From the Foundation of Mathematics to the Generalisation of Functions and the Birth of Types and Computation

A Talk dedicated to Professor Tetsuo Ida

Fairouz Kamareddine
Heriot-Watt University, Edinburgh, Scotland

June 2012

Prof Ida will always be at the top of the tree supporting and influencing us

Summary

- *General definition of function 1879* [Frege, 1879] is key to Frege's *formalisation of logic*.
- *Self-application of functions* is the heart of *Russell's paradox* [Russell, 1902].
- To *avoid paradox* Russell controled function application via *type theory*.
- Church's $\lambda\!\!\rightarrow$ [Church, 1940] is a *simply typed calculus of functions*.
- The *hierarchy of types* in $\lambda\!\!\rightarrow$ is *unsatisfactory*.
- The *notion of function adopted in the λ-calculus* is *unsatisfactory* (cf. [Kamareddine et al., 2003a]).
- Hence, birth of *different systems of functions and types*, each with *different functional power*.

- We discuss the evolution of functions and types and their use in logic and computation.

- 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.

- Furthermore, 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 formal hierarchy of types.

- We discuss the lessons learned from formalising mathematics in logic (à la Principia) and in proof checkers (à la Automath, or any modern proof checker).

- We also discuss functions and types à la de Bruijn.

The development of new research ideas have always been possible by t
encouragement, teaching and co-operation of Prof. Ida

His supportive ear and kind personality are invaluable

Here he is again giving all the encouragement possible

Prehistory of Types (Euclid)

- 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)

- 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)

- 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 (formal systems in 19th century)

In the 19th century, the need for a more *precise* style in mathematics arose, because controversial results had appeared in analysis.

- 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.

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

Prehistory of Types (Begriffsschrift's functions)

The introduction of a *very general definition of function* was the key to the formalisation of logic. Frege defined what we will call 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)

Prehistory of Types (Begriffsschrift's functions)

- Frege put *no restrictions* on what could play the role of *an argument*.
- An argument could be a *number*, a *proposition*, or a *function*.
- The *result of applying* a function to an argument need not be a number.

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)

Frege did indeed avoid the paradox in his Begriffsschrift.

Prehistory of Types (Grundgesetze's functions)

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, 1892, 1903] he described the elementary parts of arithmetics 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 *apply a function to itself*, but to its course-of-values (graph).
- Frege denoted the course-of-values of a function $\Phi(x)$ by $\grave{\varepsilon}\Phi(\varepsilon)$. The definition of equal courses-of-values could therefore be expressed as

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

Prehistory of Types (Grundgesetze's functions)

- 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*.
- All essential information of a function is contained in its graph.
- So intuitively, 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* from his system 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* (*Begriffsschrift does not suffer from a paradox*).
- Russell gave his well-known argument, defining the propositional function $f(x)$ *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.
- *Only six days later*, Frege answered Russell that *Russell's derivation of the paradox was incorrect* [Frege, 1902] but explained that *Russell's argument could be amended to a paradox* in the system of his Grundgesetze, using the *course-of-values* of functions.

When tradition fails, one needs to take a deep breath and reflect deeply

We know we are on the right path when we are in harmony with nature

The birth of type theory

- To *avoid paradox* Russell controled 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, 1910^1, 1927^2].
- *Simple theory of types* (STT): [Ramsey, 1926], [Hilbert and Ackermann, 1928].
- Church's *simply typed λ-calculus* $\lambda\!\!\rightarrow$ [Church, 1940] = λ-calculus + STT.
- The hierarchies of types (and orders) in RTT and STT are *unsatisfactory*.
- Numbers, booleans, the identity function have to be defined at every level.
- We can represent (and type) terms like $\lambda x : o.x$ and $\lambda x : \iota.x$ but not $\lambda x : \alpha.x$, where α can be instantiated to any type.
- This led to new (modern) type theories that allow more general notions of functions (e.g, *polymorphic*).

The evolution of functions with Frege, Russell and Church

- Historically, functions have long been treated as a kind of meta-objects.
- Function *values* were the important part, not abstract functions.
- In the *low level/operational approach* there are only function values.
- The sine-function, is always expressed with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- In many mathematics courses, one calls $f(x)$—and not f—the function.
- Frege, Russell and Church wrote $x \mapsto x+3$ resp. as $x+3$, $\hat{x}+3$ and $\lambda x.x+3$.
- Principia's *functions are based on Frege's Abstraction Principles* but can be first-class citizens. Frege used courses-of-values to speak about functions.
- Church made every function a first-class citizen. This is rigid and does not represent the development of logic in 20th century.

Functionalisation and Instantiation

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

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

Functionalisation and Instantiation for Frege, Russell and Church

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

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

- The λ-calculus focuses on function construction and does not employ abstraction from a subexpression. The abstraction from $2 \times 3 + 2$ to $x \times 3 + x$ is not included in the syntax.

λ-calculus does not fully represent functionalisation

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

- cannot abstract only half way: $x + 3$ is not a function, $\lambda x.x + 3$ is.
- This is why notions like parameterised functions (functions with parameters) have been introduced. To allow $x + 3$ to be a function in the lambda calculus.
- cannot instantiate $x + 3$ with argument: $(x + 3)$ cannot be instantiated to 2+3.
- This is why notions like explicit substitutions have been introduced. To make things like $(x + 3)[x := 2]$ explicit.

During our academic life, it is not the obstacles that define us

Rather, it is our long-term perserverance and how we overcome these obstacles

De Bruijn Indices [de Bruijn, 1972]

- Classical λ-calculus: $A ::= x \mid (\lambda x.B) \mid (BC)$
  $(\lambda x.A)B \rightarrow_\beta A[x := B]$
- $(\lambda x.\lambda y.xy)y \rightarrow_\beta (\lambda y.xy)[x := y] \neq \lambda y.yy$
- $(\lambda x.\lambda y.xy)y \rightarrow_\beta (\lambda y.xy)[x := y] =_\alpha (\lambda z.xz)[x := y] = \lambda z.yz$
- $\lambda x.x$ and $\lambda y.y$ are the same function. Write this function as $\lambda 1$.
- Assume a free variable list (say $x, y, z, \ldots$).
- $(\lambda\lambda 2\ 1)2 \rightarrow_\beta (\lambda 2\ 1)[1 := 2] = \lambda(2[2 := 3])(1[2 := 3]) = \lambda 3\ 1$

Classical λ-calculus with de Bruijn indices

- Let $i, n \geq 1$ and $k \geq 0$
- $A \ ::= \ n \,|\, (\lambda B) \,|\, (BC)$
  $(\lambda A)B \rightarrow_\beta A\{\!\{1 \leftarrow B\}\!\}$
- $$\begin{array}{l} U_k^i(AB) = U_k^i(A)\, U_k^i(B) \\ U_k^i(\lambda A) = \lambda(U_{k+1}^i(A)) \end{array} \qquad U_k^i(\mathtt{n}) = \begin{cases} \mathtt{n} + \mathtt{i} - \mathtt{1} & \text{if} \quad n > k \\ \mathtt{n} & \text{if} \quad n \leq k\,. \end{cases}$$
- $$\begin{array}{lcl} (A_1 A_2)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & (A_1\{\!\{\mathtt{i} \leftarrow B\}\!\})\,(A_2\{\!\{\mathtt{i} \leftarrow B\}\!\}) \\ (\lambda A)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & \lambda(A\{\!\{\mathtt{i}+\mathtt{1} \leftarrow B\}\!\}) \end{array}$$
  $$\mathtt{n}\{\!\{\mathtt{i} \leftarrow B\}\!\} = \begin{cases} \mathtt{n} - \mathtt{1} & \text{if} \quad n > i \\ U_0^i(B) & \text{if} \quad n = i \\ \mathtt{n} & \text{if} \quad n < i\,. \end{cases}$$
- Numerous implementations of proof checkers and programming languages have been based on de Bruijn indices.

From classical λ-calculus with de Bruijn indices to substitution calculus λs [Kamareddine and Ríos, 1995]

- Write $A\{\!\{n \leftarrow B\}\!\}$ as $A\sigma^n\, B$ and $U_k^i(A)$ as $\varphi_k^i A$.
- $A \;::=\; n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi_k^i B) \qquad where \quad i, n \geq 1\,, \quad k \geq 0\,.$

σ-*generation*	$(\lambda A)\, B$	$\longrightarrow$	$A\, \sigma^1\, B$
σ-λ-*transition*	$(\lambda A)\, \sigma^i B$	$\longrightarrow$	$\lambda(A\, \sigma^{i+1}\, B)$
σ-*app-transition*	$(A_1\, A_2)\, \sigma^i B$	$\longrightarrow$	$(A_1\, \sigma^i B)\, (A_2\, \sigma^i B)$
σ-*destruction*	$\mathtt{n}\, \sigma^i B$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if} \quad n > i \\ \varphi_0^i\, B & \text{if} \quad n = i \\ \mathtt{n} & \text{if} \quad n < i \end{cases}$
φ-λ-*transition*	$\varphi_k^i(\lambda A)$	$\longrightarrow$	$\lambda(\varphi_{k+1}^i\, A)$
φ-*app-transition*	$\varphi_k^i(A_1\, A_2)$	$\longrightarrow$	$(\varphi_k^i\, A_1)\, (\varphi_k^i\, A_2)$
φ-*destruction*	$\varphi_k^i\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \text{if} \quad n > k \\ \mathtt{n} & \text{if} \quad n \leq k \end{cases}$

1. The s-calculus (i.e., λs minus σ-generation) is strongly normalising,

2. The λs-calculus is confluent and simulates (in small steps) β-reduction

3. The λs-calculus preserves strong normalisation PSN.

4. The λs-calculus has a confluent extension with open terms λse.

- The λs-calculus is the only calculus of substitutions which satisfies all the above properties 1., 2., 3. and 4.

$\lambda\upsilon$ [Benaissa et al., 1996]

Terms: $\Lambda\upsilon^t ::= \mathsf{IN} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s]$
Substitutions: $\Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b/]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(FVar)	$\mathtt{1}\,[a/]$	$\longrightarrow$	a
(RVar)	$\mathtt{n}+\mathtt{1}\,[a/]$	$\longrightarrow$	$\mathtt{n}$
(FVarLift)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(RVarLift)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}\,[s]\,[\uparrow]$
(VarShift)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$

$\lambda\upsilon$ satisfies 1., 2., and 3., but does not have a confluent extension on open terms.

$\lambda\sigma_{\Uparrow}$

Terms: $\Lambda\sigma^t_{\Uparrow} ::= \mathsf{IN} \mid \Lambda\sigma^t_{\Uparrow}\Lambda\sigma^t_{\Uparrow} \mid \lambda\Lambda\sigma^t_{\Uparrow} \mid \Lambda\sigma^t_{\Uparrow}[\Lambda\sigma^s_{\Uparrow}]$

Substitutions: $\Lambda\sigma^s_{\Uparrow} ::= id \mid \uparrow \mid \Uparrow(\Lambda\sigma^s_{\Uparrow}) \mid \Lambda\sigma^t_{\Uparrow} \cdot \Lambda\sigma^s_{\Uparrow} \mid \Lambda\sigma^s_{\Uparrow} \circ \Lambda\sigma^s_{\Uparrow}.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b \cdot id]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(Varshift1)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n+1}$
(Varshift2)	$\mathtt{n}\,[\uparrow \circ s]$	$\longrightarrow$	$\mathtt{n+1}\,[s]$
(FVarCons)	$\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	a
(RVarCons)	$\mathtt{n+1}\,[a \cdot s]$	$\longrightarrow$	$\mathtt{n}\,[s]$
(FVarLift1)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(FVarLift2)	$\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{1}\,[t]$
(RVarLift1)	$\mathtt{n+1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}[s \circ \uparrow]$
(RVarLift2)	$\mathtt{n+1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{n}[s \circ (\uparrow \circ t)]$

$\lambda\sigma_{\Uparrow}$ rules continued

(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(Ass)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\longrightarrow$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(LiftId)	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a\,[id]$	$\longrightarrow$	a

$\lambda\sigma_{\Uparrow}$ satisfies 1., 2., and 4., but does not have PSN.

How is λse obtained from λs?

- $A ::= X \mid n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi^i_k B) \quad where \quad i, n \geq 1\,, \quad k \geq 0\,.$
- Extending the syntax with open terms without extending then rules loses the confluence (even local confluence):
  $((\lambda X)Y)\sigma^1 1 \rightarrow (X\sigma^1 Y)\sigma^1 1 \qquad ((\lambda X)Y)\sigma^1 1 \rightarrow ((\lambda X)\sigma^1 1)(Y\sigma^1 1)$
- $(X\sigma^1 Y)\sigma^1 1$ and $((\lambda X)\sigma^1 1)(Y\sigma^1 1)$ have no common reduct.
- But, $((\lambda X)\sigma^1 1)(Y\sigma^1 1) \twoheadrightarrow (X\sigma^2 1)\sigma^1(Y\sigma^1 1)$
- Simple: add de Bruijn's metasubstitution and distribution lemmas to the rules of λs:

σ-σ	$(A\sigma^i B)\,\sigma^j\,C$	$\longrightarrow$	$(A\,\sigma^{j+1}\,C)\,\sigma^i\,(B\,\sigma^{j-i+1}\,C)$	if	$i \leq j$
σ-φ *1*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^{i-1}_k\,A$	if	$k < j < k+i$
σ-φ *2*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^i_k(A\,\sigma^{j-i+1}\,B)$	if	$k+i \leq j$
φ-σ	$\varphi^i_k(A\,\sigma^j\,B)$	$\longrightarrow$	$(\varphi^i_{k+1}\,A)\,\sigma^j\,(\varphi^i_{k+1-j}\,B)$	if	$j \leq k+1$
φ-φ *1*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\,A)$	if	$l+j \leq k$
φ-φ *2*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^{j+i-1}_l\,A$	if	$l \leq k < l+j$

- These extra rules are the rewriting of the well-known meta-substitution ($\sigma - \sigma$) and distribution ($\varphi - \sigma$) lemmas (and the 4 extra lemmas needed to prove them).

Where did the extra rules come from?

In de Bruijn's classical λ-calculus we have the lemmas:

$(\sigma-\varphi\ 1)$ For $k<j<k+i$ we have: $U_k^{i-1}(A)=U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}$.

$(\varphi-\varphi\ 2)$ For $l\le k<l+j$ we have: $U_k^i(U_l^j(A))=U_l^{j+i-1}(A)$.

$(\sigma-\varphi\ 2)$ For $k+i\le j$ we have: $U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}=U_k^i(A\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow B\}\!\})$.

$(\sigma-\sigma)$ *[Meta-substitution lemma]* For $i\le j$ we have:
$A\{\!\{\mathtt{i}\leftarrow B\}\!\}\{\!\{\mathtt{j}\leftarrow C\}\!\}=A\{\!\{\mathtt{j}+\mathtt{1}\leftarrow C\}\!\}\{\!\{\mathtt{i}\leftarrow B\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow C\}\!\}\}\!\}$.

$(\varphi-\varphi\ 1)$ For $j\le k+1$ we have: $U_{k+p}^i(U_p^j(A))=U_p^j(U_{k+p+1-j}^i(A))$.

$(\varphi-\sigma)$ *[Distribution lemma]*
For $j\le k+1$ we have: $U_k^i(A\{\!\{\mathtt{j}\leftarrow B\}\!\})=U_{k+1}^i(A)\{\!\{\mathtt{j}\leftarrow U_{k+1-j}^i(B)\}\!\}$.

The proof of $(\sigma-\sigma)$ uses $(\sigma-\varphi\ 1)$ and $(\sigma-\varphi\ 2)$ both with $k=0$.
The proof of $(\sigma-\varphi\ 2)$ requires $(\varphi-\varphi\ 2)$ with $l=0$.
Finally, $(\varphi-\varphi\ 1)$ with $p=0$ is needed to prove $(\varphi-\sigma)$).

We know we are on the right path when we are in harmony with nature

In harmony with nature even in rainy and grey Scotland

Polymorphism: the typed λ-calculus after Church

- Instead of repeating the work, we take: $\alpha : *$ (α is an arbitrary type) and we define a polymorphic function F as follows:

  $$\lambda_{\alpha:*}.\lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x))$$

  We give F the type:

  $$\Pi_{\alpha:*}.(\alpha \to \alpha) \to (\alpha \to \alpha)$$

- This way, $F(\alpha) = \lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x)) : (\alpha \to \alpha) \to (\alpha \to \alpha)$

- We can instantiate α according to our need:
  - $F(\mathbb{N}) = \lambda_{f:\mathbb{N}\to\mathbb{N}}.\lambda_{x:\mathbb{N}}.f(f(x)) : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$
  - $F(\mathcal{B}) = \lambda_{f:\mathcal{B}\to\mathcal{B}}.\lambda_{x:\mathcal{B}}.f(f(x)) : (\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})$
  - $F(\mathcal{B} \to \mathcal{B}) = \lambda_{f:(\mathcal{B}\to\mathcal{B})\to(\mathcal{B}\to\mathcal{B})}.\lambda_{x:(\mathcal{B}\to\mathcal{B})}.f(f(x)) :$
    $((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})) \to ((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B}))$

- This way, types are like functions:
  - We can form them by abstraction
  - We can instantiate them

- But in the passage from simple to polymorphic types, we have forgotten to adapt the rule:

$$(\beta) \qquad (\lambda_{x:A}.b)C \to b[x := C]$$

  to a rule which resembles Π:

$$(\Pi) \qquad (\Pi_{x:A}.B)C \to B[x := C]$$

- Usually, if $b : B$, we take $(\lambda_{x:A}.b)C : B[x := C]$ instead of $(\lambda_{x:A}.b)C : (\Pi_{x:A}.B)C$

- De Bruijn's Automath shows that the rule Π is useful.

- It seems that the development of type theory is taking us more and more towards adopting a similar role for the binders λ et Π.
- Did we really need to separate λ et Π?
- I believe that the separation is artificial and that de Bruijn's intuition is again a winner.
- What are the properties of type theories with a single binder which represents both λ et Π?
-

Common features of modern types and functions

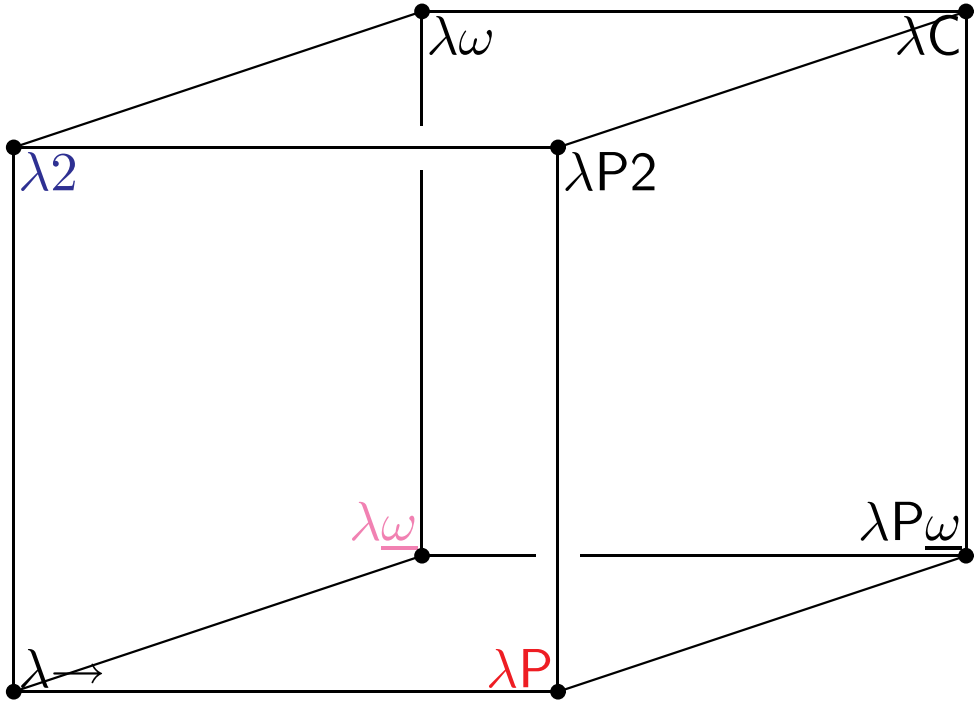
- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the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

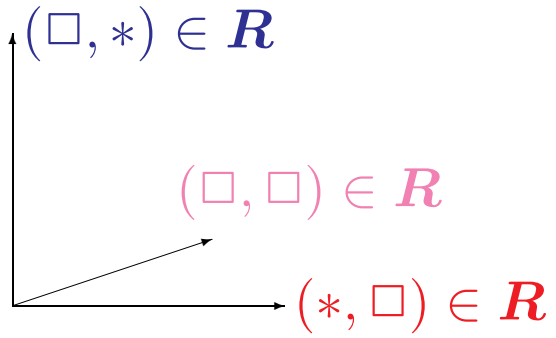
The Barendregt Cube

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

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

The Barendregt Cube





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

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

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

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

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

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

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

- ML's type system is none of those of the eight systems of the Cube.
  [Kamareddine et al., 2001] places the type system of ML on our refined Cube (between $\lambda 2$ and $\lambda\underline{\omega}$).

