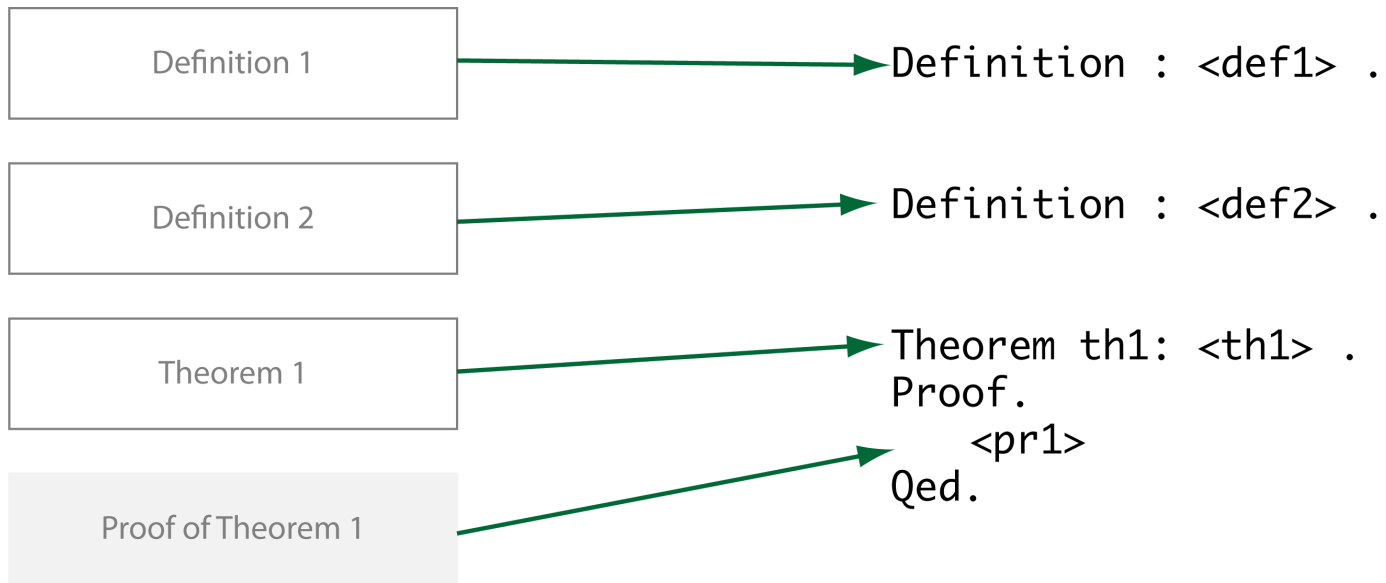
The automatic generation of a proof skeleton



