Learn-Heptagon: a web application to teach dataflow synchronous programming

Basile Pesin
Fédération ENAC ISAE-SUPAERO ONERA, Université de Toulouse, France

SYNCHRON 2025

Context

Teaching Dataflow Synchronous Programming in Lustre:

- at Sorbonne Université: M2 students
- at Ecole Nationale de l'Aviation Civile: engineers (not CS)

Context

Teaching Dataflow Synchronous Programming in Lustre:

- at Sorbonne Université: M2 students
- at Ecole Nationale de l'Aviation Civile: engineers (not CS)

Technical difficulty: installing dependencies

- Lustre v6/Heptagon + Kind 2 ...
- Short modules (8 hours): no time to waste
- Some students have Windows machines, some none at all
- Very difficult to install anything on ENAC's machines

Context

Teaching Dataflow Synchronous Programming in Lustre:

- at Sorbonne Université: M2 students
- at Ecole Nationale de l'Aviation Civile: engineers (not CS)

Technical difficulty: installing dependencies

- Lustre v6/Heptagon + Kind 2 ...
- Short modules (8 hours): no time to waste
- Some students have Windows machines, some none at all
- Very difficult to install anything on ENAC's machines

Solution: work in a web browser

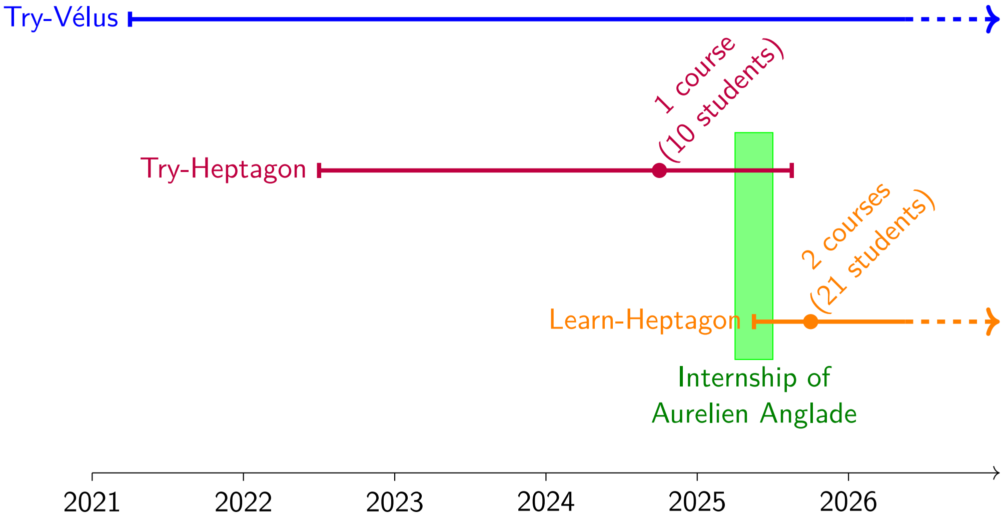
- Existing app: `https://kind.cs.uiowa.edu/app/` for Kind 2
- Limited to Kind 2 syntax + no exercise structure
- Let's develop our own (inspired by Learn-OCaml)

Demo time!

