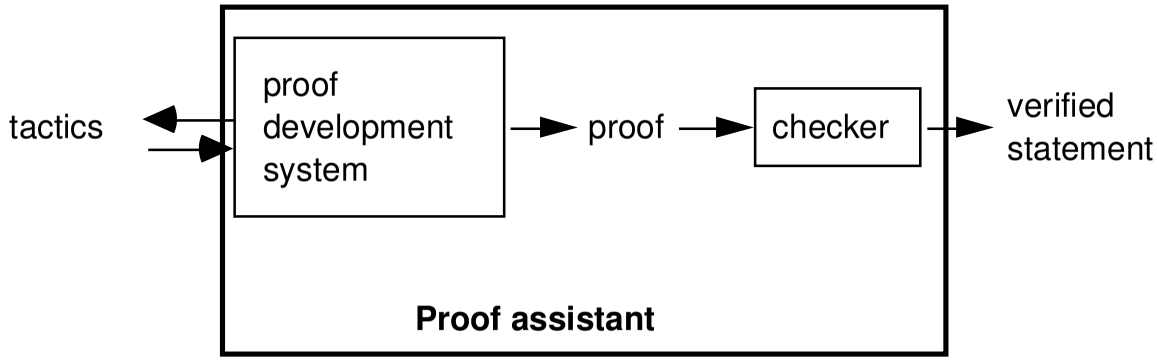
2. Representing and constructing proofs

We wanted reliability

$\vdash A$	(provable A)	undecidable
$\vdash_p A$	(p proves A)	(very much) decidable

So the problem becomes the construction of full proofs

These are also called **proof-objects**



All proof assistants have this form,

but the details differ

As the drawing suggests, we prefer systems with a small reliable checker

The axiomatic method

Logic

Types & lambda terms

Computations

Representing proofs

- Logical framework
- Formulas—as—types

Components of a proof assistant

Existing systems

The axiomatic method (Aristotle)

