HenBlocks: Structured Editing for Coq

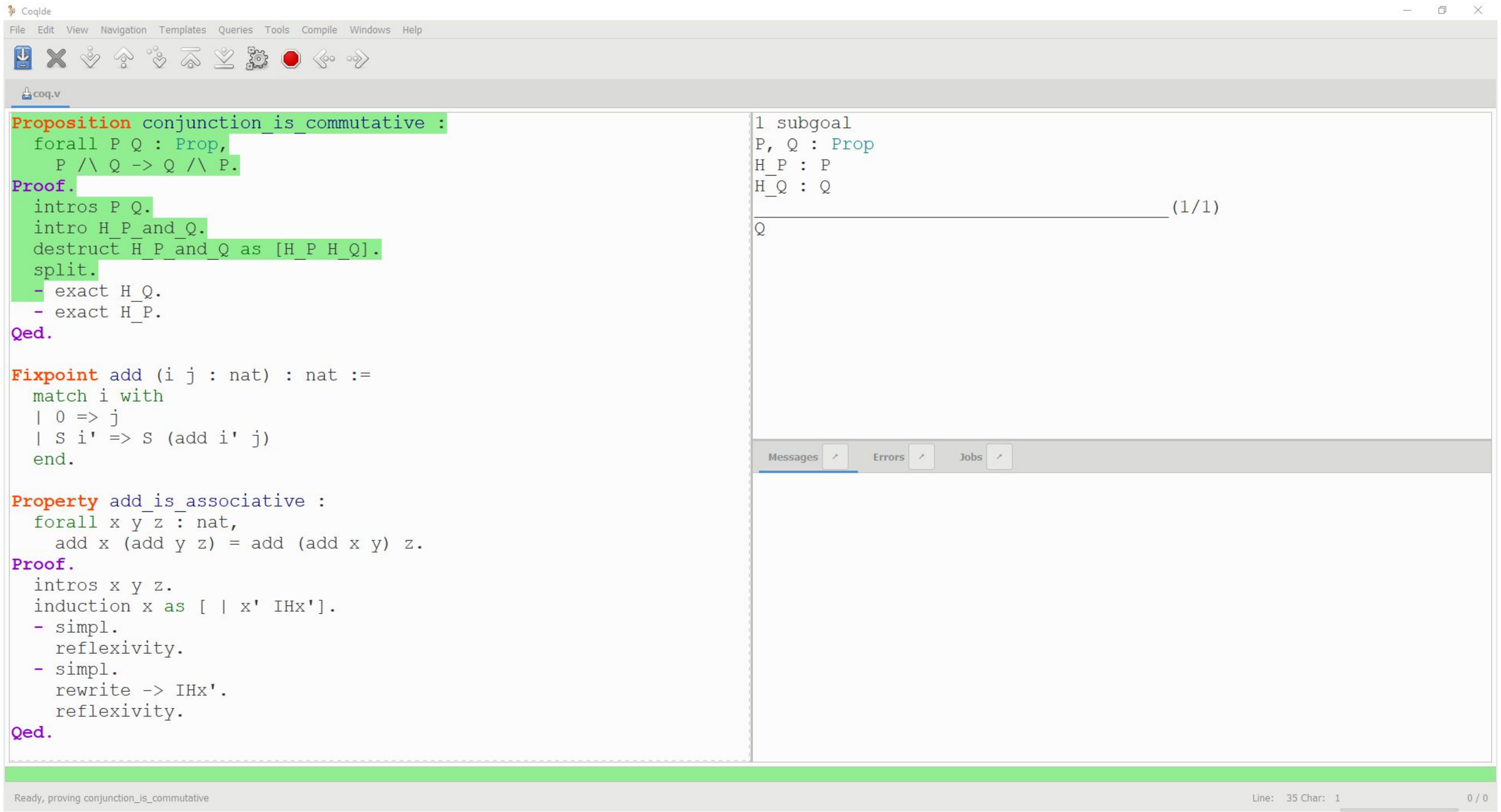
Coq Workshop 2022

Bernard Boey
Supervisor: Michael D. Adams

Contents

Background

Background - Coq Proof Assistant

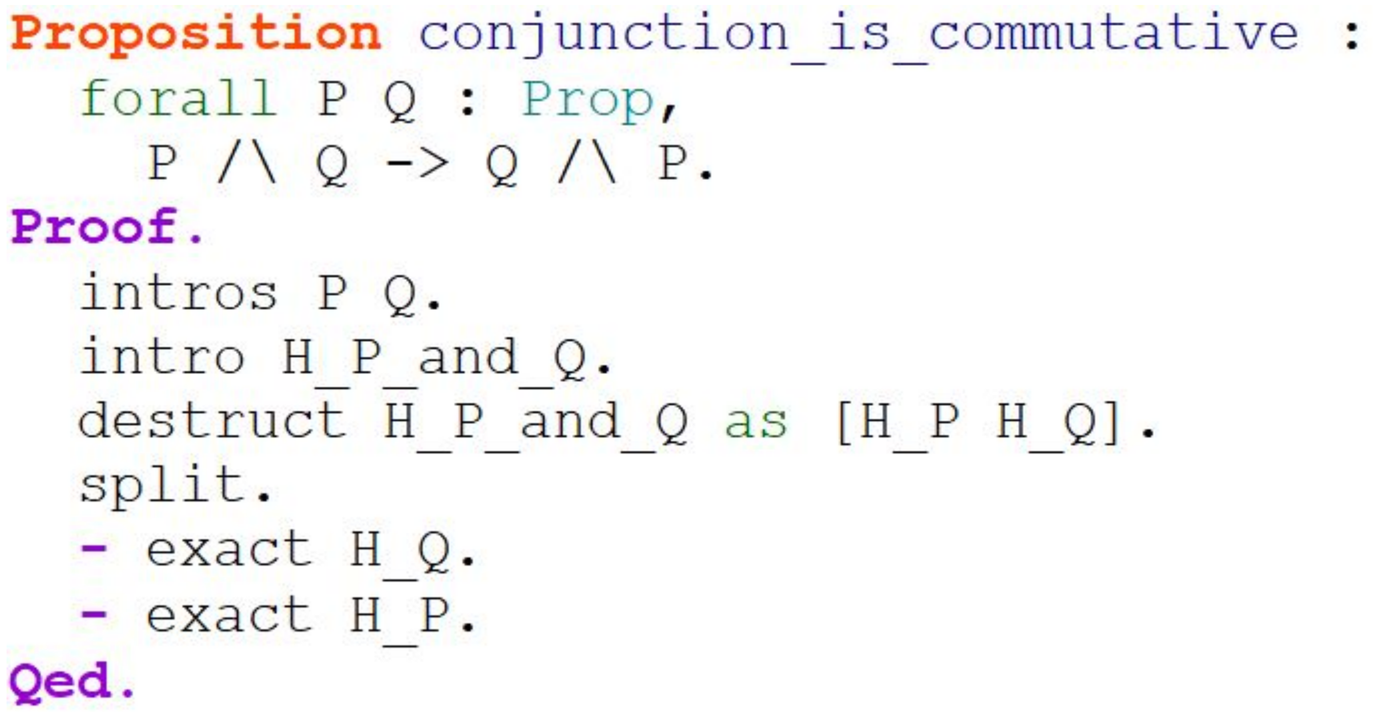

CoqIde
File Edit View Navigation Templates Queries Tools Compile Windows Help
coq.v
Messages
Errors
Jobs

```
Proposition conjunction_is_commutative :
  forall P Q : Prop,
    P /\ Q -> Q /\ P.
Proof.
  intros P Q.
  intro H_P_and_Q.
  destruct H_P_and_Q as [H_P H_Q].
  split.
  - exact H_Q.
  - exact H_P.
Qed.

Fixpoint add (i j : nat) : nat :=
  match i with
  | 0 => j
  | S i' => S (add i' j)
  end.

Property add_is_associative :
  forall x y z : nat,
    add x (add y z) = add (add x y) z.
Proof.
  intros x y z.
  induction x as [ | x' IHx'].
  - simpl.
    reflexivity.
  - simpl.
    rewrite -> IHx'.
    reflexivity.
Qed.
