From Principia to Formalised versus Computerised Maths

Fairouz Kamareddine
USEFUL LOGICS, TYPES, REWRITING and AUTOMATION
Heriot-Watt University
Edinburgh, UK

February 3, 2016

Pre-formalisation in the 19th century)

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

- 1821: Many of these controversies were solved by the work of Cauchy. E.g., he introduced *a precise definition of convergence* in his *Cours d'Analyse* [6].

- 1872: Due to the more *exact definition of real numbers* given by Dedekind [9], the rules for reasoning with real numbers became even more precise.

- 1895-1897: Cantor began formalizing *set theory* [4; 5] and made contributions to *number theory*.

- 1889: Peano formalized *arithmetic* [28], but did not treat logic or quantification.

Prehistory of Types (formal systems in 19th century)

- 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* [10], the first formalisation of logic giving logical concepts via symbols rather than natural language.

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

  (*Begriffsschrift*, Preface)

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* (as was the situation in analysis), but also a *proposition*, or a *function*.
- The *result of applying* a function to an argument did not have to be a number.
- Frege was aware of some typing rule that does not allow to substitute functions for object variables or objects for function variables:

  " Now just as functions are fundamentally different from objects, so also *functions whose arguments are and must be functions* are fundamentally different from *functions whose arguments are objects and cannot be anything else*. I call the latter *first-level*, the former *second-level*."

  (*Function and Concept*, pp. 26–27)

Prehistory of Types (Grundgesetze's functions)

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

- In Grundlagen der Arithmetik [11] he argued that mathematics can be seen as a branch of logic.
- In Grundgesetze der Arithmetik [12; 13] he described the elementary parts of arithmetics within an extension of the logical framework of *Begriffsschrift*.
- He did not *apply a function to itself*, but to its course-of-values.
  "the function $\Phi(x)$ has the same *course-of-values* as the function $\Psi(x)$" if:

  " $\Phi(x)$ and $\Psi(x)$ always have the same value for the same argument."
  (*Grundgesetze*, p. 7)
- Frege *excluded the paradox threats* by *forbidding self-application*, but due to his *treatment of courses-of-values* these threats were able to *enter his system through a back door*.
- In 1902, Russell wrote to Frege [32] *a paradox* in *Begriffsschrift*.

- (*Begriffsschrift* *does not suffer from a paradox*).

- The Russell Paradox can be derived in *Peano's system* as well, as well as on *Cantor's Set Theory*

- Logicians considered other paradoxes to be *out of the scope of logic*:
  The *Liar's Paradox* can be regarded as a problem of *linguistics*.
  The *paradoxes of Cantor and Burali-Forti* occurred in what was considered in those days a *highly questionable* part of mathematics: Cantor's Set Theory.

- The Russell Paradox, however, was *a paradox that could be formulated in all* the systems that were presented at the end of the 19th century (except for Frege's *Begriffsschrift*). A solution to it had to be found.

- Russell *avoided the paradoxes* by *avoiding all possible self-references*. He strictly implemented using *types* the *"vicious circle principle"*:

- "*Whatever involves* all *of a collection* must not be one *of the collection*."
  (Mathematical logic as based on the theory of types)

- Russell [33] *1903* gives the first type theory: the *Ramified Type Theory* (RTT).

- RTT is used in Russell and Whitehead's Principia Mathematica [36] 1910–1912.
  - The main part of the *Principia* is devoted to the development of logic and mathematics using the legal pfs of the ramified type theory.
  - *ramification*/division of simple types into orders make RTT not easy to use.
  - **(Equality)** $\mathtt{x} =_{\mathrm{L}} \mathtt{y} \stackrel{\mathrm{def}}{\leftrightarrow} \forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$.
    In order to express this general notion in RTT, we have to incorporate *all* pfs $\forall \mathtt{z} : (0^0)^n[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$ for $n > 1$, and this cannot be expressed in one pf.
  - Not possible to give a constructive proof of the theorem of the least upper bound within a ramified type theory.
  - Russell and Whitehead tried to solve problems with the axiom of reducibility: *For each formula f, there is a formula g with a* predicative *type such that f and g are (logically) equivalent.*
  - The validity of the Axiom of Reducibility has been questioned from the moment it was introduced.

- Weyl made an effort to develop Analysis within RTT (without Reducibility).

- Various parts of Maths can be developed in RTT (without Reducibility), but attitude was that RTT was too restrictive.

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

- Church's *simply typed λ-calculus* $\lambda \rightarrow$ [7] 1940 = λ-calculus + STT.

- *Unsatisfactory* hierarchies of types/orders in RTT and STT.

- The *notion of function adopted in the λ-calculus* is *unsatisfactory* [22].
  Frege's functions $\neq$ Principia's functions $\neq$ *λ-calculus* functions.

- Hence, birth of *different systems of functions and types*, each with *different functional power*.

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 and computation in 20th century.

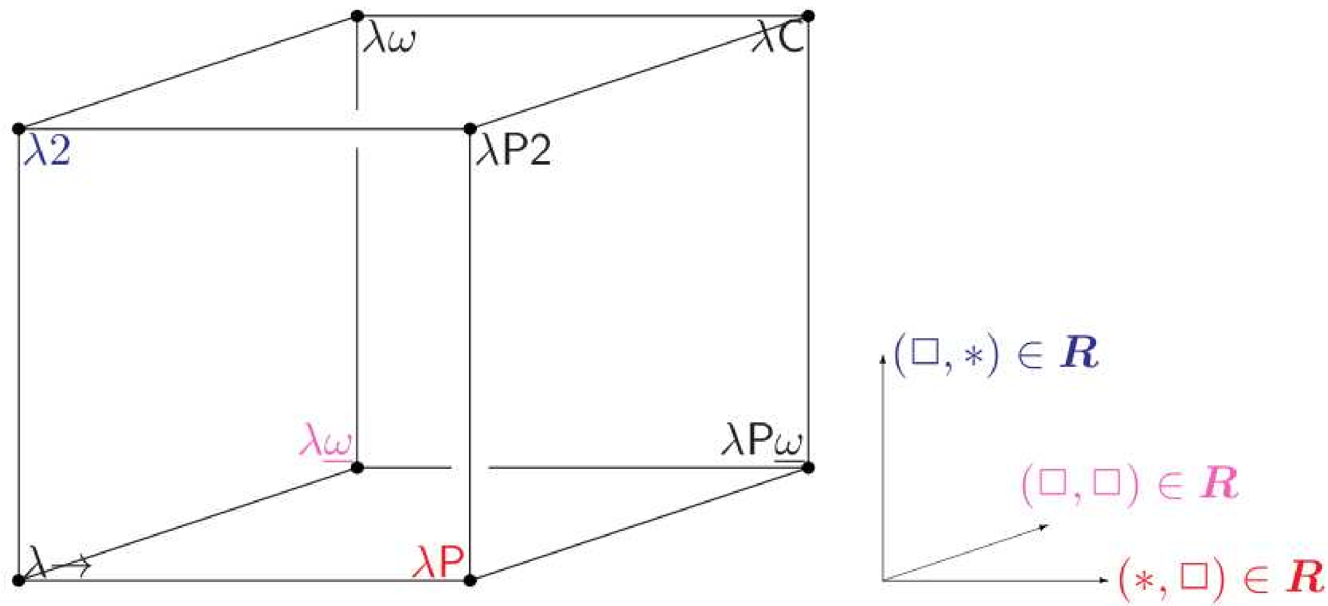
The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$

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

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\to$	$(*, *)$				λ^τ	[7; 1; 20]
$\lambda 2$	$(*, *)$	$(\Box, *)$			F	[15; 31]
λP	$(*, *)$		$(*, \Box)$		AUT-QE, LF	[3; 16]
$\lambda\underline{\omega}$	$(*, *)$			$(\Box, \Box)$	POLYREC	[30]
λP2	$(*, *)$	$(\Box, *)$	$(*, \Box)$			[26]
$\lambda\omega$	$(*, *)$	$(\Box, *)$		$(\Box, \Box)$	Fω	[15]
λP$\underline{\omega}$	$(*, *)$		$(*, \Box)$	$(\Box, \Box)$		
λC	$(*, *)$	$(\Box, *)$	$(*, \Box)$	$(\Box, \Box)$	CC	[8]

The Barendregt Cube



The β-cube: $\rightarrow_\beta$ + conv$_\beta$ + app$_\Pi$

(axiom)	$\langle\rangle \vdash * : \square$
(start)	$\dfrac{\Gamma \vdash A : s \quad x \notin \mathrm{DOM}\,(\Gamma)}{\Gamma, x{:}A \vdash x : A}$
(weak)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad x \notin \mathrm{DOM}\,(\Gamma)}{\Gamma, x{:}C \vdash A : B}$
(Π)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$
(λ)	$\dfrac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$
(conv$_\beta$)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$
(app$_\Pi$)	$\dfrac{\Gamma \vdash F : \Pi_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$

6 desirable properties of a type system with reduction r

- *Types are correct (TC)*
  If $\Gamma \vdash A : B$ then $B \equiv \Box$ or $\Gamma \vdash B : s$ for$s \in \{*, \Box\}$.

- *Subject reduction (SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_r A'$ then $\Gamma \vdash A' : B$.

- *Preservation of types (PT)* If $\Gamma \vdash A : B$ and $B \twoheadrightarrow_r B'$ then $\Gamma \vdash A : B'$.

- *Strong Normalisation (SN)* If $\Gamma \vdash A : B$ then $\mathsf{SN}_{\to_r}(A)$ and $\mathsf{SN}_{\to_r}(B)$.

- *Subterms are typable (STT)* If A is $\vdash$-legal and if C is a sub-term of A then C is $\vdash$-legal.

- *Unicity of types*
  - *(UT1)* If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $\Gamma \vdash A_1 =_r A_2$, then $\Gamma \vdash B_1 =_r B_2$.
  - *(UT2)* If $\Gamma \vdash B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s$.

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

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

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

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

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

Which type systems can be extended with features needed for mathematics?

- There was a surge for explicit substitutions. Lots of work, lots of untied results, very scattered picture.
- There was a surge for different notions of reductions. Lots of work, lots of untied results, very scattered picture.
- Explicit contexts, intersection types, Church versus Curry typing, etc. Again, lots of work, lots of untied results, very scattered picture.
- And for each small extenion, an entire machinery needs to be built and proved and many questions remain unsolved.

From the point of vue of ML

- ML is not based on all of system F (2nd order polymorphic λ-calculus).
- It was not known then if type checking and type finding are decidable in F.
- ML is based on a fragment of system F for which it was known that type checking and type finding are decidable.
- 23 years later after the design of ML, Wells showed that type checking and type finding in system F are undecidable.
- ML has polymorphism but not all the polymorphic power of system F.
- The question is, what system of functions and types does ML use?
- *A clean answer can be given when we re-incorporate the low-level function notion used by Frege and Russell (and de Bruijn) and dismissed by Church.*

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

- 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 $\lambda 2$:
  $(\lambda \mathit{id} : (\Pi\alpha{:}*.\ \alpha \to \alpha).$
  $\quad (\lambda y{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ y(\beta \to \beta)(y\,\beta))$
  $\quad (\lambda\alpha{:}*.\ \mathit{id}(\alpha \to \alpha)(\mathit{id}\,\alpha)))$
  $(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

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

- *ML's type system is none of those of the eight systems of the Cube*.
  [21] places the *type system of ML* (*between $\lambda 2 + \lambda \underline{\omega}$*).

LF

- *LF* [16] is often described as λP of the Barendregt Cube.
  However, *Use of Π-formation rule $(*, \Box)$ is restricted in LF* [14].

- 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.

- [21] precisely locate *LF* (*between $\lambda{\rightarrow}$ and λP*).

Parameters: What and Why

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

- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. *first-order with parameters* versus *second-order without* [25].

- 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 [2] 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 [21]

- 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 $S)$.