The DG for the example



Straight-forward translation of the first part

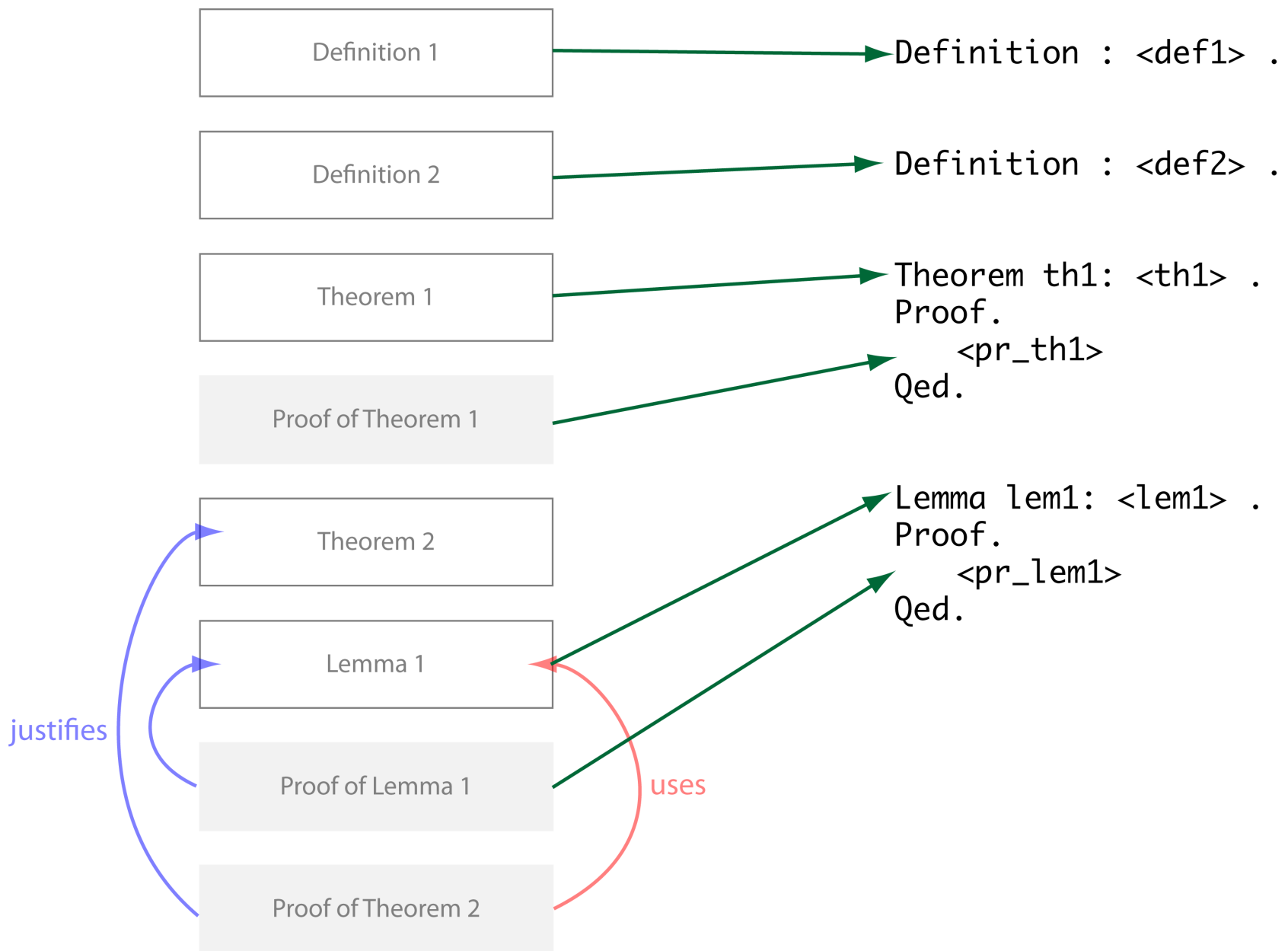


Problem

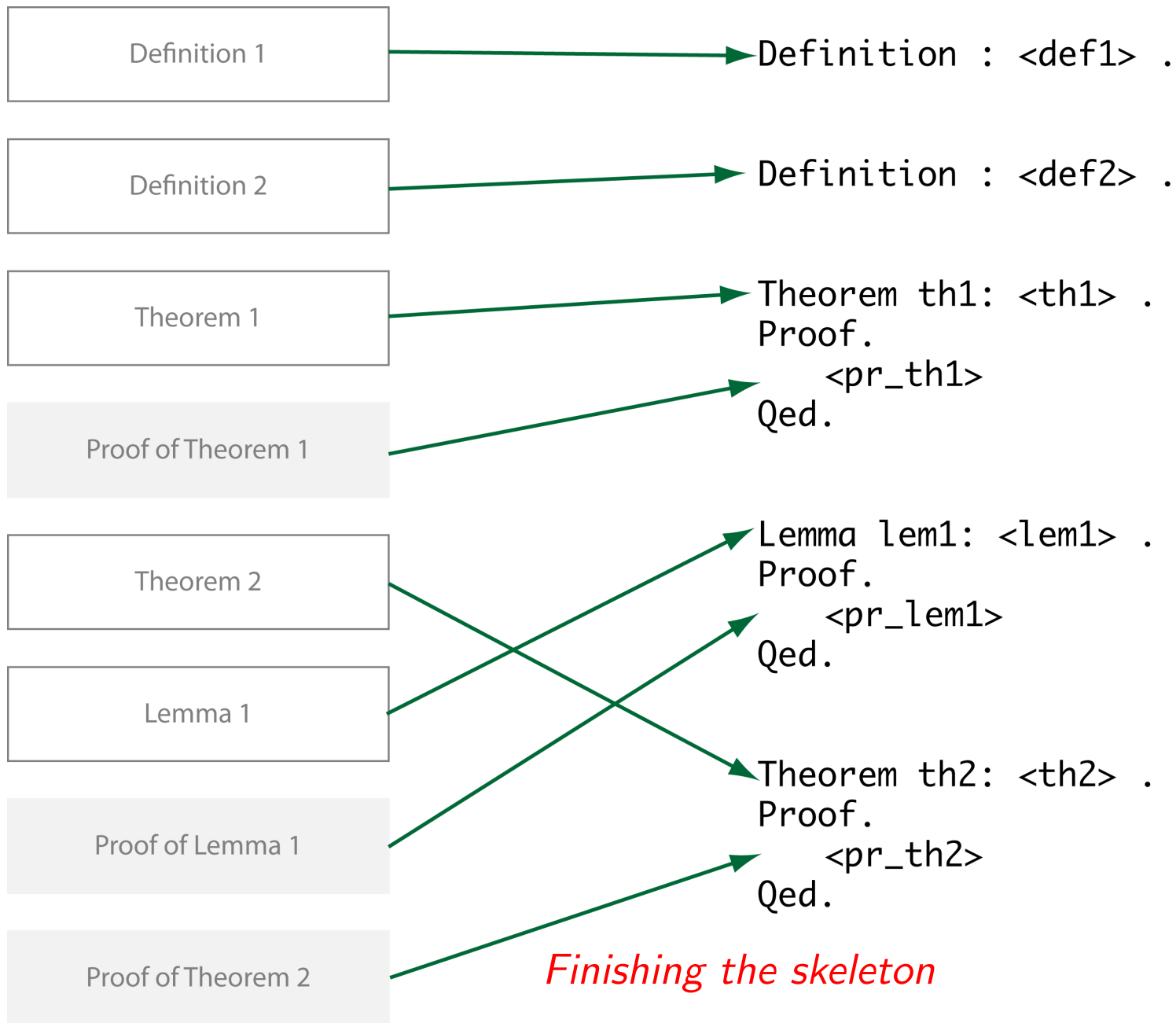
**No nested theorems/lemmas
in Coq e.g.**