```

```
1 subgoal
P, Q : Prop
H_P : P
H_Q : Q
______________________________________(1/1)
Q
```

```
Ready, proving conjunction_is_commutative                Line: 35 Char: 1      0 / 0
```

Background - Mathematical Logic in Coq



```
Proposition conjunction_is_commutative :
  forall P Q : Prop,
    P /\ Q -> Q /\ P.
Proof.
  intros P Q.
  intro H_P_and_Q.
  destruct H_P_and_Q as [H_P H_Q].
  split.
  - exact H_Q.
  - exact H_P.
Qed.
```

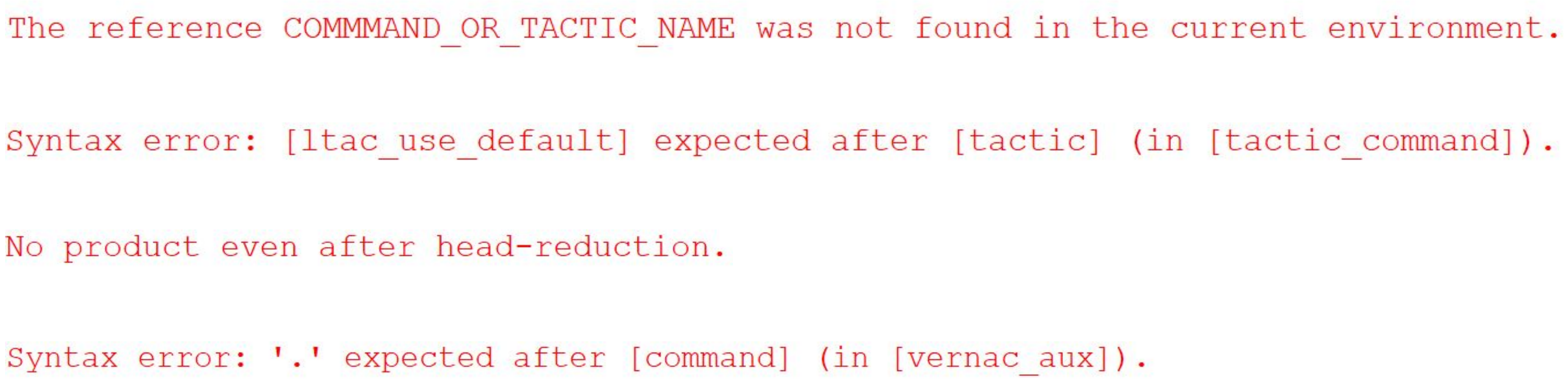
Background - Functional Programming and Proving in Coq

```
Fixpoint add (i j : nat) : nat :=
  match i with
  | 0 => j
  | S i' => S (add i' j)
  end.

Property add_is_associative :
  forall x y z : nat,
    add x (add y z) = add (add x y) z.
Proof.
  intros x y z.
  induction x as [ | x' IHx'].
  - simpl.
    reflexivity.
  - simpl.
    rewrite -> IHx'.
    reflexivity.
Qed.
```