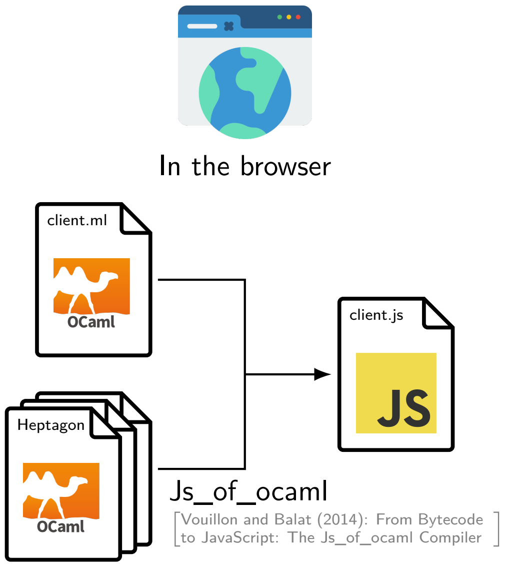
Follow along on

`http://learn-heptagon.vertmo.org`

Project history



How does it work ? Overview

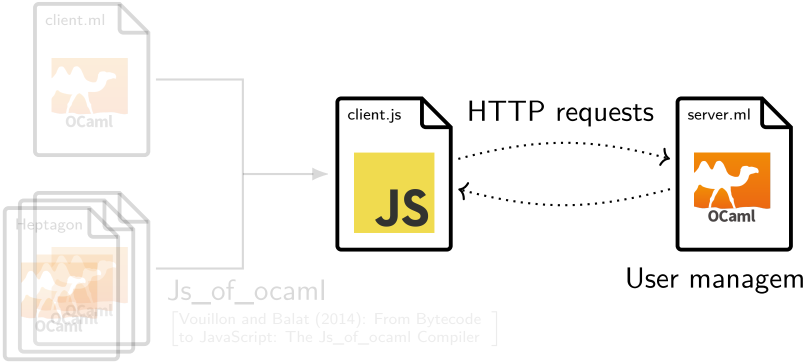


[Vouillon and Balat (2014): From Bytecode to JavaScript: The Js_of_ocaml Compiler]

How does it work ? Overview

In the browser

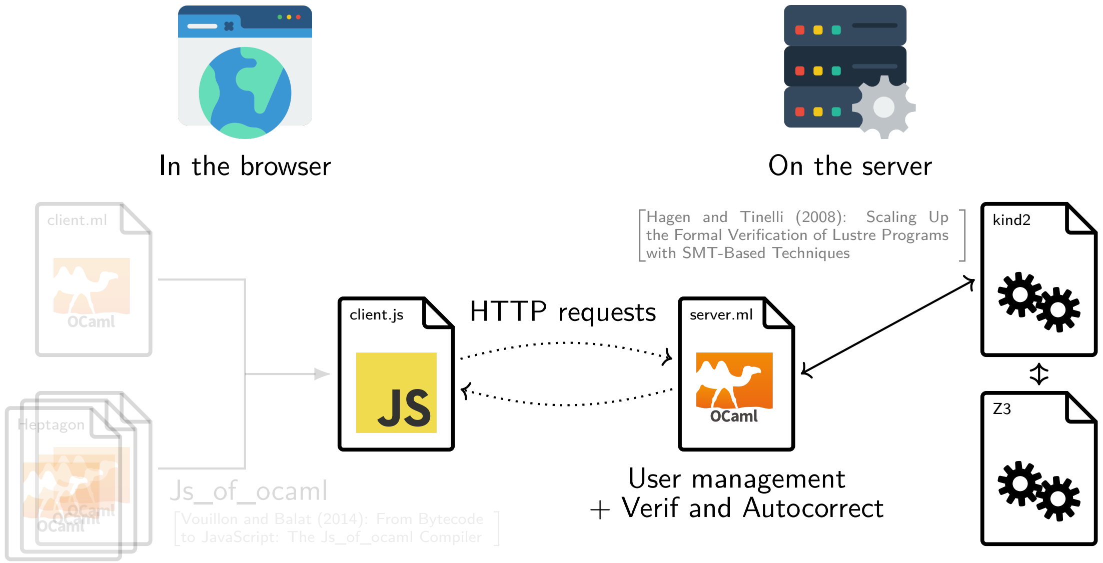
On the server



[Vouillon and Balat (2014): From Bytecode to JavaScript: The Js_of_ocaml Compiler]

User management

How does it work ? Overview


In the browser
On the server
client.ml
Heptagon
Js_of_ocaml
[Vouillon and Balat (2014): From Bytecode to JavaScript: The Js_of_ocaml Compiler]
client.js
HTTP requests
server.ml
User management
+ Verif and Autocorrect
[Hagen and Tinelli (2008): Scaling Up the Formal Verification of Lustre Programs with SMT-Based Techniques]
kind2
Z3

How does it work ? Interpreter

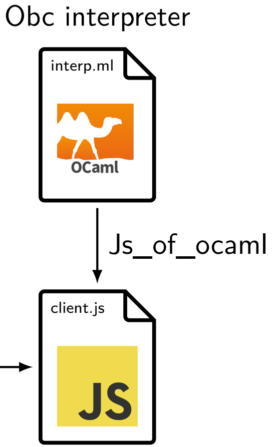
```
node counter(x : int) returns (y : int)
let
  y = x + (0 fby y)
tel
```