LF

- LF [Harper et al., 1987] is often described as λP of the Barendregt Cube. However, Use of Π-formation rule $(*, \Box)$ is restricted in LF [Geuvers, 1993].

- We only need a type $\Pi x{:}A.B : \Box$ when PAT is applied during construction of the type $\Pi\alpha{:}\mathtt{prop}.*$ of the operator Prf where for a proposition Σ, Prf(Σ) is the type of proofs of Σ.

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

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used. But, Prf is only used when applied to $\Sigma{:}\mathtt{prop}$. We never use Prf on its own.

- This use is in fact based on a parametric constant rather than on Π-formation.

- Hence, the practical use of LF would not be restricted if we present Prf in a parametric form, and use $(*, \Box)$ as a parameter instead of a Π-formation rule.

- [Kamareddine et al., 2001] precisely locate LF (between $\lambda{\to}$ and λP).

Parameters: What and Why

- We speak about *functions with parameters* when referring to functions with variable values in the *low-level* approach. The x in $f(x)$ is a parameter.
- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. first-order with parameters versus second-order without [Laan and Franssen, 2001].
- Desirable properties of the lower order theory (decidability, easiness of calculations, typability) can be maintained, without losing the flexibility of the higher-order aspects.
- This low-level approach is still worthwhile for many exact disciplines. In fact, both in logic and in computer science it has certainly not been wiped out, and for good reasons.

Automath

- The first tool for mechanical representation and verification of mathematical proofs, AUTOMATH, has a parameter mechanism.
- Mathematical text in AUTOMATH written as a finite list of *lines* of the form:

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

  Here g is a new name, an abbreviation for the expression t of type T and $x_1, \ldots, x_n$ are the parameters of g, with respective types $A_1, \ldots, A_n$.
- Each line introduces a new definition which is inherently parametrised by the variables occurring in the context needed for it.
- Developments of ordinary mathematical theory in AUTOMATH [Benthem Jutting, 1977] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

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

- We add parametric constants of the form $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types and $c \in \mathcal{C}$.
- $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.
- $\mathbf{R}$ allows several kinds of Π-constructs. We also use a set $\boldsymbol{P}$ of (s_1, s_2) where $s_1, s_2 \in \{*, \Box\}$ to allow several kinds of parametric constants.
- $(s_1, s_2) \in \boldsymbol{P}$ means that we allow parametric constants $c(b_1, \ldots, b_n) : A$ where $b_1, \ldots, b_n$ have types $B_1, \ldots, B_n$ of sort s_1, and A is of type s_2.
- If both $(*, s_2) \in \boldsymbol{P}$ and $(\Box, s_2) \in \boldsymbol{P}$ then combinations of parameters allowed. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\Box$.

The Cube with parametric constants

- Let $(*,*) \subseteq \mathbf{R}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.
- $\lambda\mathbf{R}\boldsymbol{P} = \lambda\mathbf{R}$ and the two rules **($\vec{\mathbf{C}}$-weak)** and **($\vec{\mathbf{C}}$-app)**:

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

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

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

Properties of the Refined Cube

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

Example

- $\boldsymbol{R} = \{(*,*),(*,\Box)\}$

  $\boldsymbol{P}_1 = \emptyset \qquad \boldsymbol{P}_2 = \{(*,*)\} \qquad \boldsymbol{P}_3 = \{(*,\Box)\} \qquad \boldsymbol{P}_4 = \{(*,*),(*,\Box)\}$

  All $\lambda\boldsymbol{R}\boldsymbol{P}_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

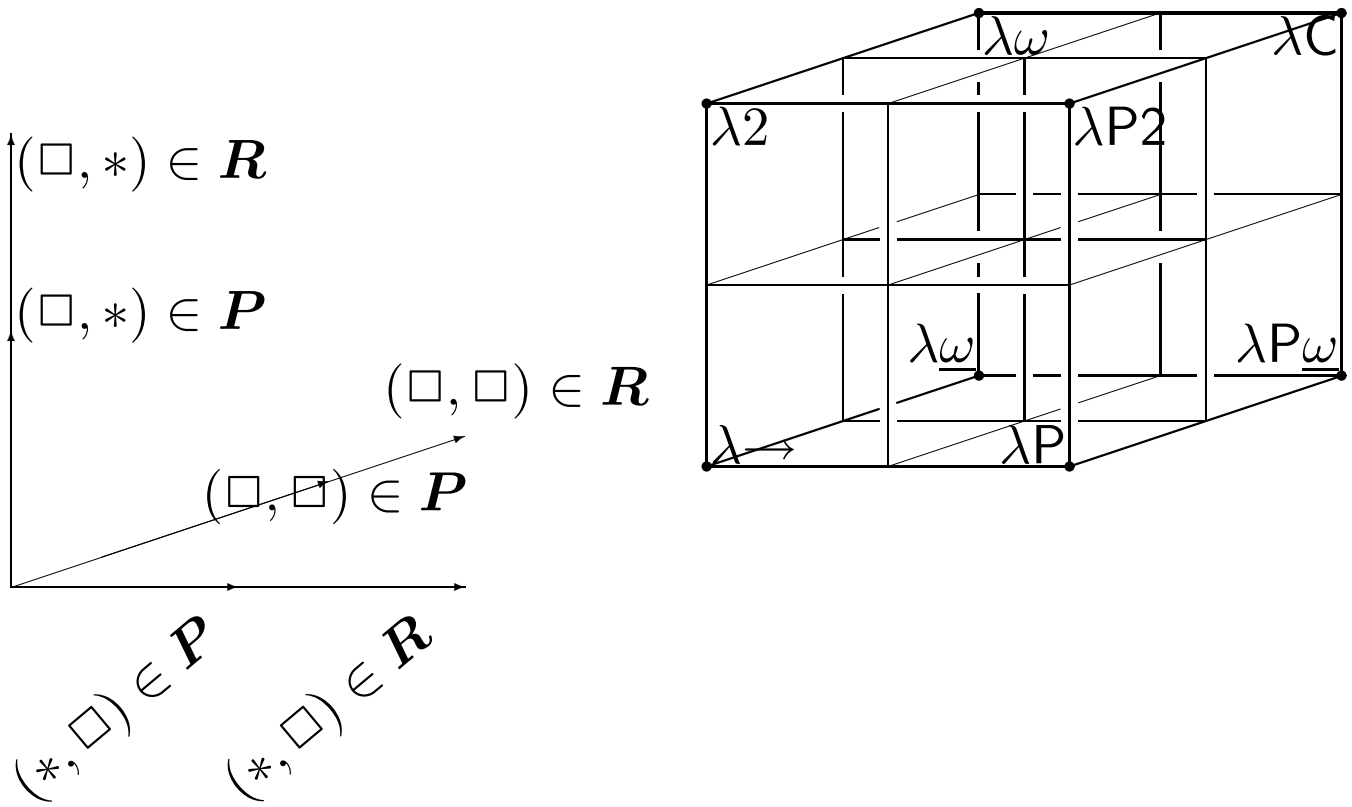
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*),(*,\Box)\}.$

  $\lambda\!\!\rightarrow \;<\; \lambda\boldsymbol{R}_5\boldsymbol{P}_5 \;<\;$ λP: we can to talk about predicates:

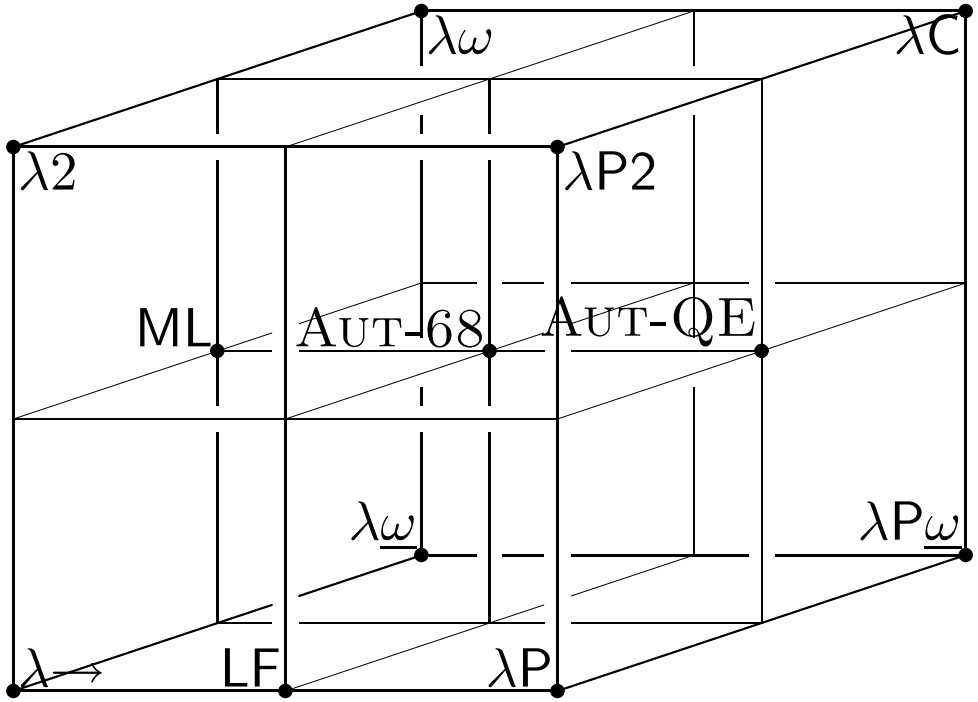
$$\begin{array}{rcl} \alpha & : & *, \\ \mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\ \mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \\ \mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & : & \mathtt{eq}(\mathtt{y},\mathtt{x}),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

  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 $\lambda\boldsymbol{R}$ where $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:

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

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

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

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

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

Logicians versus mathematicians and induction over numbers

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

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

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

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

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

Who is the logician? who is the mathematician? Who is the computer scientis

- 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.

- Parameters enable us to find an exact position of type systems in the generalised framework of type systems.

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

This is harder. Who is the logician? who is the mathematician? Who is t
developer? Who is the user?

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

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

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

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

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

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

- The type formation rule becomes

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

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

(start) $$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}\ x \notin \mathrm{DOM}\,(\Gamma)$$

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

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

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

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

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

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

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

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

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

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

We have however an organised multiplicity of types.

1. If $\Gamma \vdash_\flat A : B_1$ and $\Gamma \vdash_\flat A : B_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.
2. If $\Gamma \vdash_\flat A_1 : B_1$ and $\Gamma \vdash_\flat A_2 : B_2$ and $A_1 =_\flat A_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.
3. If $\Gamma \vdash_\flat B_1 : s_1$, $B_1 =_\flat B_2$ and $\Gamma \vdash_\flat A : B_2$ then $\Gamma \vdash_\flat B_2 : s_1$.
4. Assume $\Gamma \vdash_\flat A : B_1$ and $(\Gamma \vdash_\flat A : B_1)^{-1} = (\Gamma', A', B_1')$. Then $B_1 =_\flat B_2$ if:
   (a) either $\Gamma \vdash_\flat A : B_2$, $(\Gamma \vdash_\flat A : B_2)^{-1} = (\Gamma', A'', B_2')$ and $B_1' =_\beta B_2'$,
   (b) or $\Gamma \vdash_\flat C : B_2$, $(\Gamma \vdash_\flat C : B_2)^{-1} = (\Gamma', C', B_2')$ and $A' =_\beta C'$.

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

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

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

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

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

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

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

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

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

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

Failure of correctness of types and subject reduction

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

Solving the problem

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

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

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

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

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

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

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

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

Correctness of types holds.

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

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

Subject Reduction holds.

- We demonstrate this with the earlier example.

- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \rightarrow_\beta x$ and we need to show that $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.

- Here is how the latter formula now holds:

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

Lesson: it is important to respect history

Common Mathematical Language of mathematicians: CML

+ CML is *expressive*: it has linguistic categories like *proofs* and *theorems*.
+ CML has been refined by intensive use and is rooted in *long traditions*.
+ CML is *approved* by most mathematicians as a communication medium.
+ CML *accommodates many branches* of mathematics, and is adaptable to new ones.
– Since CML is based on natural language, it is *informal* and *ambiguous*.
– CML is *incomplete:* Much is left implicit, appealing to the reader's intuition.
– CML is *poorly organised:* In a CML text, many structural aspects are omitted.
– CML is *automation-unfriendly:* A CML text is a plain text and cannot be easily automated.

A CML-text

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

Theorem 6. [Commutative Law of Addition]

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

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

$$y + 1 = y',$$

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

$$1 + y = y',$$

so that

$$1 + y = y + 1$$

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

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

Therefore

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

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

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

hence

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

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

The problem with formal logic

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

- *So, mathematicians kept to* CML*.*

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

What are the options for computerization?

Computers can handle mathematical text at various levels:

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

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

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

The issues with typesetting systems

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

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

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

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

LaTeX example

draft documents	✓
public documents	✓
computations and proofs	✗

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

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

\end {theorem}

\begin{proof}

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

  \begin{enumerate}

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

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

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

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

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

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

  \end{enumerate}

  The assertion therefore holds for all $x$.

\end{proof}
```

Full formalization difficulties: choices

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

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

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

Full formalization difficulties: informality

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

- A single (big) step may need to expand into a (series of) syntactic proof expressions. *Very long expressions can replace a clear* CML*-text.*
- The entire CML-text may need *reformulation* in a fully *complete* syntactic formalism where every detail is spelled out. New details may need to be woven throughout the entire text. The text may need to be *turned inside out.*
- Reasoning may be obscured by *proof tactics*, whose meaning is often *ad hoc* and implementation-dependent.

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

Coq example

draft documents	✗
public documents	✗
computations and proofs	✓

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

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

Proof.

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

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

Qed.
```

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

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

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

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

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

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

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

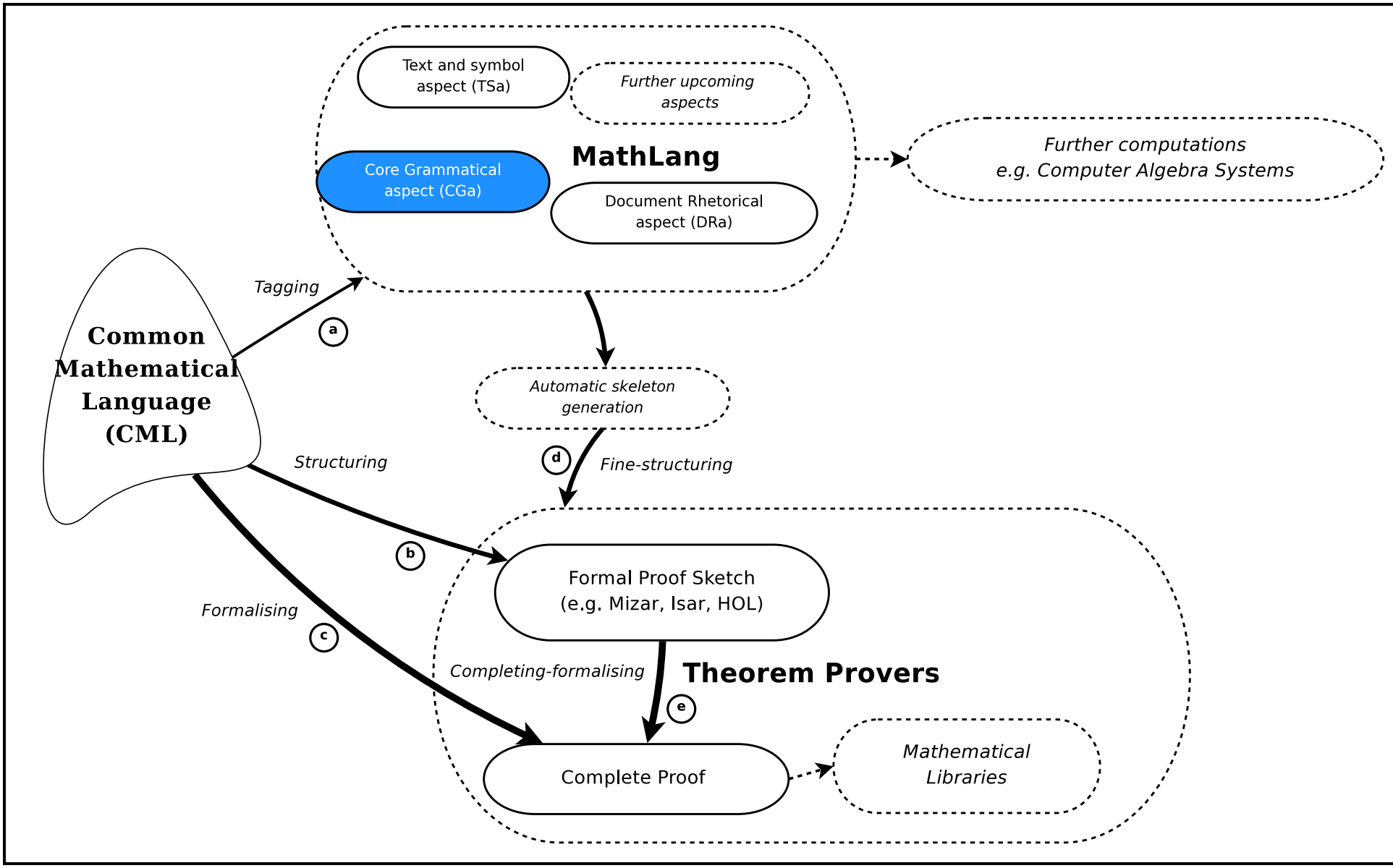
4. This formalisation of text is not much harder for the ordinary mathematician than LaTeX. *Full formalization down to a foundation of mathematics is not required*, although allowing and supporting this is one goal.

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

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.


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

What is CGa? (Maarek's PhD thesis)

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

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

  term set noun adjective statement declaration definition step context .