Background - Pain Points of Coq

1. Type system complex and difficult to understand (Robert 2018)
2. Difficulty in learning new specification & tactic languages (Böhne & Kreitz 2018)
3. Friction in user experience (Robert 2018)

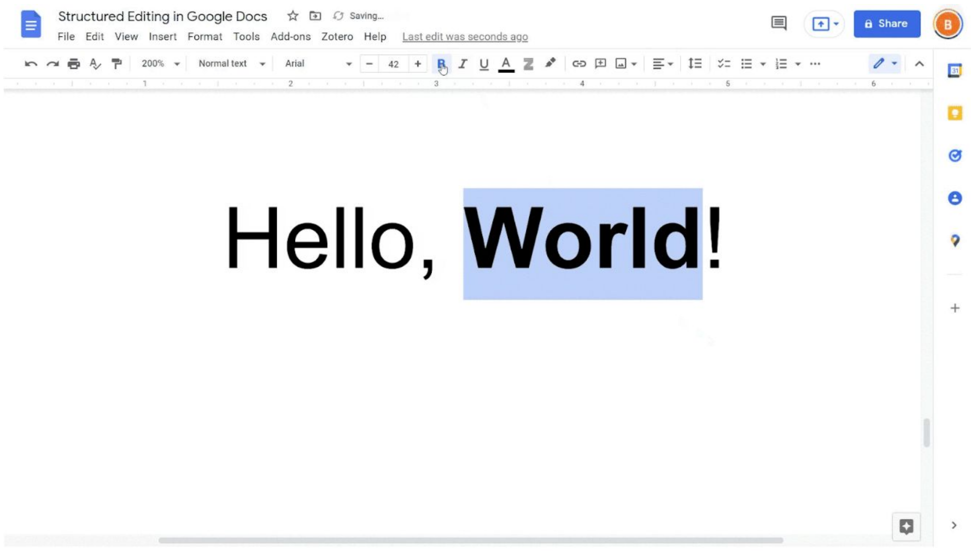
These 4 error messages are all due to the same kind of syntax error: missing a period (full stop) after a command/tactic.



```
The reference COMMMAND_OR_TACTIC_NAME was not found in the current environment.

Syntax error: [ltac_use_default] expected after [tactic] (in [tactic_command]).

No product even after head-reduction.

Syntax error: '.' expected after [command] (in [vernac_aux]).
```

Background - Structured Editing

Manipulation of underlying text content in a syntax-directed manner.


Structured Editing in Google Docs
Saving...
File Edit View Insert Format Tools Add-ons Zotero Help
Last edit was seconds ago
Share
200%
Normal text
Arial
42
Hello, World!

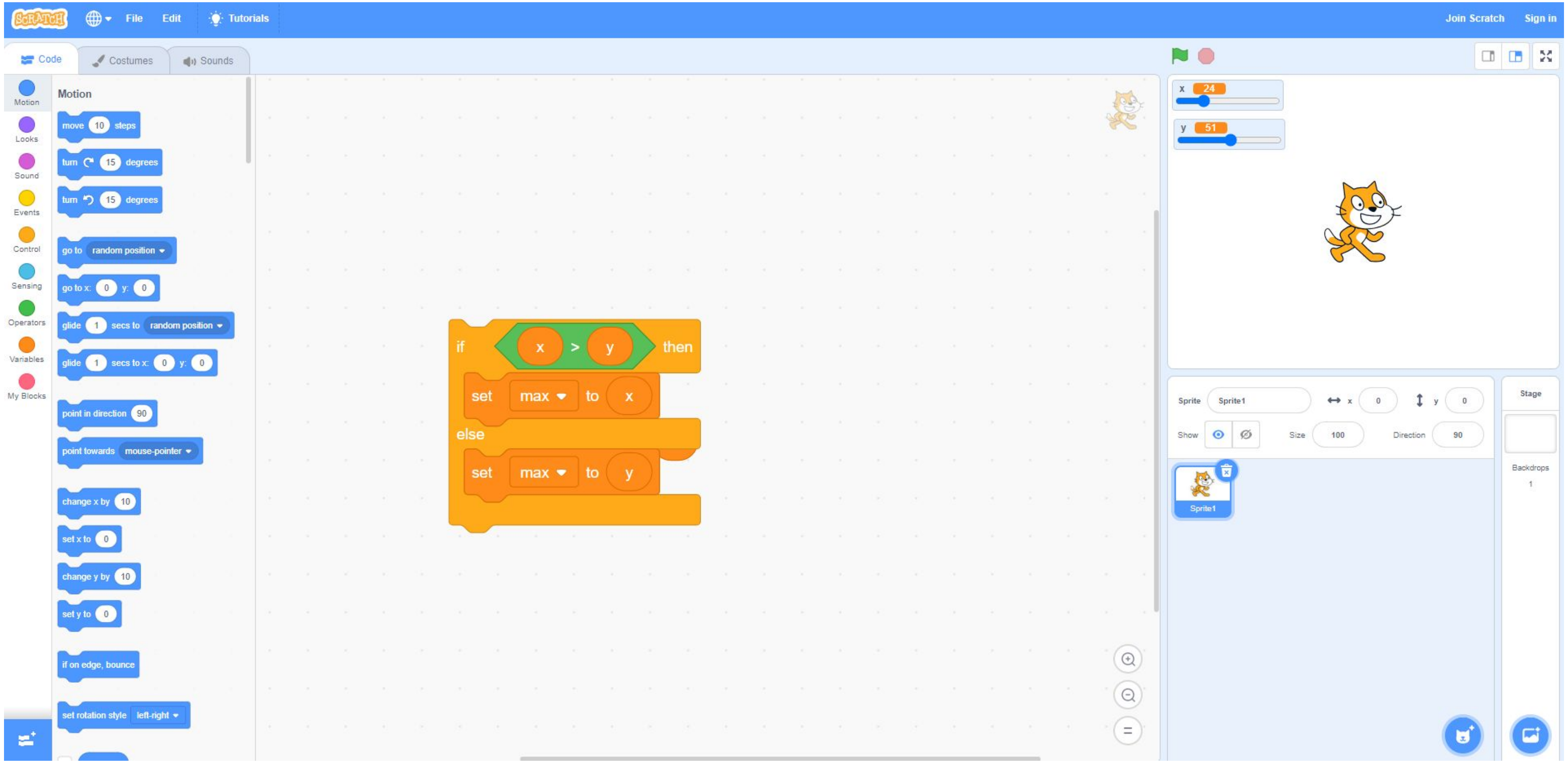
Structured Representation	Underlying HTML
Hello, World!	`<p>Hello, World!</p>`
Hello, **World**!	`<p>Hello, <b>World</b>!</p>`

