

Talk at Carnegie Mellon University, Dept. of Computer Science, October 16, 2018

James Carlson, jxxcarlson@gmail.com

MiniLatex Project

- **Bring Latex to the Browser**
- **Typed functional programming**

Elm

Evan Czaplicki, thesis 2012

- elm-lang.org
- **elm/parser**



Two parts of LaTeX

(1)

$$\begin{aligned} & \int_0^1 x^n dx \\ &= \frac{1}{n+1} \end{aligned}$$

$$\int_0^1 x^n dx = \frac{1}{n+1}$$

(2)

`\begin{theorem}` The indefinite integral of e^x is $e^x + C$. `\end{theorem}`

Theorem 2.1
The indefinite integral of e^x is $e^x + C$.

\strong{Newton} says

\begin{theorem}

The indefinite integral of e^x
is $e^x + C$.

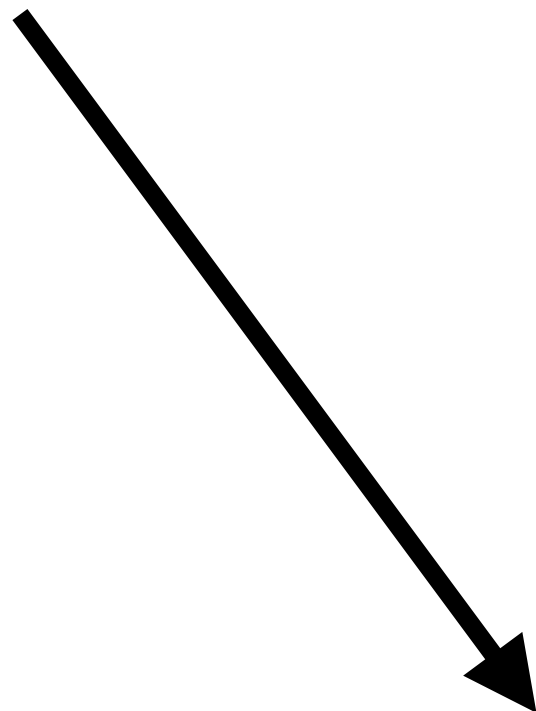
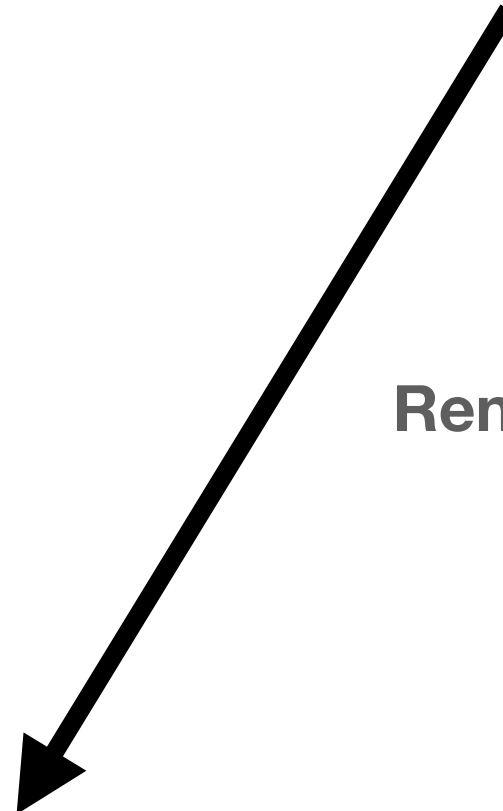
\end{theorem}

Parse



AST

Render



Newton says

Theorem 2.1

The indefinite integral of e^x is $e^x + C$.

What is its
Type?



Type of the AST

```
type LatexExpr
```

```
  = LXString String
```

```
  | InlineMath String
```

```
  | DisplayMath String
```

```
  | Macro String (List LatexExpr)
```

```
  | Environment String LatexExpr
```

```
  | LatexList (List LatexExpr)
```

```
  | Comment String
```

```
  | Item (LXString String)
```

```
  | LXError (List DeadEnd)
```

Once upon a time ...

$a^2 + b^2 = c^2$

$$a^2 + b^2 = c^2$$

Wow!

`\begin{foo} ... \end{foo}`

[a, b, c, ...]

% la di dah do day!