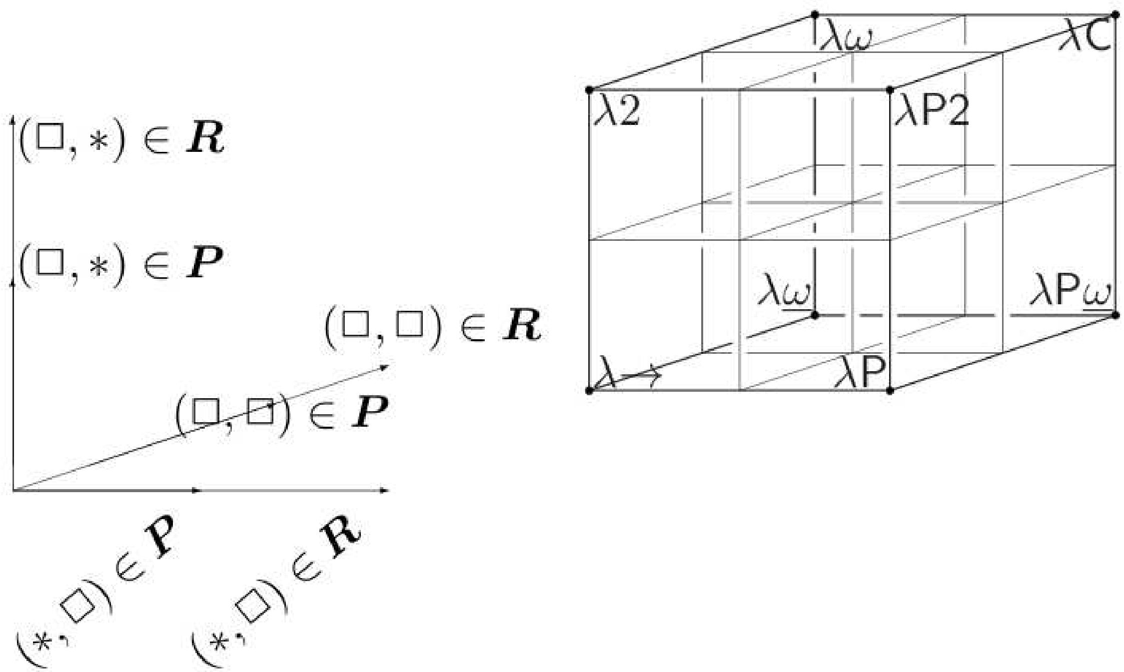
- *(Subject Reduction SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$

- *(Strong Normalisation)* For all $\vdash$-legal terms M, we have $\mathrm{SN}_{\twoheadrightarrow_\beta}(M)$.

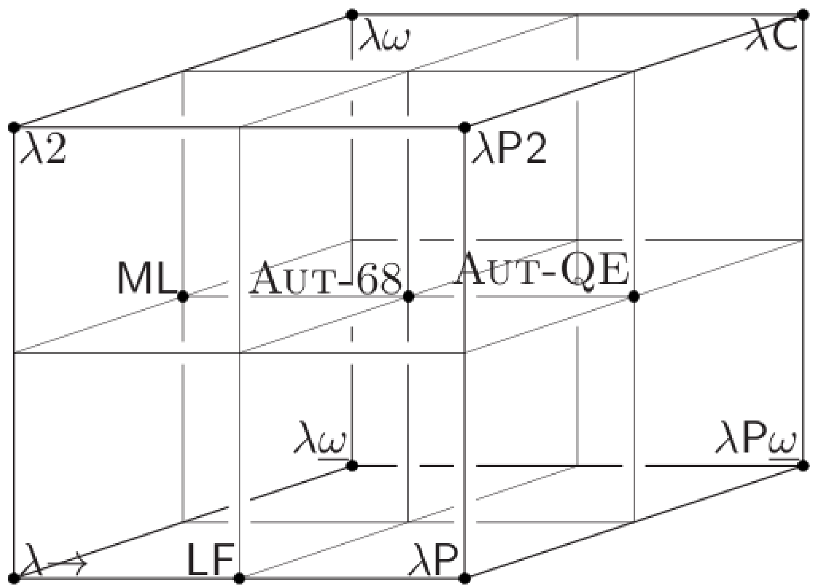
- Other properties such as *Uniqueness of types* and *typability of subterms* hold.

- $\lambda \mathbf{R}\boldsymbol{P}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.

The refined Barendregt Cube



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

- 

Logicians versus mathematicians: induction over numbers

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

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}{\to}*).p0{\to}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm){\to}\Pi n{:}\mathbb{N}.pn \qquad (1)$$

- Mathematician uses $\texttt{ind}$ 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):

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

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

Logicians versus mathematicians: induction over numbers

- *Mathematician 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.

- 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* [25].

- 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.

Common features of modern types and functions

- Write $A \rightarrow A$ as $\Pi_{y:A}.A$ when y not free in A.
- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $\Pi_{y:A}.A$*, i.e.* $A \rightarrow A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \rightarrow A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \rightarrow A)\mathbb{N} = (A \rightarrow A)[A := \mathbb{N}]$ or $\mathbb{N} \rightarrow \mathbb{N}$.
- More clearly

Term	$\lambda_{y:A}.y$	$\lambda_{A:*}.\lambda_{y:A}.y$	$(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$
Type	$\Pi_{y:A}.A$	$\Pi_{A:*}.\Pi_{y:A}.A$	$(\Pi_{A:*}.\Pi_{y:A}.A)\mathbb{N}$
shorthand	$A \rightarrow A$	$\Pi_{A:*}.A \rightarrow A$	$(\Pi_{A:*}.A \rightarrow A)\mathbb{N}$

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

The π-cube: $R_\pi = R_\beta \backslash$ (conv$_\beta$) $\cup$ (conv$_{\beta\Pi}$), $\rightarrow_{\beta\Pi}$

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

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$)

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

Lemma: $\Gamma \vdash_\beta A : B$ iff $\Gamma \vdash_\pi A : B$

Lemma: The β-cube and the π-cube satisfy the six properties that are desirable for type systems.

The π_i-cube: $R_{\pi_i} = R_\pi \setminus$ (app$_\Pi$) $\cup$ (i-app$_\Pi$), $\to_{\beta\Pi}$

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

(axiom) (start) (weak) (Π) (λ)

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

(i-app$_\Pi$) $$\frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

Lemma:

- If $\Gamma \vdash_\beta A : B$ then $\Gamma \vdash_{\pi_i} A : B$.
- If $\Gamma \vdash_{\pi_i} A : B$ then $\Gamma \vdash_\beta A : [B]_\Pi$
  where $[B]_\Pi$ is the Π-normal form of B.

The π_i-cube

- The π_i-cube loses three of its six properties
  Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_i} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.
  - *We do not have TC* $(\Pi_{y:z}.z)x \not\equiv \Box$ and $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$.
  - *We do not have SR* $(\lambda_{y:z}.y)x \to_{\beta\Pi} x$ but $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$.
  - *We do not have UT2* $\vdash_{\pi_i} * : \Box$, $* =_{\beta\Pi} (\Pi_{z:*}.*)\alpha$, $\alpha : * \vdash_{\pi_i} (\lambda_{z:*}.*)\alpha : (\Pi_{z:*}.*)\alpha$ and $\not\vdash_{\pi_i} (\Pi_{z:*}.*)\alpha : \Box$

- But we have:
  - *We have UT1*
  - *We have STT*
  - *We have PT*
  - *We have SN*
  - *We have a weak form of TC* If $\Gamma \vdash_{\pi_i} A : B$ and *B does not have a Π-redex* then either $B \equiv \Box$ or $\Gamma \vdash_{\pi_i} B : s$.
  - *We have a weak form of SR* If $\Gamma \vdash_{\pi_i} A : B$, *$B$ is not a Π-redex* and $A \twoheadrightarrow_{\beta\Pi} A'$ then $\Gamma \vdash_{\pi_i} A' : B$.

The problem can be solved by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function)

(start-a)	$\dfrac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A}$	$x \notin \textsc{dom}\,(\Gamma)$
(weak-a)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B}$	$x \notin \textsc{dom}\,(\Gamma)$

Figure 1: Basic abbreviation rules BA

$(\mathbf{let}_{\backslash})$	$\dfrac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\backslash_{x:A}.C)B : D[x := B]}$

Figure 2: $(\text{let}_{\backslash})$ where $\backslash = \lambda$ or $\backslash = \Pi$

The β_a-cube: $R_{\beta_a} = R_\beta$ + BA + let$_\beta$, $\to_\beta$

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$) (conv$_\beta$)

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

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

(let$_\beta$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\lambda_{x:A}.C)B : D[x := B]}$$

Lemma: The β_a-cube satisfies the desirable properties except for typability of subterms.
If A is $\vdash$-legal and B is a subterm of A such that every bachelor $\lambda_{x:D}$ in B is also bachelor in A, then B is $\vdash$-legal.

The π_a-cube: $R_{\pi_a} = R_\pi$ + BA + let$_\beta$ + let$_\Pi$, $\rightarrow_{\beta\Pi}$

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$) (conv$_{\beta\Pi}$)

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

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

(let$_\beta$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\lambda_{x:A}.C)B : D[x := B]}$$

(let$_\Pi$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\Pi_{x:A}.C)B : D[x := B]}$$

Lemma: The π_a-cube satisfies the same properties as the β_a.

The π_{ai}-cube: $R_{\pi_{ai}} = R_{\pi_a} \setminus$ app$_\Pi$ + i-app$_\Pi$, $\rightarrow_{\beta\Pi}$

Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_{ai}} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.

- *We NOW have TC* although $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$, we have $\Gamma \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : s$

  By (weak-a) $z : *, x : z, y = x : z \vdash_{\pi_{ai}} z : *$.

  Hence by (let$_\Pi$) $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *[y := x] \equiv *$.

- *We NOW have SR* $(\lambda_{y:z}.y)x \rightarrow_{\beta\Pi} x$.
  Although $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$, we have $\Gamma \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$

  Since $z : *, x : z \vdash_{\pi_{ai}} x : z$, and $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *$ and
  $z : *, x : z \Vdash z =_{\beta\Pi} (\Pi_{y:z}.z)x$, we use (conv$_{\beta\Pi}$) to get:
  $z : *, x : z \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$.

Identifying λ and Π (see [23])

- In the cube, the syntax for terms (functions) and types was intermixed with the only distinction being λ- versus Π-abstraction.

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

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

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

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

- The type formation rule becomes

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

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

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

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

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

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

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

Consequences of unifying λ and Π

- A term can have many distinct types. E.g., in λP we have:

  $$\alpha : * \vdash_\beta (\lambda x{:}\alpha.\alpha) : (\Pi x{:}\alpha.*) \qquad \text{and} \qquad \alpha : * \vdash_\beta (\Pi x{:}\alpha.\alpha) : *$$

  which, when we give up the difference between λ and Π, result in:

  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : [x{:}\alpha]\,*$ and
  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : *$

- More generally, in AUT-QE we have the dervived rule:

$$\frac{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \quad 0 \leq m \leq n \qquad (3)$$

This derived rule (3) has the following equivalent derived rule in λP (and hence in the higher systmes like $\lambda P\omega$):

$$\frac{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_n{:}A_n.* \qquad 0 \leq m \leq n}{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_m{:}A_m.\Pi x_{m+1}{:}A_{m+1}.\cdots\Pi x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_m{:}A_m.*}$$

However, $\mathrm{A{\scriptsize UT}\text{-}QE}$ goes further and generalises (3) to a rule of *type inclusion*:

$$\frac{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \quad 0 \leq m \leq n \qquad (Q)$$

The β_Q-cube $=$ β-cube $+$ (Q$_\beta$)

$$(\mathsf{Q}_\beta) \qquad \frac{\Gamma \vdash \lambda^{i:1..k}_{x_i:A_i}.A : \Pi^{i:1..n}_{x_i:A_i}.*}{\Gamma \vdash \lambda^{i:1..m}_{x_i:A_i}.\Pi^{i:m+1..k}_{x_i:A_i}A : \Pi^{i:1..m}_{x_i:A_i}.*} \quad 0 \leq m \leq n,\ A \not\equiv \lambda_{x:B}.C$$

- *Lemma:*
  - *The β_Q-cube enjoys all the properties of the cube except the unicity of types.*
  - *Rule Q$_\beta$ and rule $(s, \Box)$ for $s \in \{*, \Box\}$ imply rule $(s, *)$.*
    This means that the type systems $\lambda_{\mathsf{Q}}\underline{\omega}$ and $\lambda_{\mathsf{Q}}\omega$ are equal, and that $\lambda_{\mathsf{Q}}P\underline{\omega}$ and $\lambda_{\mathsf{Q}}P\omega$ are equal as well.

- Unicity of types fails for the β_Q-cube. Take: $A : *, x : \Pi_{y:A}.* \vdash x : \Pi_{y:A}.*$ and hence by Q$_\beta$, $A : *, x : \Pi_{y:A}.* \vdash x : *$.

Cubes								
β	$\rightarrow_\beta$	BT	conv_β	app				
π	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	app				
β_a	$\rightarrow_\beta$	BT	conv_β	app	BA	let_λ		
π_a	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	app	BA	let_λ	let_Π	
π_i	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	i-app				
π_{ai}	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	i-app	BA	let_λ	let_Π	
β_Q	$\rightarrow_\beta$	BT	conv_β	app				Q
π_{iQ}	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	i-app				Q
β_{aQ}	$\rightarrow_\beta$	BT	conv_β	app	BA	let_λ		Q
π_{aiQ}	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	i-app	BA	let_λ	let_Π	Q
π_Q	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	app				Q
π_{aQ}	$\rightarrow_{\beta\Pi}$	BT	$\text{conv}_{\beta\Pi}$	app	BA	let_λ	let_Π	Q
$c\pi$	$\rightarrow_{\beta\Pi}$	BT_c		appc				
$c\pi_a$	$\rightarrow_{\beta\Pi}$	BT_c		appc	BA_c	letc_λ	letc_Π	
$c\pi_Q$	$\rightarrow_{\beta\Pi}$	BT_c		appc				Qc
$c\pi_{aQ}$	$\rightarrow_{\beta\Pi}$	BT_c		appc	BA_c	letc_λ	letc_Π	Qc

Figure 3: Canonical and Non Canonical Type Systems

Type and function systems using de Bruijn indices and/or different notations

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

- Also, PTSs with de Bruijn indices.
- Also, PTSs with Curry style typing.
- PTSs with intersection types.