↓ Heptagon front/middle end

```
machine counter =
  var v: int;

  step(x: int) returns (y: int) {
    y = ((+) x mem(v));
    mem(v) = y
  }

  reset() returns () {
    mem(v) = 0
  }
```



How does it work ? Interpreter

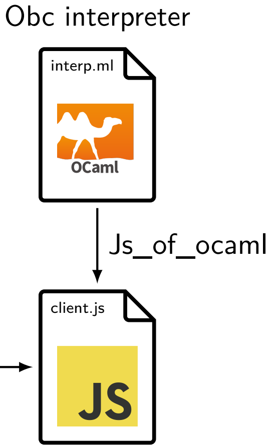
```
node counter(x : int) returns (y : int)
let
  y = x + (0 fby y)
tel
```

↓ Heptagon front/middle end

```
machine counter =
  var v: int;

  step(x: int) returns (y: int) {
    y = ((+) x mem(v));
    mem(v) = y
  }

  reset() returns () {
    mem(v) = 0
  }
```



Not efficient! Especially for programs with arrays

How does it work ? Interpreter

```
node counter(x : int) returns (y : int)
let
  y = x + (0 fby y)
tel
```

Heptagon front/middle end

```
machine counter =
  var v: int;

  step(x: int) returns (y: int) {
    y = ((+) x mem(v));
    mem(v) = y
  }

  reset() returns () {
    mem(v) = 0
  }
```

Obc to JS backend

```
class Counter {
  constructor() { this.v = 0; }

  step(x) {
    let y = null;
    y = (x + this.v);
    this.v = y;
    return y;
  }

  reset() { this.v = 0; }
}
```

printing

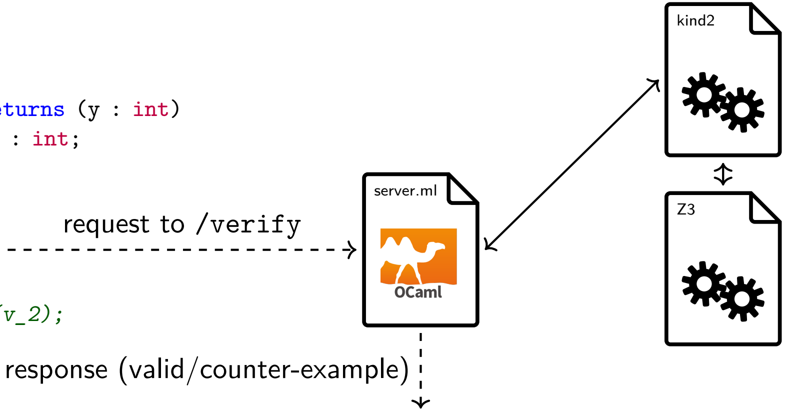
```
eval("...;␣new␣Counter()")
```

How does it work ? Verification and Autocorrect

```
node counter(x : int) returns (y : int)
contract
  assume always(x >= 0)
  enforce y >= 0
let
  y = x + (0 fby y)
tel
```

Kind2 printer

```
node counter(x : int) returns (y : int)
var v_1, v_2 : bool; v_3 : int;
let
  v_2 = (y >= 0);
  v_1 = always(x >= 0);
  y = (x + v_3);
  v_3 = 0 -> pre y;
  --%PROPERTY (v_1) => (v_2);
tel
```



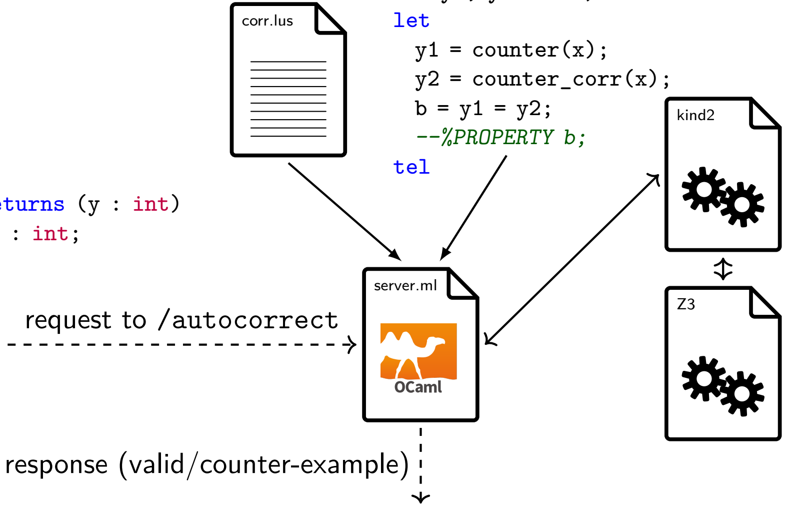
How does it work ? Verification and Autocorrect

```
node counter(x : int) returns (y : int)
contract
  assume always(x >= 0)
  enforce y >= 0
let
  y = x + (0 fby y)
tel
```