`\item whatever...`

Oops!

```
\strong{Newton} says  
\begin{theorem}  
The indefinite integral of  $e^x$   
is  $e^x + C$ .  
\end{theorem}
```



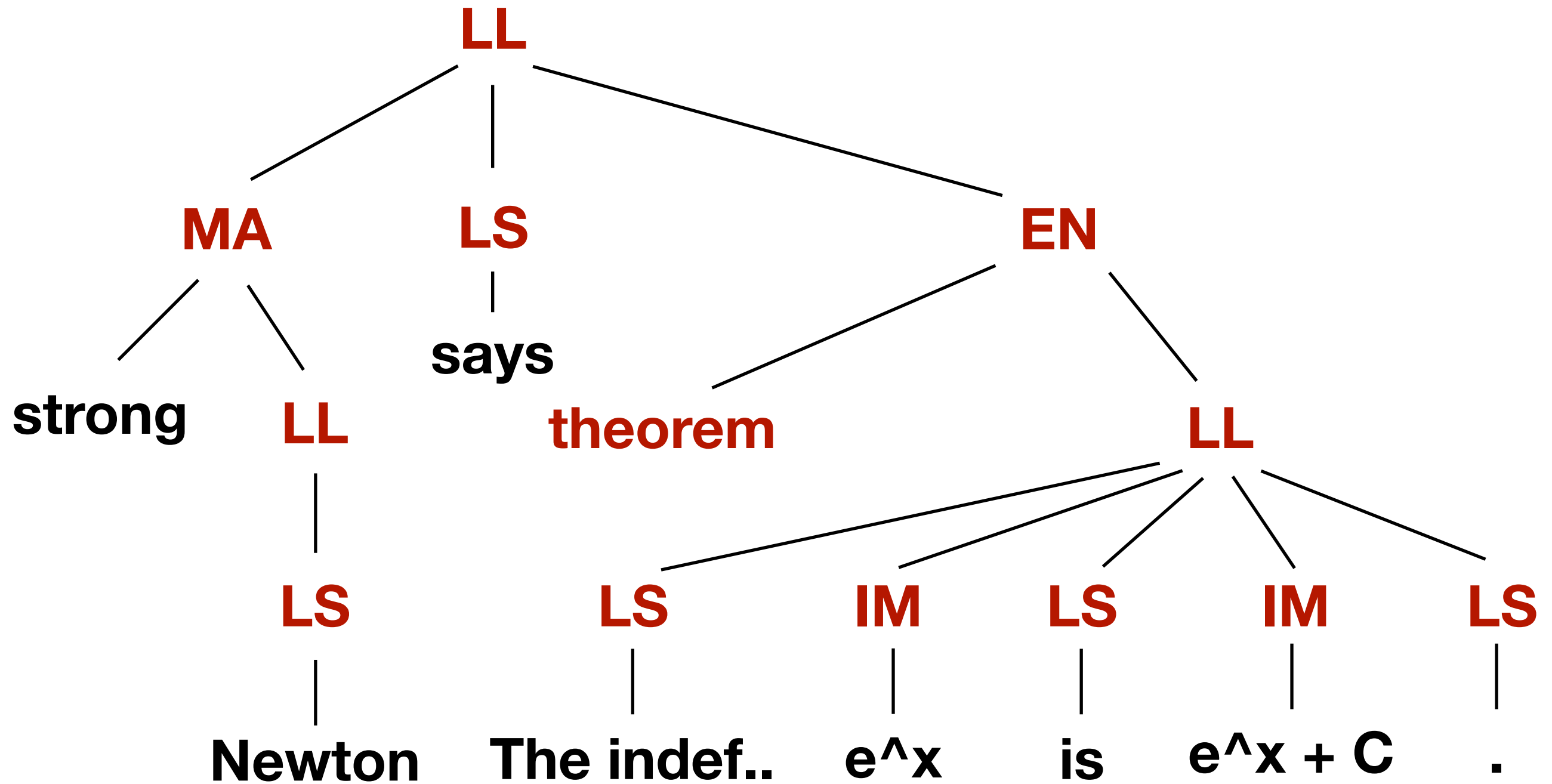
```
LatexList ( [  
  [Macro "strong" ( [LatexList ( [LXString "Newton" ] ) ] )  
    ,LXString "says"  
    ,Environment "theorem" (LatexList ( [  
      ,LXString "The indefinite integral of"  
      ,InlineMath "e^x"  
      ,LXString "is"  
      ,InlineMath "e^x + C"  
      ,LXString "." ] ) ) ]  
  ] )
```