Background - Structured Editing

Varies in scope and scale

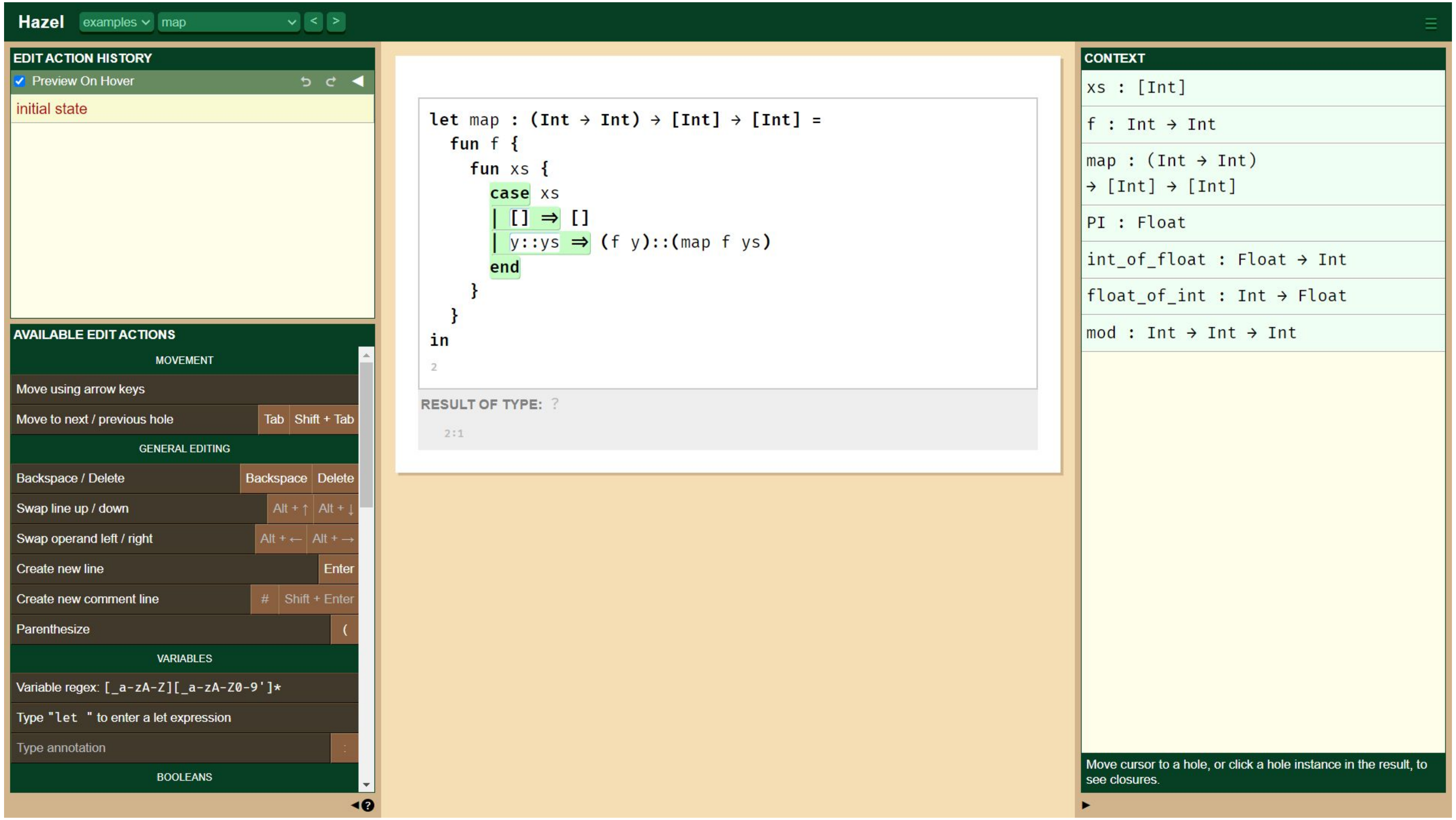
Plain Text Editors	Text Editors with Some Structured Editing Support	Fully-Fledged Structured Editors
Windows Notepad	JetBrains IDEs (e.g. IntelliJ IDEA, PyCharm)	WYSIWYG Editors (What You See Is What You Get) (e.g. PowerPoint, Wix)
	Other IDEs (e.g. Emacs, VSCode)	Scratch, Hazel