There are concepts and theorems about them

Concepts can be defined from earlier concepts

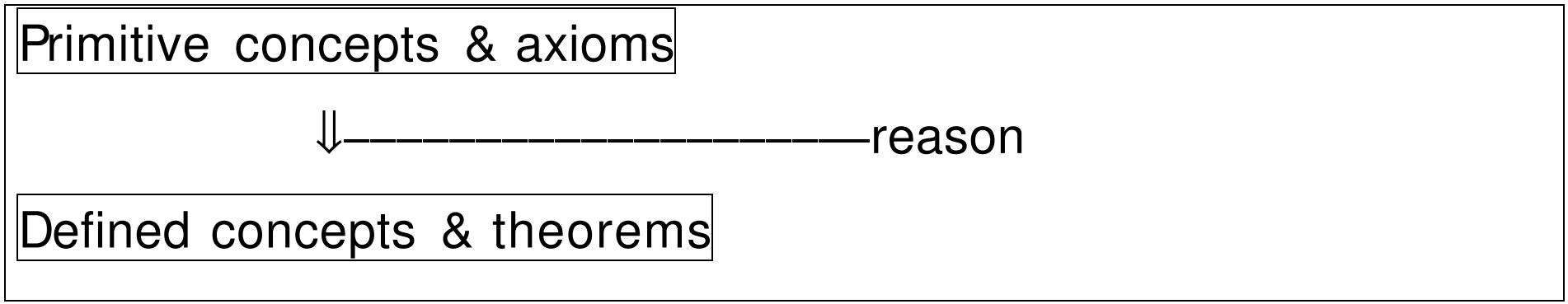
We need primitive concepts

Theorems can be proved from earlier ones

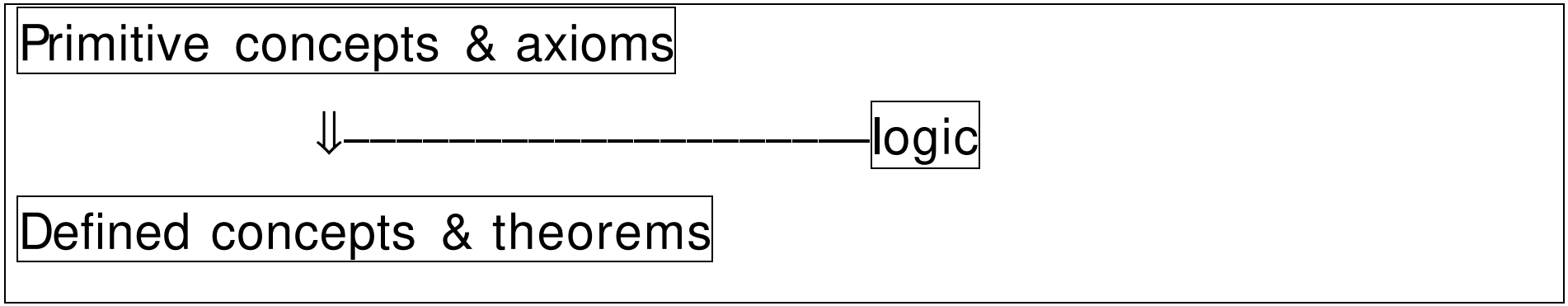
We need axioms

This must have inspired Euclid to compose The Elements

Logic



was replaced by



Logic was initiated by Aristotle and finished by

Frege (for classical logic)

Heyting (for intuitionistic/constructive logic)

From the completion on it was possible *in principle*

to give fully formalised proofs and to do proof-checking

Types and lambda terms

Intuitive types

π : Real

sin : Real $\rightarrow$ Real

$\int_a^b$: (Real $\rightarrow$ Real) $\rightarrow$ Real

Intuitive lambda terms

f does x ---> x^2+1

$f=\lambda x.x^2+1$

So $(\lambda x.x^2+1)(3) =_\beta 3^2+1$

In general $(\lambda x.M[x])N =_\beta M[N]$

$(\lambda x.M)N =_\beta M[x:=N]$

A:set, R:A→A→prop, a:A, b:A |- Rab : prop

Def

Binrel A = A→A→prop

Binrel = λA:set. A→A→prop

: A → (A→A→prop)

Then intuitively we could have written R:(Binrel A)

It is convenient to add types and lambda terms to logic

Representation of proofs

- Logical frameworks (first automath translation)
- Formulas—as—types (Curry-Howard-de Bruijn)

Logical frameworks

imp:prop→prop→prop, p,q:prop |- imp p q : prop

we want “imp p p” to be “valid”

introduce axiomatically T

[intended meaning T p = “set of proofs of p”]

T:prop→set and the axioms

Γ={T:prop→set, imp:prop→prop→prop,

Ax1: T(imp p q) → T p → T q,

Ax2: (T p → T q) → T(imp p q)}

A formula Z is <u>valid</u> iff T Z has an inhabitant (… : T Z)

Then we have that (imp p p) is valid

Γ |- (λu:(T p).u) : (T p) $\rightarrow$ (T p)

Γ |- Ax2(λu:(T p).u) : T(imp p p) So imp p p is valid indeed

In this way on could formalise logic and mathematics.

But it was realised that T did not do much

Second automath translation

Formulas—as—types

Take prop=set then the Γ becomes superfluous

(definable/derivable from the axioms of

typed lambda calculus)

$T = \lambda p{:}prop.p$ $\qquad\qquad$ $T\ p = p$

$Imp = \lambda p{:}prop \lambda q{:}prop.p \to q$ $\quad imp\ p\ q = p \to q$

$\vdash (\lambda x{:}p.x) : p \to p$

In mathematics

Given context Γ one has A

In logic

Γ |- A

In type theory

Γ |- proof : A

Proof-checking becomes type-checking

Poincaré principle

"If in a mathematical argument we need

2 + 2 = 4

this is not a proof, just a verification."

In type theory this becomes

$\underline{p : A(t)}$ if $t =_R s$

$p : A(s)$

de Bruijn emphasised this for R is β

$$\frac{R:\text{Binrel } A}{R:(A \to A \to \text{prop})}$$

Martin-Löf emphasised this for R is ι

ι is the equality (reduction) corresponding to primitive recursion over data types

$$\frac{p : A(\text{fac } 4)}{p : A(24)}$$

Inductive types

Nat = zero | suc Nat

Comes with a recursor that gives at the same time primitive recursion and induction

Not just for Nat, but for arbitrary such data types

Proof assistants

- Proof environment: text editor, ...
- Vernacular: tactics, ...
- Theory libraries: bootstrap
- Special logics: can be internalised using the LF method

  notably this is the case for equational logic

  whereby the Poincaré principle is used heavily

  (reflection, two-level approach)

Choice of type theory

Logical force

Allowed lambda abstractions ($\rightarrow$ allowed quantification)

Computational force

Allowed equality relation in the Poincaré principle

Extensional type theories, like in Nuprl, have an equality that is not decidable

Some existing proof assistants

Coq	Hol	Mizar	PVS
Lego	Isabelle		
Half	Nuprl*		
λ-terms as proofs	tactics as proofs *extensional TT	based on ZF set-theory	not de Bruijn