Source



AST

AST



Grammar



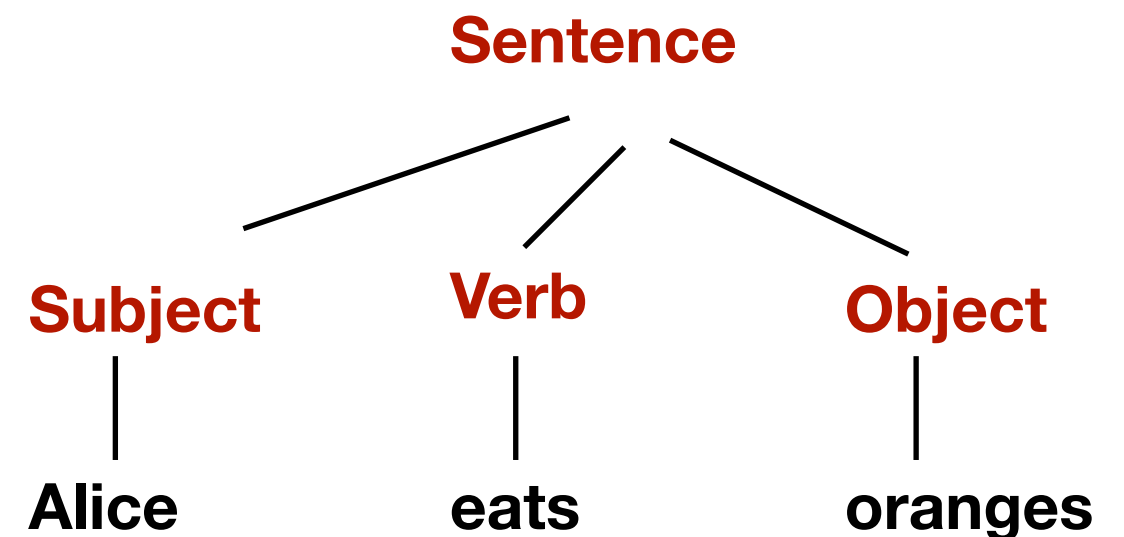
Rules

Sentence -> **Subject Verb Object**

Subject -> Alice | Bob

Verb -> eats | likes

Object -> apples | oranges



Alice eats oranges

Bob likes apples

MiniLaTeX Parser — Productions

LE \rightarrow **DM** | **IM** | **Macro** | **Words** | **LL** | **Env**

DM \rightarrow **\$\$ mathSymbols+ \$\$**

IM \rightarrow **\$ mathSymbols+ \$**

Words \rightarrow Word+

Word \rightarrow WS WordChar+ WS

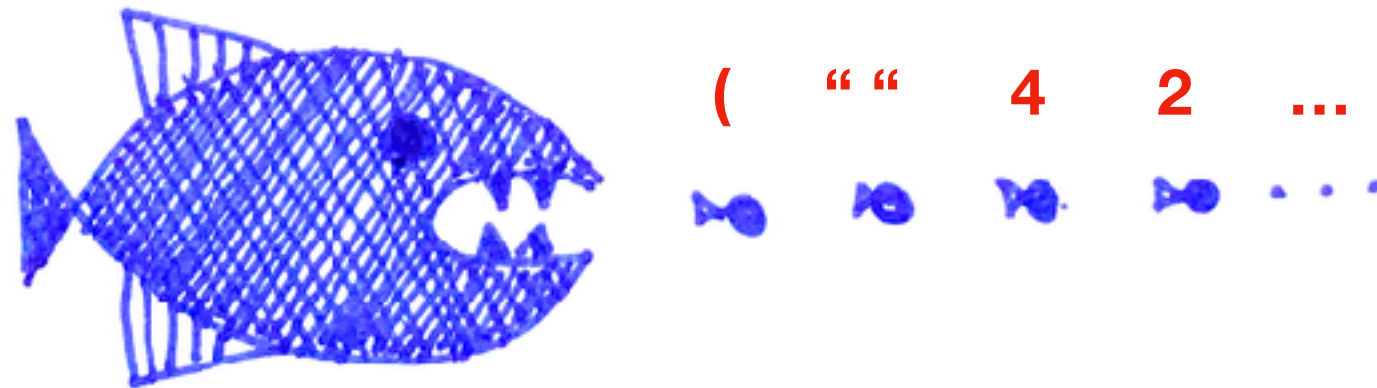
Macro \rightarrow **MacroName Arg* WS**

LL \rightarrow **[]** | **[LE]** | **LL ++ [LE]**

Env envName \rightarrow **\begin{ envName }**
LE **\end{ envName }**

Parsing

A kind of inverse of production

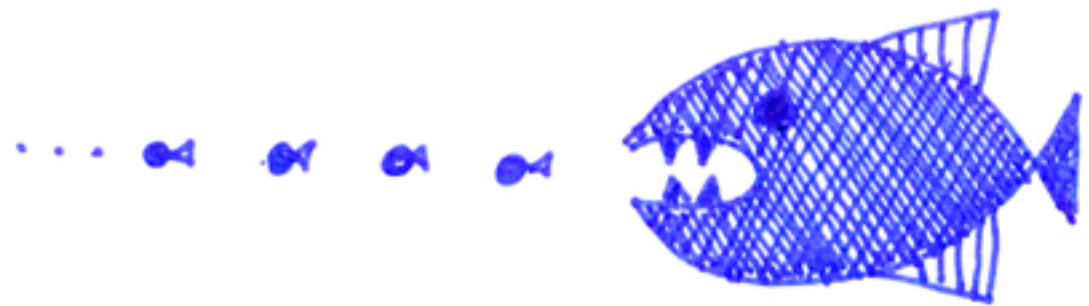


Elementary parsers

symbol "(" : Parser ()

```
> run ( symbol "(" ) "(aaa"  
Ok ()
```

```
> run (symbol "(") "aaa"  
Err [...]
```



float : Parser Float

```
> run float "1.2"  
Ok 1.2
```

spaces : Parser ()

spaces =

chompWhile (\c -> c == ' ')

succeed : a -> Parser a

> run (succeed "a") "a"
Ok "a"

> run (succeed "a") "b"
Ok "a" :

Parser combinators

Alternation

oneOf : List (Parser a) -> Parser a

oneOf [symbol "a", symbol "b"]

Sequencing

(|.) : Parser keep -> Parser ignore -> Parser keep

(|=) : Parser (a -> b) -> Parser a -> Parser b

Exercise

type alias Point = { x : Float, y : Float }

Point : Float -> Float -> Point

Point 1.2 3.6 == Point 1.2 3.6

Succeed Point

|= Float

|= Float

What is the type??

Succeed Point

$\begin{array}{l} \models \text{Float} \\ \models \text{Float} \end{array} \quad \longleftrightarrow \quad ((\text{succeed Point}) \models \text{Float}) \models \text{Float}$

$(\text{Parser } (a \rightarrow b) \models (\text{Parser } a) : \text{Parser } b)$

$((\text{succeed Point}) \models \text{Float}) \models \text{Float}$

$(\text{Parser}(\underset{a}{\text{Float}} \rightarrow (\underset{b}{\text{Float}} \rightarrow \text{Point}))) \models \underset{a}{\text{Float}} \models \text{Float}$

$(\text{Parser}(\text{Float} \rightarrow \text{Point})) \models \text{Float}$

Parser Point

