Designing a programming language for local reasoning and easy debugging

Whoami

- Robin Heggelund Hansen
- Consultant at Bekk
- Creator of Gren

Gren

- Is a purely functional, staticly typed programming language with ML-syntax
- It aims to be small and easy to learn, while still being performant and expressive enough for general use
- Gren targets JavaScript, for maxium portability
- Still in early development

Why?

There is no such thing as a perfect programming language …

… but you can get pretty close within a specific domain

The sort of projects I work on

- Big corporations in public and private sectors
- Not technically advanced (rest services)
- Relatively low-traffic
- Important services that can make the news when there are issues

An interesting thing about consultants

- Don't tend to stick around
- I tend to work in codebases I have little experience with

My ideal programming language

- Make it easy to understand what code does and doesn't do, without requiring that I know the entire codebase
- Has guarantees and tooling that make it easy to pinpoint and fix problems
- Performance isn't terribly important, but it shouldn't get in my way

My ideal programming language

Enables local reasoning, and easy debugging

Local reasoning

What does that mean?

Error handling

```
canViewAccount : Request -> Account -> Bool
canViewAccount req account =
    let
        userDetails =
            decodeUserDetails req
    in
    case userDetails of
        Admin _ ->
            True

        User details ->
            List.member account.id details.accountIds
```

Managing side-effects

```
loadFromCache :: Key -> IO (Maybe Value)
```

Managing side-effects

```
module FileSystem (..)

openForRead : Permission -> String -> Task AccessError (ReadableFileHandle a)
```

Tradeoffs

- Code size
- But to me, that is a tradeoff worth making

Debugging

When local reasoning won't do

Step-Debuggers are useful

- Makes for easy exploration of the running application
- Makes it easier to learn how the language works
- Sometimes, reasoning fails

Challenges of lazy evaluation

```
encodeHelp :: Int -> String -> String
encodeHelp num acc =
  let clamped =
        num .&. 31

      newNum =
        num `Bit.shiftR` 5

      newClamped =
        if newNum > 0
          then clamped .|. 32
          else clamped

      newAcc =
        base64Table ! newClamped : acc
   in if newNum > 0
        then encodeHelp newNum newAcc
        else List.reverse newAcc
```

Challenges of lazy evaluation

```
encodeHelp :: Int -> String -> String
encodeHelp num acc =
  let clamped =
        num .&. 31

      newNum =
        num `Bit.shiftR` 5

      newClamped =
        if newNum > 0
          then clamped .|. 32
          else clamped

      newAcc =
        base64Table ! newClamped : acc
   in if newNum > 0 --< BREAK HERE
        then encodeHelp newNum newAcc
        else List.reverse newAcc
```

Challenges of lazy evaluation

```
encodeHelp :: Int -> String -> String
encodeHelp num acc =
  let clamped =
        num .&. 31

      newNum =
        num `Bit.shiftR` 5 --< STEPS TO HERE

      newClamped =
        if newNum > 0
          then clamped .|. 32
          else clamped

      newAcc =
        base64Table ! newClamped : acc
   in if newNum > 0
        then encodeHelp newNum newAcc
        else List.reverse newAcc
```

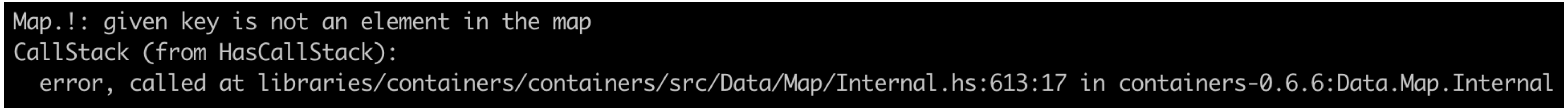
Challenges of lazy evaluation

```
encodeHelp :: Int -> String -> String
encodeHelp num acc =
  let clamped =
        num .&. 31

      newNum =
        num `Bit.shiftR` 5

      newClamped =
        if newNum > 0
          then clamped .|. 32
          else clamped

      newAcc =
        base64Table ! newClamped : acc
   in if newNum > 0
        then encodeHelp newNum newAcc
        else List.reverse newAcc
```