Kind2 printer

```
node counter(x : int) returns (y : int)
var v_1, v_2 : bool; v_3 : int;
let
  v_2 = (y >= 0);
  v_1 = always(x >= 0);
  y = (x + v_3);
  v_3 = 0 -> pre y;
tel
```

```
node eq(x: int) returns (b: bool)
var y1, y2 : int;
let
  y1 = counter(x);
  y2 = counter_corr(x);
  b = y1 = y2;
  --%PROPERTY b;
tel
```



Experience report

First used this year with two classes

- 2nd year of Master IATSED (8 students)
- 3rd year of ENAC engineer students-apprenctices (13 students)

What went...

Well :)

- Straight to the action!
- No big usability problem

Less well...

- Heptagon's syntax (floating points, weird priority rules)
- Error messages (Syntax Error ?)
- Found some bugs !

Experience report

First used this year with two classes

- 2nd year of Master IATSED (8 students)
- 3rd year of ENAC engineer students-apprenctices (13 students)

What went...

Well :)

- Straight to the action!
- No big usability problem

Less well...

- Heptagon's syntax (floating points, weird priority rules)
- Error messages (Syntax Error ?)
- Found some bugs !

Bugs found:

- Bug in the server software – fixed
- Limitations of the autocorrect – understood, need some redesign
- Nasty bug with Heptagon's array updates – TODO with the Parisians

Future improvements

Some improvements:

- Better handling of saving:
  - Currently, the notebook is saved (sent to the server) at every change
  - Lots of requests (and writes on the server) !
  - Some considered approaches:
    - Saving manually $\Rightarrow$ students might forget and cry later
    - Saving when window is left/closed $\Rightarrow$ does not work consistently

Future improvements

Some improvements:

- Better handling of saving:
  - Currently, the notebook is saved (sent to the server) at every change
  - Lots of requests (and writes on the server) !
  - Some considered approaches:
    - Saving manually ⇒ students might forget and cry later
    - Saving when window is left/closed ⇒ does not work consistently
- Translate Heptagon features for Kind 2 ?
  - Control structures ⇒ when/merge ⇒ renormalize ?
  - Records ⇒ flatten ?
  - Arrays ⇒ translate or flatten ?
- Notebook structure
  - Dependencies between cells (all or selected ?)
  - Need a redesign of the client + notebooks config files

How you can use it / install it / improve it

Please don't use my instance for a big group !

- currently hosted on the **lowest tier of VPS** at a french hosting company
- I do not know how it scales !

How you can use it / install it / improve it

Please don't use my instance for a big group !

- currently hosted on the **lowest tier of VPS** at a french hosting company
- I do not know how it scales !

Everything you need to install/customise your own:

- Server: `https://github.com/learn-heptagon/learn-heptagon-server`
- Client: `https://github.com/learn-heptagon/learn-heptagon`
- a server with OCaml + Z3 + a few opam dependencies

How you can use it / install it / improve it

Please don't use my instance for a big group !

- currently hosted on the **lowest tier of VPS** at a french hosting company
- I do not know how it scales !

Everything you need to install/customise your own:

- Server: `https://github.com/learn-heptagon/learn-heptagon-server`
- Client: `https://github.com/learn-heptagon/learn-heptagon`
- a server with OCaml + Z3 + a few opam dependencies

You can add your own notebooks with

- `.html` and `.lus` files for the text / editor cells
- a JSON file that gives the sequence of cells

Feel free to reach out if you need help, and to share your exercises :)