Point -> “(“ SP Float SP “,” SP Float SP “)”



(23.1, 67.5) (23.1, 67.5)


Etc.

(|.) : Ignore RHS

(|=) : Keep RHS

point : Parser Point
point =

succeed Point



|. symbol “(“
|. spaces
|= float
|. spaces
|. symbol “,”
|. spaces
|= float
|. spaces
|. symbol “)”

Top level parser


LE -> DisplayMath | InlineMath | Macro | Words | Env



latexExpression : Parser LatexExpression

latexExpression =

oneOf



**[displayMath
, inlineMath
, macro
, words
, lazy (_ -> environment)
]**

Macro parser

\italic{Wow!}

Macro -> MacroName Arg* WS



macro : Parser () -> Parser LatexExpression

macro **wsParser** =

(succeed Macro

|= macroName

|= itemList arg

|. **wsParser**

)



$x \mid > f = f \ x$

$x \mid > f \mid > g = g (f \ x)$

environment : Parser LatexExpression

environment =

envName $\mid >$ **andThen** environmentOfType

envName : Parser String

environmentOfType : String -> Parser LatexExpression

andThen : (a -> Parser b) -> Parser a -> Parser b

andThen environmentOfType envName : **Parser LE**

environmentOfType : String -> Parser LatexExpression

environmentOfType envType =

let

theEndWord = "\\end{" ++ envType ++ "}"

in

succeed (Environment envType)

|. ws

|= (nonemptyItemList latexExpression) |> map LatexList)

|. ws

|. symbol endWord_

|. ws

The M Word

Monads in Elm

andThen : (a -> Parser b) -> Parser a -> Parser b

andThen : (a -> M b) -> M a -> M b

(>>=) : M a -> (a -> M b) -> M b

Composition of “monadic functions”

$f : a \rightarrow M\ b$

$g : b \rightarrow M\ c$

$g \circ f : a \rightarrow M\ c$

$g \circ f = \lambda x \rightarrow \text{andThen } g\ (f\ x)$

$g \circ f = \lambda x \rightarrow (f\ x) \mid\!> \text{andThen } g$

Kleisli categories

Testing

```
suite : Test
suite =
  describe "MiniLatex Parser"
    [ doTest
      "(1) Comment"
      (run latexExpression "% This is a comment\n")
      (Ok (Comment "% This is a comment\n"))
    , doTest
      "(2) InlineMath"
      (run latexExpression "$a^2 = 7$")
      (Ok (InlineMath "a^2 = 7"))
    , doTest
      "(3) DisplayMath"
      (run latexExpression "$$b^2 = 3$$")
      (Ok (DisplayMath "b^2 = 3"))
```