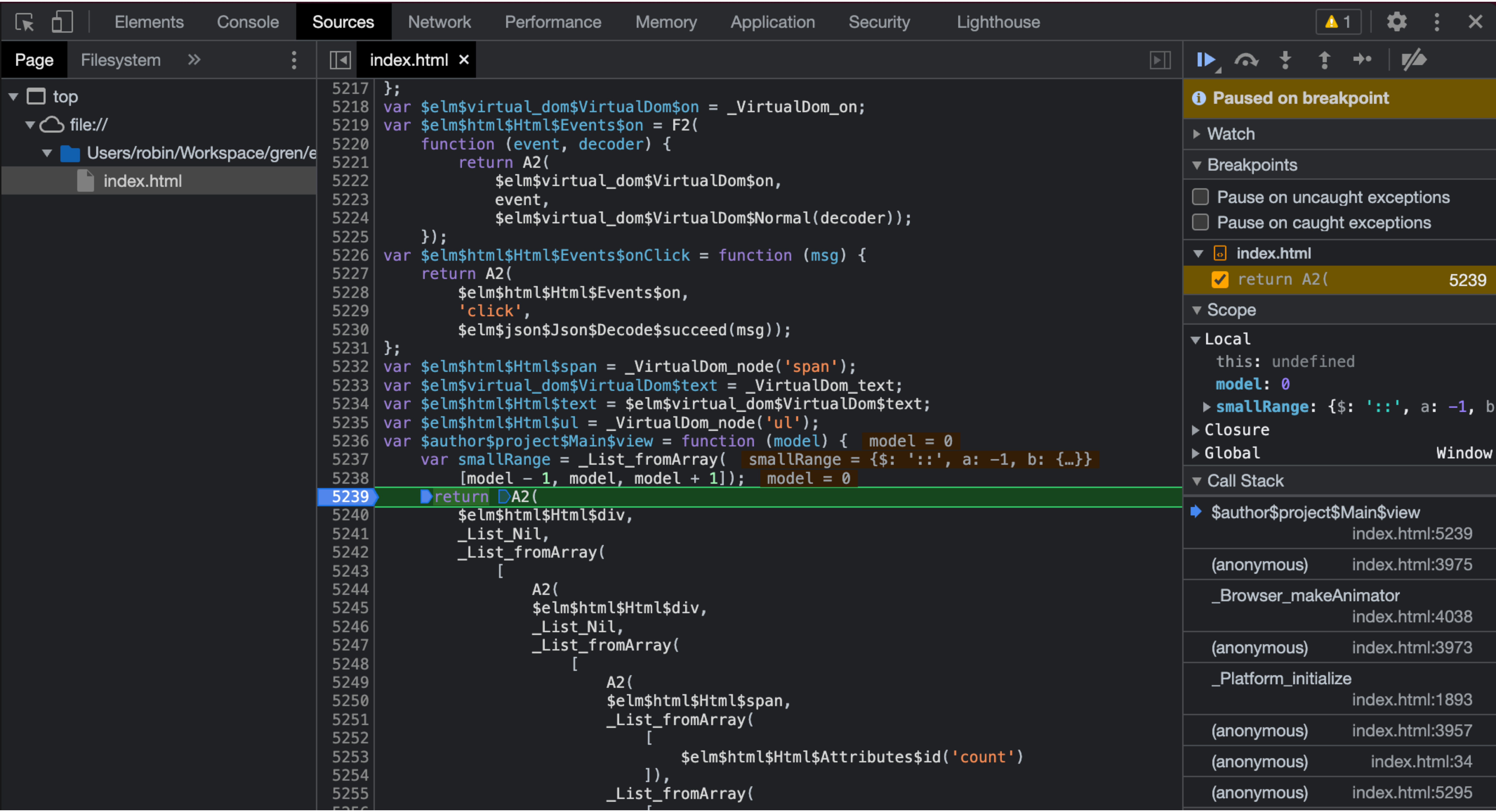
Challenges of lazy evaluation

```haskell
encodeHelp :: Int -> String -> String
encodeHelp num acc =
  let clamped =
        num .&. 31

      newNum =
        num `Bit.shiftR` 5

      newClamped =
        if newNum > 0
          then clamped .|. 32
          else clamped

      newAcc =
        base64Table ! newClamped : acc
   in if newNum > 0
        then encodeHelp newNum newAcc
        else List.reverse newAcc --< NOW HERE
```