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, Theorema, etc.) can be used.

I will briefly describe our experience at developing a system named Mathlang which aimed to bridge the latter 3 levels.

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.

draft documents	✓	
public documents	✓	
computations and proofs	✗	

LaTeX example

```
\begin{theorem}[Commutative Law of Addition]\label{theorem:6}
  $$x+y=y+x.$$
\end {theorem}
\begin{proof}
  Fix $y$, and $\mathfrak{M}$ be the set of all $x$ for which
  the assertion holds.
  \begin{enumerate}
  \item We have $$y+1=y',$$
    and furthermore, by the construction in
    the proof of Theorem~\ref{theorem:4}, $$1+y=y',$$
    so that $$1+y=y+1$$
    and $1$ belongs to $\mathfrak{M}$.
  \item If $x$ belongs to $\mathfrak{M}$, then $$x+y=y+x,$$
    Therefore $$(x+y)'=(y+x)'=y+x'.$$
    By the construction in the proof of
    Theorem~\ref{theorem:4}, we have $$x'+y=(x+y)',$$
    hence $$x'+y=y+x',$$
    so that $x'$ belongs to $\mathfrak{M}$.
  \end{enumerate}
  The assertion therefore holds for all $x$.
\end{proof}
```

Full formalization difficulties: choices

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

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

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

Full formalization difficulties: informality

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

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

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

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

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

Coq example

draft documents	✗
public documents	✗
computations and proofs	✓

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

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

Proof.