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

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

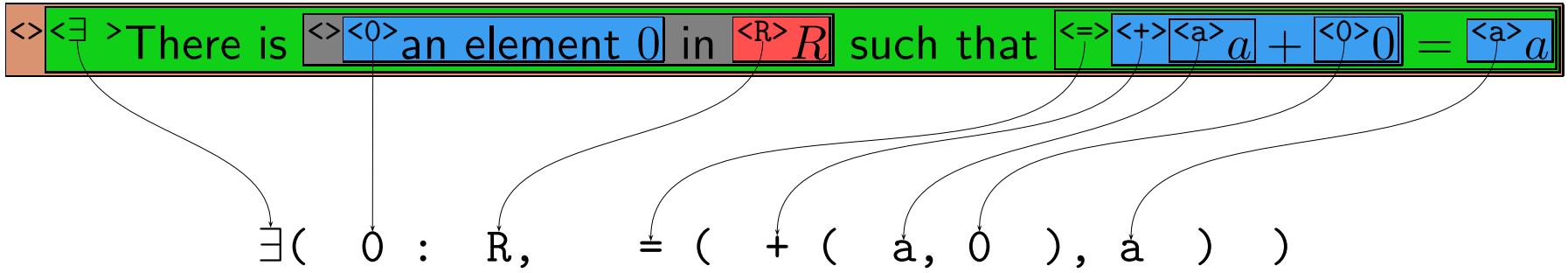


Figure 1: Example of CGa encoding of CML text

Weak Type Theory

In Weak Type Theory (or WTT) we have the following linguistic categories:

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

Categories of syntax of WTT

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

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

Derivation rules

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

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

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

$$B;\Gamma\ \vdash\ t :: T, \qquad B;\Gamma\ \vdash\ s :: S, \qquad B;\Gamma\ \vdash\ n :: N,$$
$$B;\Gamma\ \vdash\ a :: A, \qquad B;\Gamma\ \vdash\ p :: P, \qquad B;\Gamma\ \vdash\ d :: D$$

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

Examples of derivation rules

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

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

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

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

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

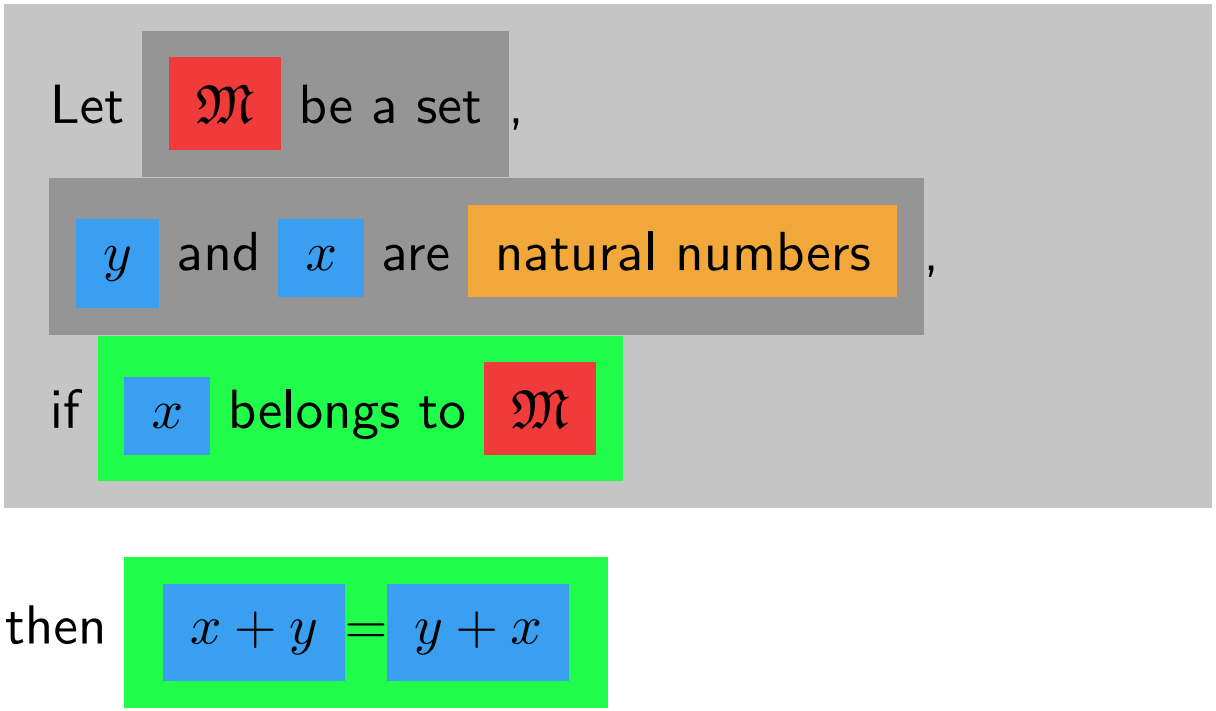
Properties of WTT

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

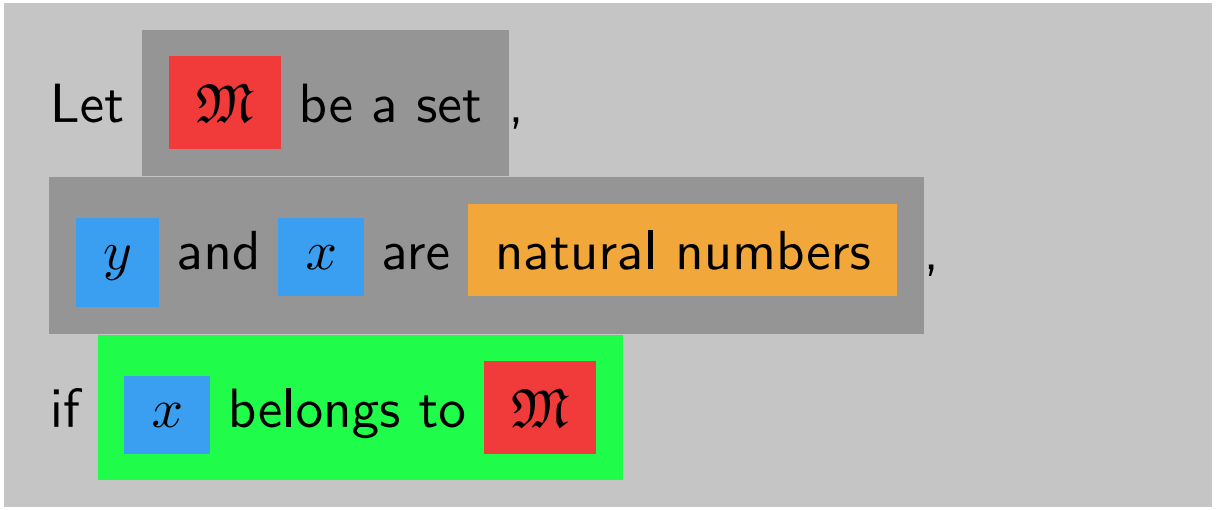
Definition unfolding

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

CGa Weak Type Checking



CGa Weak Type checking detects grammatical errors



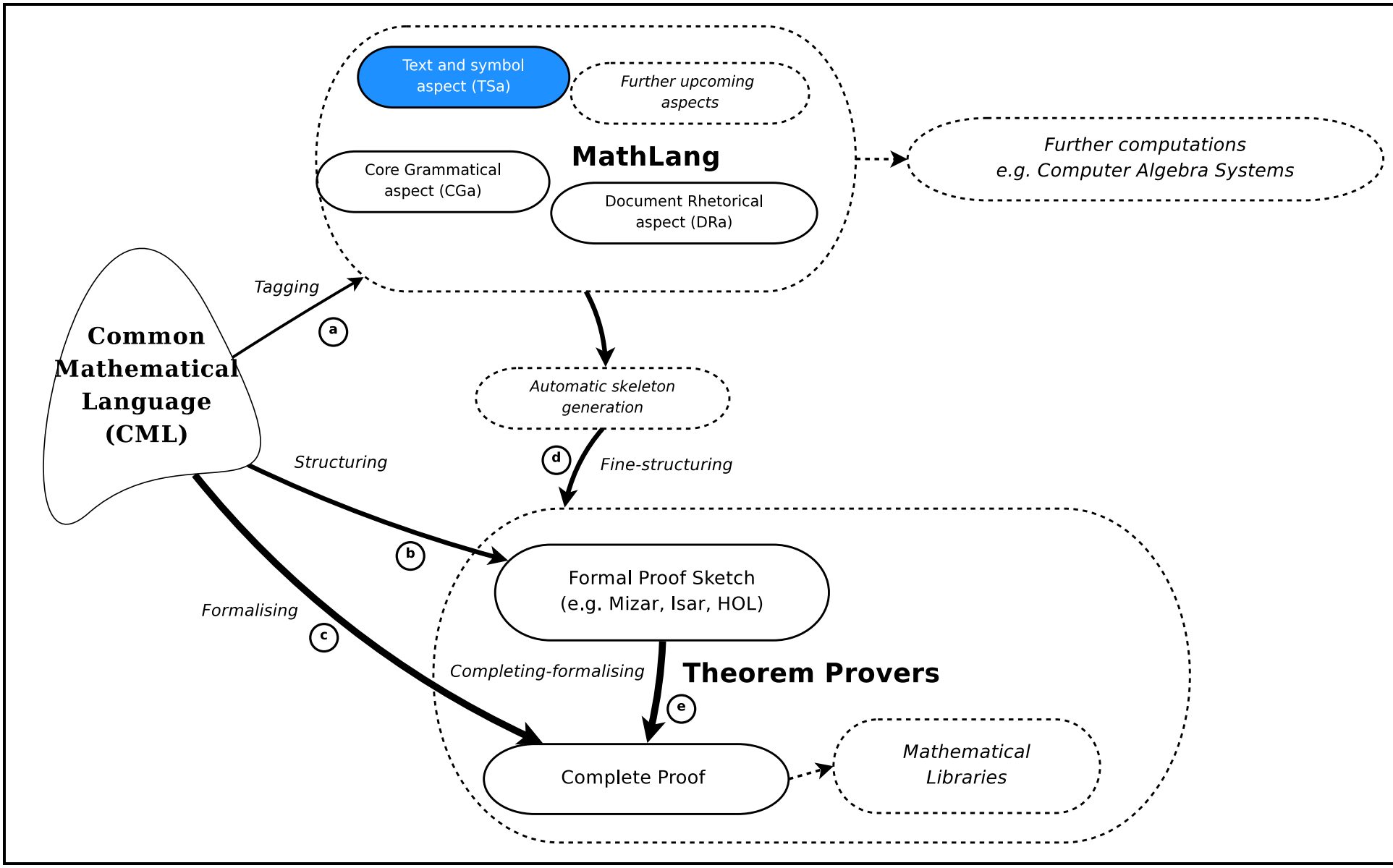
then $x + y$ $\Leftarrow$ **error**

How complete is the CGa?

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

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

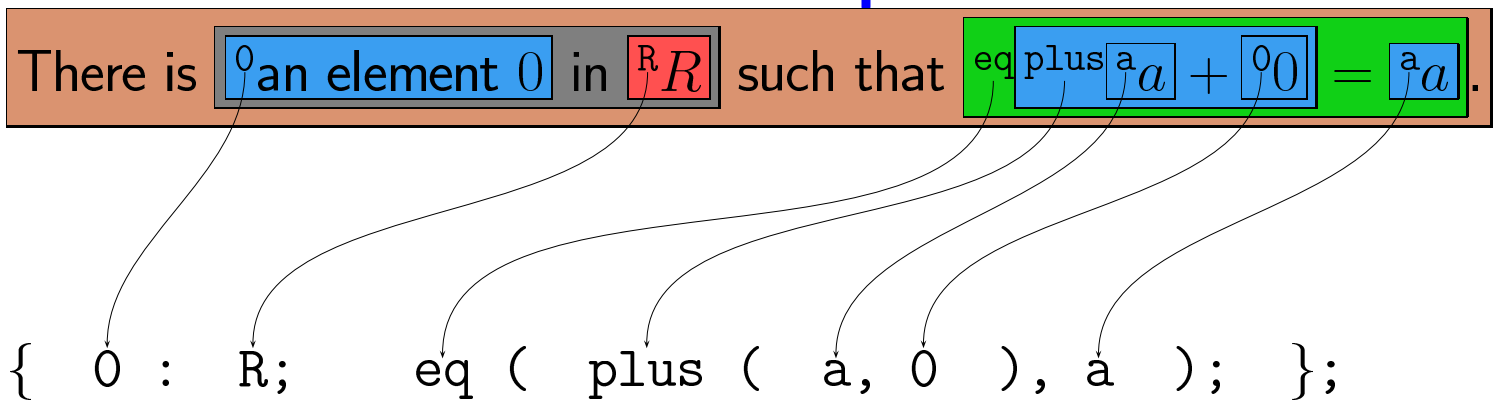
- The type checker for CGa works well and gives some useful error messages. Error messages should be improved.


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

What is TSa? Lamar's PhD thesis

- TSa builds the bridge between a CML text and its grammatical interpretation and adjoins to each CGa expression a string of words and/or symbols which aims to act as its CML representation.
- TSa plays the role of a user interface
- TSa can flexibly represent natural language mathematics.
- The author wraps the natural language text with boxes representing the grammatical categories (as we saw before).
- The author can also give interpretations to the parts of the text.

Interpretations



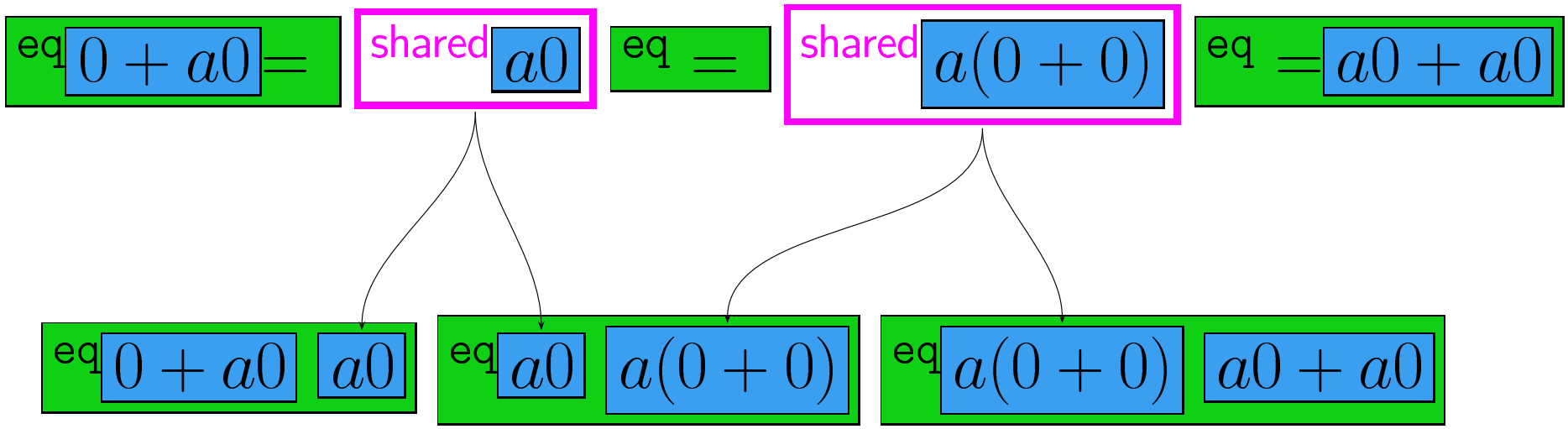
There is ^{0}an element 0 in ^{R}R such that $^{eq}\,^{plus}\,^{a}a + {}^{0}0 = {}^{a}a$.

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

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

Rewrite rules enable natural language representation

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



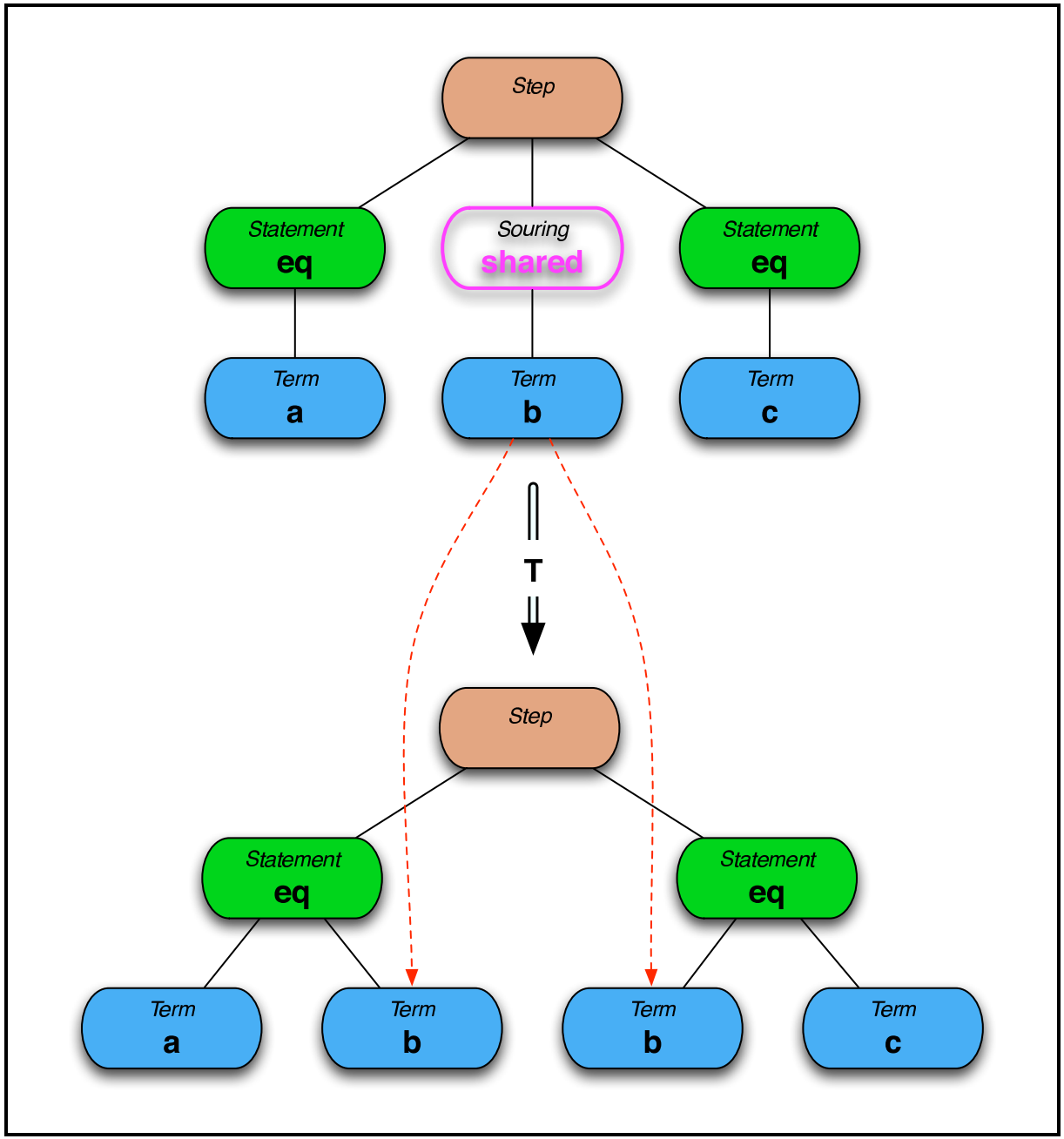
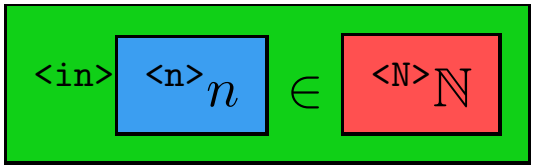
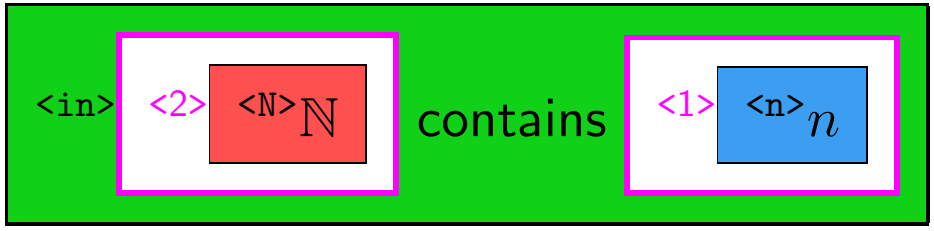