→ **Re-Ordering !**

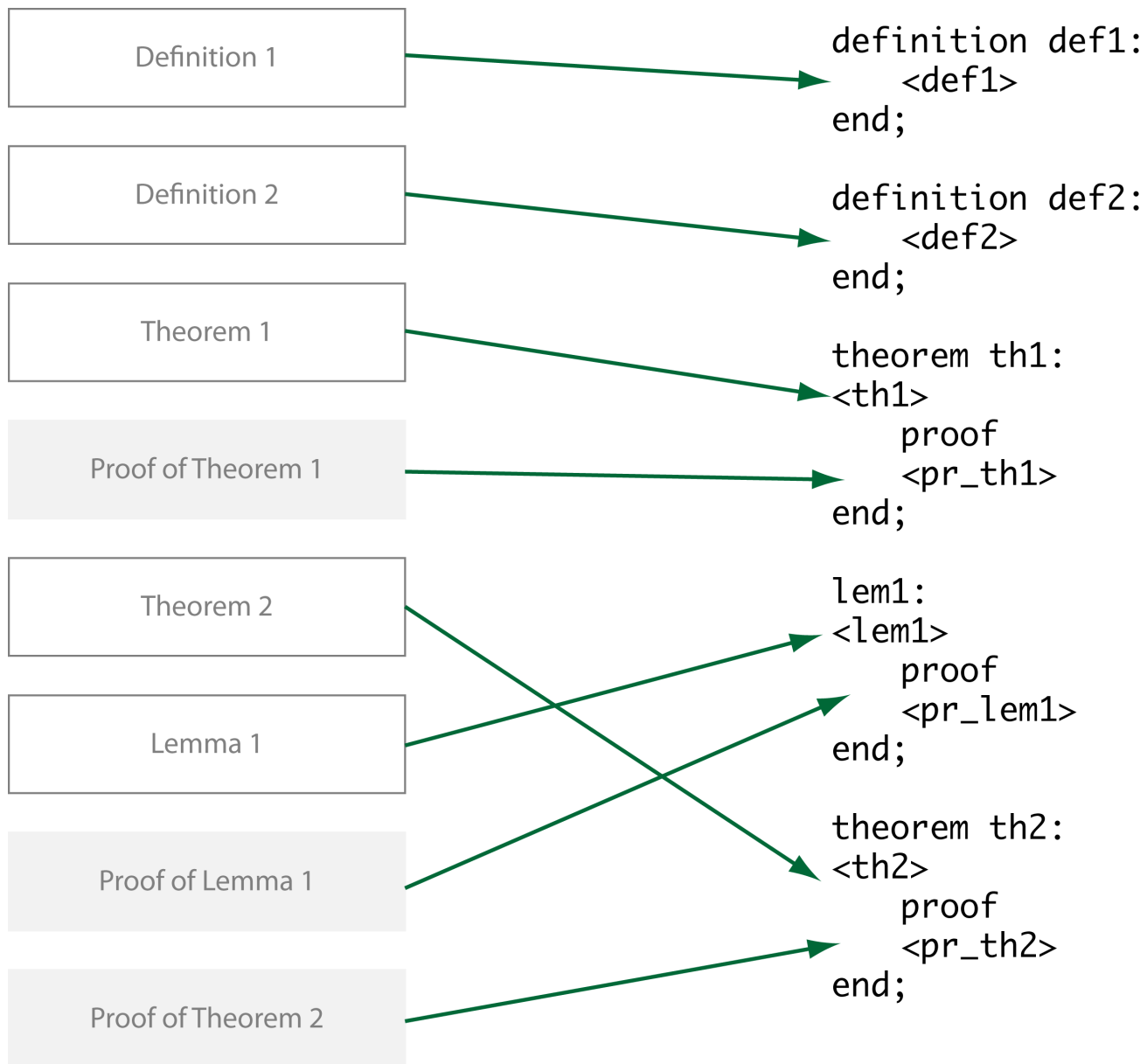
Problem: nested theorems



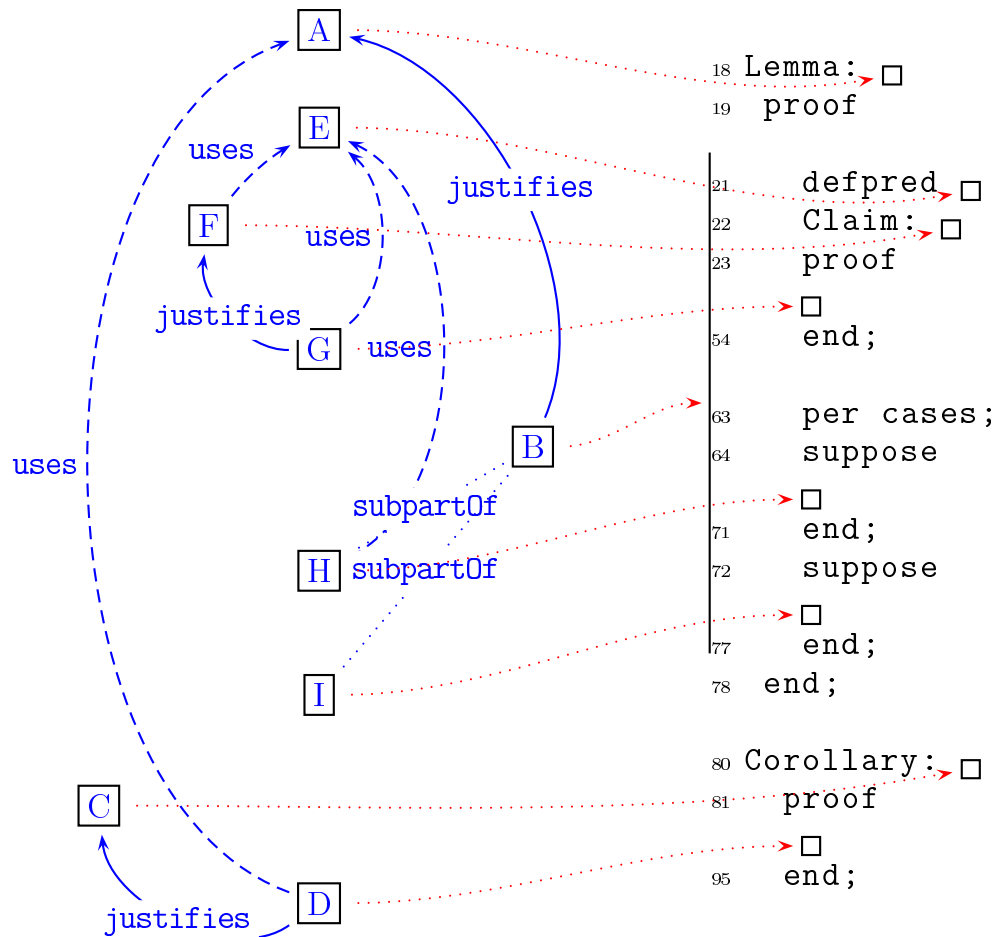
Solution: Re-ordering



Skeleton for Mizar



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