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

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

Qed.
```

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

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

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

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

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

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

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

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

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

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

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

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

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

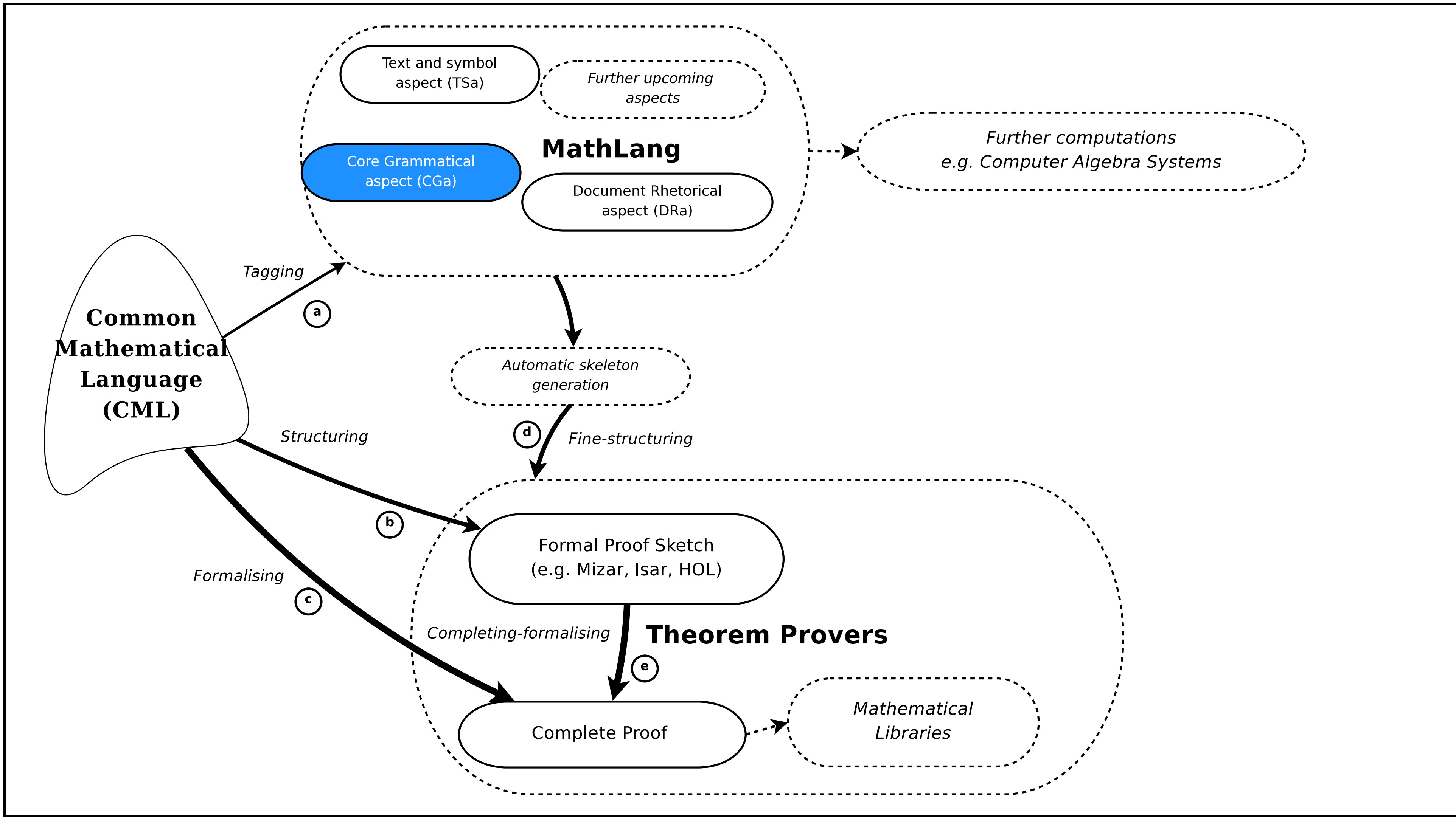
- We have been developing ways to weave natural language text into Mathlang.

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

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

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

Example of a MathLang Path



What is CGa?

- 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.

Weak Type Theory

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

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