Figure 2: Example for a simple shared souring

reordering/position Souring



$ann =$



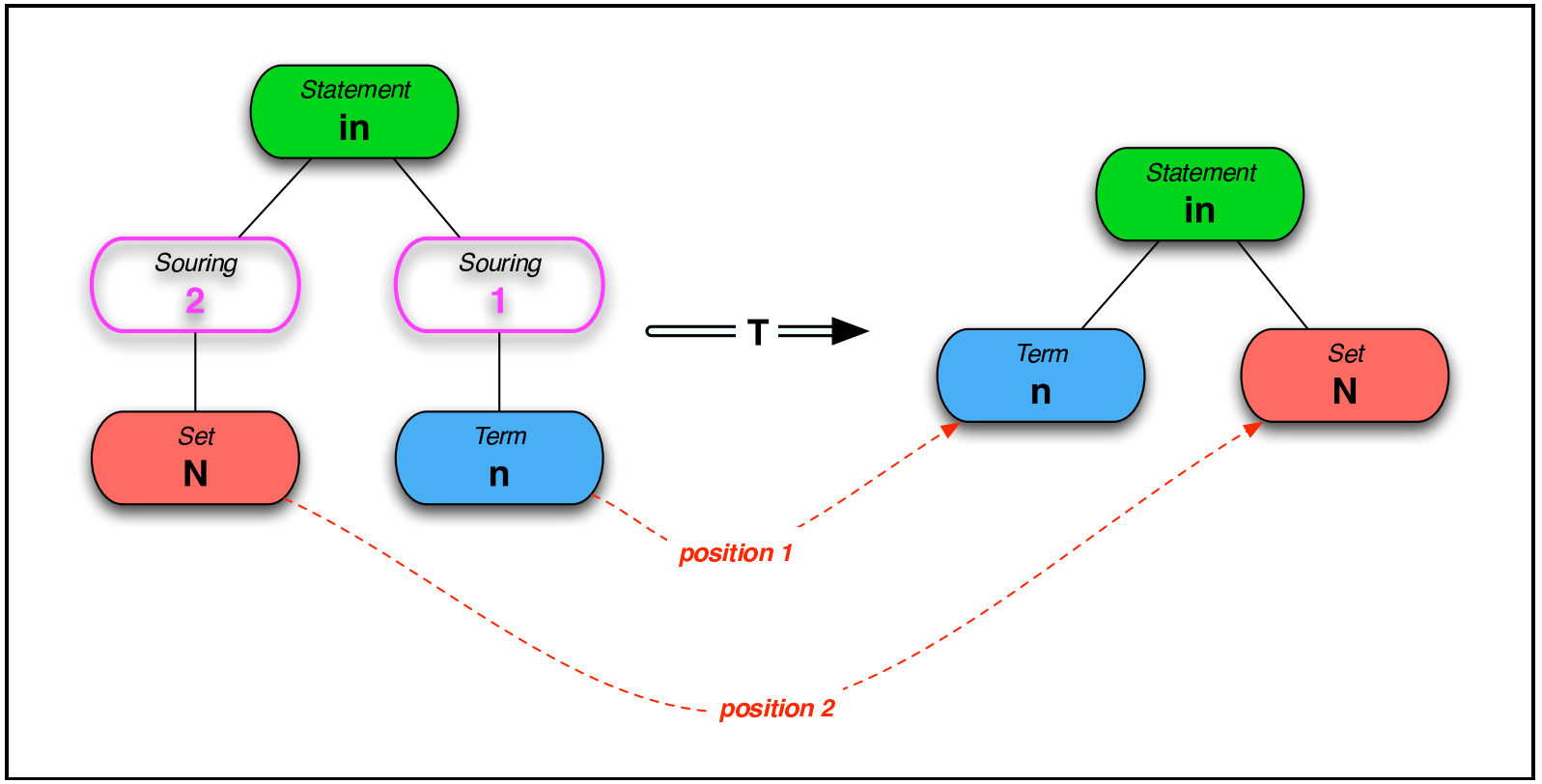
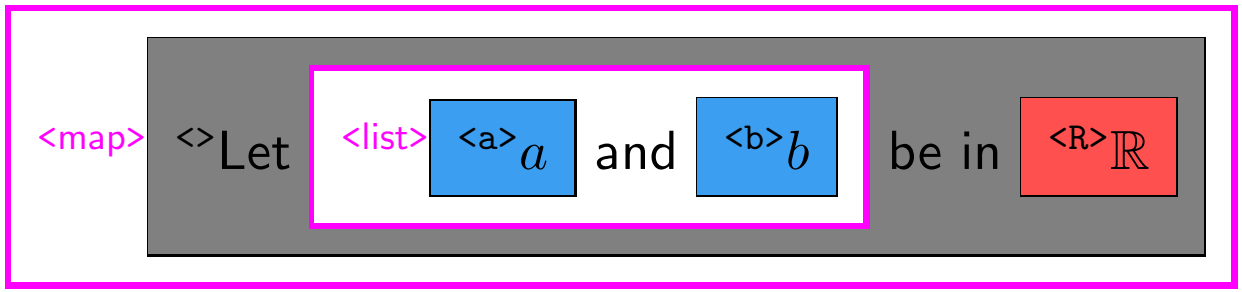
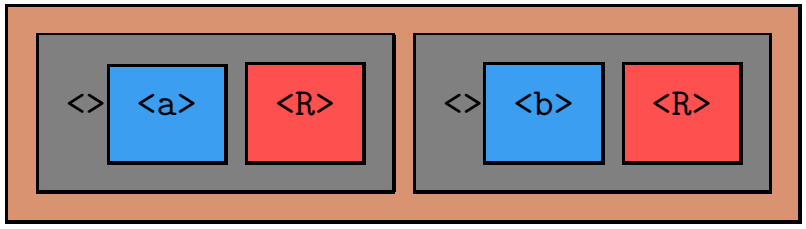


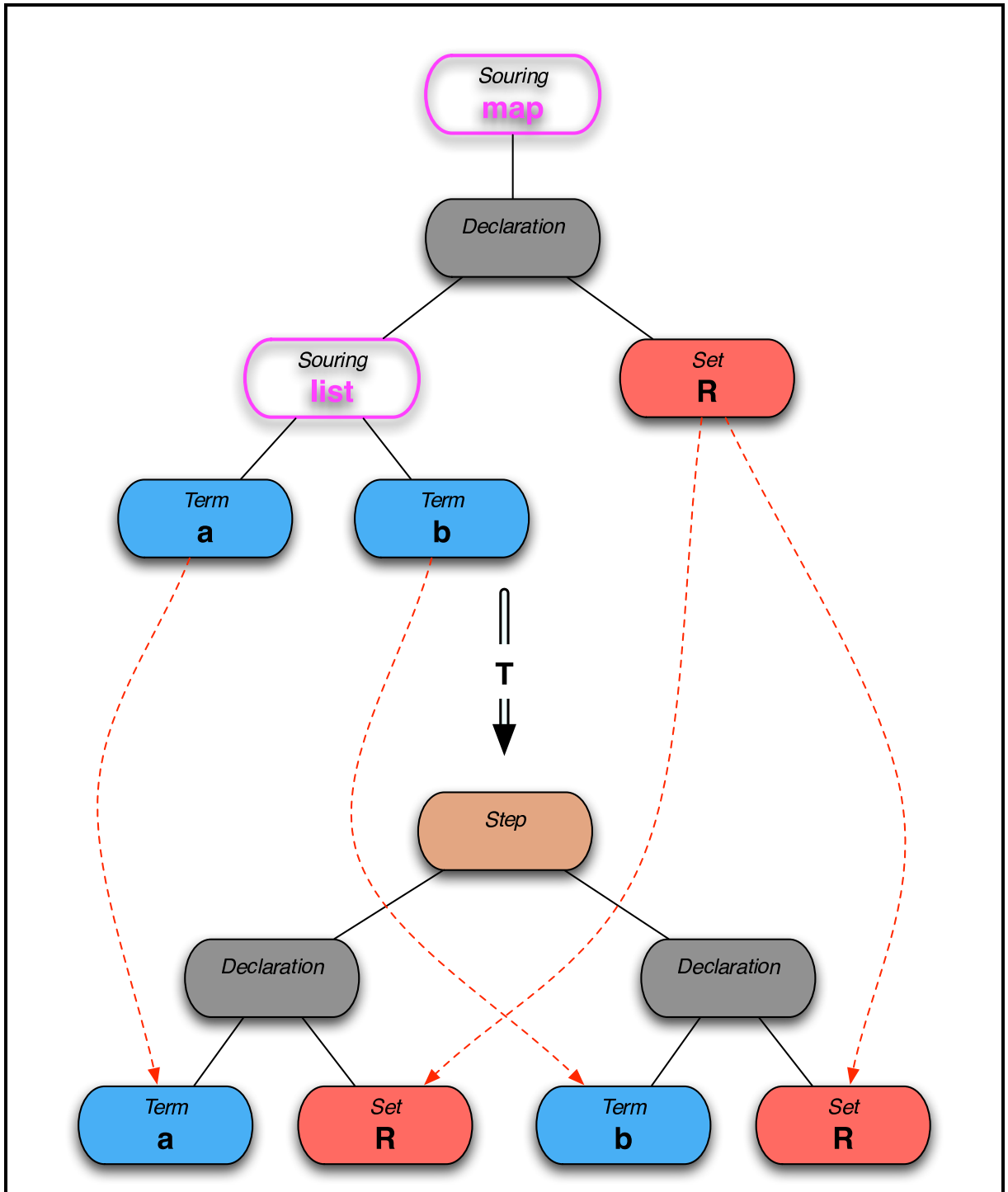
Figure 3: Example for a position souring

map souring

$ann =$ 

This is expanded to

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

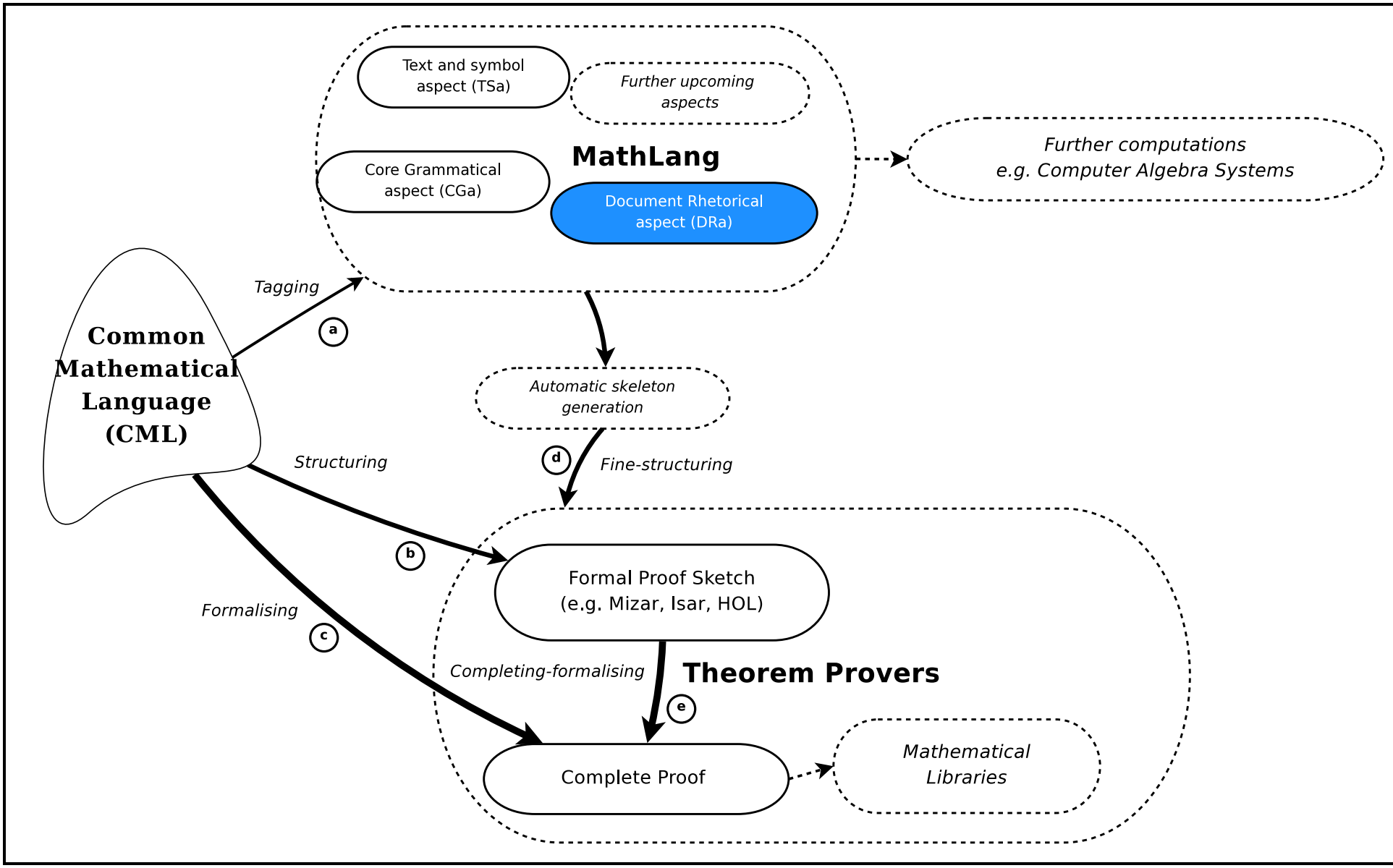

Souring
map
Declaration
Souring
list
Set
R
Term
a
Term
b
T
Step
Declaration
Declaration
Term
a
Set
R
Term
b
Set
R

How complete is TSa?

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

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

- The TSa theory and metatheory need development.


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

What is DRa? Retel's PhD thesis

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

Relations

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

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

What does the mathematician do?

- The mathematician wraps into boxes and uniquely names chunks of text
- The mathematician assigns to each box the structural and/or mathematical rhetorical roles
- The mathematician indicates the relations between wrapped chunks of texts

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

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

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

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

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

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

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

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

Lemma 1.

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

Proof.

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

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

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

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

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

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

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

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

□

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

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

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

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

Lemma 1.

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

Proof.

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

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

justifies

uses

Claim. $P(m) \Longrightarrow \exists m' < m.P(m')$. **F**

uses

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 $2n^2 = m^2 = 4k^2 \Longrightarrow n^2 = 2k^2$ 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 $m' = n$. **G**

justifies

uses

uses

B

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

subpartOf

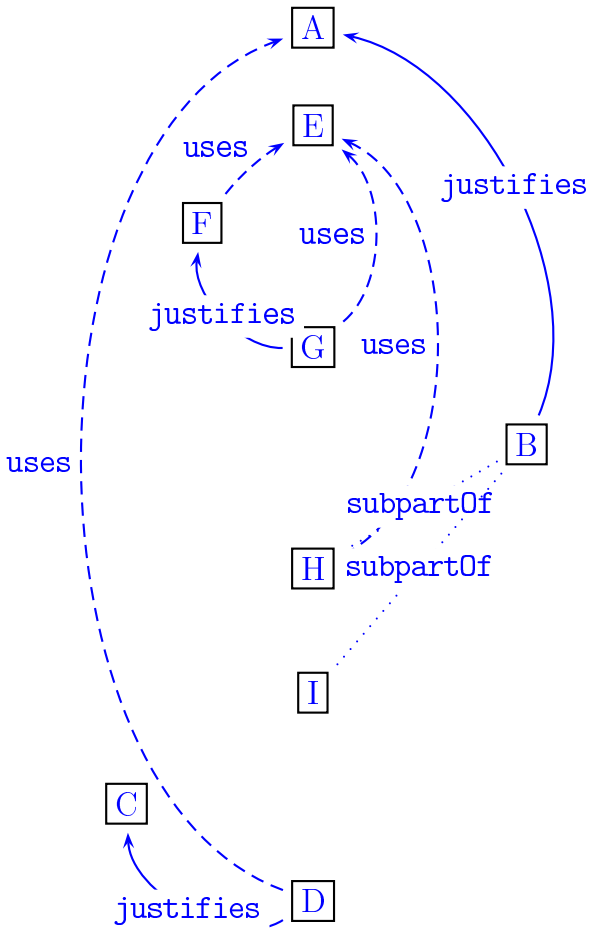
subpartOf

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

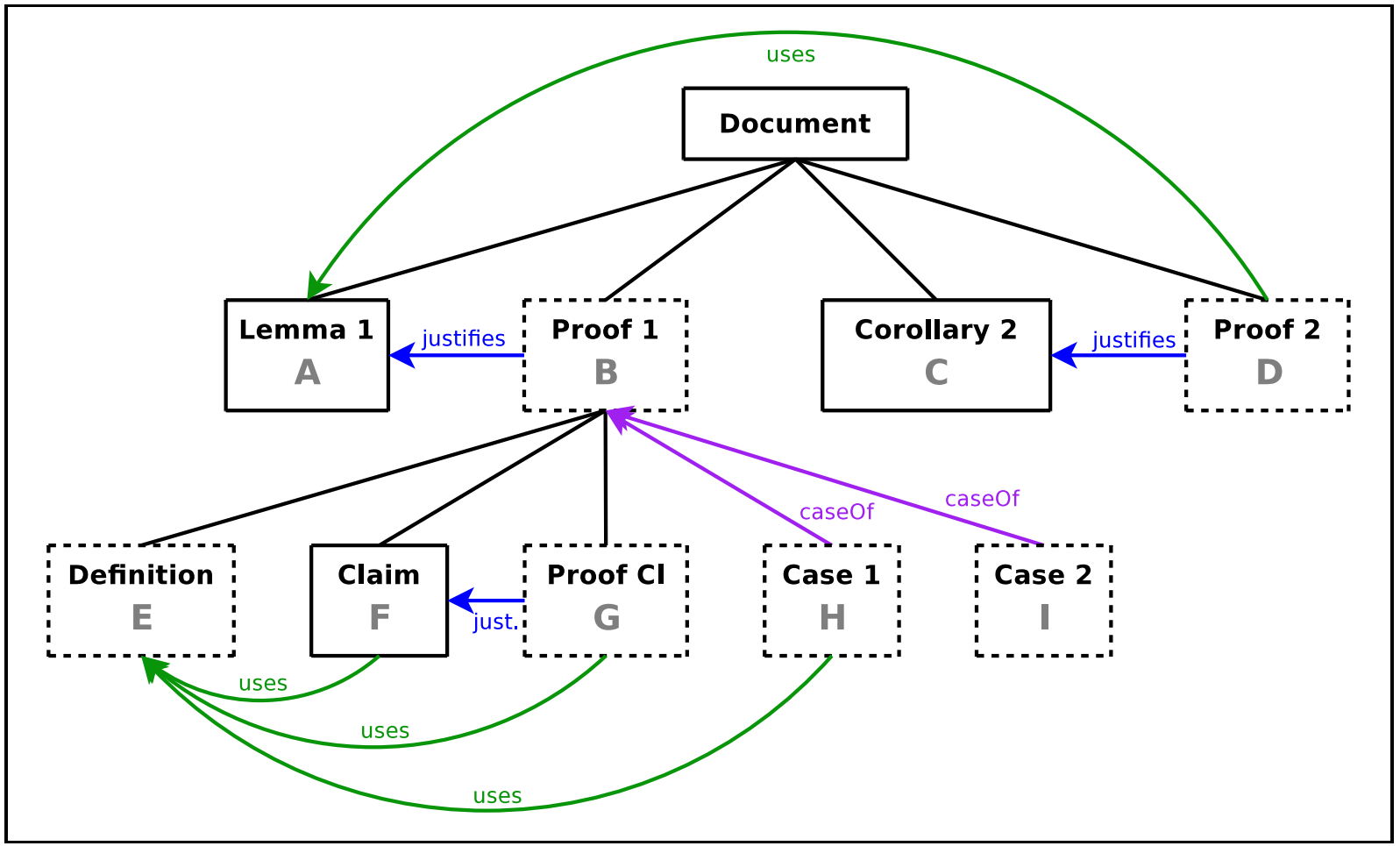
with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction. **H**