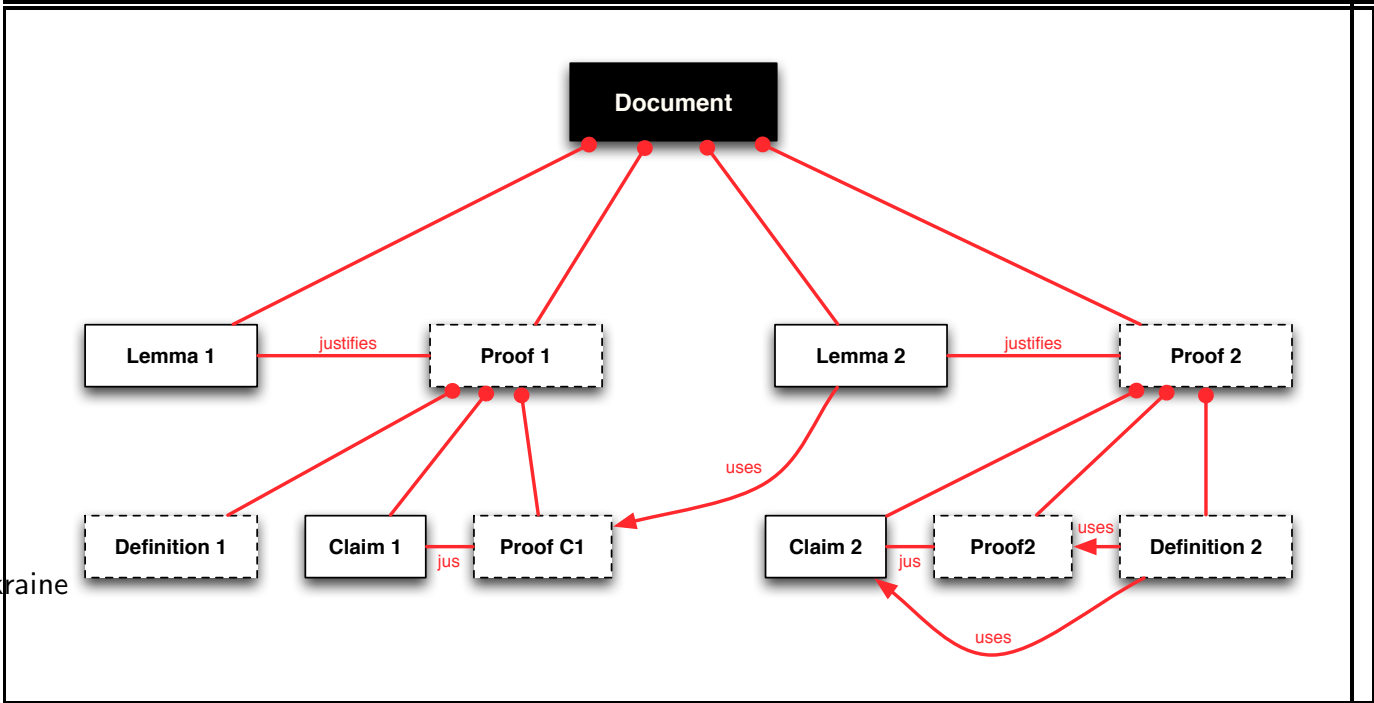
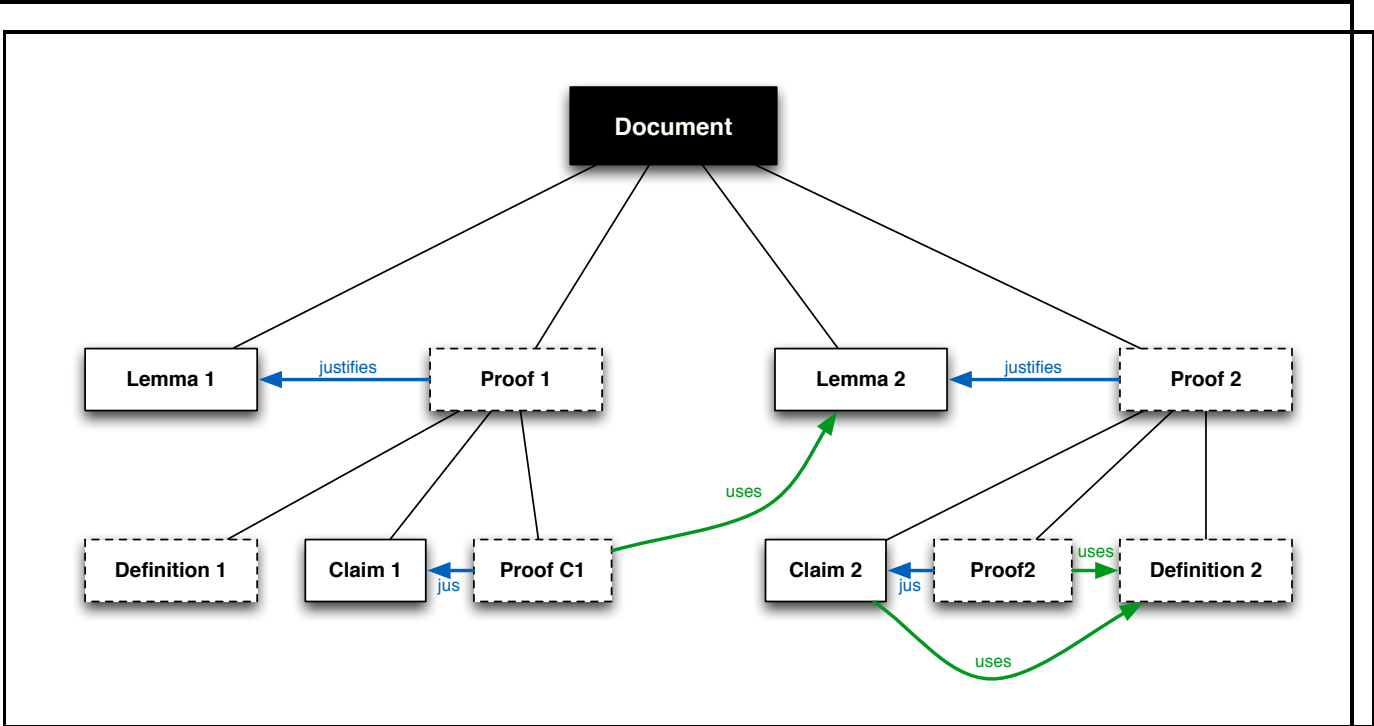


The generic algorithm for generating the proof skeleton (SGa, Zengler's thesis)

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:



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

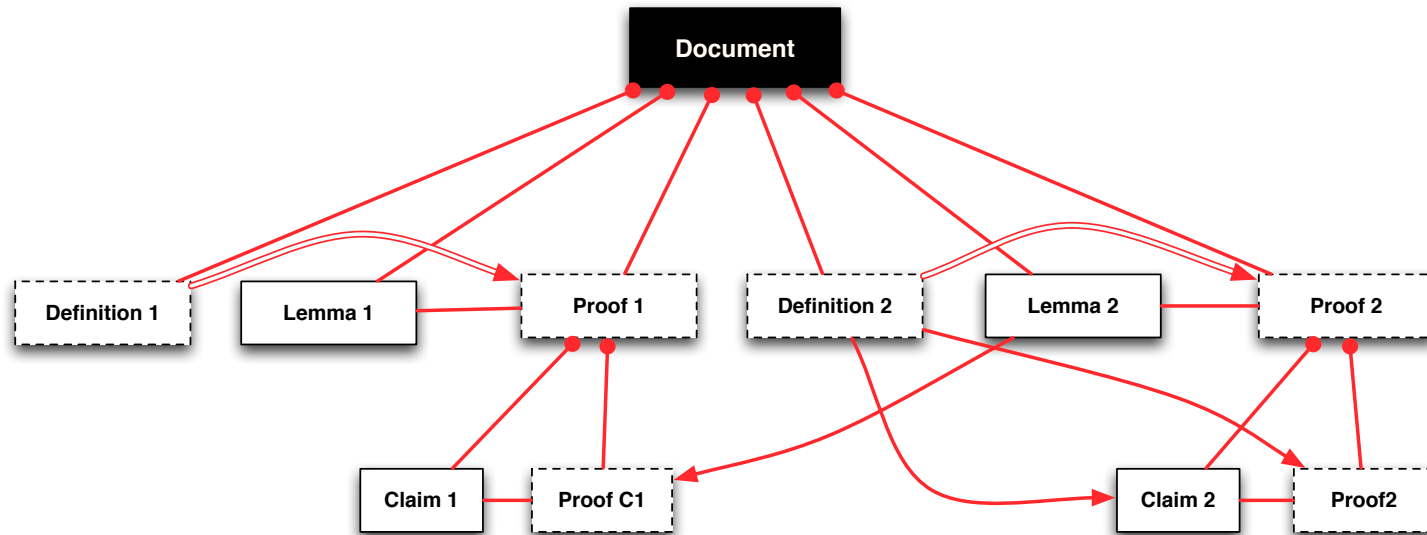


Figure 6: A flattened graph of the GoTO of figure 5 without nested definitions

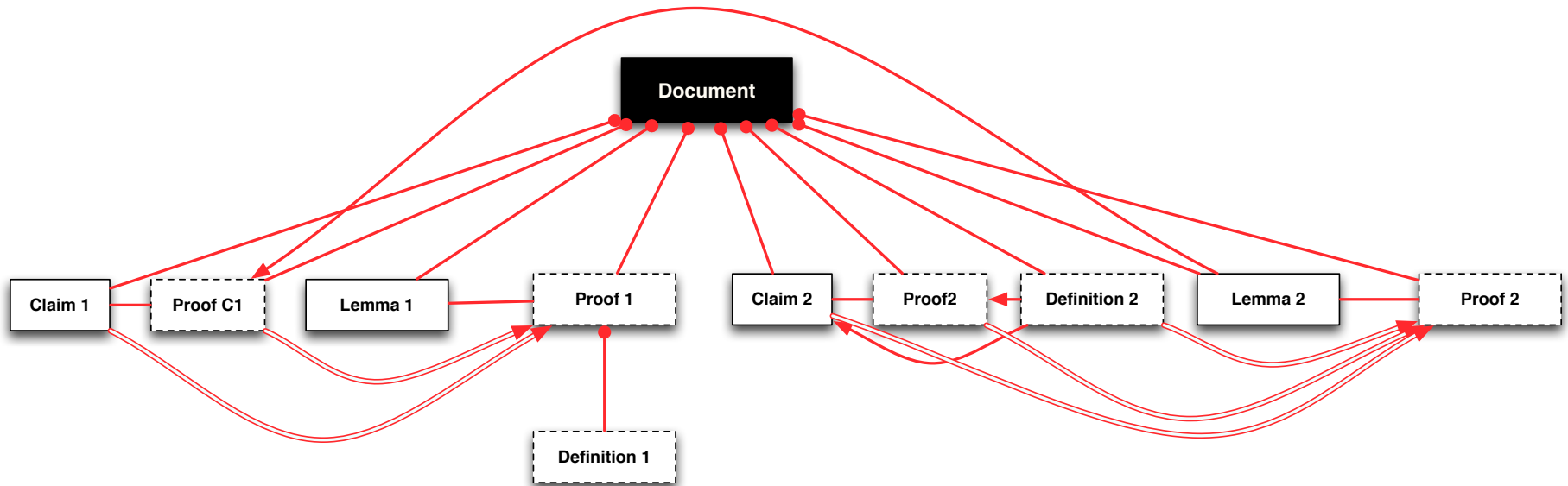


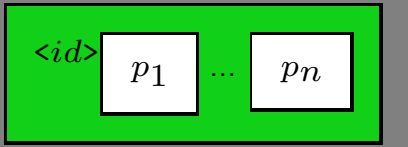
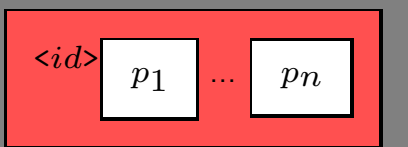
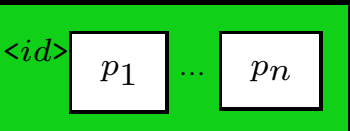
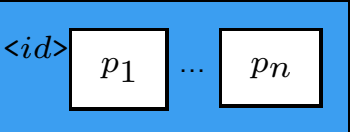
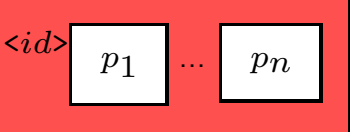

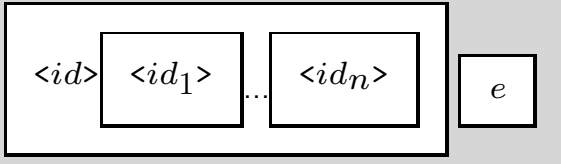
Figure 7: A flattened graph of the GoTO of figure 5 without nested claims

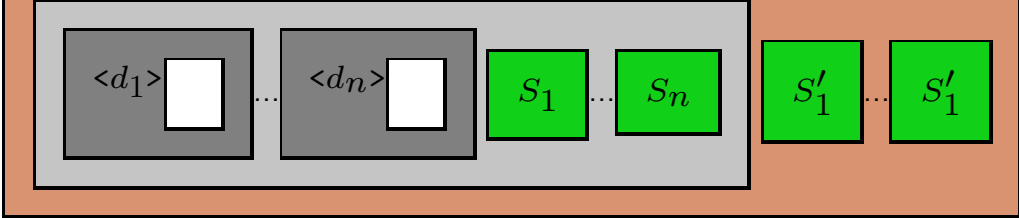
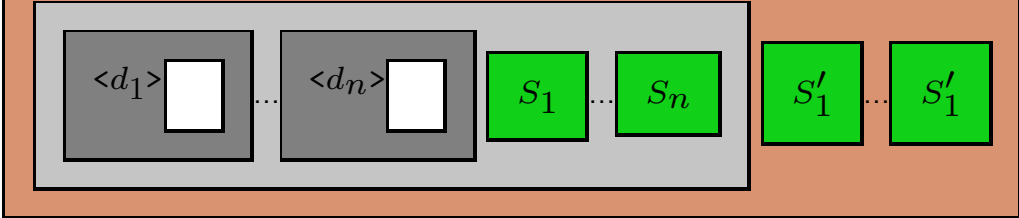
The Mizar and Coq rules for the dictionary

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

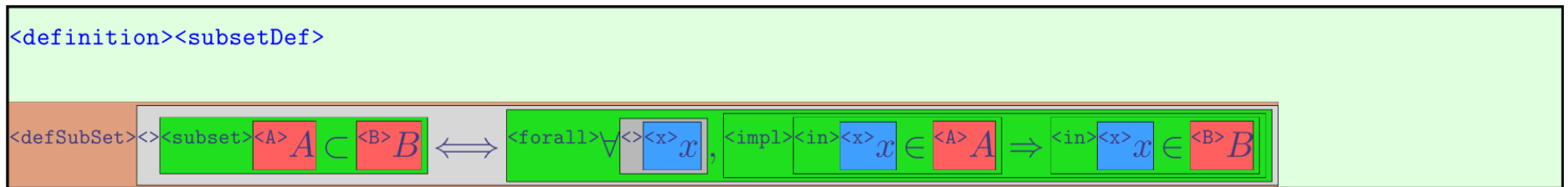
Rich skeletons for Coq

Rule N^0	Annotation ann	Coq translation $\mathcal{S}_{Coq}(ann)$
coq1)	<#>	Set
coq2)	<#>	Prop
coq3)	<div style="display: inline-block; border: 1px solid blue; background-color: lightblue; padding: 2px 5px;"><id></div> <div style="display: inline-block; border: 1px solid orange; background-color: orange; padding: 2px 5px; margin-left: 10px;"><N></div>	id : N
coq4)	<div style="display: inline-block; border: 1px solid blue; background-color: lightblue; padding: 2px 5px;"><id></div> <div style="display: inline-block; border: 1px solid red; background-color: red; padding: 2px 5px; margin-left: 10px;"><S></div>	id : S
coq5)	<div style="border: 1px solid blue; background-color: lightblue; padding: 2px 5px; display: inline-block;"><id></div>	id
coq6)	<div style="border: 1px solid blue; background-color: lightblue; padding: 5px; display: inline-block;"> <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;"><id></div> <div style="display: inline-block; margin: 0 5px;"> <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;">p₁</div> ... <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;">p_n</div> </div> <div style="display: inline-block; border: 1px solid orange; background-color: orange; padding: 2px 5px; margin-left: 10px;"><N></div> </div>	id : $\mathcal{S}_{Coq} \left(\boxed{p_1} \right) \rightarrow \dots \rightarrow \mathcal{S}_{Coq} \left(\boxed{p_n} \right) \rightarrow N$
coq7)	<div style="border: 1px solid blue; background-color: lightblue; padding: 5px; display: inline-block;"> <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;"><id></div> <div style="display: inline-block; margin: 0 5px;"> <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;">p₁</div> ... <div style="border: 1px solid black; background-color: white; padding: 2px 5px; display: inline-block;">p_n</div> </div> <div style="display: inline-block; border: 1px solid red; background-color: red; padding: 2px 5px; margin-left: 10px;"><S></div> </div>	id : $\mathcal{S}_{Coq} \left(\boxed{p_1} \right) \rightarrow \dots \rightarrow \mathcal{S}_{Coq} \left(\boxed{p_n} \right) \rightarrow S$

