# RacketCon 2017 talk

# Proust: A Nano Proof Assistant

Prabhakar Ragde
University of Waterloo

Marcel Proust

Proust:

A series of small Racket programs to be used as proof assistants.

Eventual application:

Teaching a required sophomore course in logic and computation.

Proust does not use:

Macros, modules,
`#lang` DSLs, contracts,
continuations, futures...

Proust does use:

S-exprs, structs, `match`, `format`.

Racket shows that
“difficult” ideas
can be easier to understand
than “simple” ones.

Conventional approach:

Propositional logic
Predicate logic
Equality, mathematical objects
Computation

## Student complaints:

Inconsistent content.
Confining.
Boring.
Irrelevant.
Impractical.

## A conventional proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, A \to B \vdash A \to B}\,(\mathsf{Ass}) \quad \dfrac{}{A, A \to B \vdash A}\,(\mathsf{Ass})}{A, A \to B \vdash B}\,(\to_E)}{A \vdash (A \to B) \to B}\,(\to_I)}{\vdash A \to (A \to B) \to B}\,(\to_I)$$

Proust approach:

Propositions as types.
Proofs as programs.

A proof of $T \longrightarrow W$ is a function that consumes a proof of $T$ and produces a proof of $W$.

A proof of $A \to (A \to B) \to B$ is $\lambda x \, . \, \lambda f \, . \, f \; x$.

## Proust:

parser, pretty-printer,
type checking/synthesis
(bidirectional type inference)
syntax-driven
refinement in REPL

Other logical connectives yield natural programming constructs.

We work out rules in lecture, students implement them in Proust, then use it for proofs.

Things get more complicated with predicate logic.

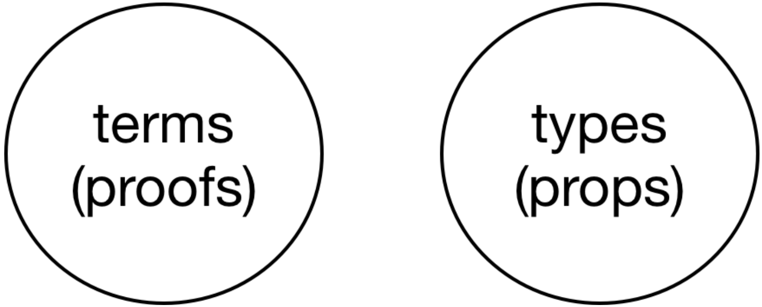
Conventional predicate logic is first-order (quantification over unspecified universe).

Proust uses higher-order logic (domain of quantification specified).

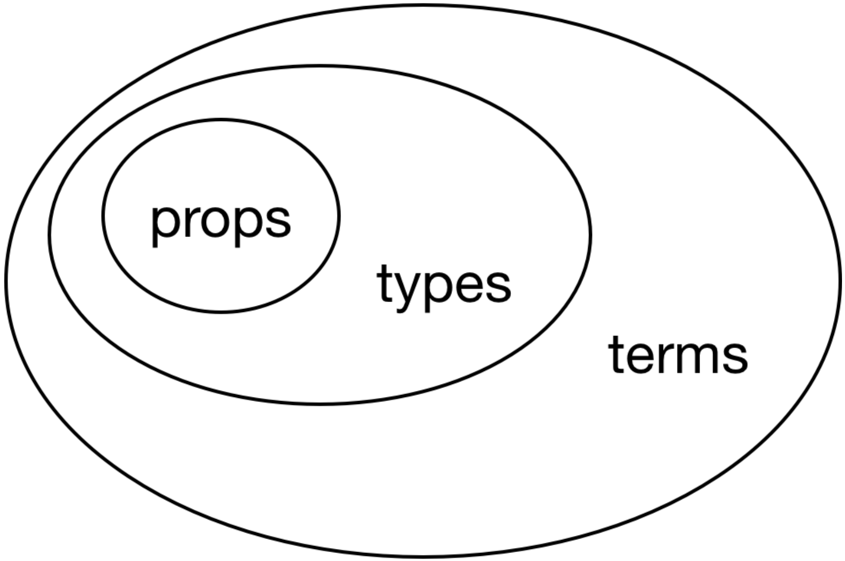
A proof of $\forall(X : T) \to W$ is a function that consumes an object $x$ of type $T$ and produces a proof of $W[X \mapsto x]$.

(Dependent types.)

Previously:



## Now:



Proust must implement:

Proper substitution
Renaming, equivalence
Reduction (computation)

Proofs continue to be more intuitive and shorter than conventional ones.

It’s also easy to add datatypes.

Polymorphic eliminators give both structural induction and structural recursion.

We move to Coq for the last part of the course.

Proust demystifies Coq!

Students see Coq as compelling and potentially useful.

Summary:

Small simple Racket programs help in effectively teaching logic and computation, using the ideas of proofs as programs and dependent types.