Background - Structured Editing - Scratch



```
if x > y then
    set max to x
else
    set max to y
```

Background - Structured Editing - Hazel



EDIT ACTION HISTORY

Preview On Hover

initial state

AVAILABLE EDIT ACTIONS

MOVEMENT

Move using arrow keys	
Move to next / previous hole	Tab / Shift + Tab

GENERAL EDITING

Backspace / Delete	Backspace / Delete
Swap line up / down	Alt + ↑ / Alt + ↓
Swap operand left / right	Alt + ← / Alt + →
Create new line	Enter
Create new comment line	# / Shift + Enter
Parenthesize	(

VARIABLES

Variable regex: `[_a-zA-Z][_a-zA-Z0-9']*`

Type "`let `" to enter a let expression

Type annotation :

BOOLEANS

```
let map : (Int → Int) → [Int] → [Int] =
  fun f {
    fun xs {
      case xs
      | [] ⇒ []
      | y::ys ⇒ (f y)::(map f ys)
      end
    }
  }
in
2
```

RESULT OF TYPE: ?

```
2:1
```

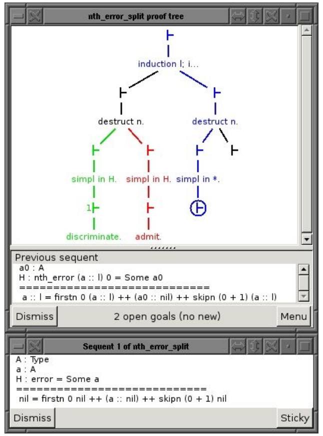
CONTEXT

```
xs : [Int]
f : Int → Int
map : (Int → Int)
→ [Int] → [Int]
PI : Float
int_of_float : Float → Int
float_of_int : Int → Float
mod : Int → Int → Int
```

Move cursor to a hole, or click a hole instance in the result, to see closures.

Background - Existing Approaches to Coq Interfaces

Prooftree, Proof-by-pointing, Actema, PeaCoq, Chick

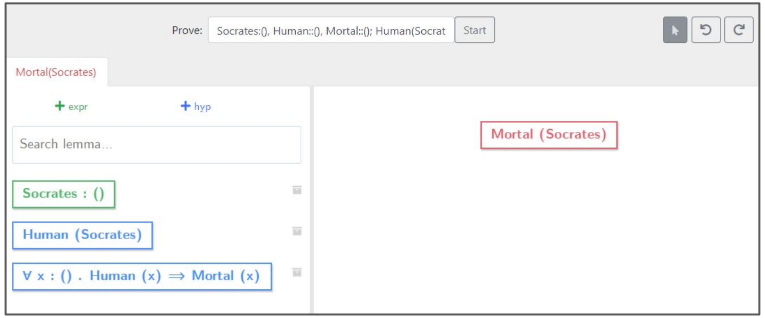


Previous sequent

```
 a0 : A
 H : nth_error (a :: l) 0 = Some a0
 ===============================
  a :: l = firstn 0 (a :: l) ++ (a0 :: nil) ++ skipn (0 + 1) (a :: l)
```

2 open goals (no new)

```
A : Type
a : A
H : error = Some a
==============================
 nil = firstn 0 nil ++ (a :: nil) ++ skipn (0 + 1) nil
```



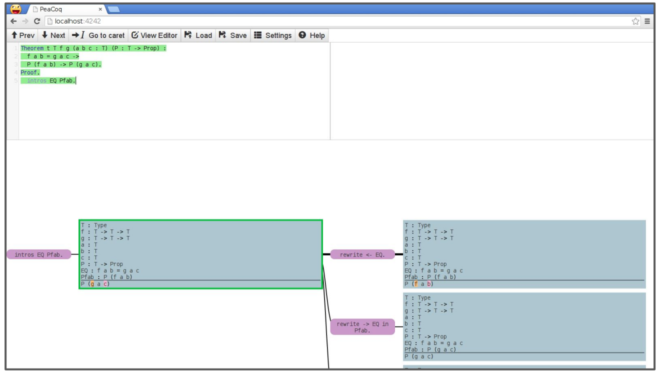
Socrates:(), Human::(), Mortal::(); Human(Socrat

Socrates : ()

Human (Socrates)

∀ x : () . Human (x) ⟹ Mortal (x)