The automatically generated dependency Graph

Dependency Graph (DG)



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



The Graph of Textual Order: GoTO
Zengler's thesis

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

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

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

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

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

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

Graph of Textual Order

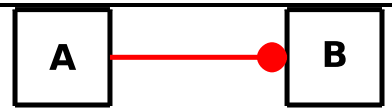
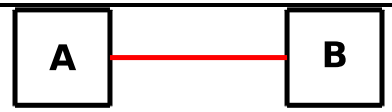
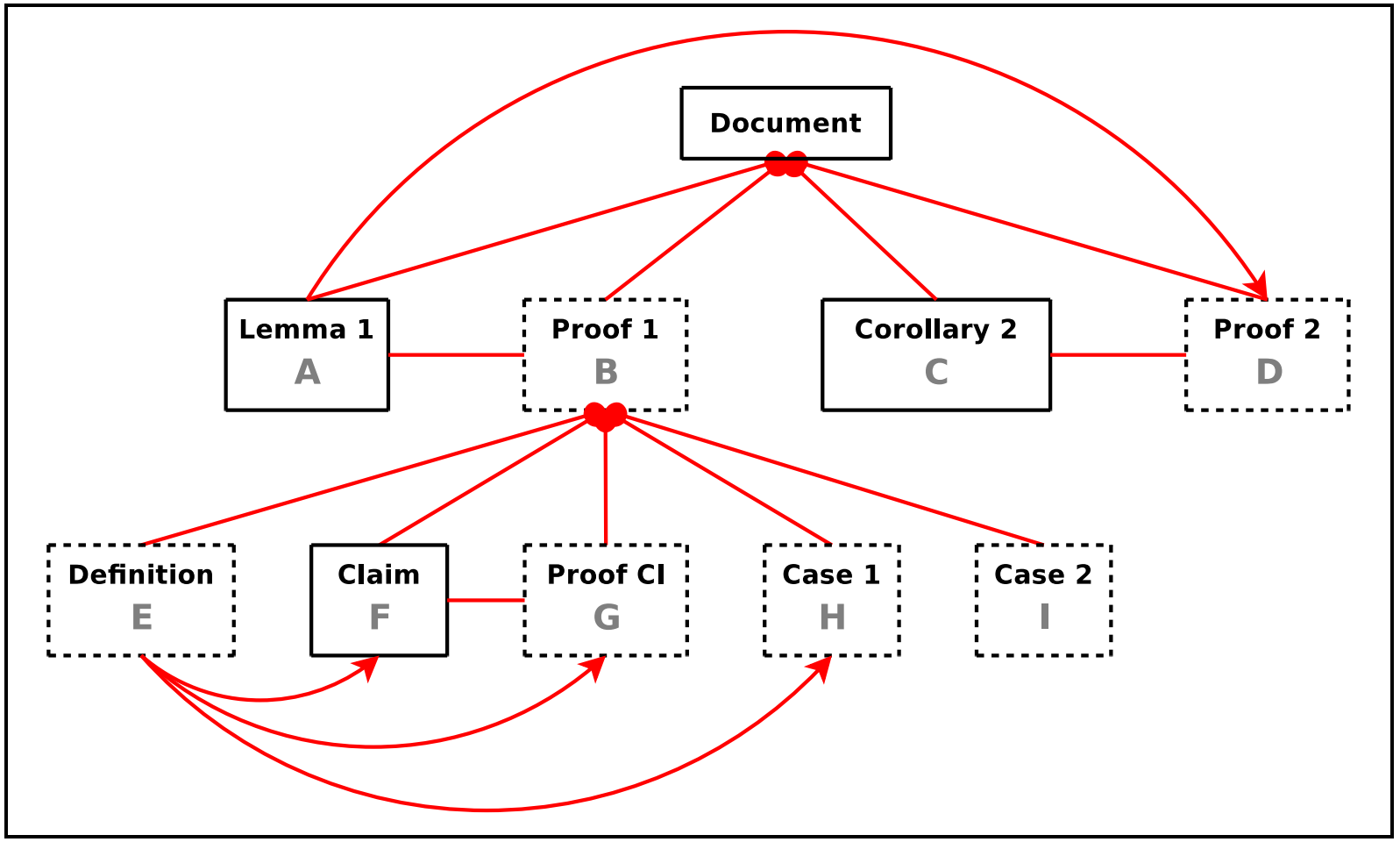
(A, uses, B)	$A \succ B$	
(A, caseOf, B)	$A \preceq B$	
$(A, \text{justifies}, B)$	$A \leftrightarrow B$	

Table 1: Graphical representation of edges in the GoTO

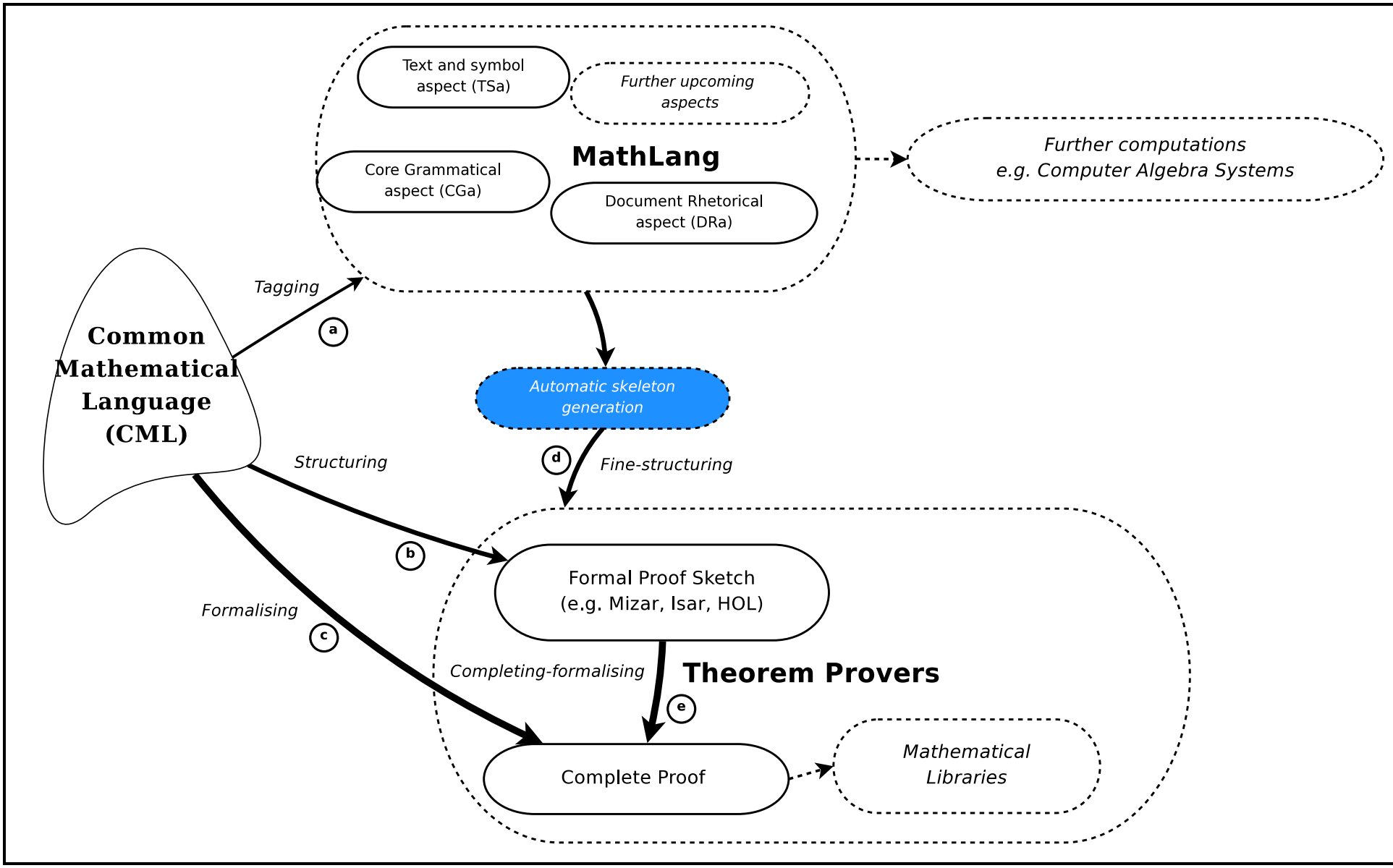
The GoTO can be generated automatically from the DG and therefore (since the DG can be produced automatically from an annotated document) automatically from an annotated document.

Graph of Textual Order for the DRa tree example



How complete is DRa?

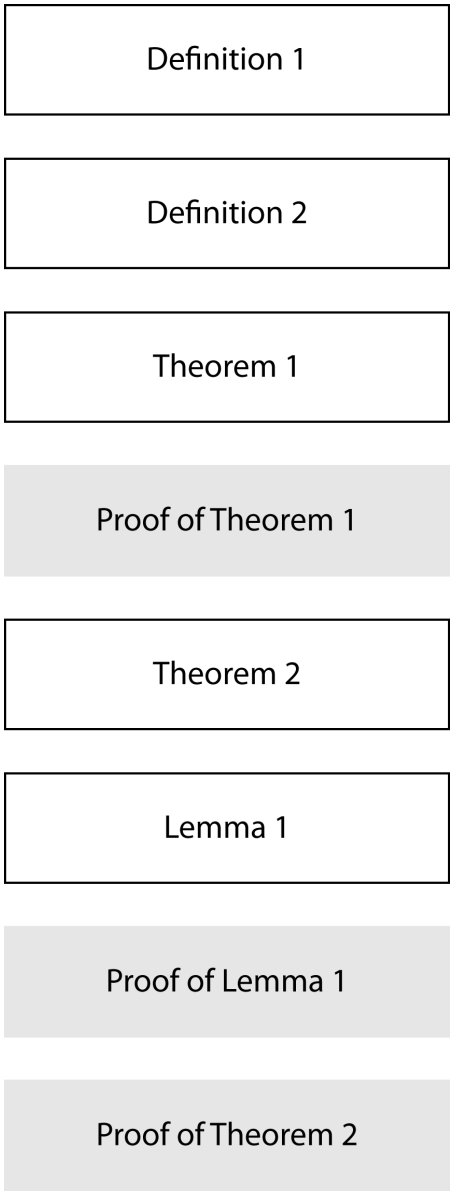
- The dependency graph can be used to check whether the logical reasoning of the text is coherent and consistent (e.g., no loops in the reasoning).
- However, both the DRa language and its implementation need more experience driven tests on natural language texts.
- Also, the DRa aspect still needs a 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.


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

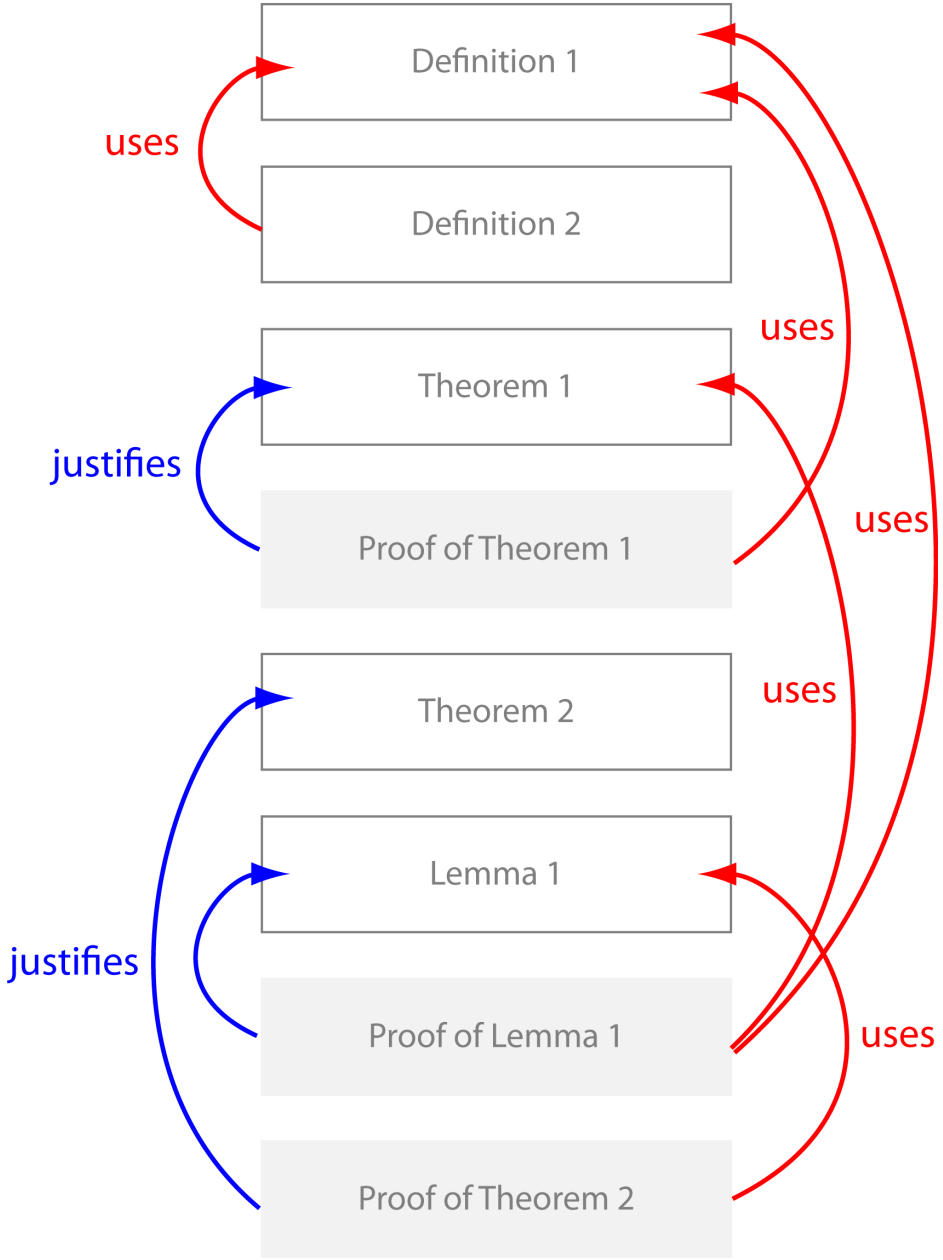
Different provers have

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

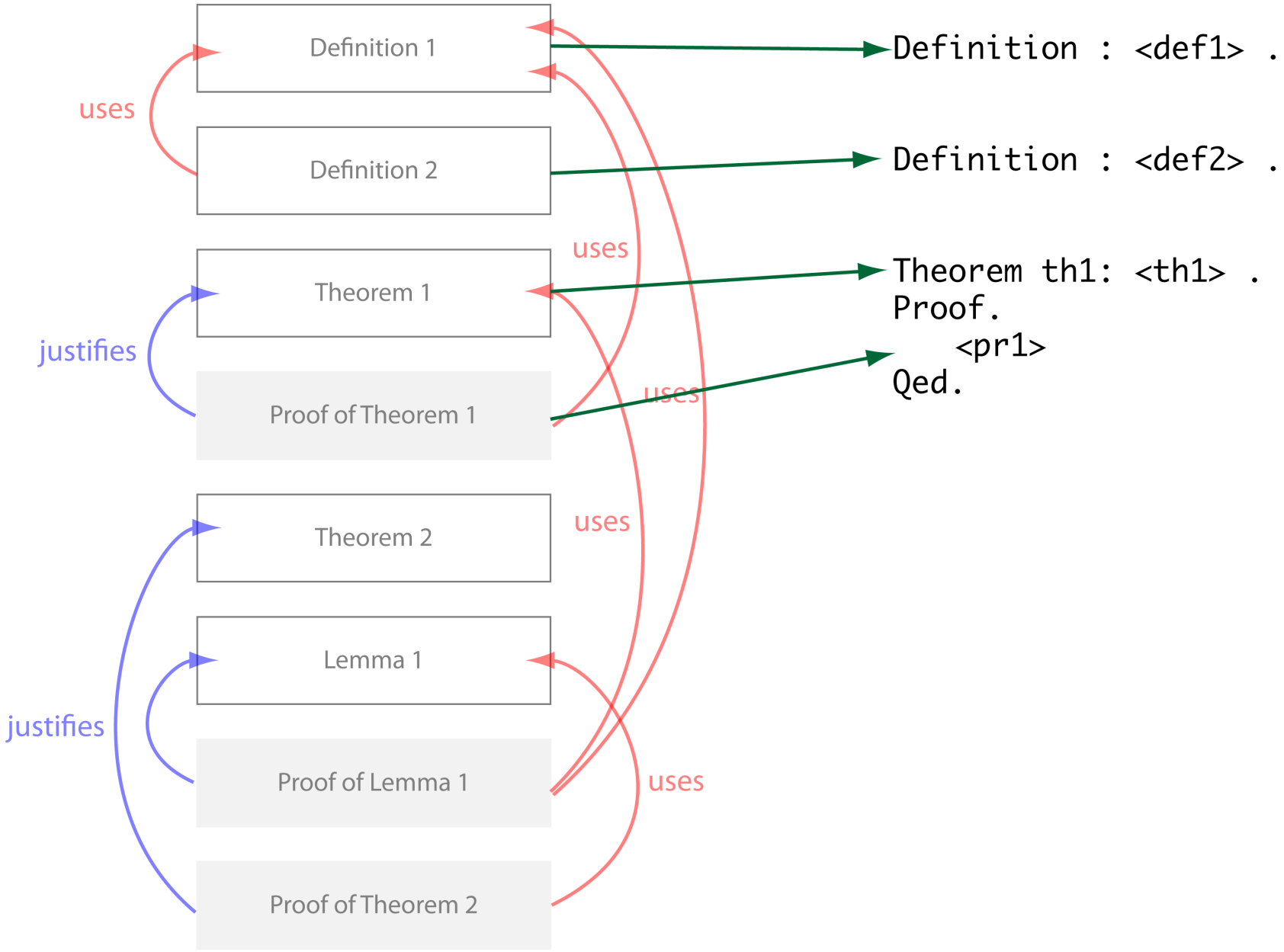
Example of nested theorems/lemmas (Moller, 03, Chapter III,2)