Importance of stack traces



```
Map.!: given key is not an element in the map
CallStack (from HasCallStack):
  error, called at libraries/containers/containers/src/Data/Map/Internal.hs:613:17 in containers-0.6.6:Data.Map.Internal
```



```
};
var $elm$virtual_dom$VirtualDom$on = _VirtualDom_on;
var $elm$html$Html$Events$on = F2(
    function (event, decoder) {
        return A2(
            $elm$virtual_dom$VirtualDom$on,
            event,
            $elm$virtual_dom$VirtualDom$Normal(decoder));
    });
var $elm$html$Html$Events$onClick = function (msg) {
    return A2(
        $elm$html$Html$Events$on,
        'click',
        $elm$json$Json$Decode$succeed(msg));
};
var $elm$html$Html$span = _VirtualDom_node('span');
var $elm$virtual_dom$VirtualDom$text = _VirtualDom_text;
var $elm$html$Html$text = $elm$virtual_dom$VirtualDom$text;
var $elm$html$Html$ul = _VirtualDom_node('ul');
var $author$project$Main$view = function (model) {  model = 0
    var smallRange = _List_fromArray(  smallRange = {$: '::', a: -1, b: {…}}
        [model - 1, model, model + 1]);  model = 0
    return A2(
        $elm$html$Html$div,
        _List_Nil,
        _List_fromArray(
            [
                A2(
                $elm$html$Html$div,
                _List_Nil,
                _List_fromArray(
                    [
                        A2(
                        $elm$html$Html$span,
                        _List_fromArray(
                            [
                                $elm$html$Html$Attributes$id('count')
                            ]),
                        _List_fromArray(
```