Mortal (Socrates)



```
Theorem t T f g (a b c : T) (P : T -> Prop) :
  f a b = g a c ->
  P (f a b) -> P (g a c).
Proof.
  intros EQ Pfab.
```

```
T : Type
f : T -> T -> T
g : T -> T -> T
a : T
b : T
c : T
P : T -> Prop
EQ : f a b = g a c
Pfab : P (f a b)
P (g a c)
```

```
T : Type
f : T -> T -> T
g : T -> T -> T
a : T
b : T
c : T
P : T -> Prop
EQ : f a b = g a c
Pfab : P (f a b)
P (f a b)
```

```
T : Type
f : T -> T -> T
g : T -> T -> T
a : T
b : T
c : T
P : T -> Prop
EQ : f a b = g a c
Pfab : P (g a c)
P (g a c)
```

Some intended for advanced users

Some unrelated to Coq (separate custom system)

Some old and no longer maintained

Objective

- To explore the use of structured editing in writing Coq proofs by building an interactive GUI, and evaluate whether it can help alleviate the pain points

Solution

Solution - Methods

Text editor with structured editing support vs **fully-fledged structured editor**

Desktop app vs **online web app**

Backend Coq API: jsCoq

Frontend library: Blockly

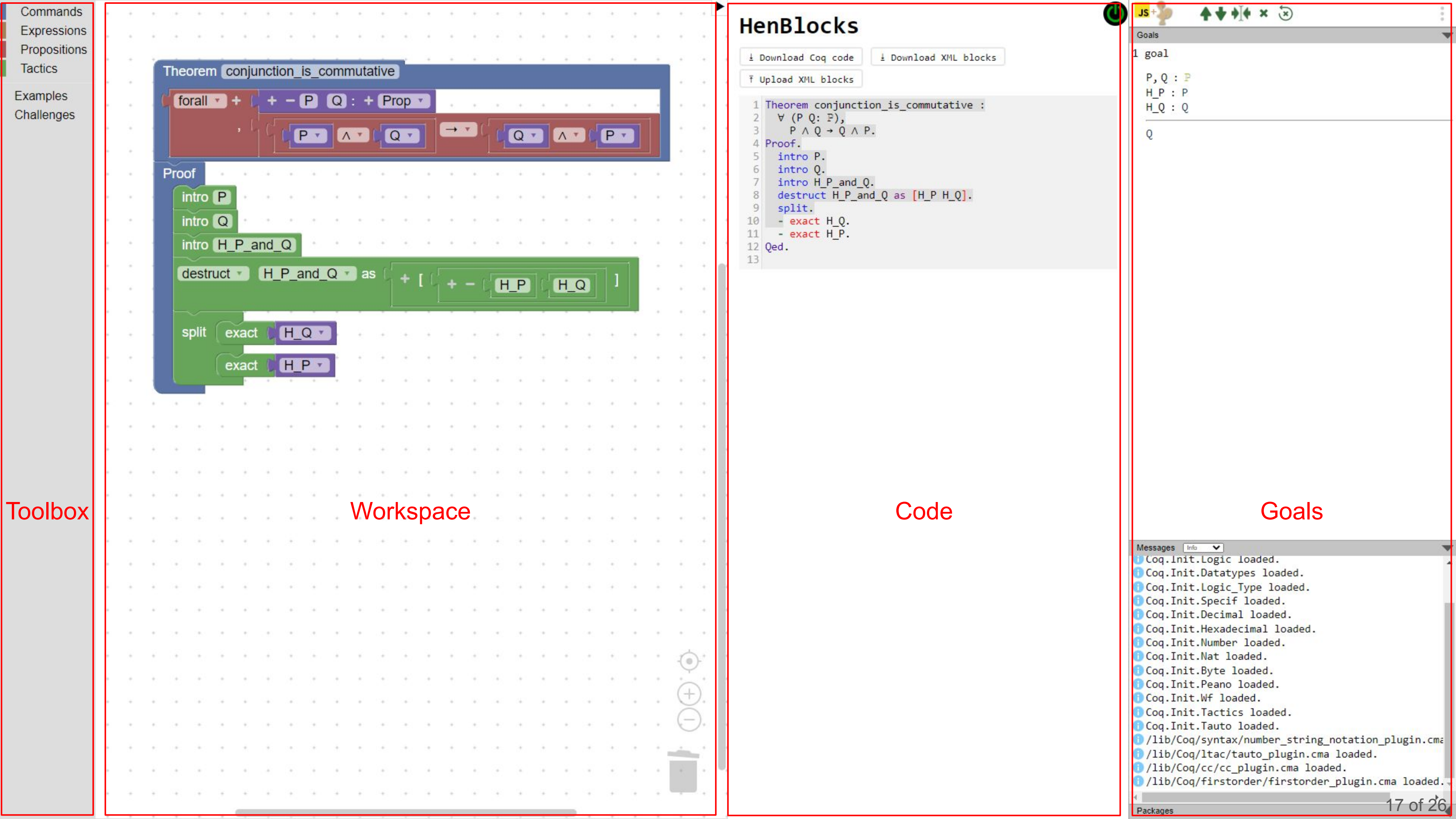
Built on Blockly

Solution - HenBlocks

Target audience: Undergrad students with experience in functional programming but little/no experience in proving

Use case: Learn, discover, and practise proving, and eventually transition to writing textual proofs with text editors

Available at https://henblocks.github.io (desktop only)



```
Theorem conjunction_is_commutative :
  ∀ (P Q: ℙ),
    P ∧ Q → Q ∧ P.
Proof.
  intro P.
  intro Q.
  intro H_P_and_Q.
  destruct H_P_and_Q as [H_P H_Q].
  split.
  - exact H_Q.
  - exact H_P.
Qed.
```