Rendering



One rendering function

per

nonterminal symbol

```
render : LatexState -> LatexExpression -> Html msg
```

```
render latexState latexExpression =
```

```
  case latexExpression of
```

```
    Macro name optArgs args ->
```

```
      renderMacro latexState name optArgs args
```

```
    InlineMath str ->
```

```
      Html.span [] [ oneSpace, inlineMathText str ]
```

```
    DisplayMath str ->
```

```
      displayMathText str
```

```
    Environment name args body ->
```

```
      renderEnvironment latexState name args body
```

```
    LatexList latexList ->
```

```
      renderLatexList latexState latexList
```

```
    LXString str ->
```

```
      Html.span [] [ Html.text str ]
```

```
    LXError error ->
```

```
      -- List.map ErrorMessage.renderError error |> String.join ";" "
```

```
      Html.p [] [ Html.text <| "((ERROR))" ]
```

LE

MA

IM

DM

Env

LL

LS

```
renderMacro : LatexState -> String  
              -> List LatexExpression -> Html msg
```

```
renderMacro latexState name args =  
  case Dict.get name renderMacroDict of
```

```
    Just f -> f latexState args
```

```
    Nothing ->  
      reproduceMacro name latexState args
```

"\foo{bar}"



"\foo{bar}"

The full parse-render pipeline





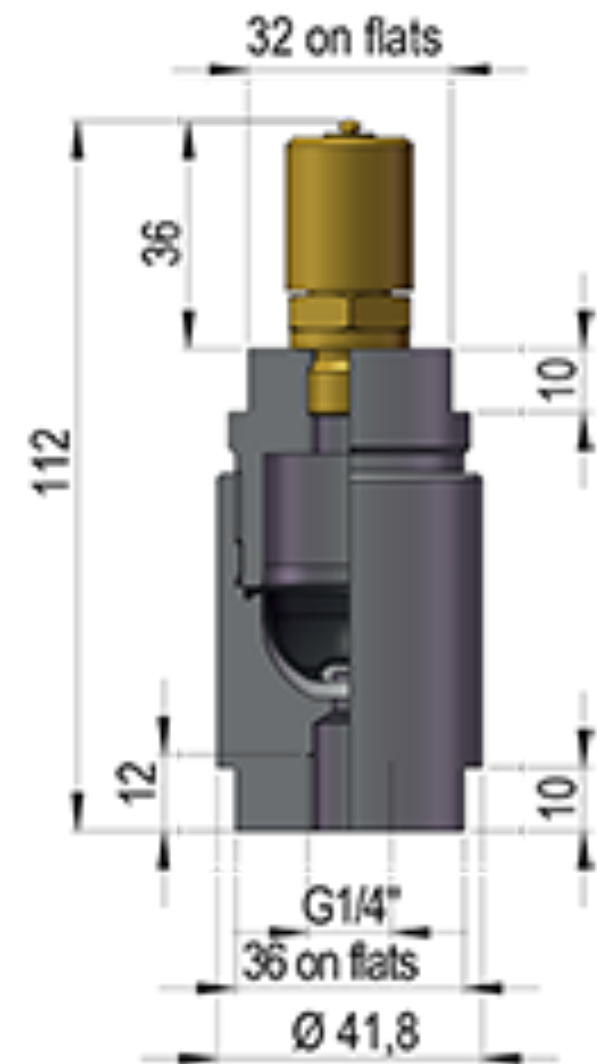
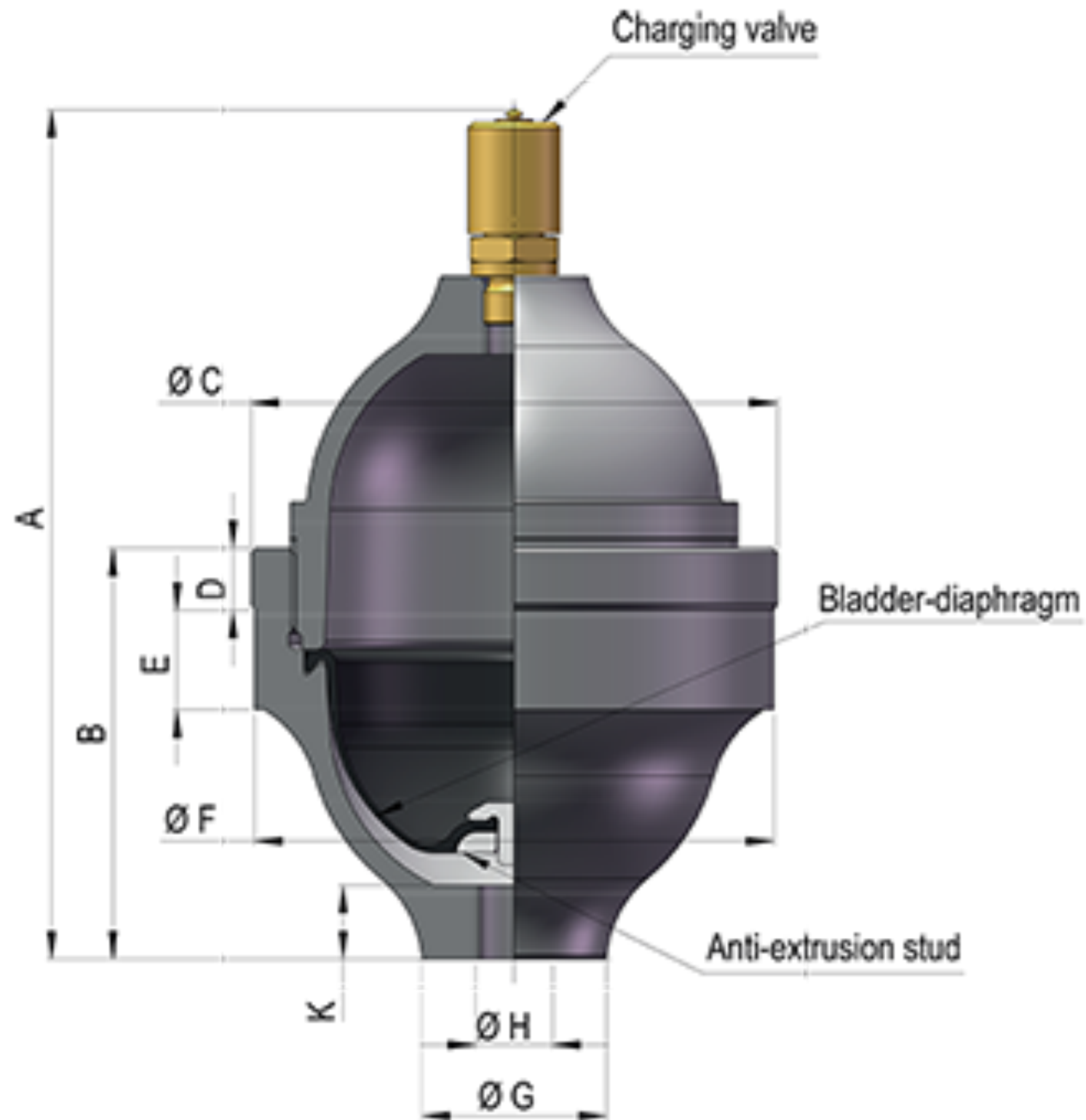
Source text