Categories of syntax of WTT

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

Main categories of syntax of WTT

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

Derivation rules

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

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

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

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

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

Examples of derivation rules

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

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

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

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

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

Properties of WTT

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

Definition unfolding

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

CGa Weak Type Checking

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

y and x are natural numbers ,

if x belongs to $\mathfrak{M}$

then $x + y = y + x$

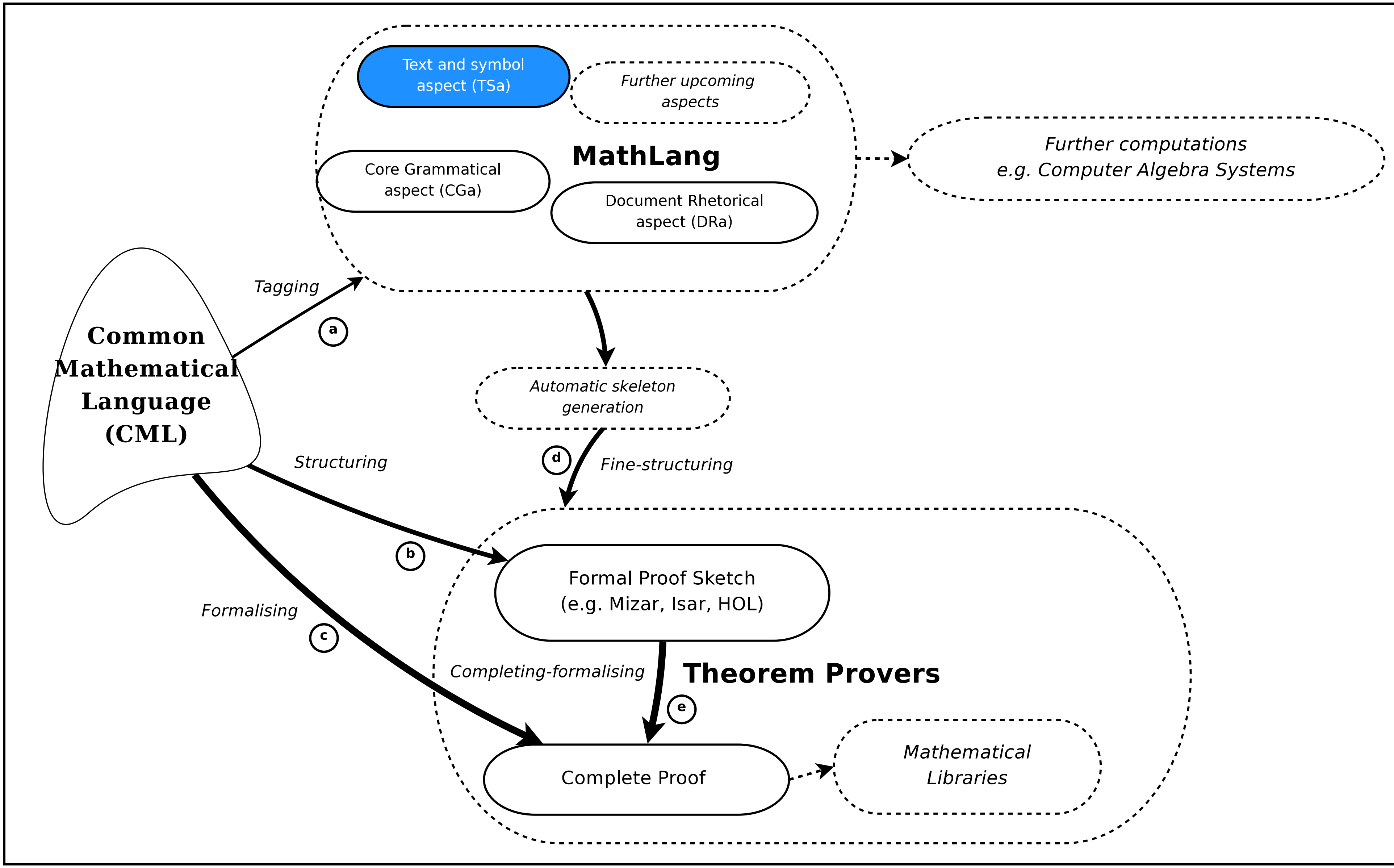
CGa Weak Type checking detects grammatical errors

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

$\boxed{\boxed{y}$ and $\boxed{x}$ are $\boxed{\text{natural numbers}}}$,

if $\boxed{\boxed{x}$ belongs to $\boxed{\mathfrak{M}}}$