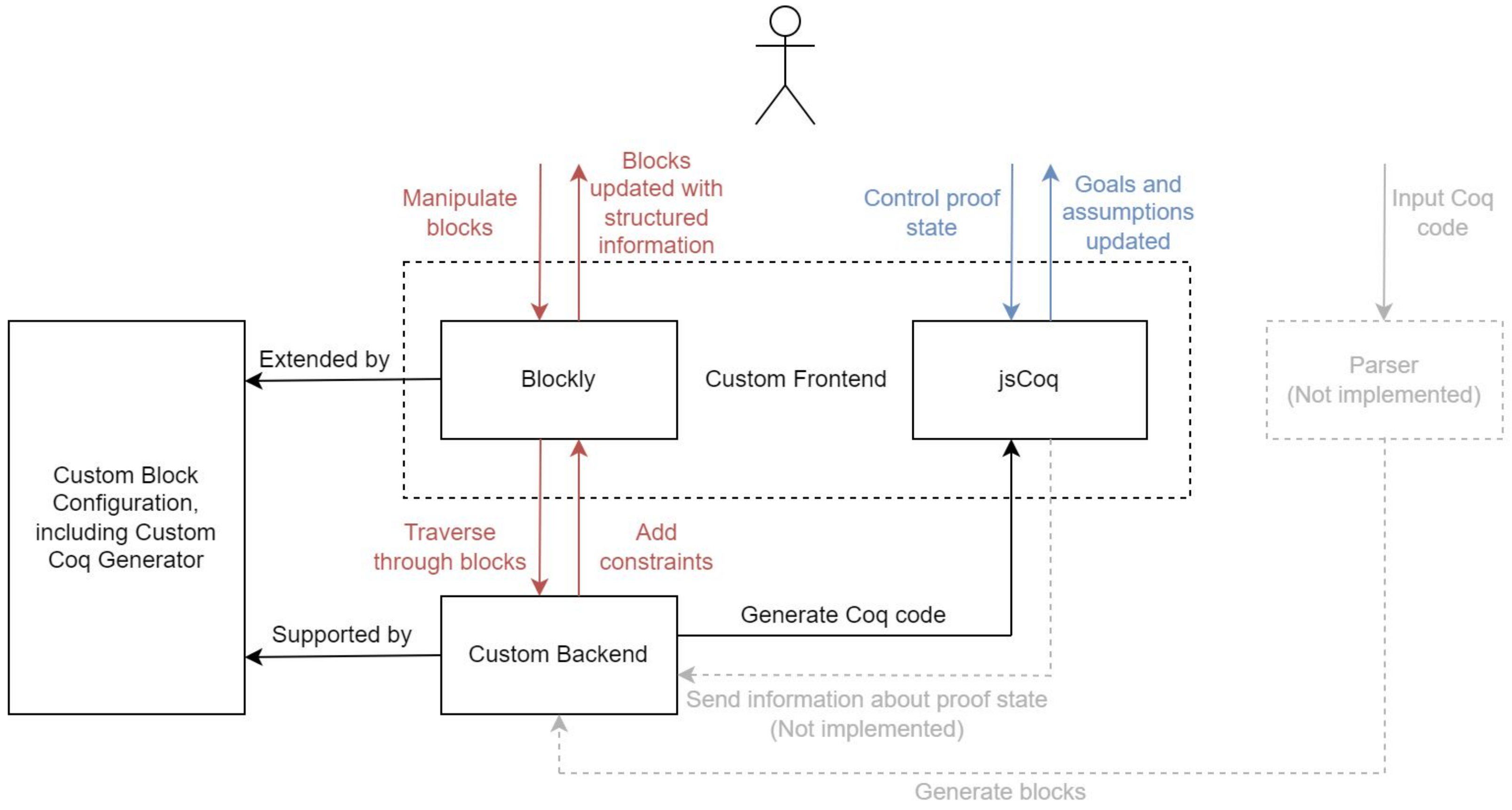
```
1 goal

  P, Q : ℙ
  H_P : P
  H_Q : Q
  ______________
  Q
```

```
Coq.Init.Logic loaded.
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Peano loaded.
Coq.Init.Wf loaded.
Coq.Init.Tactics loaded.
Coq.Init.Tauto loaded.
/lib/Coq/syntax/number_string_notation_plugin.cma
/lib/Coq/ltac/tauto_plugin.cma loaded.
/lib/Coq/cc/cc_plugin.cma loaded.
/lib/Coq/firstorder/firstorder_plugin.cma loaded.
```