- | | |
|--|----------------|
| (1) => List of logical paragraphs | :: FSM |
| (2) => (LatexState, List (List LatexExpr)) | :: Acc.parse |
| (3) => (LatexState, List (Html msg)) | :: Acc.render |
| (4) => Html msg | :: Concatenate |
| (5) => DOM | :: Elm runtime |
| (6) => DOM | :: Mathjax.js |

Also

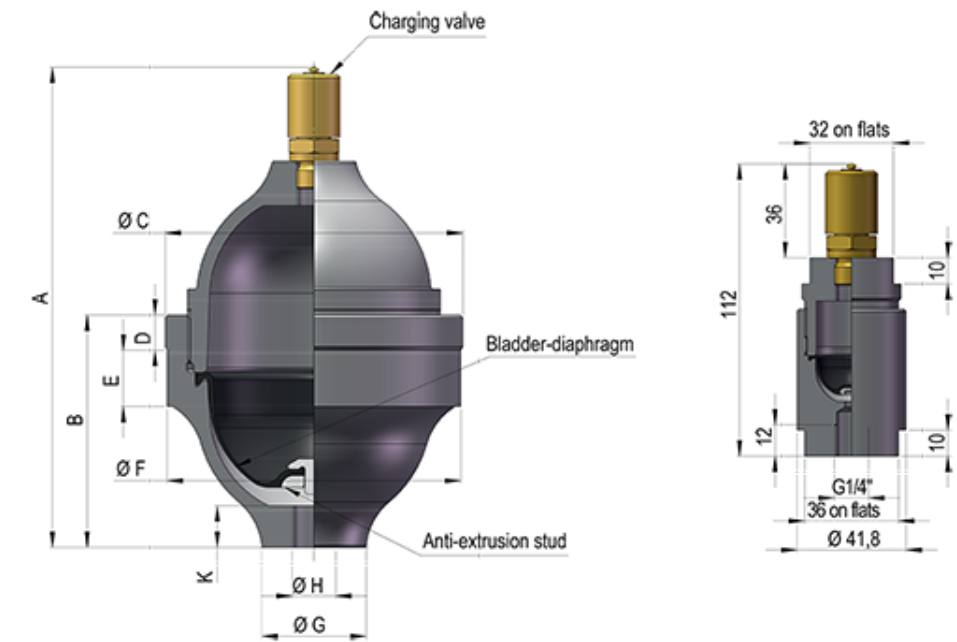
- The **joining problem** for **List (LatexExpression)**
 - use a List Machine
- **Diffing**