then $\boxed{x + y}$ $\Leftarrow$ **error**


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

What is TSa?

- 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

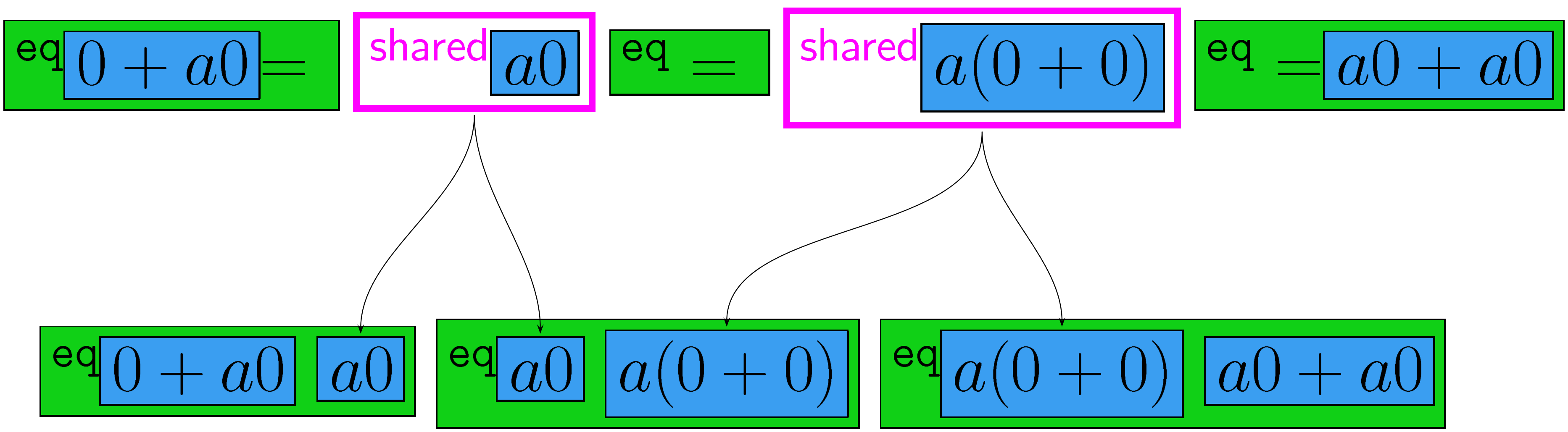
- TSa can flexibly represent natural language mathematics.

- 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.

Rewrite rules enable natural language representation

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



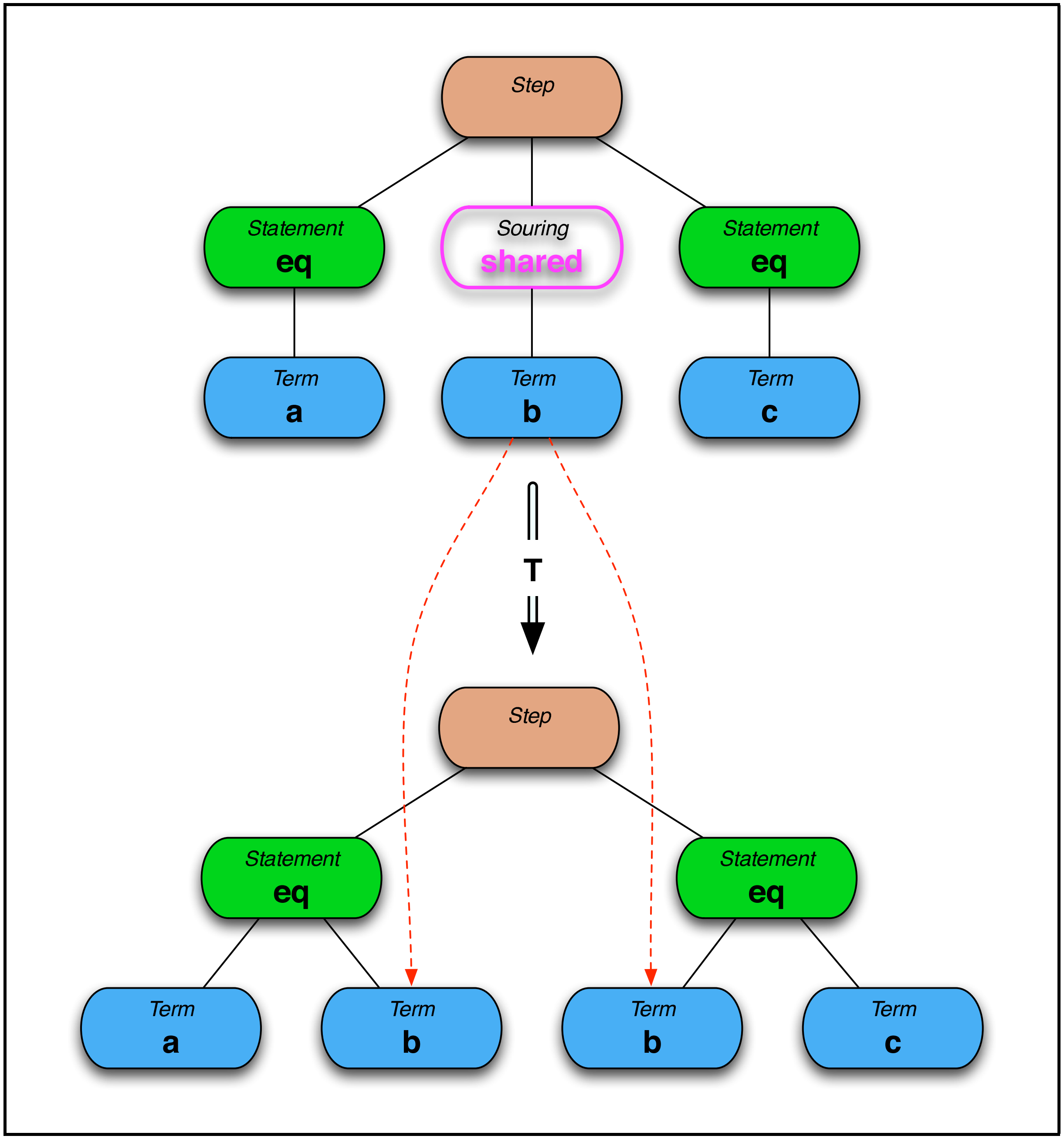
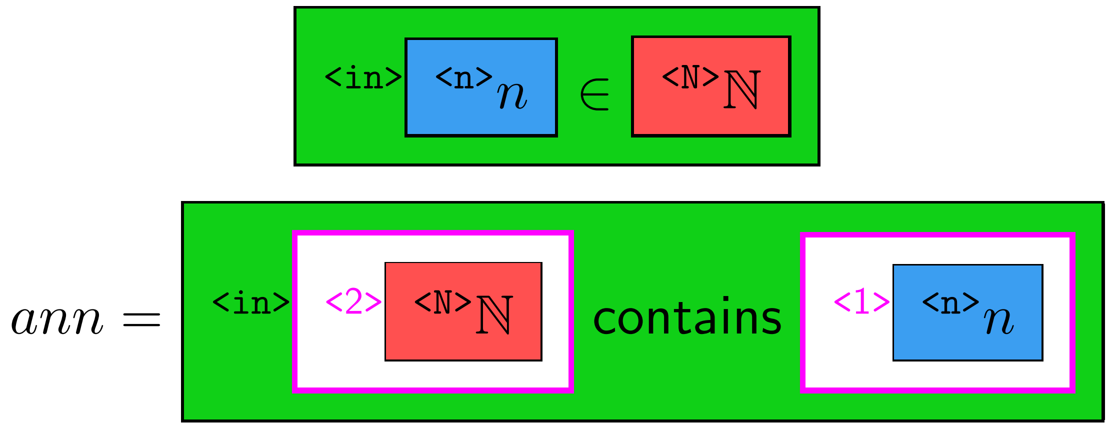


Figure 4: Example for a simple shared souring

reordering/position Souring



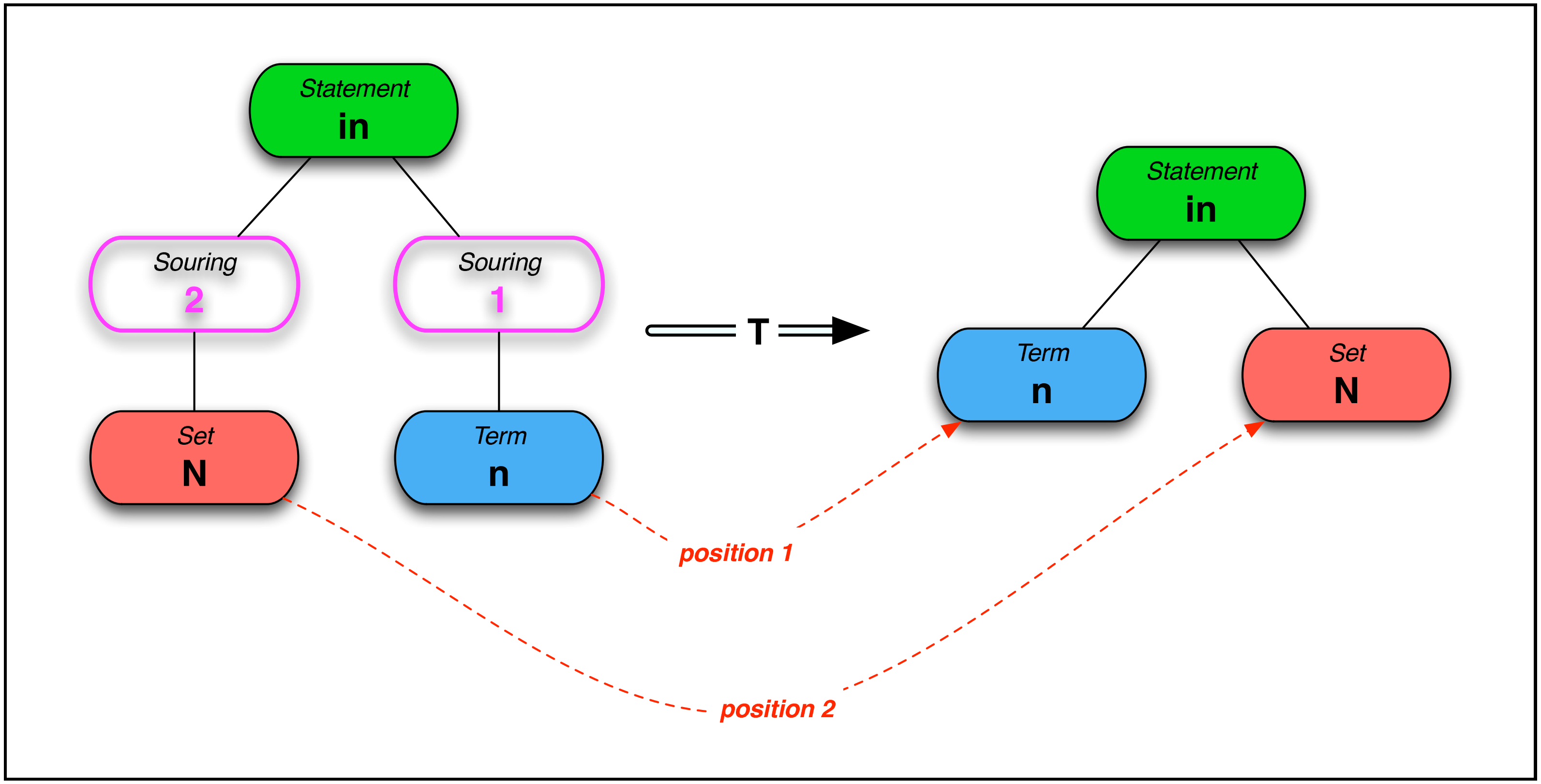
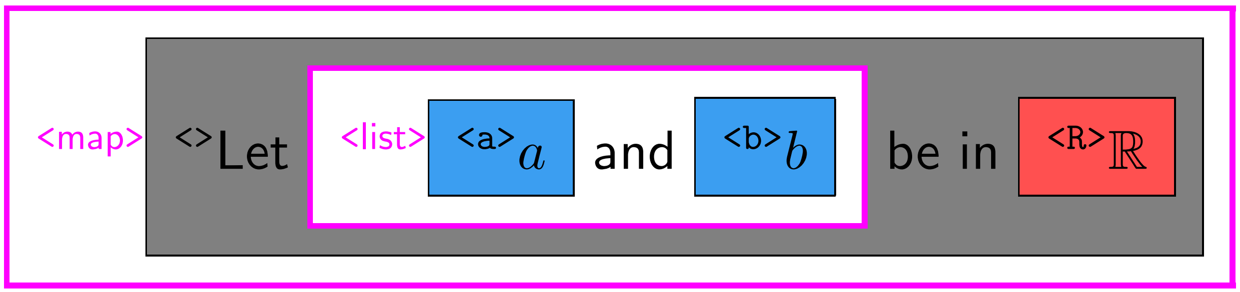


Figure 5: Example for a position souring

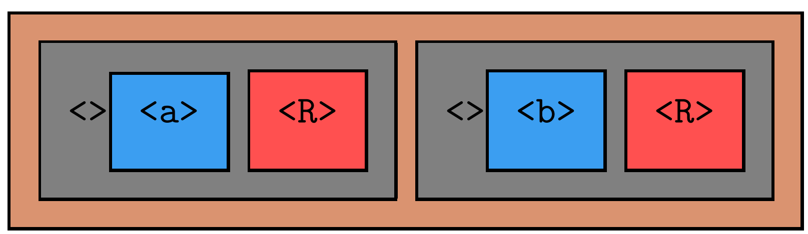
map souring

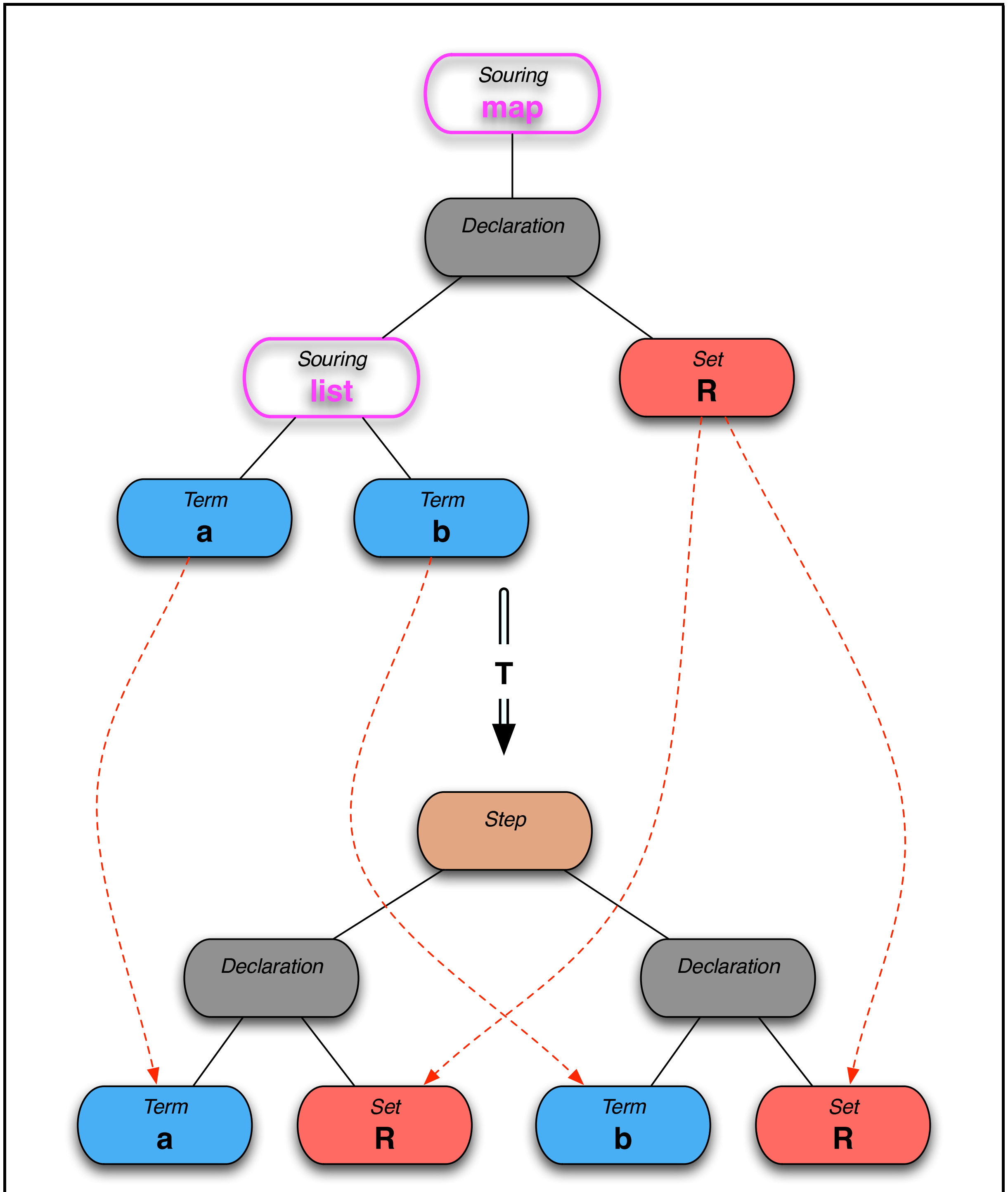
$ann =$

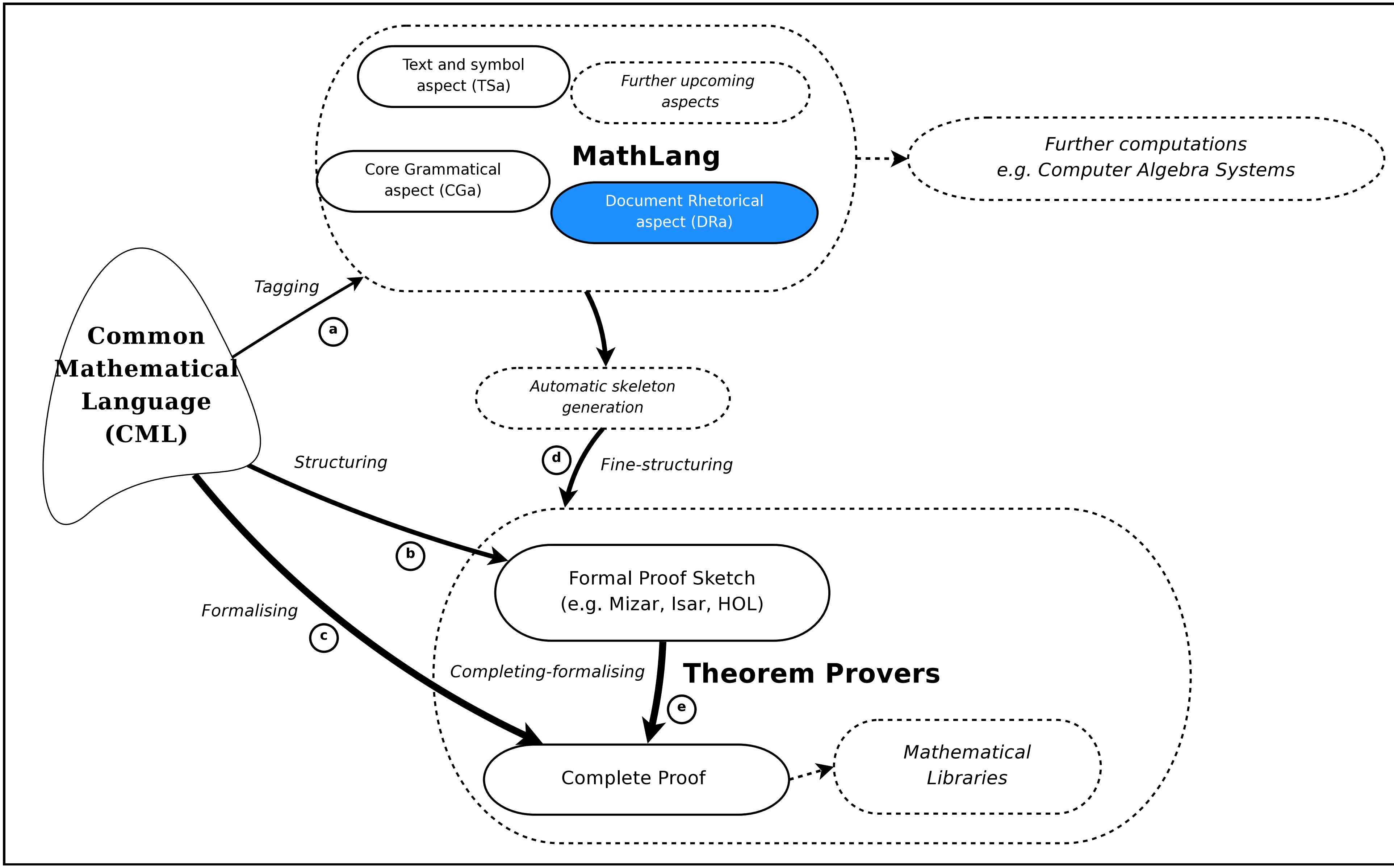


This is expanded to

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




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


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

What is DRa?

- 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 $\sqrt{2} \notin \mathbb{Q}$.

Barendregt

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.

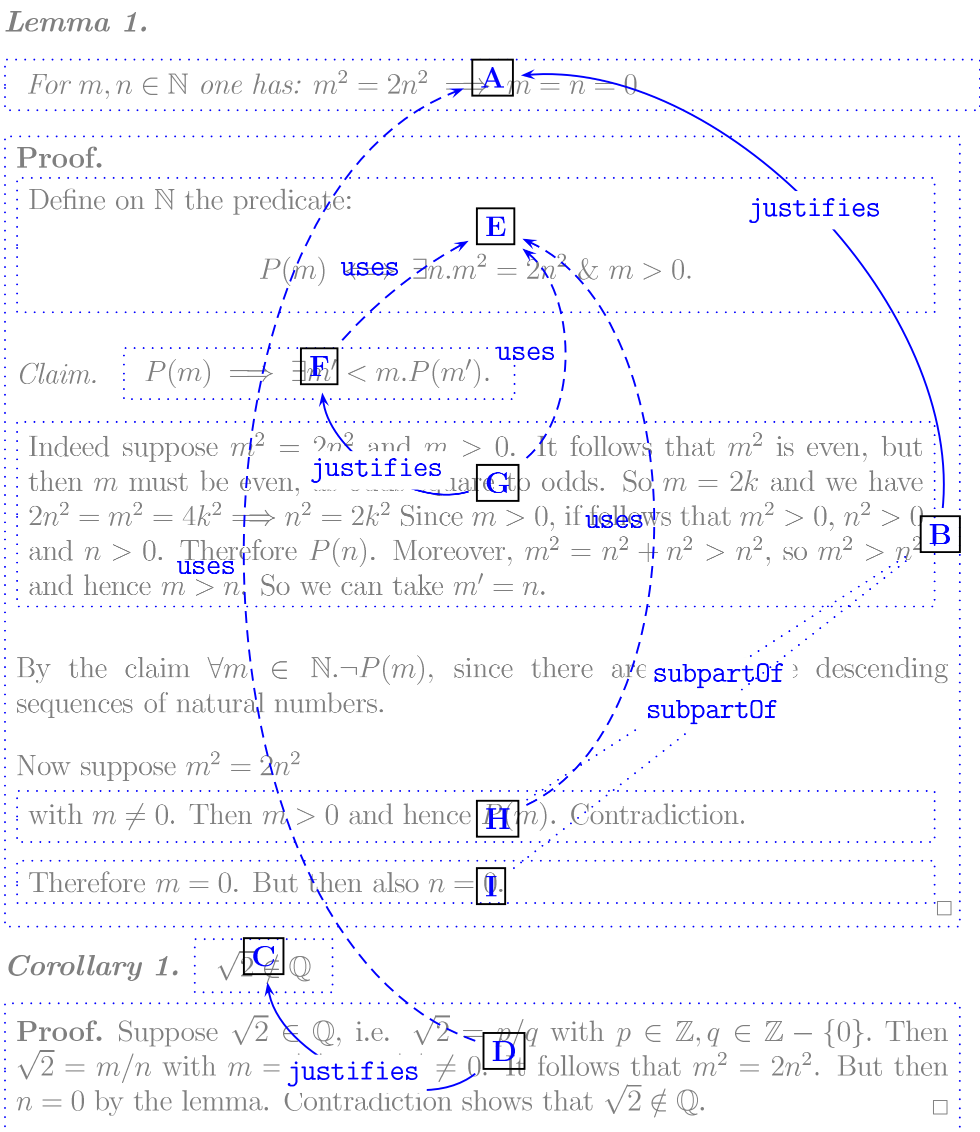
Now suppose $m^2 = 2n^2$