coq8)		$id : \mathcal{S}_{Coq} \left(\boxed{p_1} \right) \rightarrow \dots \rightarrow \mathcal{S}_{Coq} \left(\boxed{p_n} \right) \rightarrow \text{Prop}$
coq9)		$id : \mathcal{S}_{Coq} \left(\boxed{p_1} \right) \rightarrow \dots \rightarrow \mathcal{S}_{Coq} \left(\boxed{p_n} \right) \rightarrow \text{Set}$
coq10)		$(id \mathcal{S}_{Coq} \left(\boxed{p_1} \right) \dots \mathcal{S}_{Coq} \left(\boxed{p_n} \right))$
coq11)		$(id \mathcal{S}_{Coq} \left(\boxed{p_1} \right) \dots \mathcal{S}_{Coq} \left(\boxed{p_n} \right))$
coq12)		$(id \mathcal{S}_{Coq} \left(\boxed{p_1} \right) \dots \mathcal{S}_{Coq} \left(\boxed{p_n} \right))$
coq13)		id
coq14)		$id \ id_1 \ \dots \ id_n := \mathcal{S}_{Coq} \left(\boxed{e} \right)$

coq15)	 <p>for a surrounding unproved <i>DRa</i> annotation</p>	$\text{forall } \mathcal{S}_{Coq} \left(\langle d_1 \rangle \right) \dots \mathcal{S}_{Coq} \left(\langle d_n \rangle \right) \dots \wedge \mathcal{S}_{Coq} \left(S_n \right) \rightarrow \mathcal{S}_{Coq} \left(S'_1 \right)$
coq16)	 <p>for a surrounding proved <i>DRa</i> annotation</p>	$\mathcal{S}_{Coq} \left(\langle d_1 \rangle \right) \dots \mathcal{S}_{Coq} \left(\langle d_n \rangle \right) \wedge \mathcal{S}_{Coq} \left(S_n \right) \rightarrow \mathcal{S}_{Coq} \left(S'_1 \right) /$

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))

`<theorem><th117>`

`<Th117>`

Theorem 1.

`<x>` `<y>` `<z>` *If*

`<leq><x>x ≤ <y>y, <leq><y>y ≤ <z>z,`

then

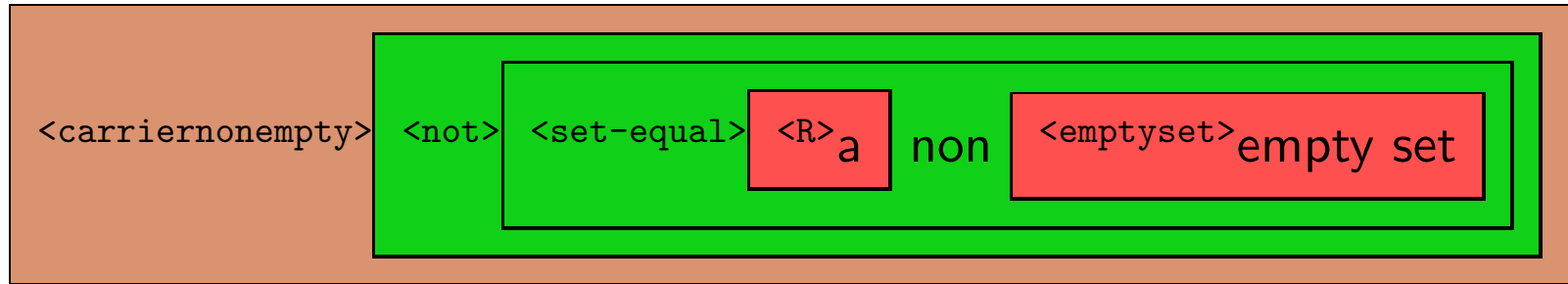
`<leq><x>x ≤ <z>z.`

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

The automatic translation is:

Theorem th117 $x y z : (leq x y \wedge leq y z) \rightarrow leq x z .$

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

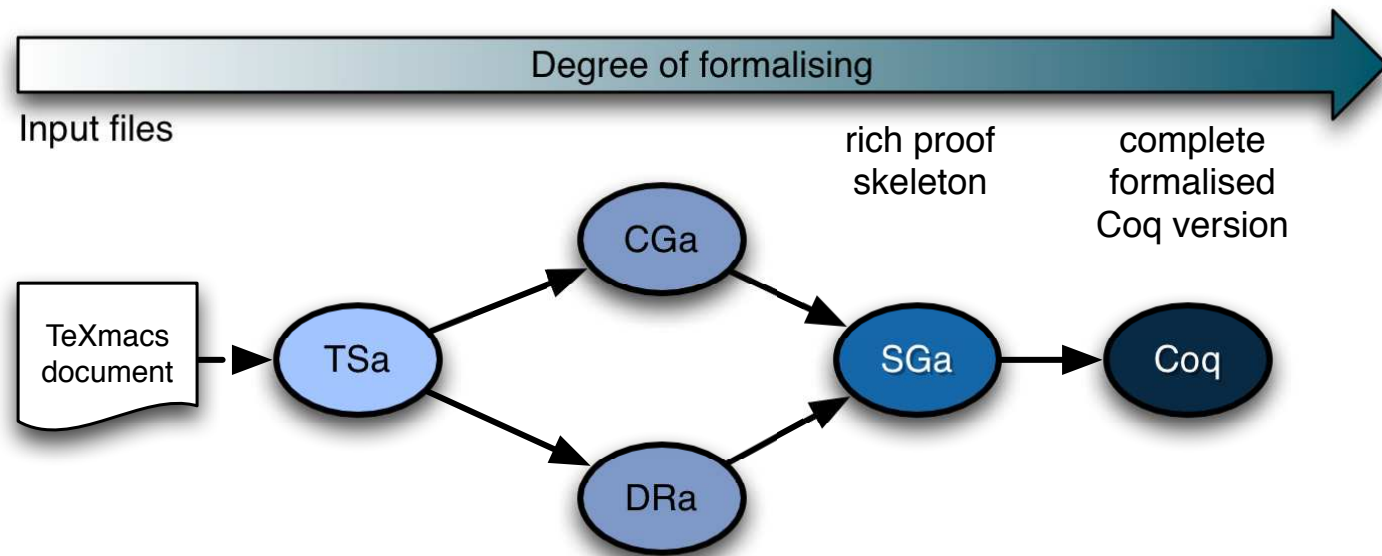


Figure 9: The path for processing the Landau chapter

<Th16>

Theorem 1.6. COMMUTATIVE LAW OF ADDITION 

$$\text{<eq>plus<x>x + <y>y = plus<y>y + <x>x.}$$

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

<Th116>

Theorem 1.16.