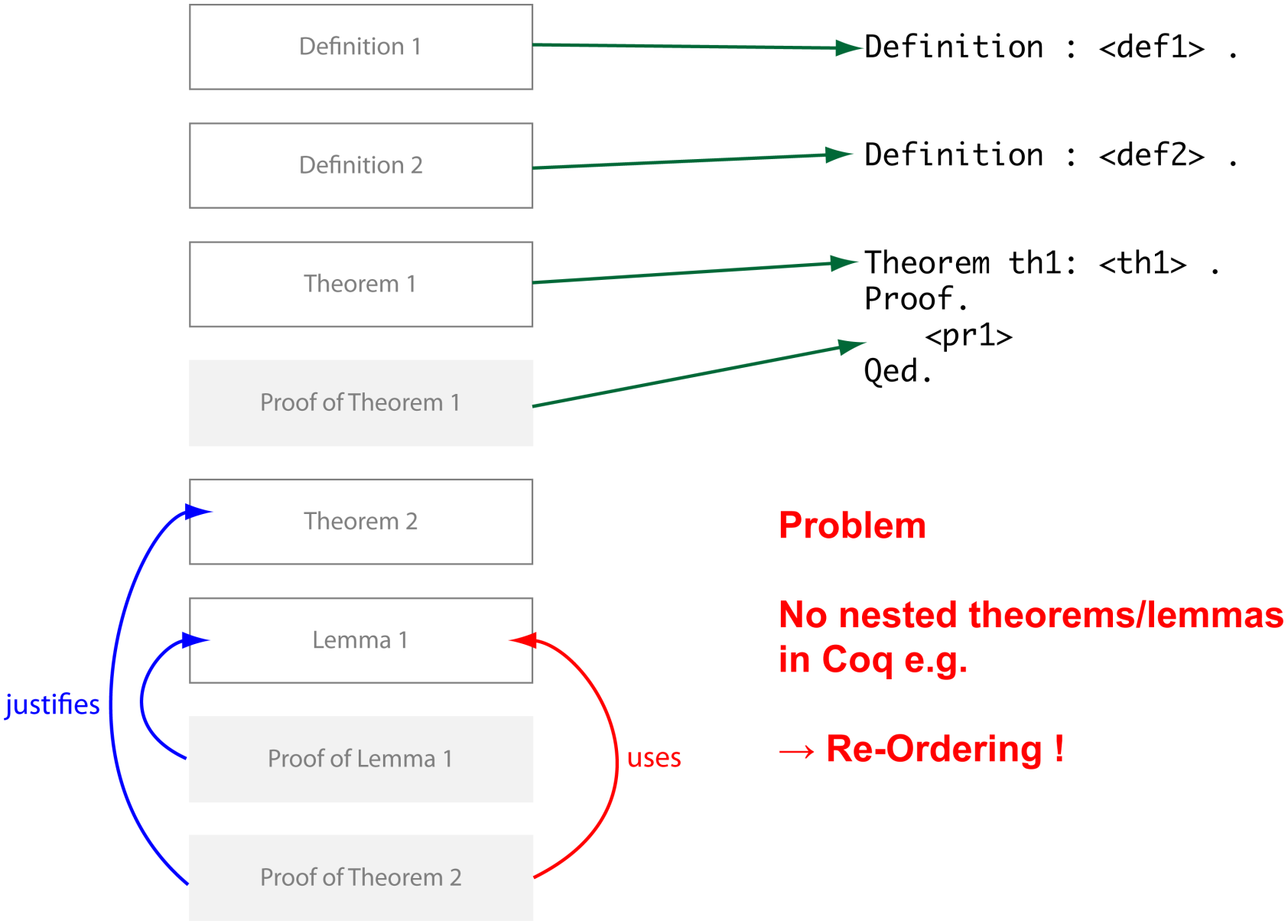
The automatic generation of a proof skeleton



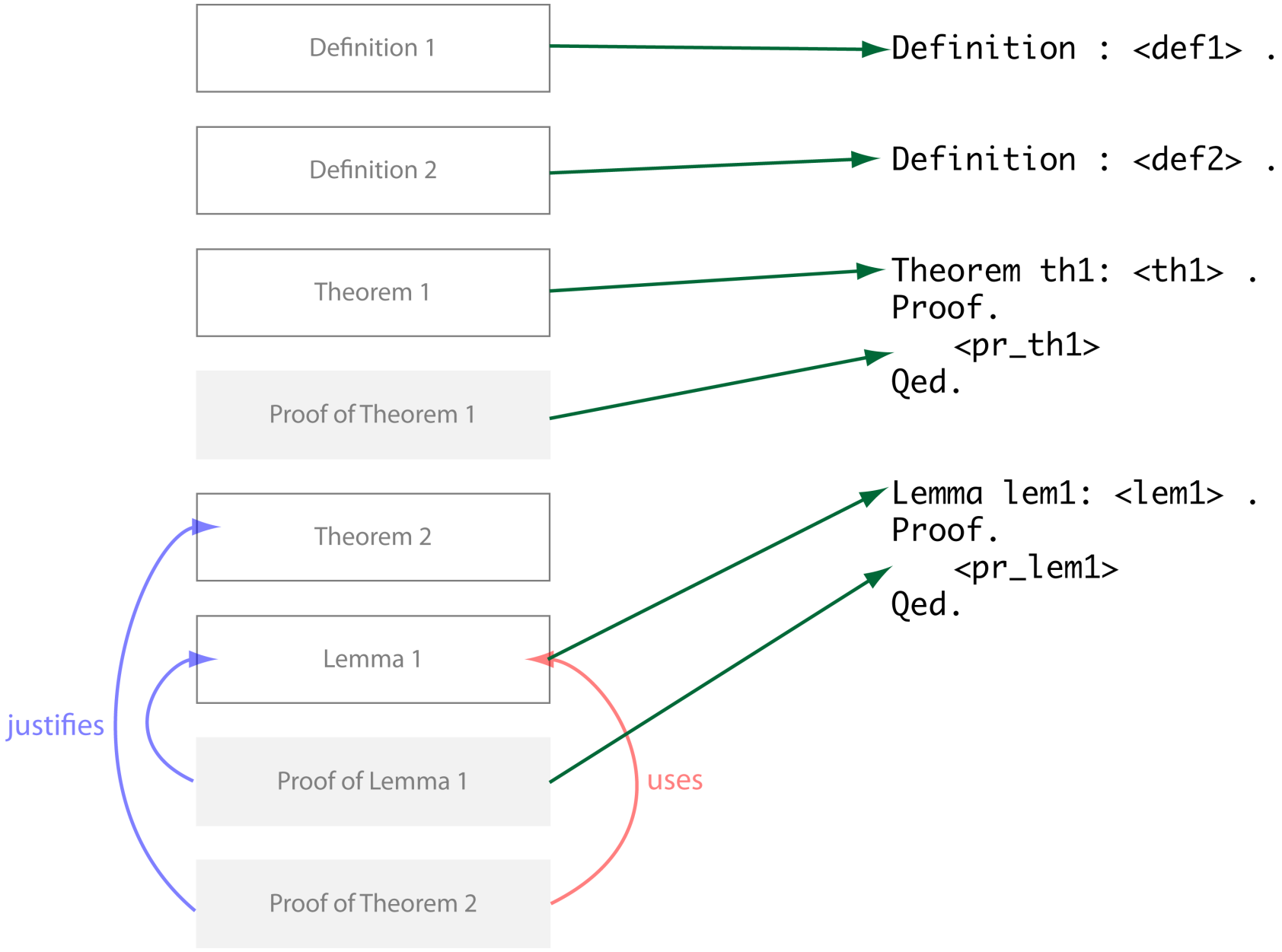
The DG for the example



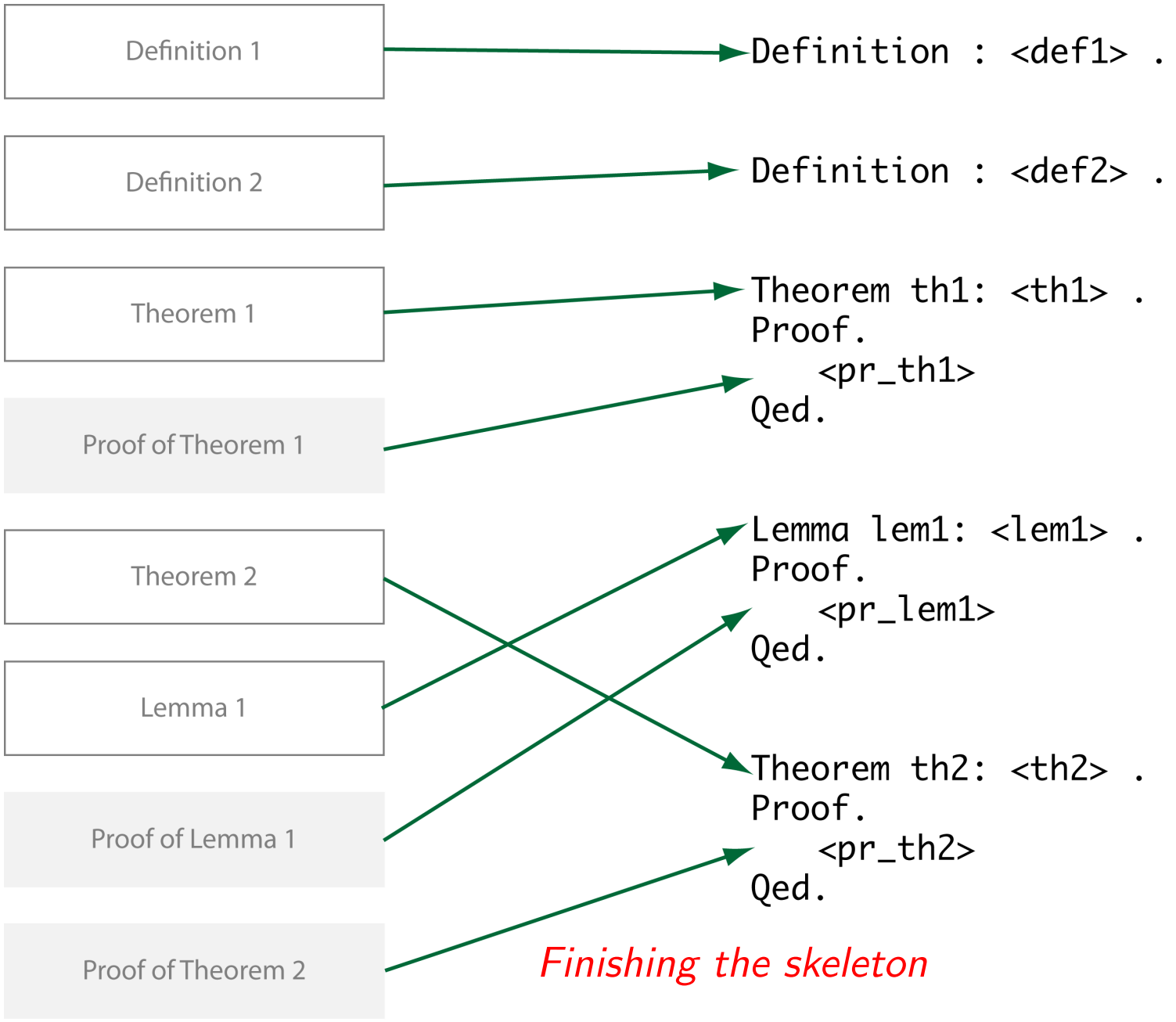
Straight-forward translation of the first part



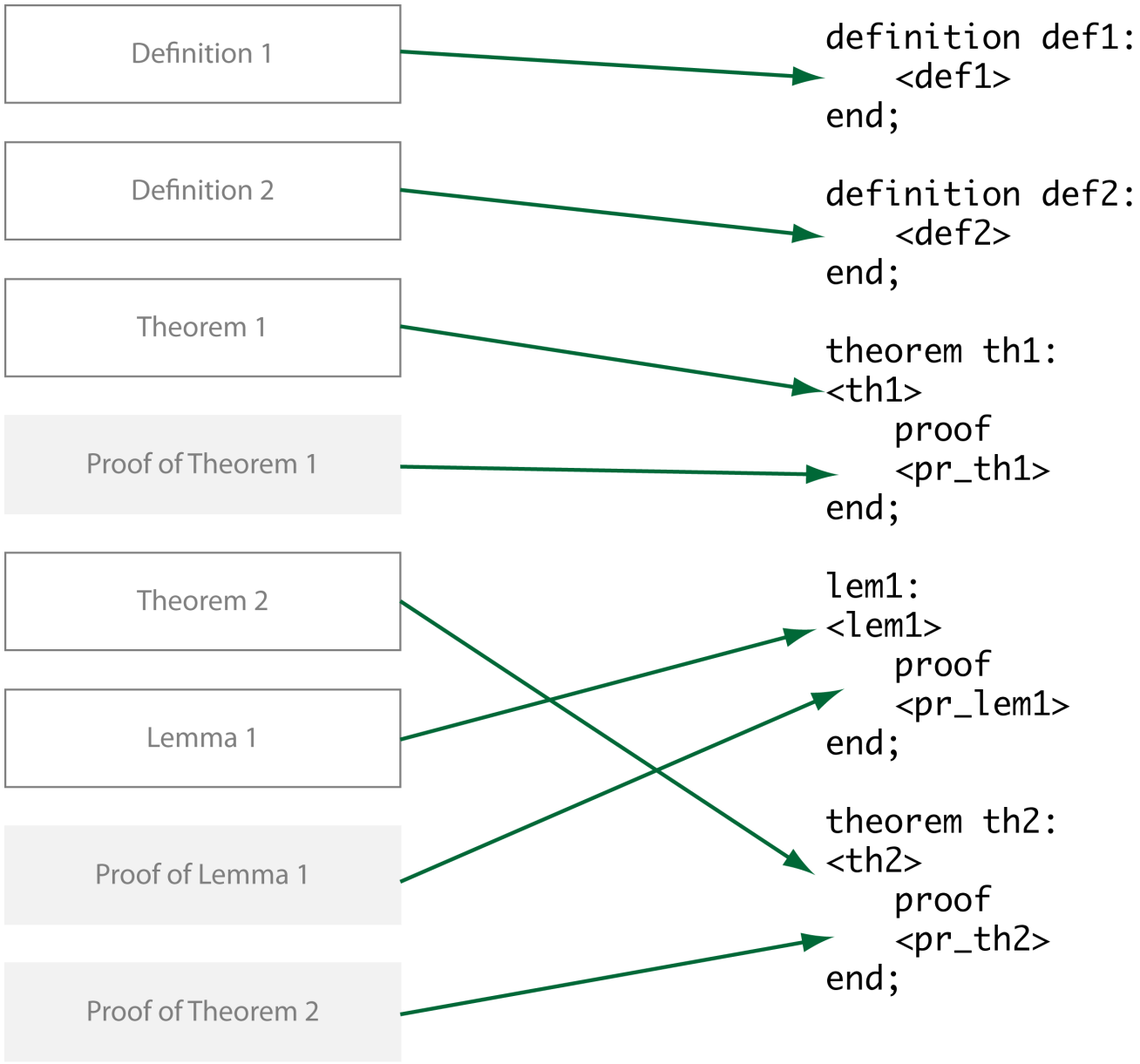
Problem: nested theorems



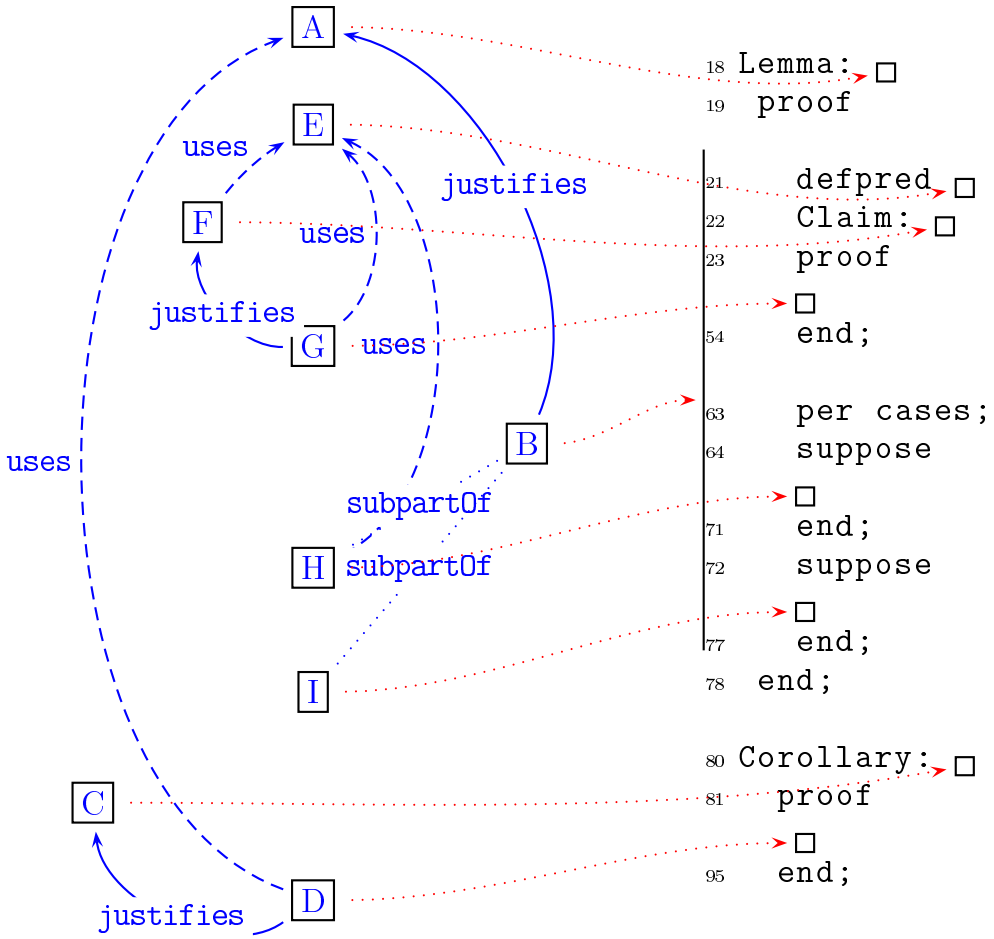
Solution: Re-ordering


Definition 1
Definition 2
Theorem 1
Proof of Theorem 1
Theorem 2
Lemma 1
Proof of Lemma 1
Proof of Theorem 2
Definition : <def1> .
Definition : <def2> .
Theorem th1: <th1> .
Proof.
<pr_th1>
Qed.
Lemma lem1: <lem1> .
Proof.
<pr_lem1>
Qed.
Theorem th2: <th2> .
Proof.
<pr_th2>
Qed.
Finishing the skeleton

Skeleton for Mizar


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

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

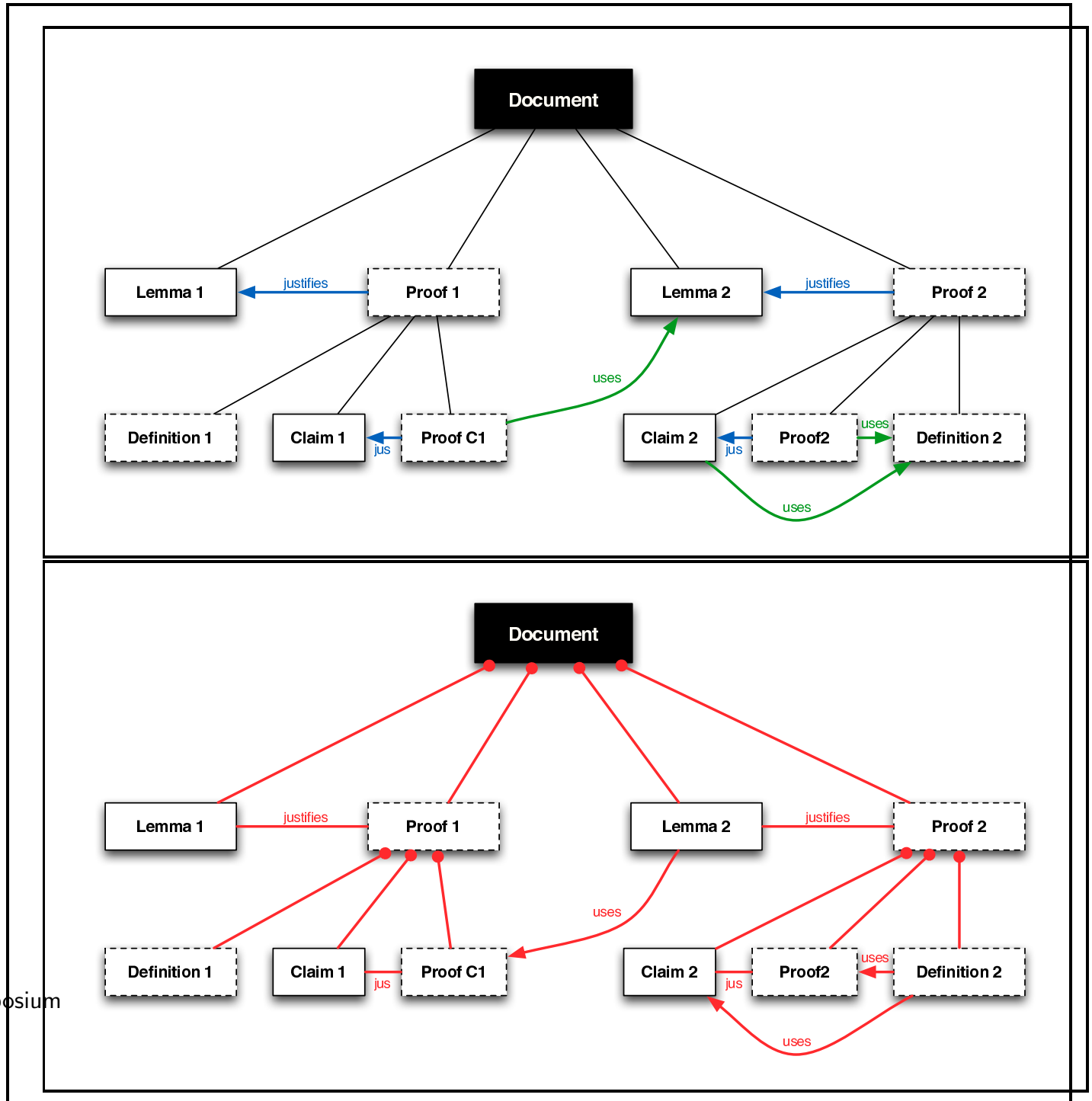


The generic algorithm for generating the proof skeleton (SGa, Zengler's thesis)