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 $\sqrt{2} \notin \mathbb{Q}$. □



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.

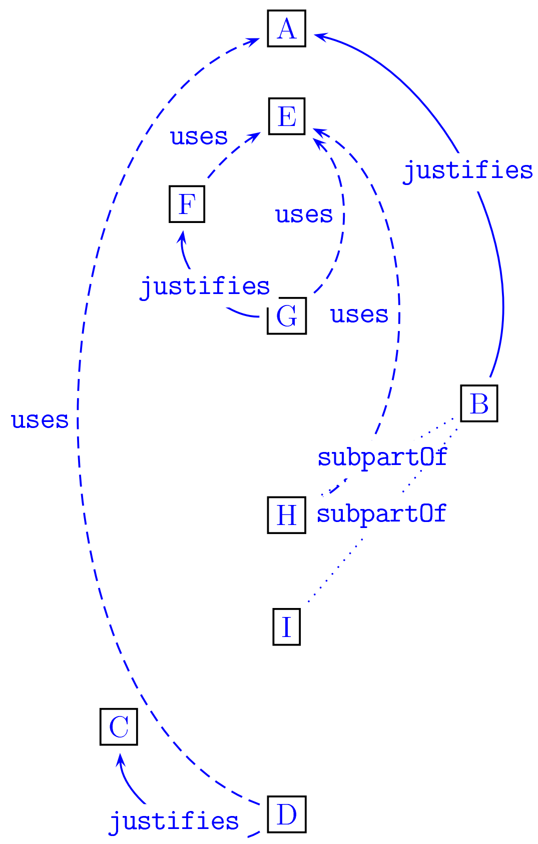
Therefore $m = 0$. But then also $n = 0$. □

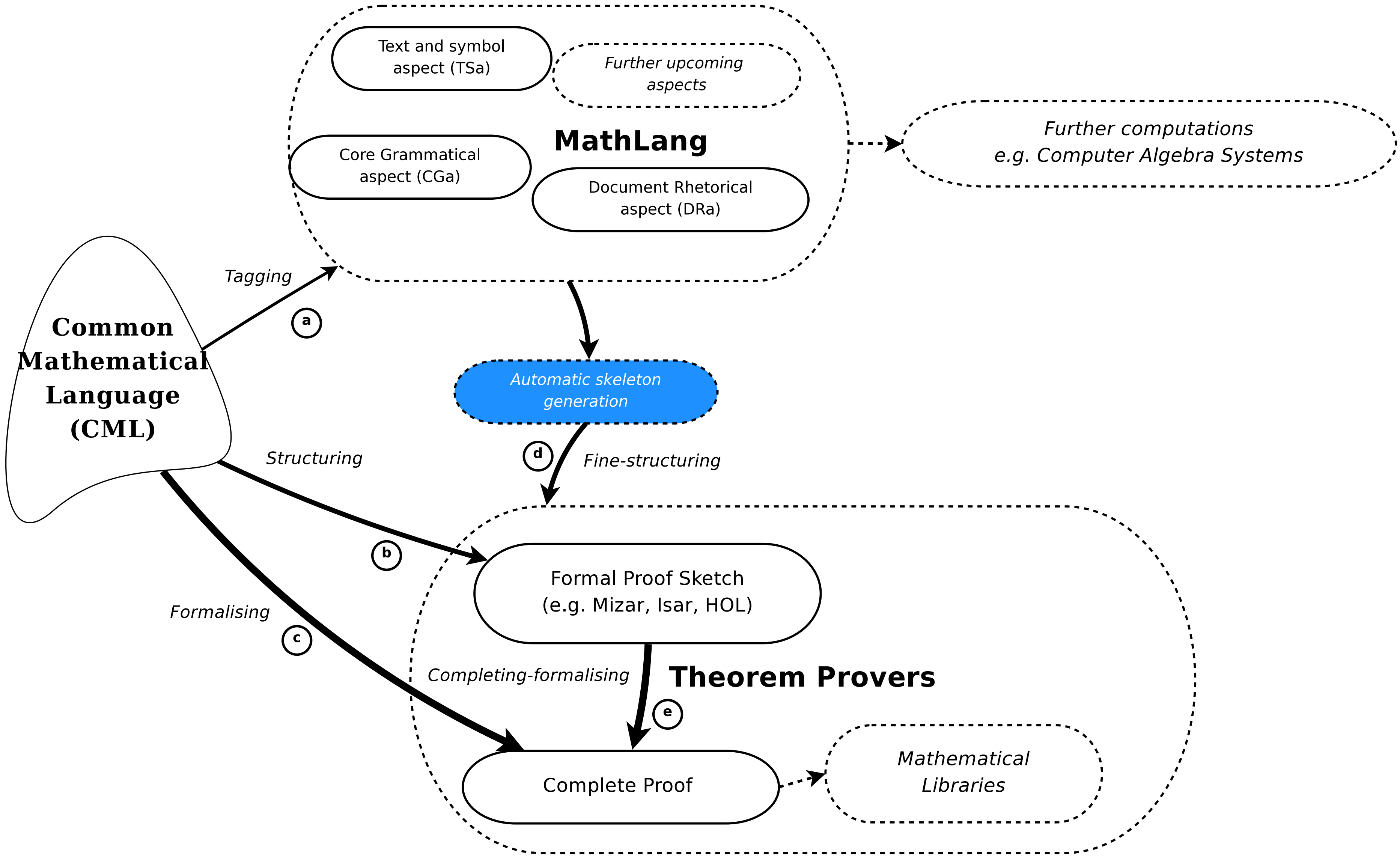
Corollary 1. $\sqrt{2} \notin \mathbb{Q}$

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 $\sqrt{2} \notin \mathbb{Q}$. □

The automatically generated dependency Graph

Dependency Graph (DG)





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

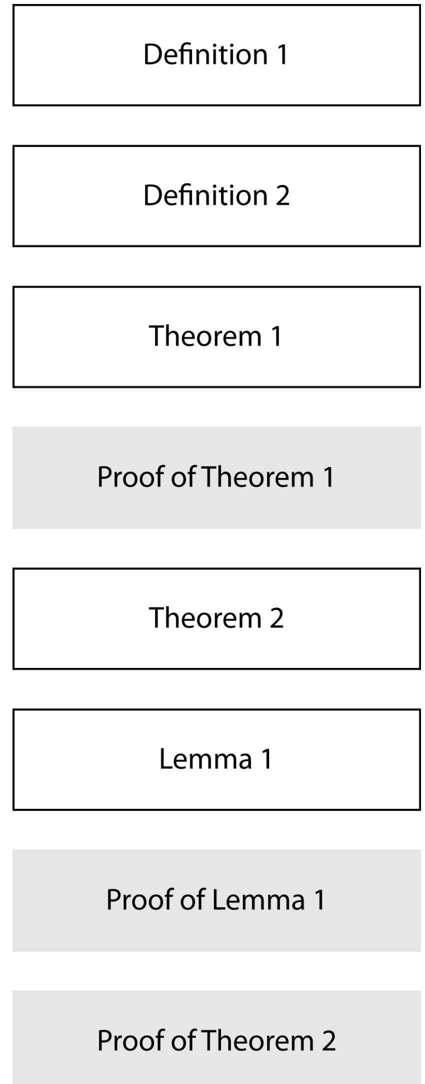
Different provers have

- different syntax

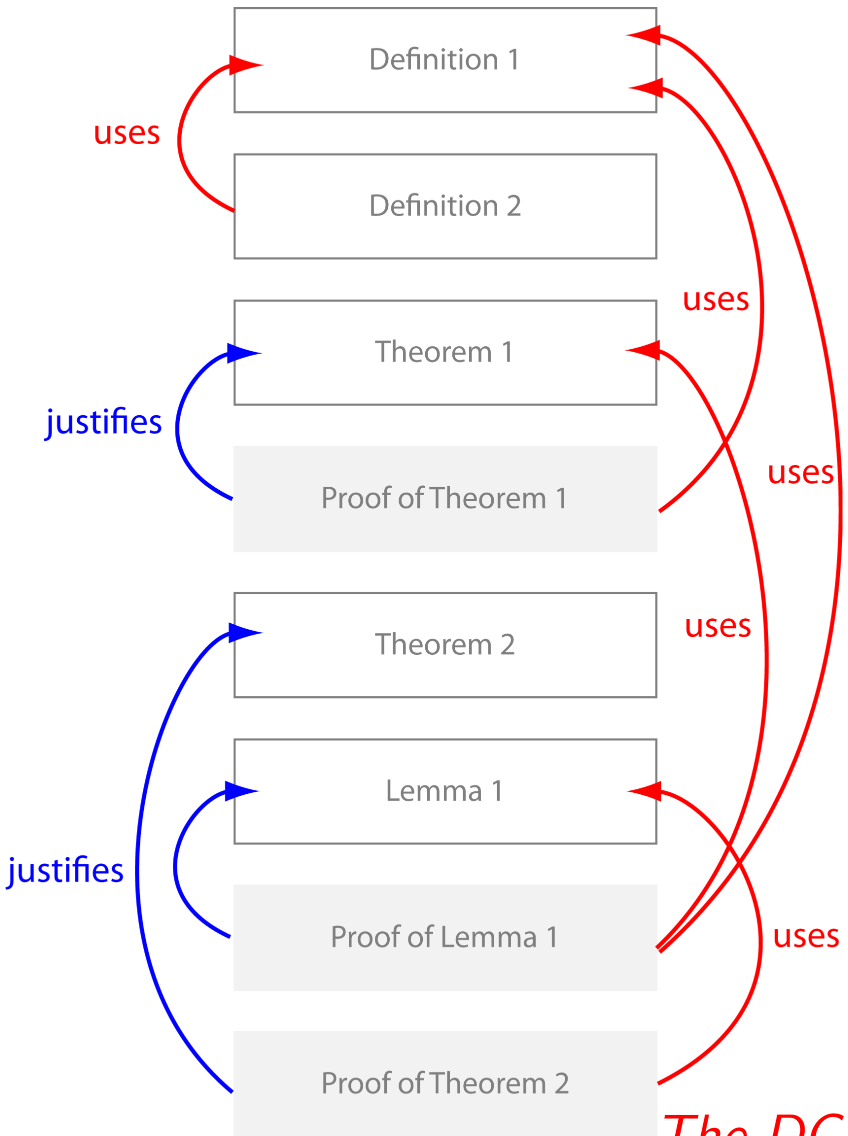
- different requirements to the structure of the text
  e.g.
  - no nested theorems/lemmas
  - only backward references
  - ...

- Aim: Skeleton should be as close as possible to the mathematician's text but with re-arrangements when necessary

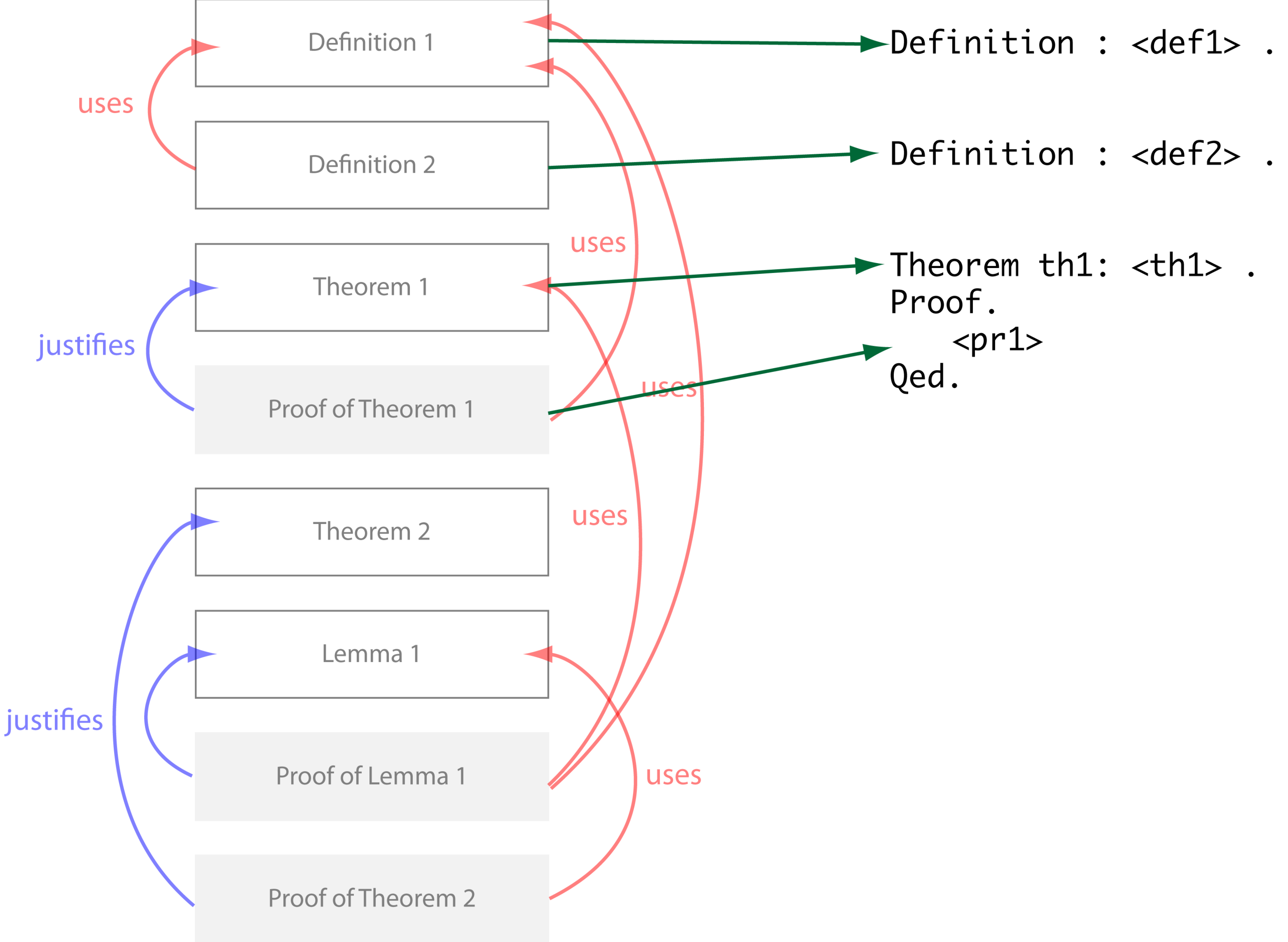
Example of nested theorems/lemmas (Moller, 03, Chapter III,2)



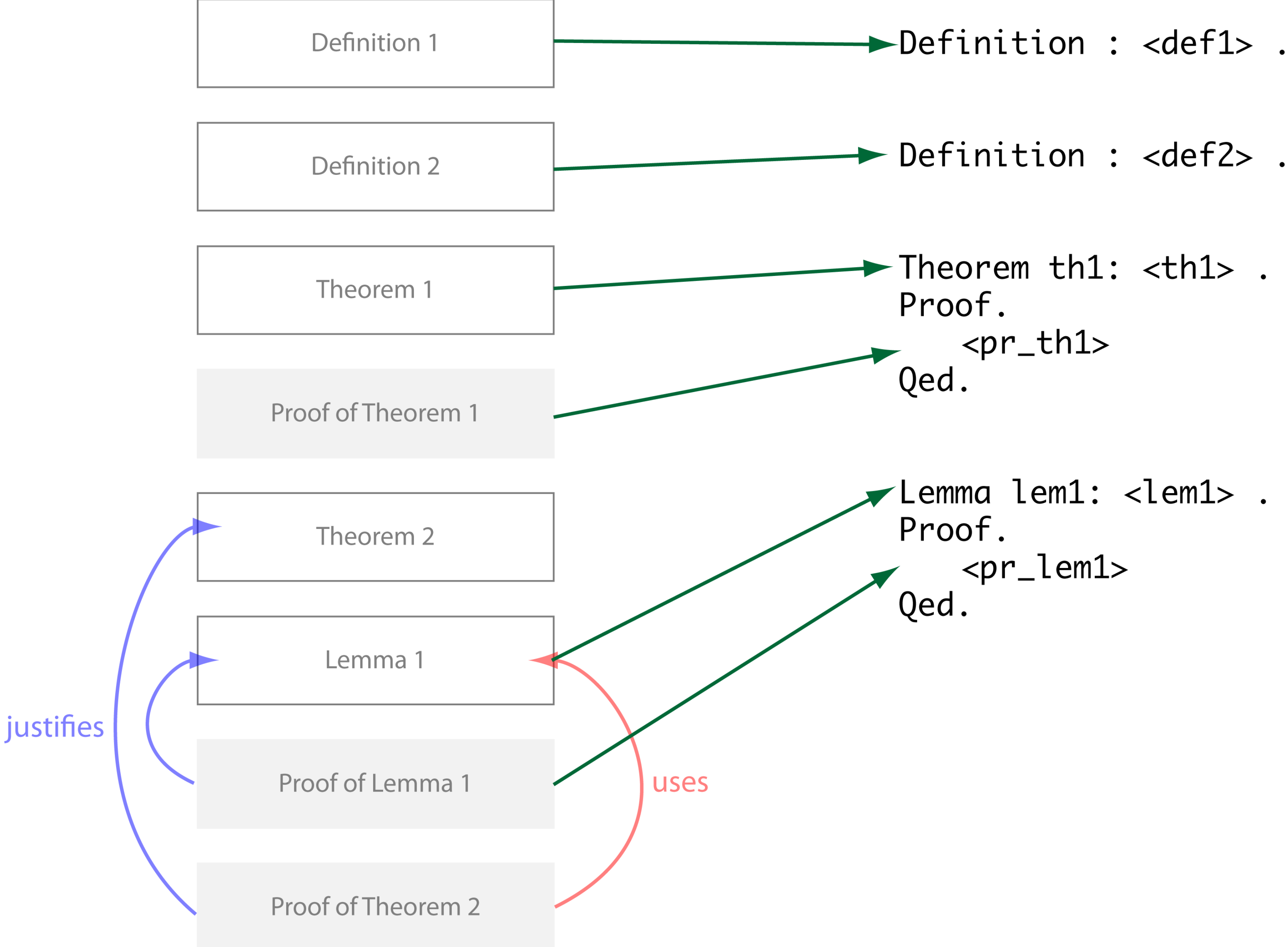
The automatic generation of a proof skeleton



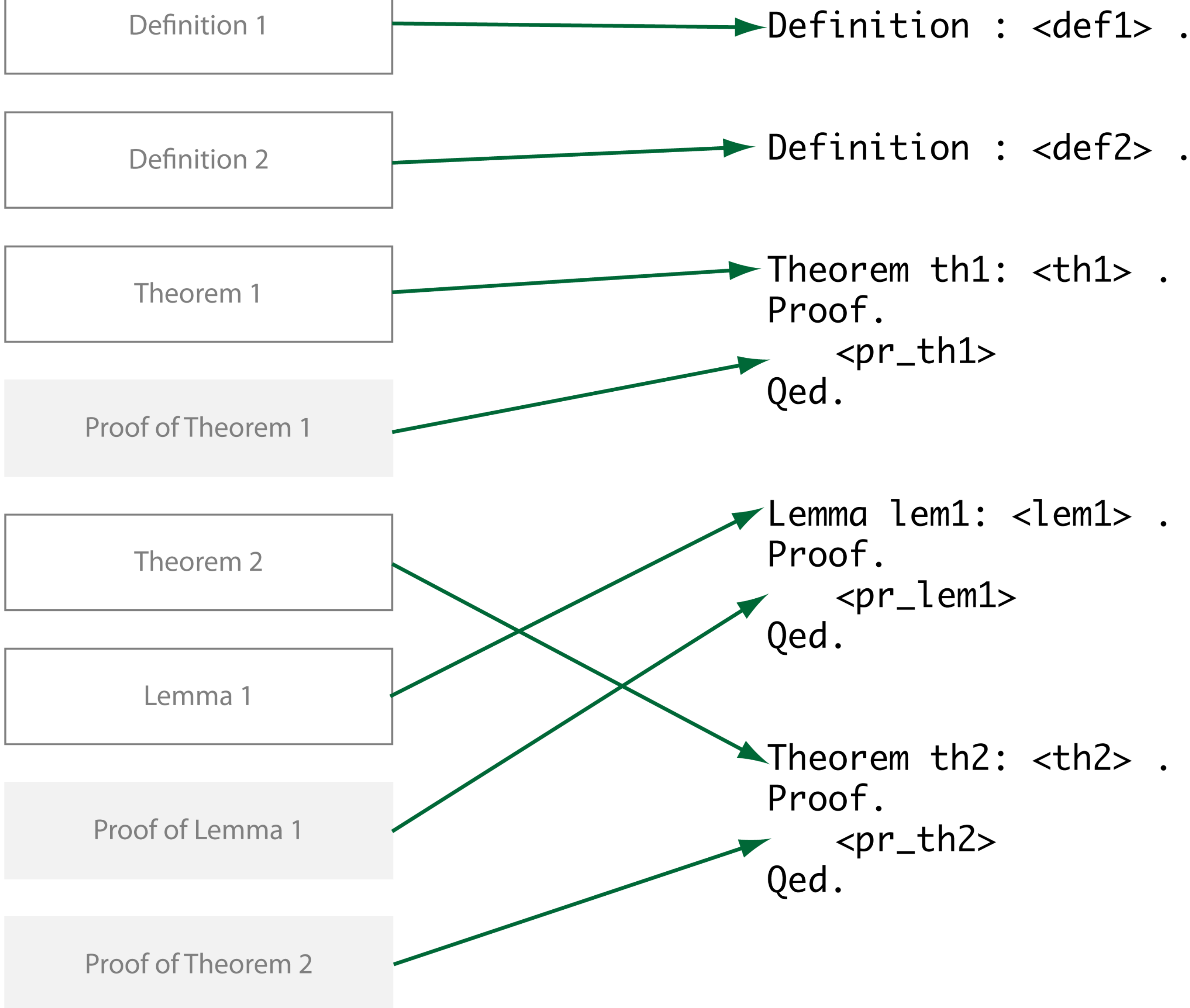
The DG for the example



Straight-forward translation of the first part

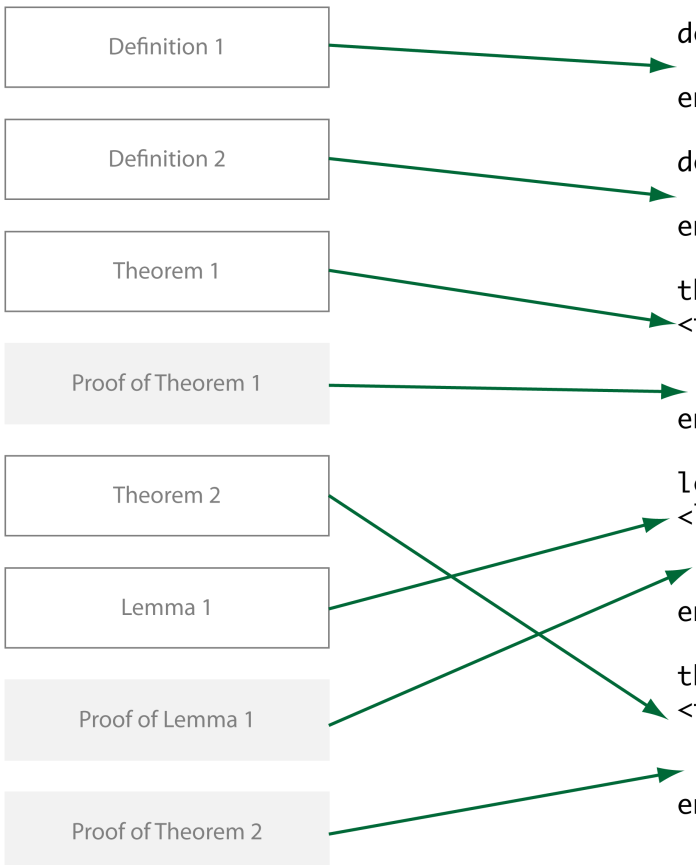


Solution: Re-ordering



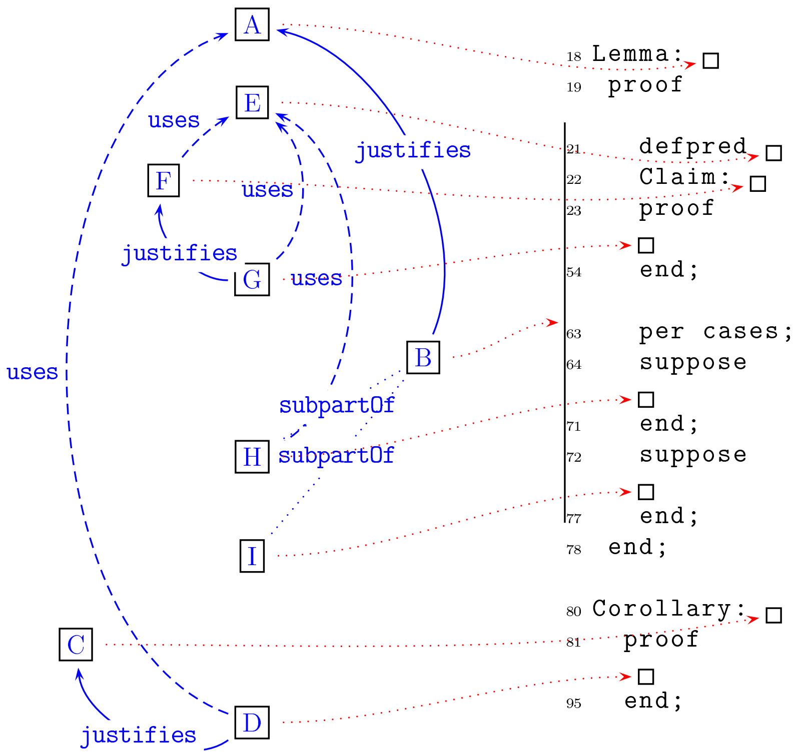
Finishing the skeleton

Skeleton for Mizar