<>

<><x> <><y> <><z> *If*

<or><and><leq><x> $x \leq y$, <lt><y> $y < z$, OR <and><lt><x> $x < y$, <leq><y> $y \leq z$,

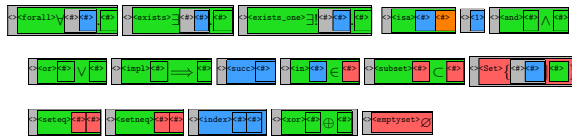
then

<lt><x> $x < z$.

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:

$\exists A$ set (i.e. totality) of objects called natural_numbers 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

$$x \neq y$$

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

Accordingly, the following are true on purely logical grounds:

$\forall x, y$ for every x, y

If $x = y$ then $y = x$

If $x = y$ and $y = z$ then $x = z$

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”.

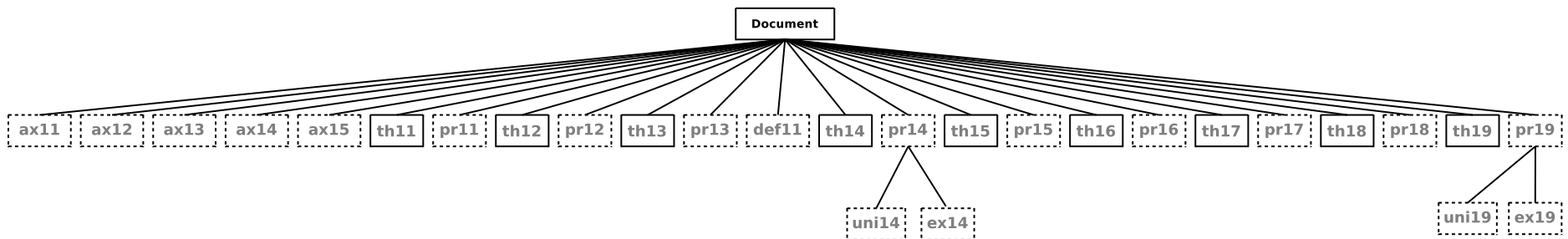


Figure 12: The DRa tree of sections 1 and 2 of chapter 1 of Landau’s book

- 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.