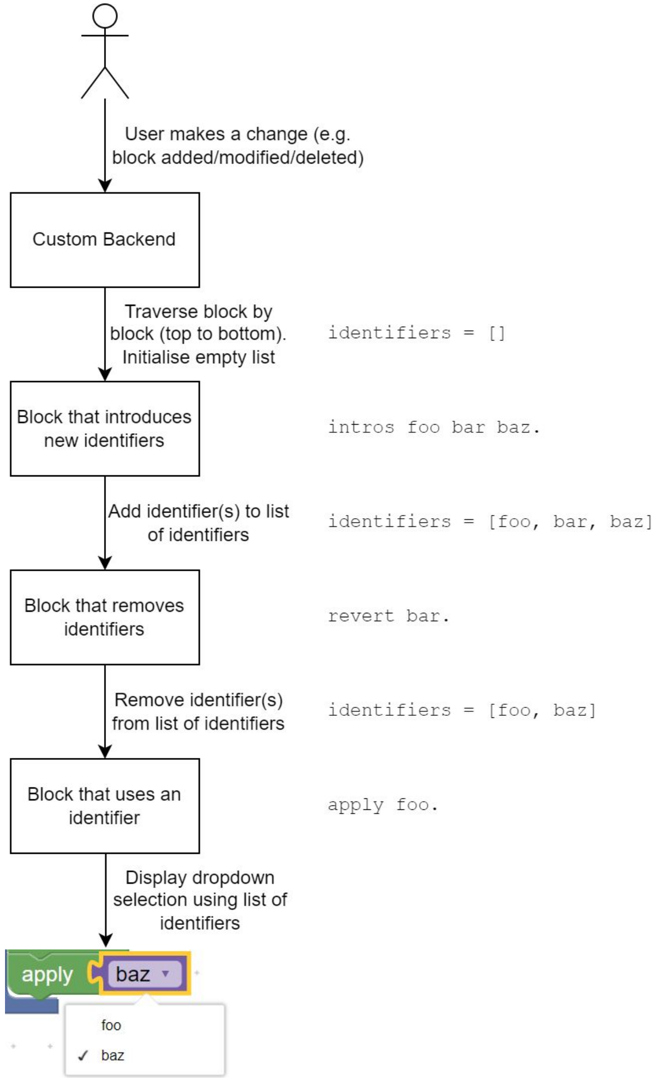
Advanced Features

- Variable Dropdowns
- Automatic Renaming of Variables
- Automatic Slots for Subgoals

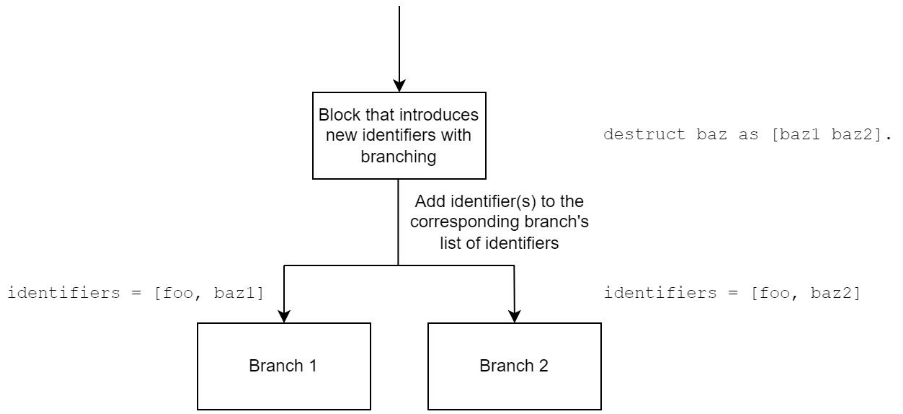
HenBlocks Architecture Diagram



Variable Dropdowns Diagram


User makes a change (e.g. block added/modified/deleted)
Custom Backend
Traverse block by block (top to bottom). Initialise empty list
Block that introduces new identifiers
Add identifier(s) to list of identifiers
Block that removes identifiers
Remove identifier(s) from list of identifiers
Block that uses an identifier
Display dropdown selection using list of identifiers
apply
baz
foo
✓ baz

```
identifiers = []

intros foo bar baz.

identifiers = [foo, bar, baz]

revert bar.

identifiers = [foo, baz]

apply foo.
```


Block that introduces new identifiers with branching
Add identifier(s) to the corresponding branch's list of identifiers
Branch 1
Branch 2

```
destruct baz as [baz1 baz2].

identifiers = [foo, baz1]

identifiers = [foo, baz2]
```

Discussion

Discussion - Findings

- Existing interfaces for Coq have drawbacks
- Existing (non-Coq) fully-fledged structured editors do not go far enough
- We can go further than removing syntax errors by reducing semantic errors
    - HenBlocks attempts to resolve the limitations, and alleviate the pain points
- We have to make compromises and simplifications to achieve a flatter learning curve

Discussion - Limitations

Potential for visual clutter

Slower than typing

Limited vocabulary

Discussion - Future Work

Testing

A/B
Longitudinal
(Randomised Control)

Development

Parsing Coq code to generate blocks

User Experience

Customisation for teaching

Conclusion

- Novel Contributions
    - Applied fully-fledged structured editing to proof writing
    - Developed advanced structured editing features

Fully-fledged structured editing is a promising approach to proof writing that warrants more exploration, development, and testing.

References

Sebastian Böhne and Christoph Kreitz. “Learning how to Prove: From the Coq Proof Assistant to Textbook Style”. In: *Electronic Proceedings in Theoretical Computer Science* 267 (2018), pp. 1–18. DOI: 10.4204/eptcs. 267.1.

Neil Fraser. “Ten things we’ve learned from Blockly”. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond).* 2015, pp. 49–50. DOI: 10.1109/BLOCKS.2015.7369000.

Valentin Robert. “Front-end tooling for building and maintaining dependently-typed functional programs”. PhD thesis. University of California San Diego, 2018.

Acknowledgements

Michael D. Adams
Olivier Danvy
Emilio Jesús Gallego Arias

Full Report

bboey.com/henblocks