```
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 Mizar and Coq rules for the dictionary

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

Rich skeletons for Coq

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

coq8)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	id : $\mathcal{S}_{Coq}\left(\boxed{p_1}\right) \longrightarrow \ldots \longrightarrow \mathcal{S}_{Coq}\left(\boxed{pn}\right) \longrightarrow$ Prop
coq9)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	id : $\mathcal{S}_{Coq}\left(\boxed{p_1}\right) \longrightarrow \ldots \longrightarrow \mathcal{S}_{Coq}\left(\boxed{pn}\right) \longrightarrow$ Set
coq10)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	(id $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$)
coq11)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	(id $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$)
coq12)	<id> $\boxed{p_1}$ ... $\boxed{pn}$	(id $\mathcal{S}_{Coq}\left(\boxed{p_1}\right)$ ... $\mathcal{S}_{Coq}\left(\boxed{pn}\right)$)
coq13)	<id>	id
coq14)		

<definition><subsetDef>

<defSubSet><><subset><A> $A \subset$ <B> $B \iff$ <forall>$\forall$<><x> x, <impl><in><x> $x \in$ <A> $A \Rightarrow$ <in><x> $x \in$ <B> B

the left hand side of the definition is translated according to rule (coq14)) withsubset 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))

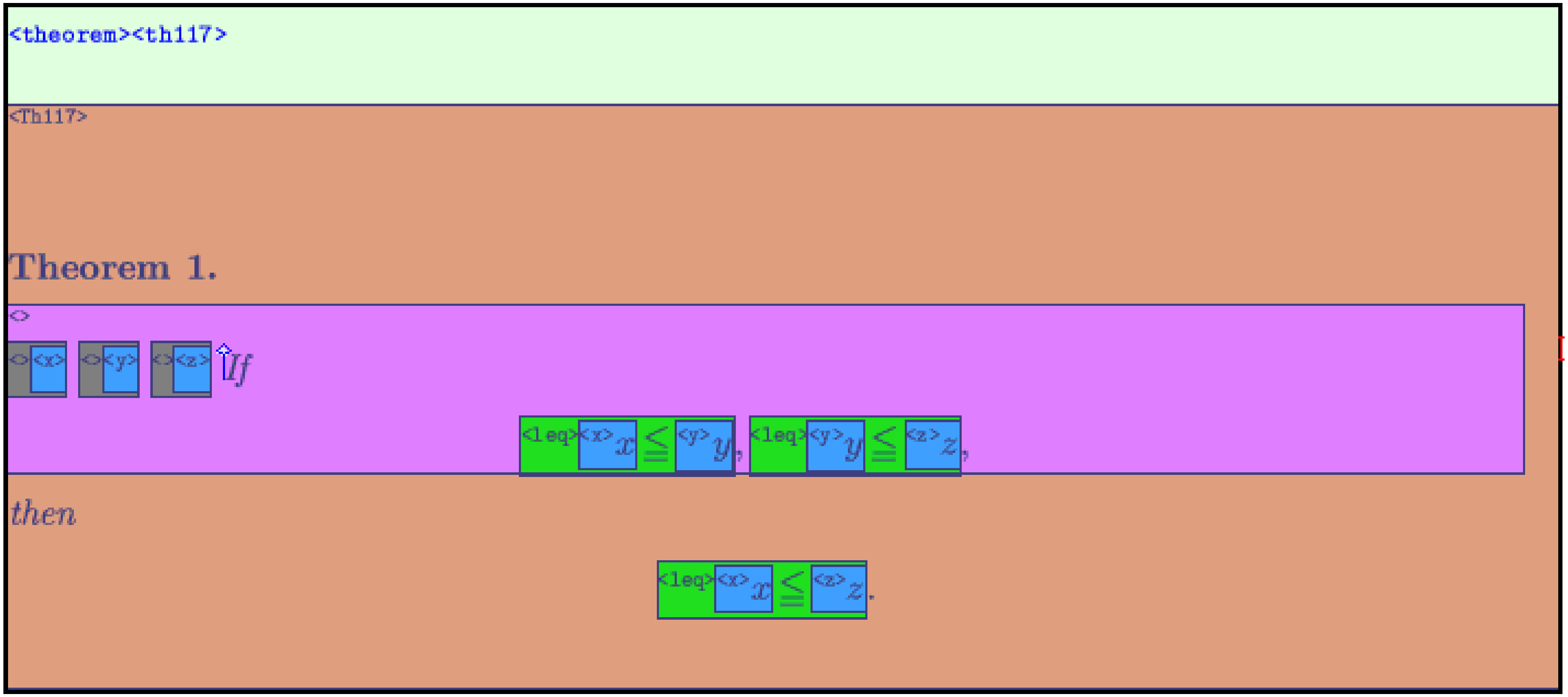
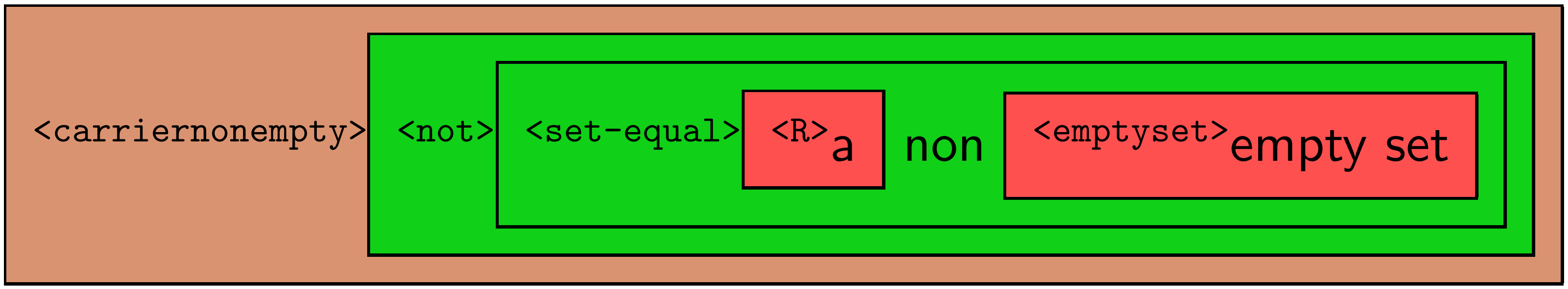


Figure 7: Theorem 17 of Landau's "Grundlagen der Analysis"

The automatic translation is:

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

Rich skeletons for Isabelle



The corresponding translation into Isabelle is:

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

An example of a full formalisation in Coq via MathLang

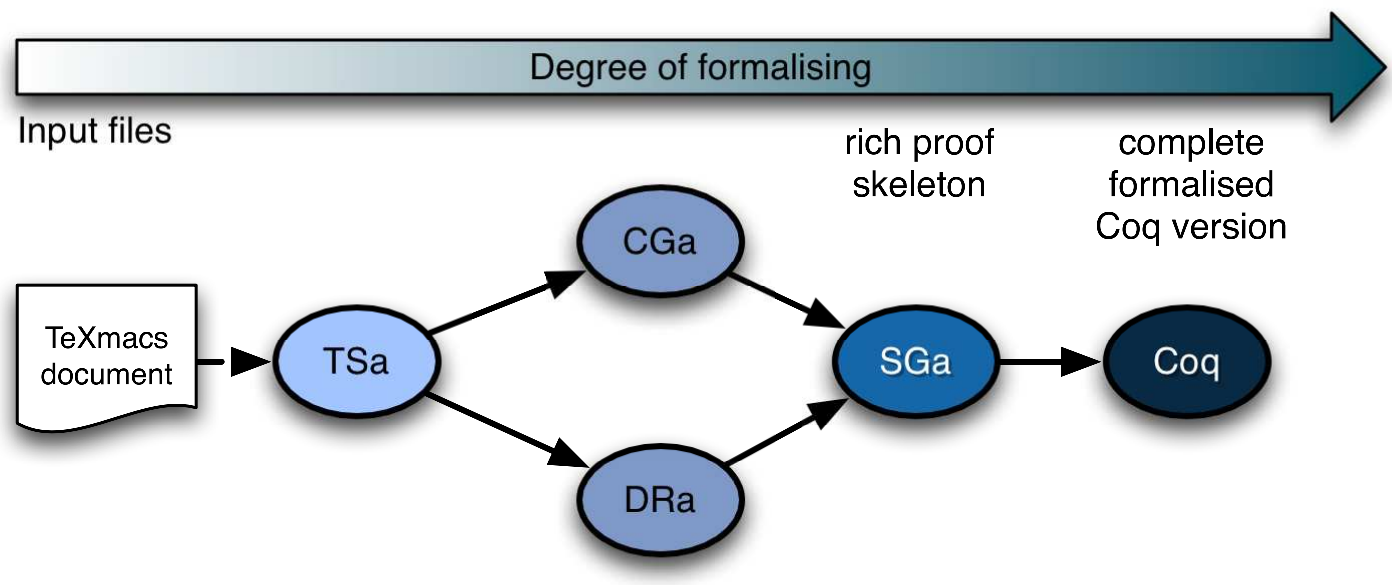


Figure 8: The path for processing the Landau chapter

Chapter 1

Natural Numbers

<><forall> $\forall$ <#><#> , <#> <><exists> $\exists$ <#><#> , <#> <><exists_one> $\exists!$ <#><#> , <#> <><isa><#> <#> <><1> <><and><#> $\wedge$ <#>

<><or><#> $\vee$ <#> <><impl><#> $\longrightarrow$ <#> <><succ><#> <><in><#> $\in$ <#> <><subset><#> $\subset$ <#> <><Set> { <#><#> <#> }

<><seteq><#><#> <><setneq><#><#> <><index><#><#> <><xor><#> $\oplus$ <#> <><emptyset> $\varnothing$

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 ≠ 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

$$1x \neq y$$

(≠ to be read "is not equal to").

Accordingly, the following are true on purely logical grounds:

$x = x$ for every x

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

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

Chapter 1 of Landau

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

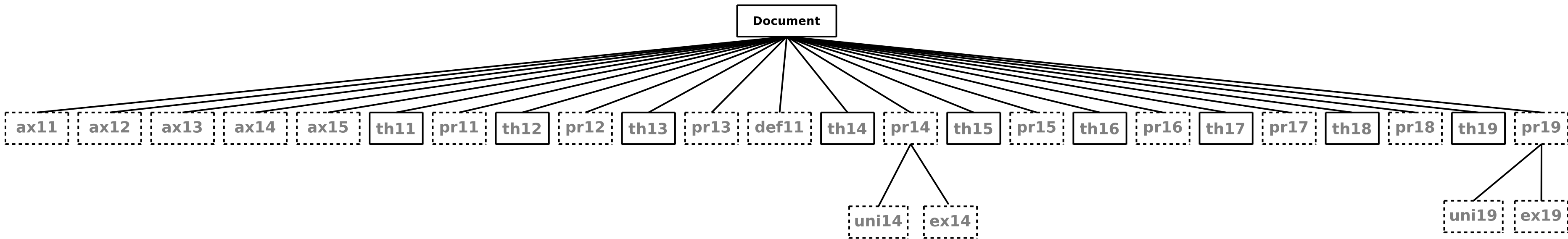


Figure 9: The DRa tree of sections 1 and 2 of chapter 1 of Landau's book

- 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.

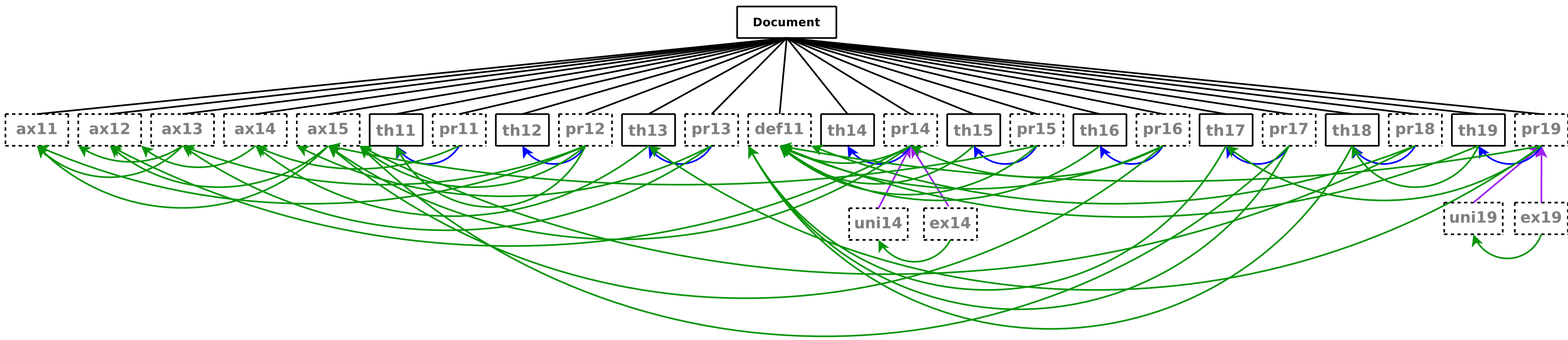
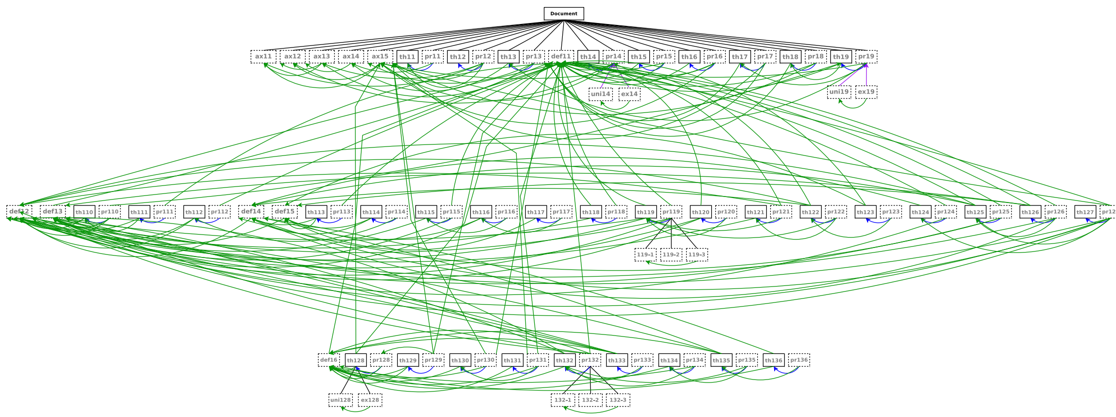


Figure 10: The DG of sections 1 and 2 of chapter 1 of Landau's book

DG of sections 1..4



With the help of the Caa 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.

[1] 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.

[2] 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).

[3] 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 [27], pages 73–100.

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

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

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

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

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

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

[10] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [18], pages 1–82.

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

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

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

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

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

[16] 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.

[17] Robert Harper and Greg Morrisett. Compiling polymorphism using intensional type analysis. [17].

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

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

[20] 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.

[21] 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.

[22] F. Kamareddine, T. Laan, and R. Nederpelt. Revisiting the notion of function. *Logic and Algebraic programming*, 54:65–107, 2003.

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

[24] 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.

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

[26] G. Longo and E. Moggi. Constructive natural deduction and its modest

interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.

[27] 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.

[28] G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [18], pages 83–97.

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

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

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

[32] B. Russell. Letter to Frege. English translation in [18], pages 124–125, 1902.

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

[34] M. Takahashi. λ-calculi with conditional rules. [34].

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

[36] 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.