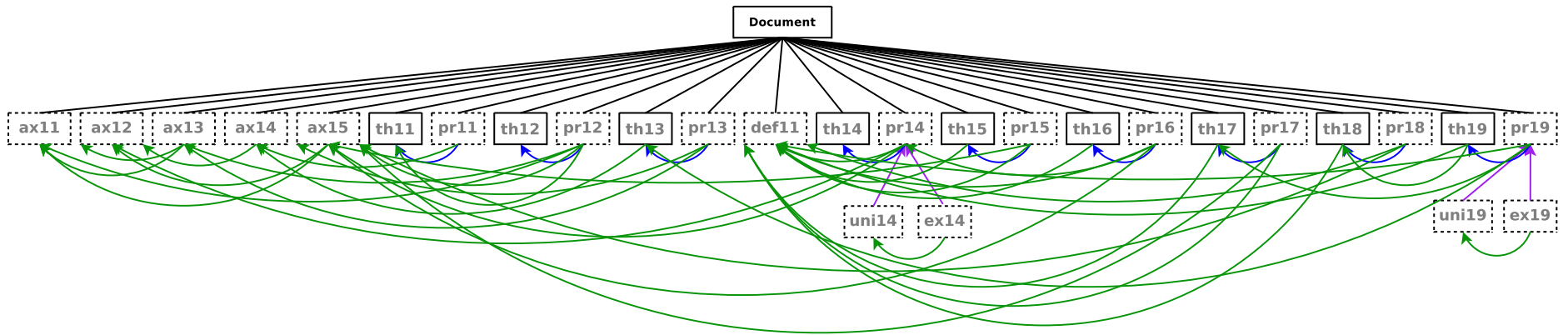
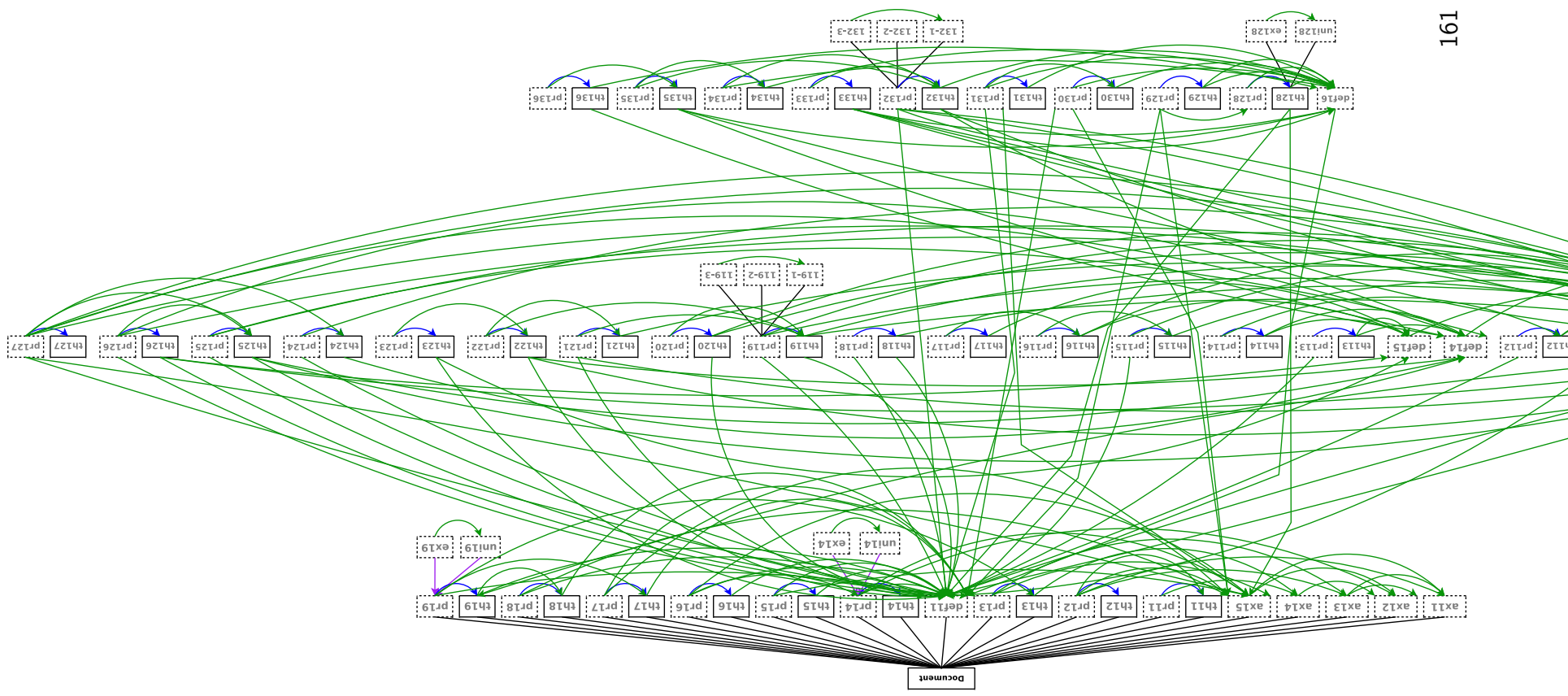
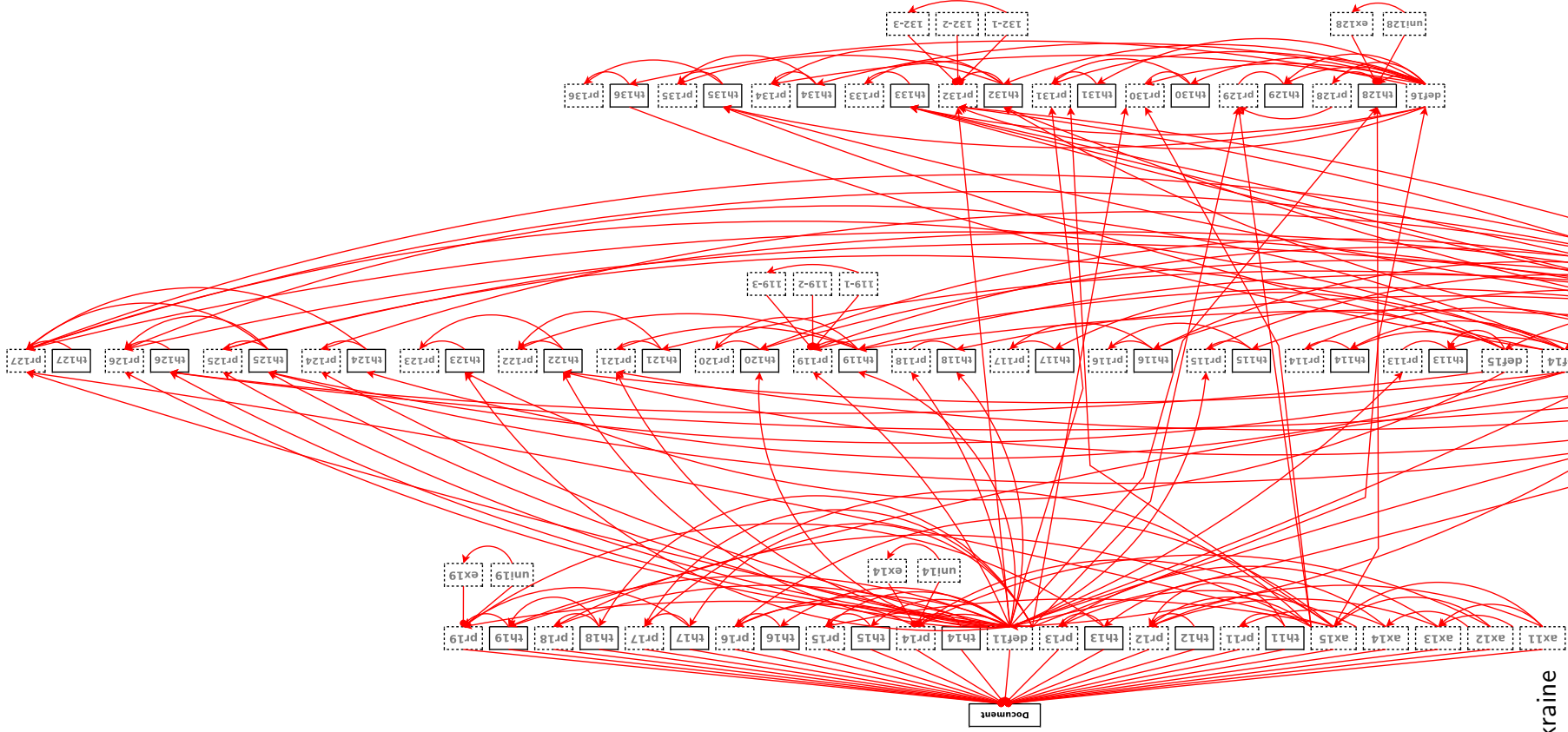


Figure 13: The DG of sections 1 and 2 of chapter 1 of Landau's book



The GoTO of section 1 - 4



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

----- (2)
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

----- (3)
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

- Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pages 233–246, 1995.
- 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.
- 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).
- Roel Bloo, Fairouz Kamareddine, and Rob Nederpelt. The Barendregt cube with definitions and generalised reduction. *Inform. & Comput.*, 126(2):123–143, May 1996.
- 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 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.
- Philippe de Groote. The conservation theorem revisited. In *Proc. Int'l Conf. Typed Lambda Calculi and Applications*, pages 163–178. Springer, 1993.
- 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, begriffsschriftlich 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, begriffsschriftlich 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 Begriffsschrift 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 fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.
- 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, R. Bloo, and R. Nederpelt. On Π -conversion in the λ -cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1–3):27–45, 1999a.
- 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. Revisiting the notion of function. *Logic and Algebraic programming*, 54:65–107, 2003.
- 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. Postponement, conservation and preservation of strong normalisation for generalised reduction. *J. Logic Comput.*, 10(5):721–738, 2000.
- Fairouz Kamareddine and Rob Nederpelt. Refining reduction in the λ -calculus. *J. Funct. Programming*, 5(4): 637–651, October 1995.
- Fairouz Kamareddine and Rob Nederpelt. A useful λ -notation. *Theoret. Comput. Sci.*, 155(1):85–109, 1996.
- Fairouz Kamareddine, Alejandro Ríos, and J. B. Wells. Calculi of generalised β -reduction and explicit substitutions: The type free and simply typed versions. *J. Funct. Logic Programming*, 1998(5), June 1998.
- 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, 1999b.
- A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ -calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pages 196–207, 1994. ISBN 0-89791-643-3.
- A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of β -strong normalization in typed λ -calculi. In *Proc. 10th Ann. IEEE Symp. Logic in Comput. Sci.*, pages 311–321, 1995. ISBN 0-8186-7050-9. URL <http://www.church-project.org/reports/electronic/Kfo+Wel:LICS-1995.pdf.gz>.

Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, March 1994.

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 Mellon University, Pittsburgh, USA, 1988.

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

Rob Nederpelt. *Strong Normalization in a Typed Lambda Calculus With Lambda Structured Types*. PhD thesis, Eindhoven, 1973.

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*.

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

Laurent Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.

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.

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.

A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910¹, 1927². All references are to the first volume, unless otherwise stated.