Acc = Accumulator



Accumulator state a b :

state -> List a -> (state, List b)



Constructing an accumulator

type alias Reducer a b : a -> b -> b

List.foldl : (a -> b -> b) -> b -> List a -> b

Special reducers ...

Reducer **state a b** = a -> (state, List b) -> (state, List b)

Reducer state a b -> (state, List b) -> List a -> (state, List b)

state -> (state, [])

(state, List b) -> List a -> (state, List b)

state -> List a -> (state, List b)

Accumulator.parse :

LatexState

-> List String

-> (LatexState, List (List LatexExpression))

**Accumulator.parse latexState paragraphs =
paragraphs**

|> List.foldl parseReducer (latexState, [])

parseReducer :

String

-> (LatexState, List (List LatexExpression))

-> (LatexState, List (List LatexExpression))

parseReducer inputString (latexState, llexpr) =

let

 parsedInput = Parser.parse inputString

 newLatexState =

 latexStateReducer parsedInput latexState

in

 (newLatexState, llexpr ++ [parsedInput])

latexStateReducer :

List LatexExpression -> LatexState -> LatexState

latexStateReducer parsedParagraph latexState =

let

theInfo =

parsedParagraph

|> List.head

|> Maybe.map info

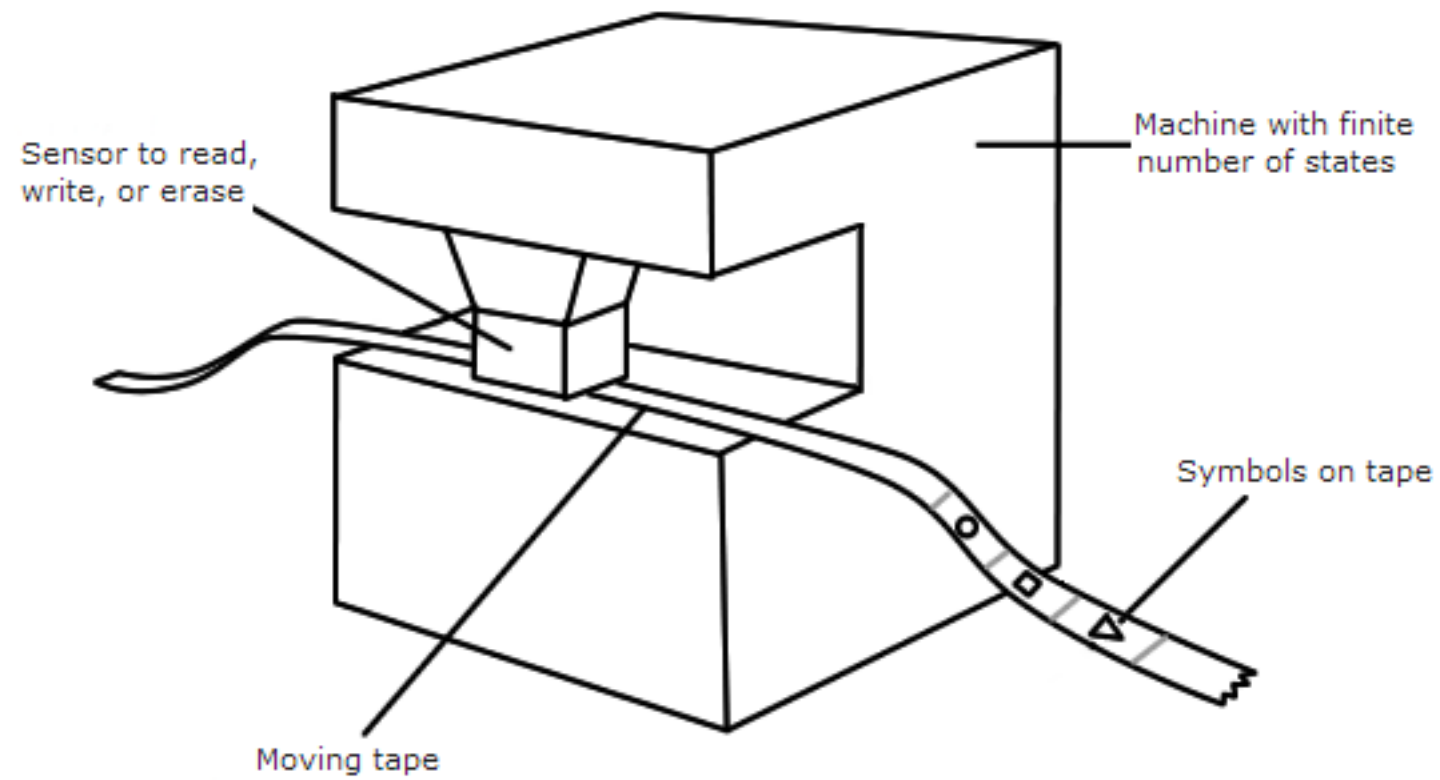
|> Maybe.withDefault (LatexInfo "null" "null" [] [])