Paused on breakpoint

▸ Watch

▾ Breakpoints

☐ Pause on uncaught exceptions

☐ Pause on caught exceptions

▾ index.html

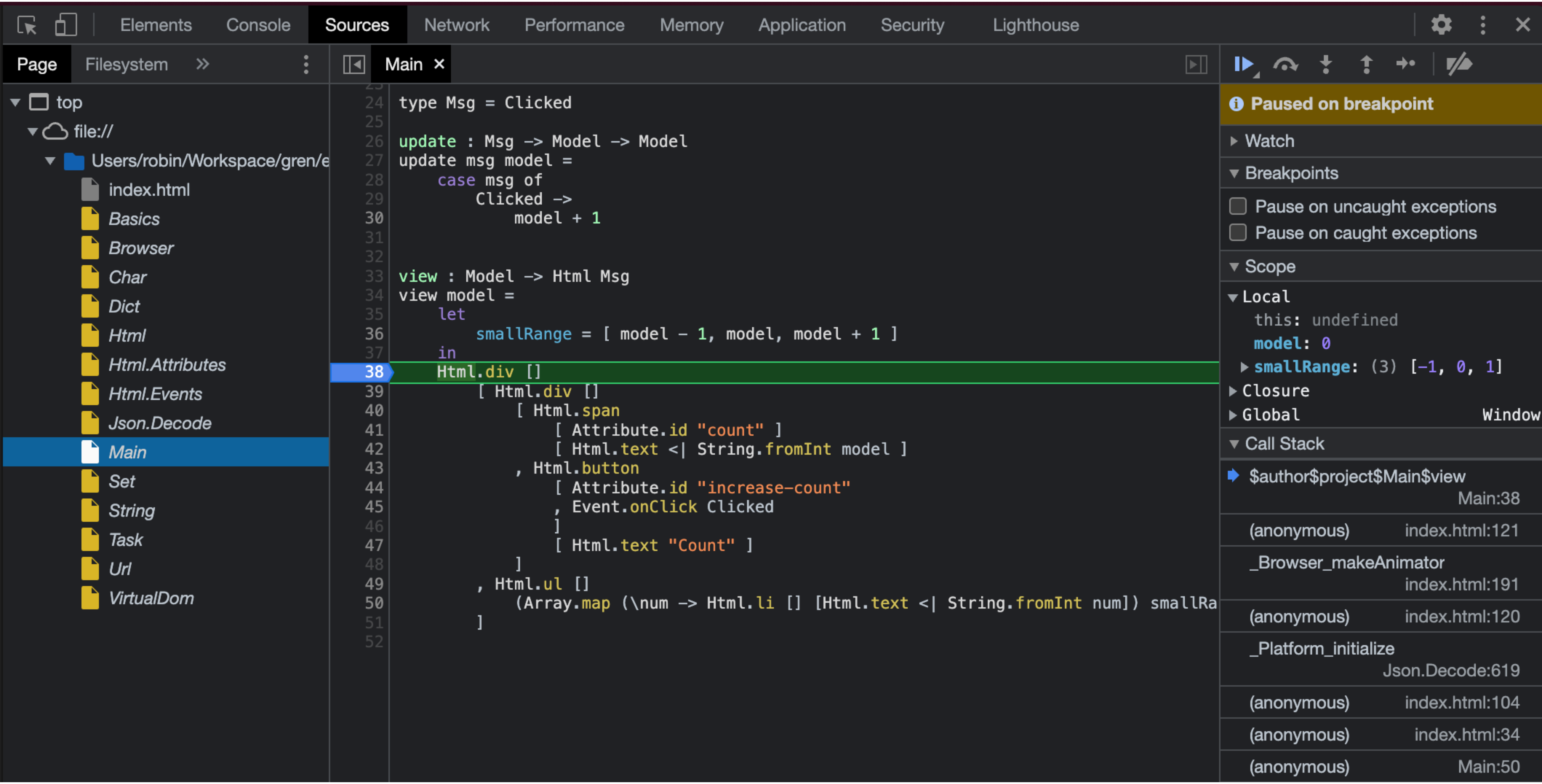
☑ return A2(5239

▾ Scope

```
▾ Local
    this: undefined
    model: 0
  ▸ smallRange: {$: '::', a: -1, b
▸ Closure
▸ Global                    Window
```

▾ Call Stack

```
$author$project$Main$view
                    index.html:5239
(anonymous)         index.html:3975
_Browser_makeAnimator
                    index.html:4038
(anonymous)         index.html:3973
_Platform_initialize
                    index.html:1893
(anonymous)         index.html:3957
(anonymous)         index.html:34
(anonymous)         index.html:5295
```



```
type Msg = Clicked

update : Msg -> Model -> Model
update msg model =
    case msg of
        Clicked ->
            model + 1


view : Model -> Html Msg
view model =
    let
        smallRange = [ model - 1, model, model + 1 ]
    in
    Html.div []
        [ Html.div []
            [ Html.span
                [ Attribute.id "count" ]
                [ Html.text <| String.fromInt model ]
            , Html.button
                [ Attribute.id "increase-count"
                , Event.onClick Clicked
                ]
                [ Html.text "Count" ]
            ]
        , Html.ul []
            (Array.map (\num -> Html.li [] [Html.text <| String.fromInt num]) smallRa
        ]

```

Paused on breakpoint

□ Pause on uncaught exceptions

□ Pause on caught exceptions

```
Local
  this: undefined
  model: 0
▸ smallRange: (3) [-1, 0, 1]
▸ Closure
▸ Global                    Window
```

```
$author$project$Main$view
                          Main:38
(anonymous)       index.html:121
_Browser_makeAnimator
                  index.html:191
(anonymous)       index.html:120
_Platform_initialize
                  Json.Decode:619
(anonymous)       index.html:104
(anonymous)       index.html:34
(anonymous)       Main:50
```

Tradeoffs

- Strict evaluation makes it easier to step through the code
- Readable stack traces make it easy to locate grivious errors
- Being able to debug the actual source code complicates and slows down the compiler
- Using the target platform's primitive types makes it easier to inspect state

Do we need a new language for this?

Problems with new languages

- Learning them takes time and commitment
- People usually have limited time to learn new things
- Few are willing to bet on a language without a future
- To be successfull the language needs to be small, and have a low complexity budget.
- Also, should be portable.

Why make a new language?

- Haskell is big and complex. Could do better on local reasoning and debugging.
- Elm is great! ... but it's hard to use for backends or terminal applications. Also, debugging experience could be better.

Gren

- Small, has simple but powerful features that compose, and aims to be learnable with a low time investment
- Great for local reasoning
- Integrates well with the JS debugger
- Can use it almost everywhere

Questions?

Gren: https://gren-lang.org

Mastodon: @robinheghan@snabelen.no