A vertex is ready to be processed iff:

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

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


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

The final order of the vertices is:

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

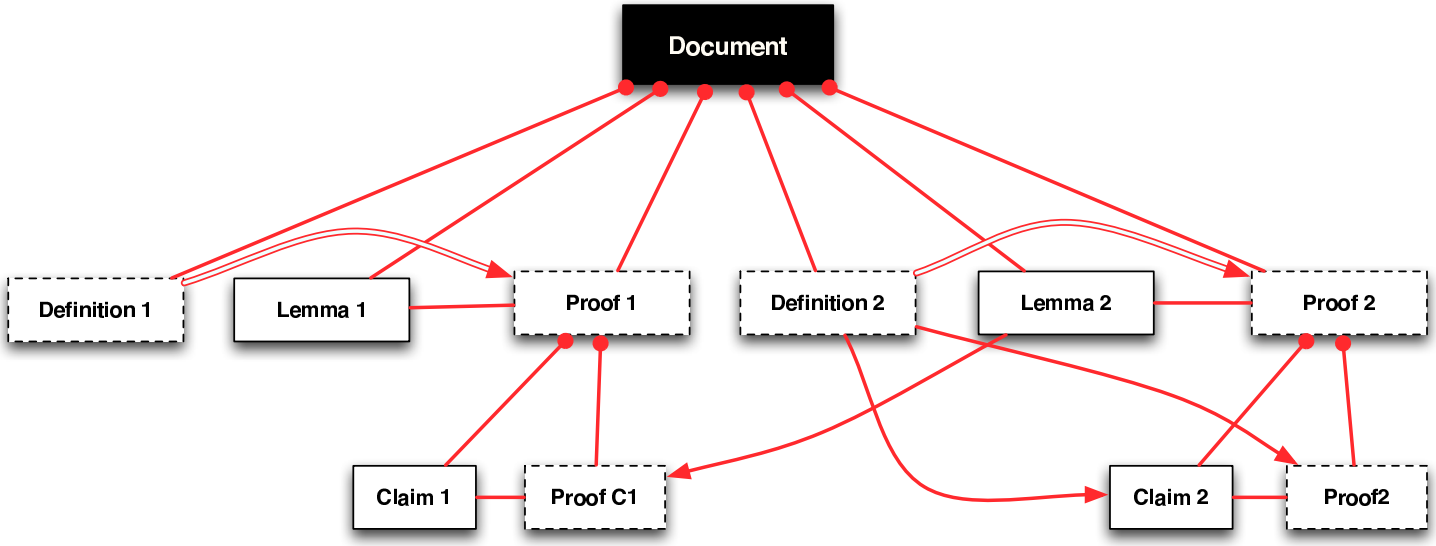


Figure 6: A flattened graph of the GoTO of figure 5 without nested definitions

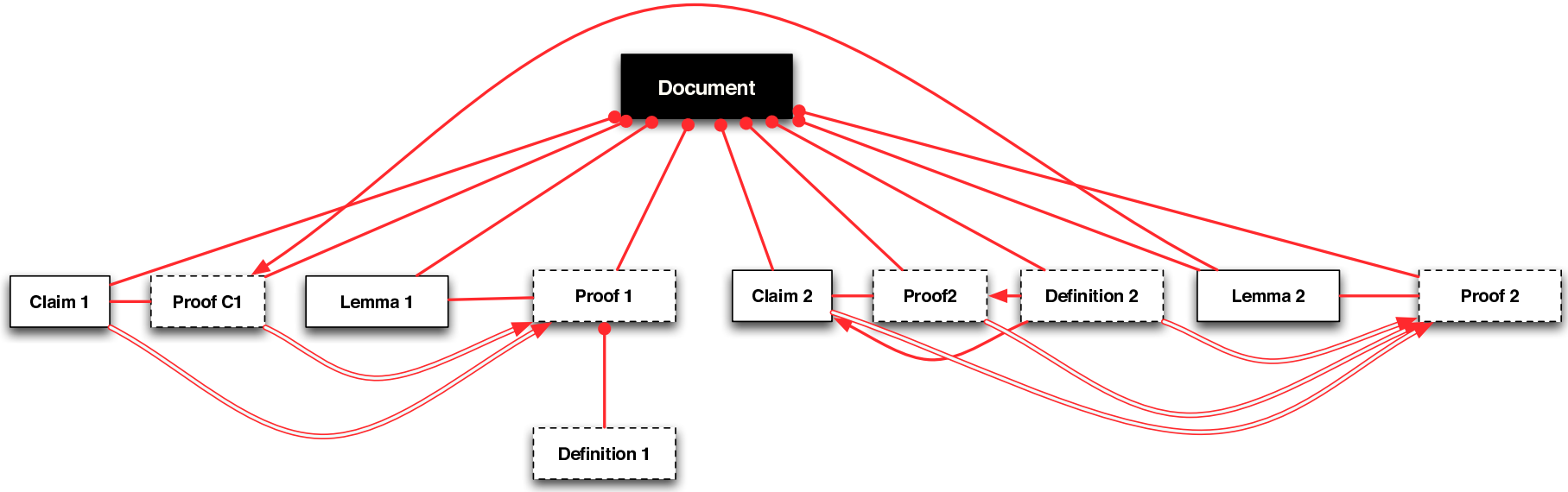
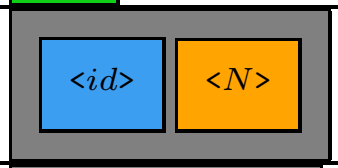
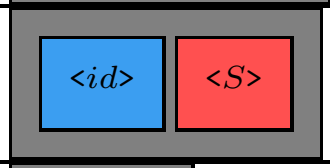
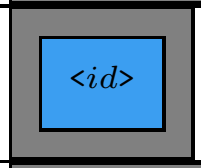
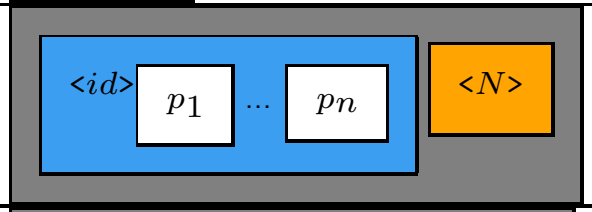
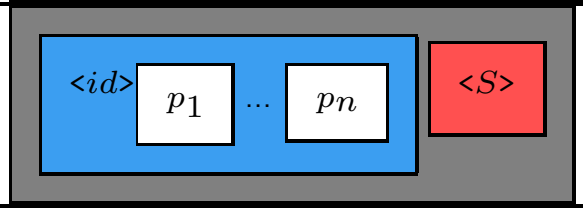


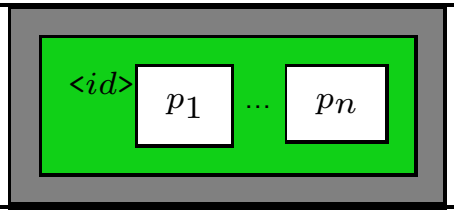
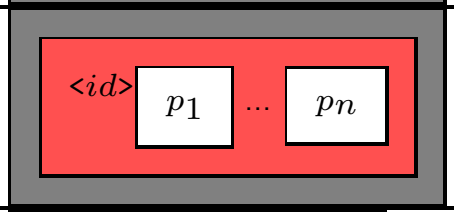
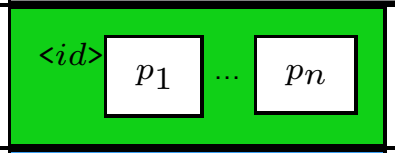
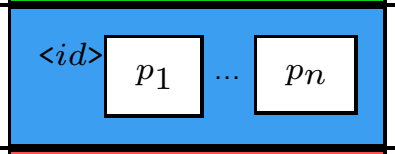
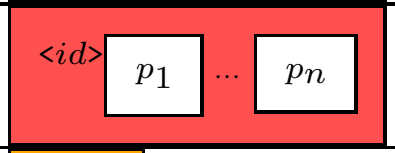
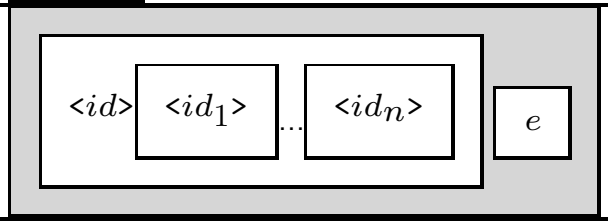
Figure 7: A flattened graph of the GoTO of figure 5 without nested claims

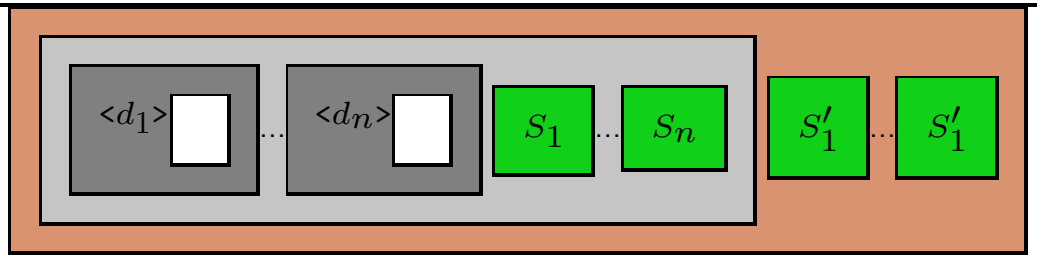
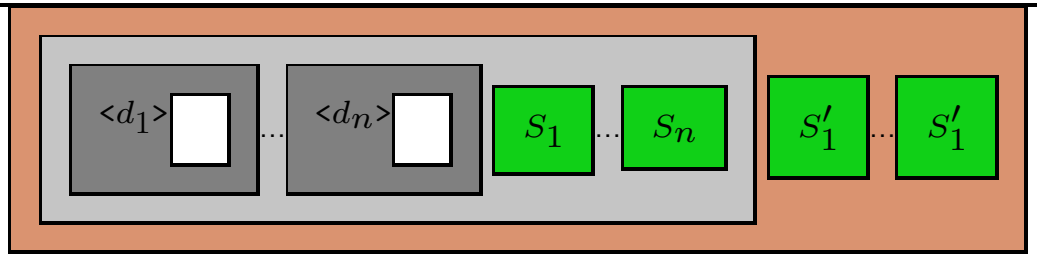
The Mizar and Coq rules for the dictionary

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

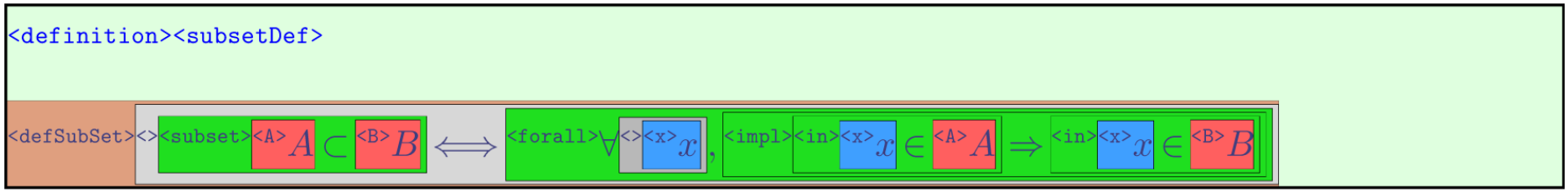
Rich skeletons for Coq

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

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

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

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



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

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

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

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

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

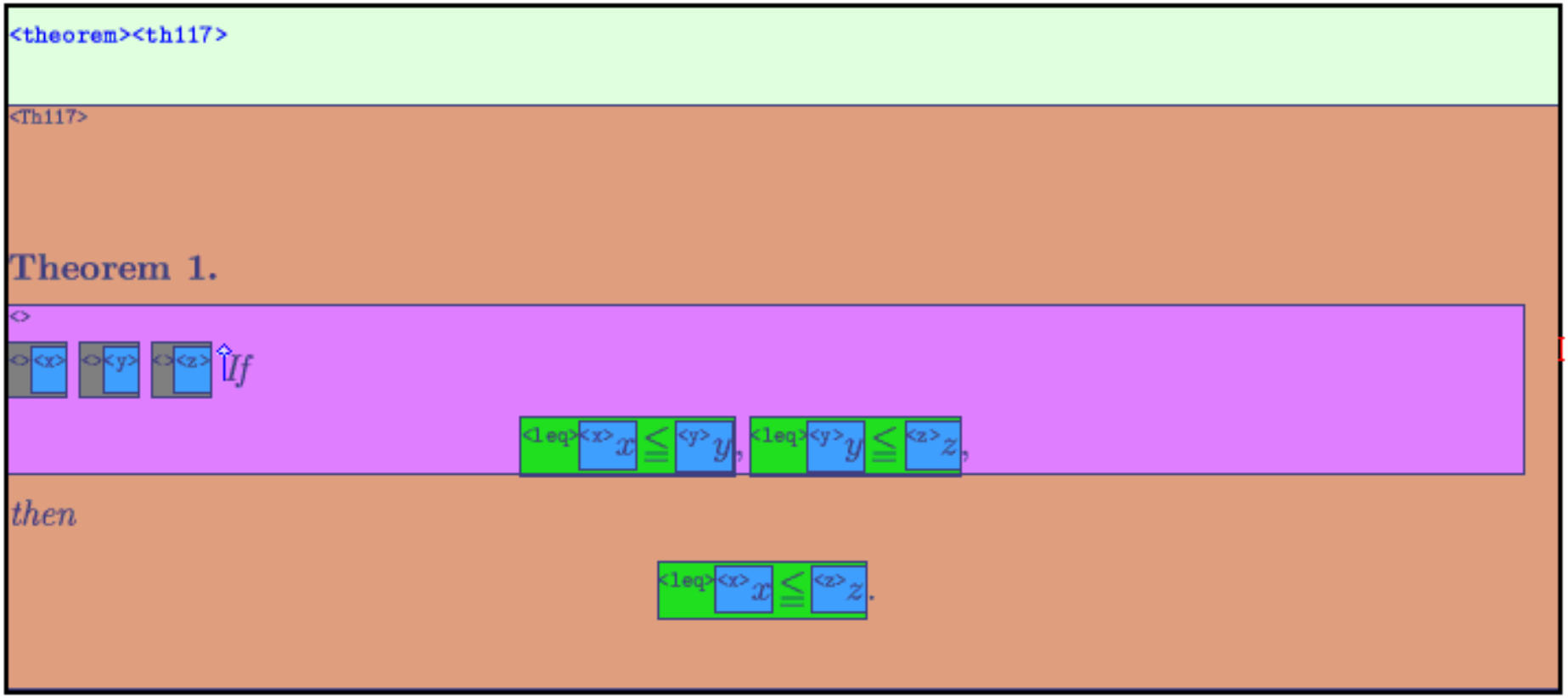
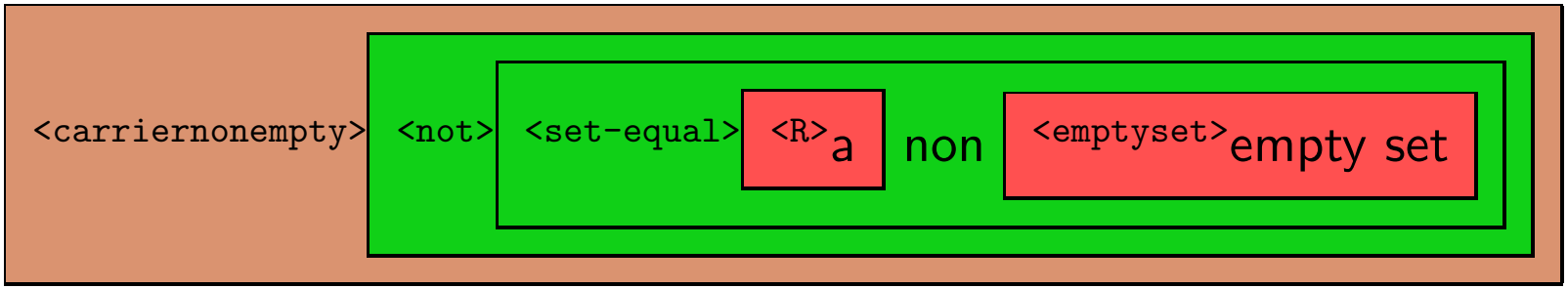


Figure 8: Theorem 17 of Landau's "Grundlagen der Analysis"

The automatic translation is:

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

Rich skeletons for Isabelle



The corresponding translation into Isabelle is:

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

An example of a full formalisation in Coq via MathLang

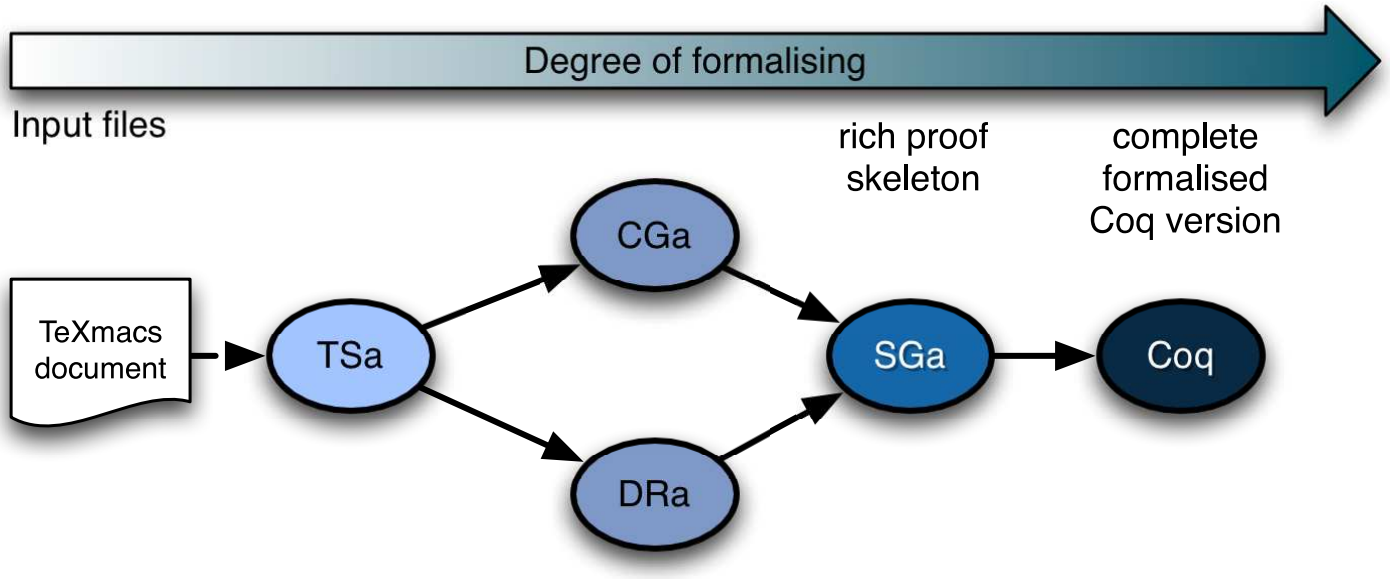


Figure 9: The path for processing the Landau chapter

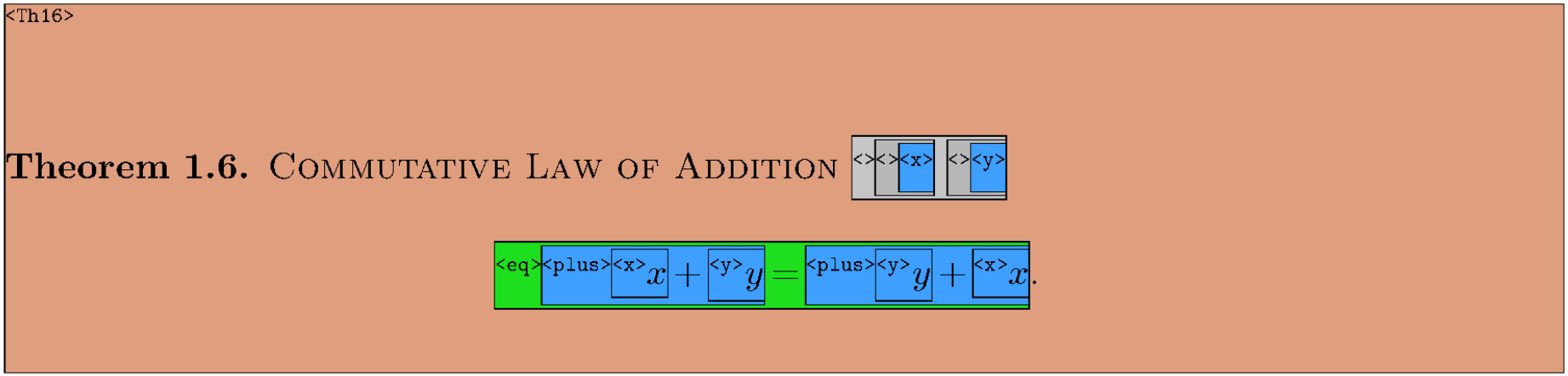


Figure 10: Simple theorem of the second section of Landau's first chapter

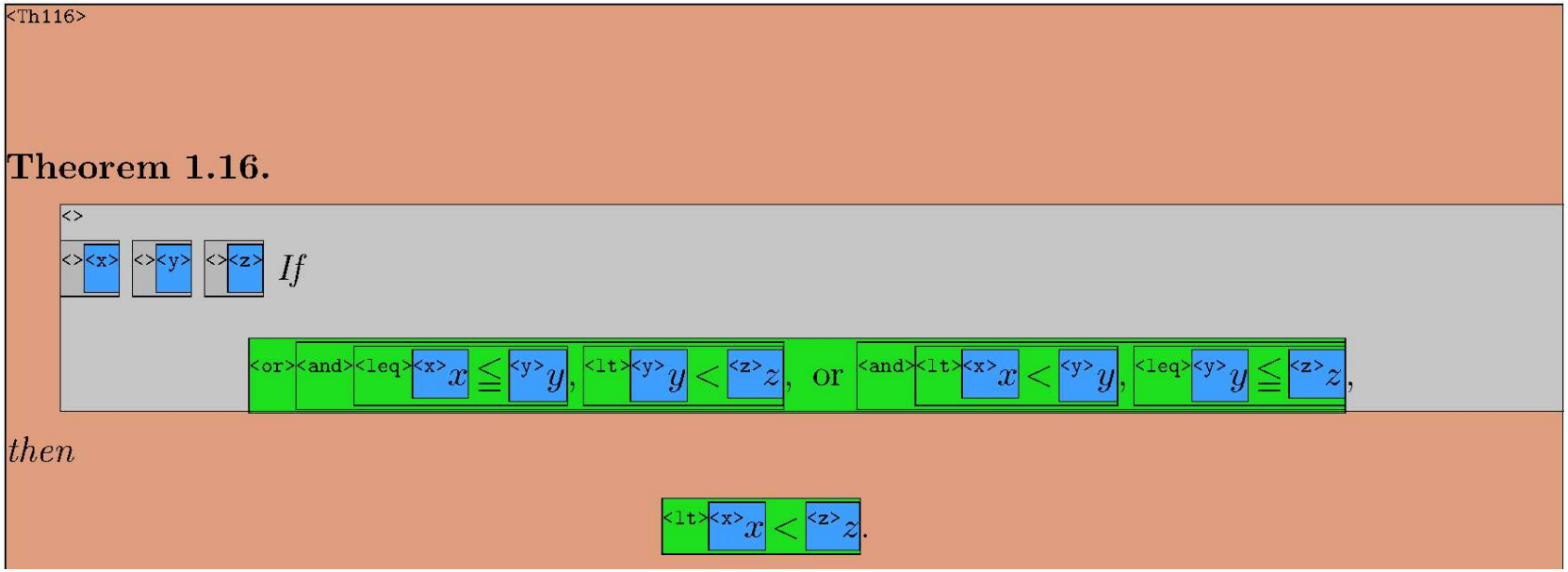
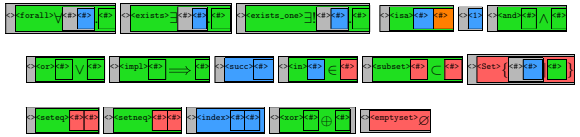


Figure 11: The annotated theorem 16 of the Landau's first chapter

Chapter 1

Natural Numbers



1.1 Axioms

We assume the following to be given:

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

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

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

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

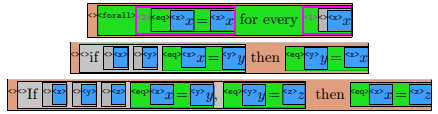
$$x = y$$

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

$$1x \neq y$$

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

Accordingly, the following are true on purely logical grounds:



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”.

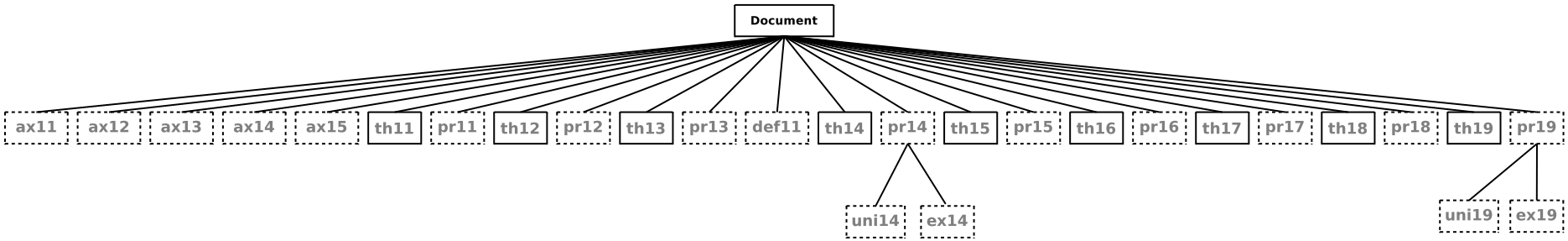


Figure 12: The DRa tree of sections 1 and 2 of chapter 1 of Landau’s book

- 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.

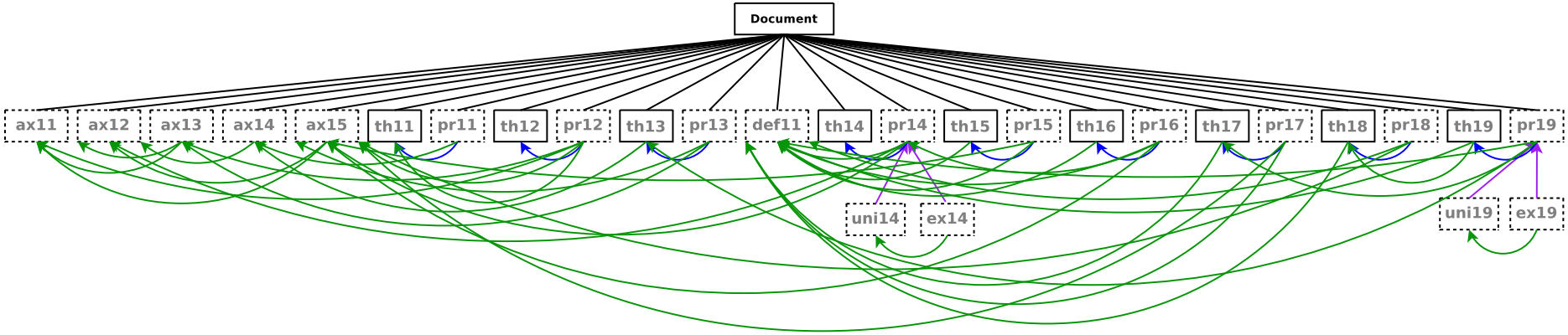
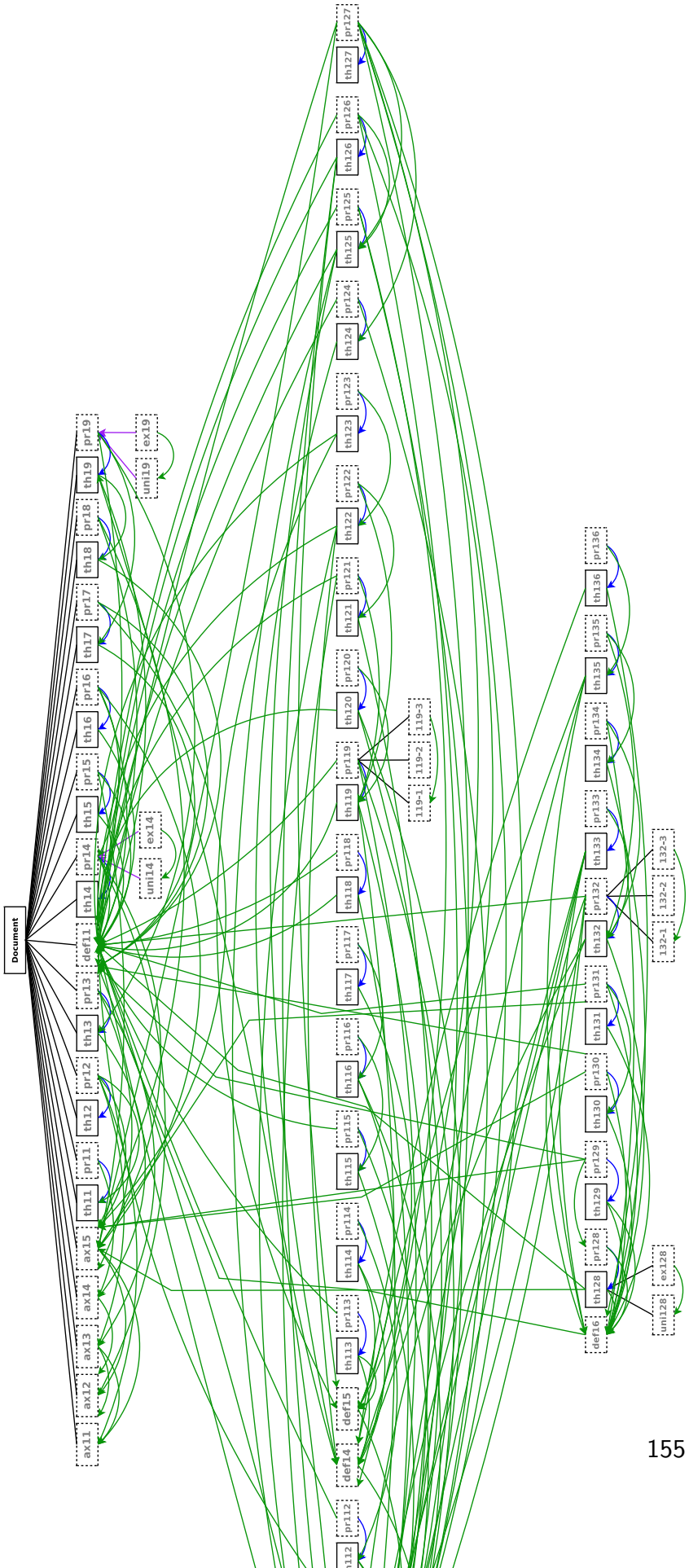
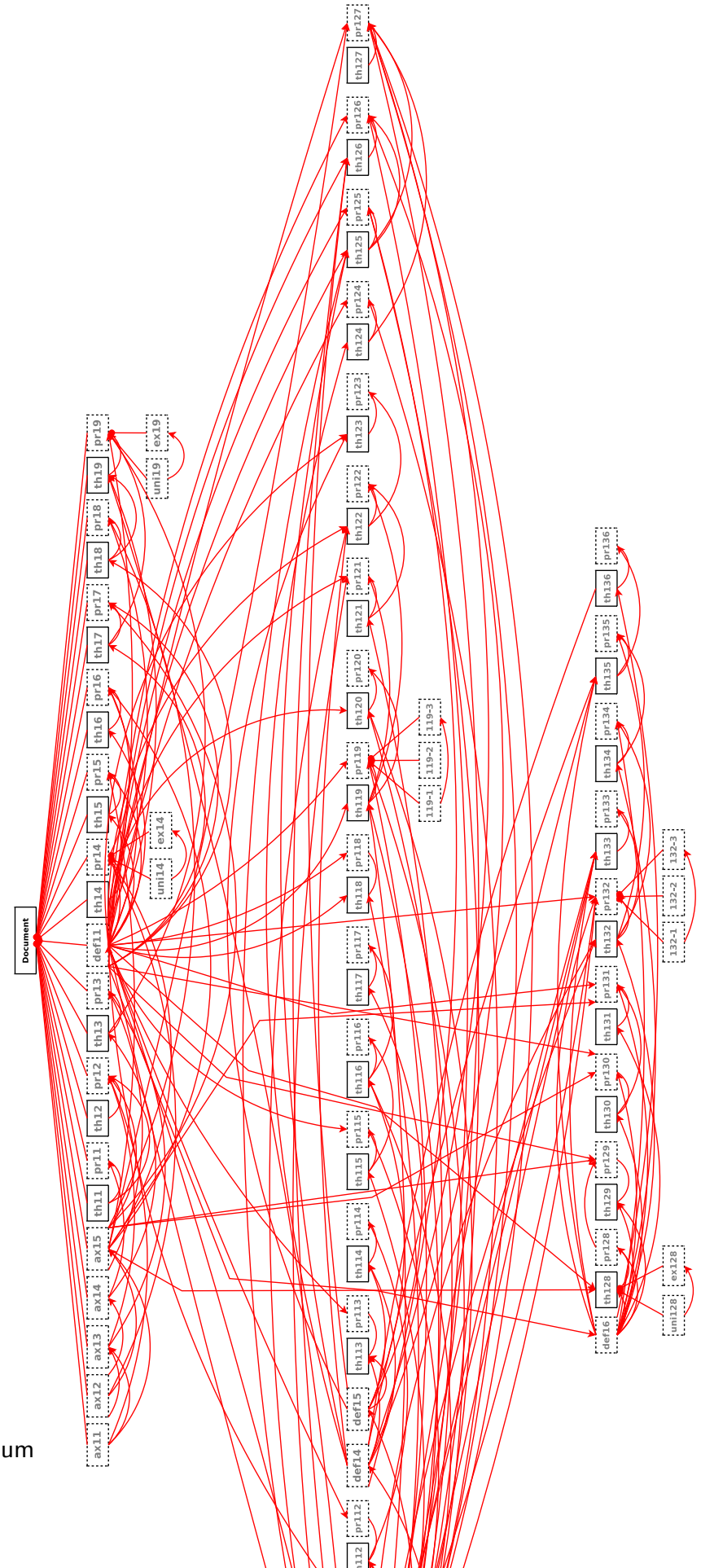


Figure 13: The DG of sections 1 and 2 of chapter 1 of Landau's book


Document
ax11
ax12
ax13
ax14
ax15
def11
def14
def15
def16
ex14
ex19
ex128
l19-1
l19-2
l19-3
l32-1
l32-2
l32-3
pr11
pr12
pr13
pr14
pr15
pr16
pr17
pr18
pr19
pr112
pr113
pr114
pr115
pr116
pr117
pr118
pr119
pr120
pr121
pr122
pr123
pr124
pr125
pr126
pr127
pr128
pr129
pr130
pr131
pr132
pr133
pr134
pr135
pr136
th11
th12
th13
th14
th15
th16
th17
th18
th19
th112
th113
th114
th115
th116
th117
th118
th119
th120
th121
th122
th123
th124
th125
th126
th127
th128
th129
th130
th131
th132
th133
th134
th135
th136
uni14
uni19
uni128

The GoTO of section 1 - 4


Document
def11
th11
pr11
th12
pr12
th13
pr13
th14
pr14
th15
pr15
th16
pr16
th17
pr17
th18
pr18
th19
pr19
ex19
uni19
ex14
uni14
ax11
ax12
ax13
ax14
ax15
def14
def15
th113
pr113
th114
pr114
th115
pr115
th116
pr116
th117
pr117
th118
pr118
th119
pr119
l19-1
l19-2
l19-3
th120
pr120
th121
pr121
th122
pr122
th123
pr123
th124
pr124
th125
pr125
th126
pr126
th127
pr127
def16
th128
pr128
th129
pr129
th130
pr130
th131
pr131
th132
pr132
l32-1
l32-2
l32-3
th133
pr133
th134
pr134
th135
pr135
th136
pr136
uni128
ex128

An extract of the automatically generated rich skeleton

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

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

Qed.

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

Qed.

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

Qed.
```

Completing the proofs in Coq

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

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

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

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

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

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

```
Proof.
intro x. elim x.

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

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

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

```
apply ax13.

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

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

```
intros n H.
```

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

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

```
apply th11.
```

```
1 subgoal
x : nats
n : nats
```

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

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

```
assumption.
Proof completed.
```

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

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

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

Some points to consider

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

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

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

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

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

- MathLang aims to support automated processing of mathematical knowledge.

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

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

What is important in all this are the good teachers, the good motivators and the makers of the next generation

Prof Ida has influenced generations of researchers and academics in Tunisia, Austria, Romania, Japan, etc.

He will always be on top of the education ladder protecting and leading the rest

And what a pleasure it is to know Prof. Ida and to work with him.

References

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.

Z.E.A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

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

N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968.

Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [Nederpelt et al., 1994], pages 73–100.

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.

N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the chuirch rosser theorem. In *Indagationes Math*, pages 381–392. 1972. Also in [Nederpelt et al., 1994].

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

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

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

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

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

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

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

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 and A. Ríos. A λ-calculus à la de bruijn with explicit substitutions. *Proceedings of Programming Languages Implementation and the Logic of Programs PLILP'95, Lecture Notes in Computer Science*, 982:45–62, 1995.

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. Revisiting the notion of function. *Logic and Algebraic programming*, 54:65–107, 2003a.

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

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

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

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

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

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.

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

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

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

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

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

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