in

(latexStateReducerDispatcher theInfo)

theInfo latexState

List Machines



A Turing Machine

```
type alias State a = {  
  before: Maybe a,  
  current: Maybe a,  
  after: Maybe a,  
  inputList: List a  
}
```

outputFunction : State a -> b

ListMachine.run : (State a -> b) -> List a -> List b

Diffing

u = text v = edited text

$u = a \ x \ b$ $v = a \ y \ b$

a = greatest common prefix

b = greatest common suffix

$u' = \text{render } u$

$u' = a' \ x' \ b'$ $v' = a' \ y' \ b'$

Store $u' = a' \ x' \ b'$

To compute v' , only compute $y' = \text{render } y$

Contents (18)

NOTES ON QUANTUM FIELD

1. Trajectories and Uncertainty!

2. Wave packets and the disp

3. Wave Packets and Schrodin

4. Bound States

5. Hydrogen Atom

6. Harmonic Oscillator

7. Anharmonic Oscillator

8. Bras and Kets

9. Fourier Transform

10. Evolution and Propagation

11. Time-dependent perturbatio

12. N-Particle Systems: Bosons

13. Special Relativity

14. Lagrangian and Hamiltonian

15. Classical String

16. Discrete Fourier Analysis

17. Heisenberg Picture and Har

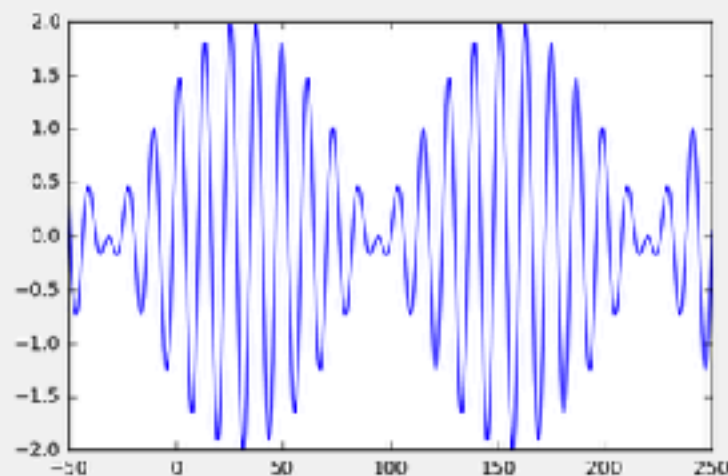
Notes on Quantum Field Theory

(jxxcarlson)

Wave packets and the dispersion relation

2.1 A two-frequency packet: beats

Consider a wave $\psi = \psi_1 + \psi_2$ which is the sum of two terms with slightly different frequencies. If the waves are sound waves, then then what one will hear is a pitch that corresponding to the average of the two frequencies modulated in such a way that the volume goes up and down at a frequency corresponding to their difference.



Two-frequency beats

Let us analyze this phenomenon mathematically, setting

$$\psi_1(x, t) = \cos((k - \Delta k/2)x - (\omega - \Delta\omega/2)t) \quad (2.1)$$

and

Signed in as jxxcarlson

Preferences

REFERENCES

jxxcarlson@gmail.com

DEMO

<https://jxxcarlson.github.io/app/miniLatexLive/index.html>

<https://knode.io>

ARTICLES

<https://minilatem.gitbook.io>

<https://hackernoon.com/towards-latex-in-the-browser-2ff4d94a0c08>

REPOS

<https://package.elm-lang.org/packages/jxxcarlson/minilatem/latest>

<https://github.com/jxxcarlson/meenylatem>