Speaking for the Trees

A New (Old) Approach to Languages and Syntax

Moss Prescott

Preamble

syntax is important

current approaches are limited…

- in what they can express
- or they’re hard to extend

try something new

This Talk

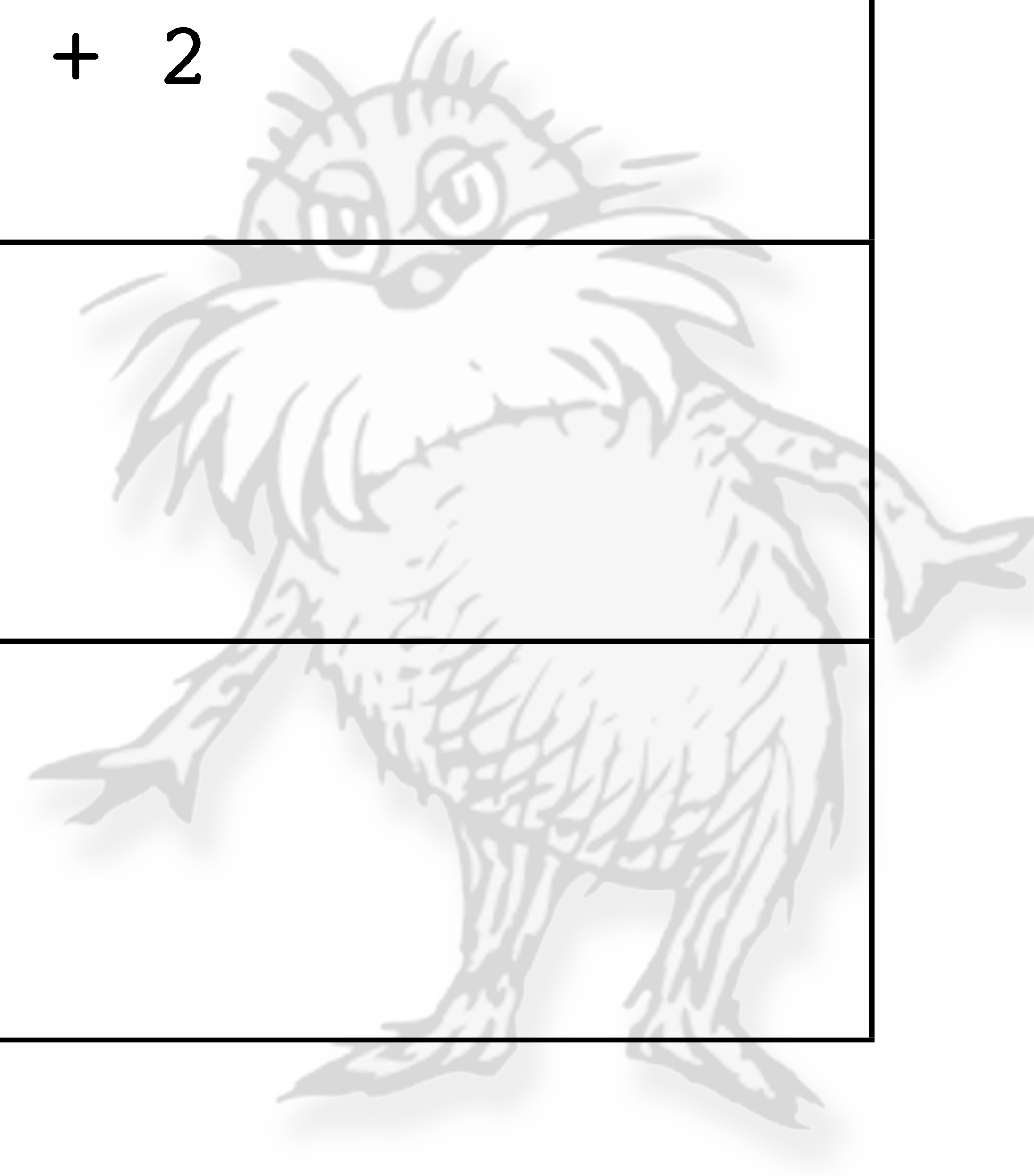
background and motivation

a new approach

demo

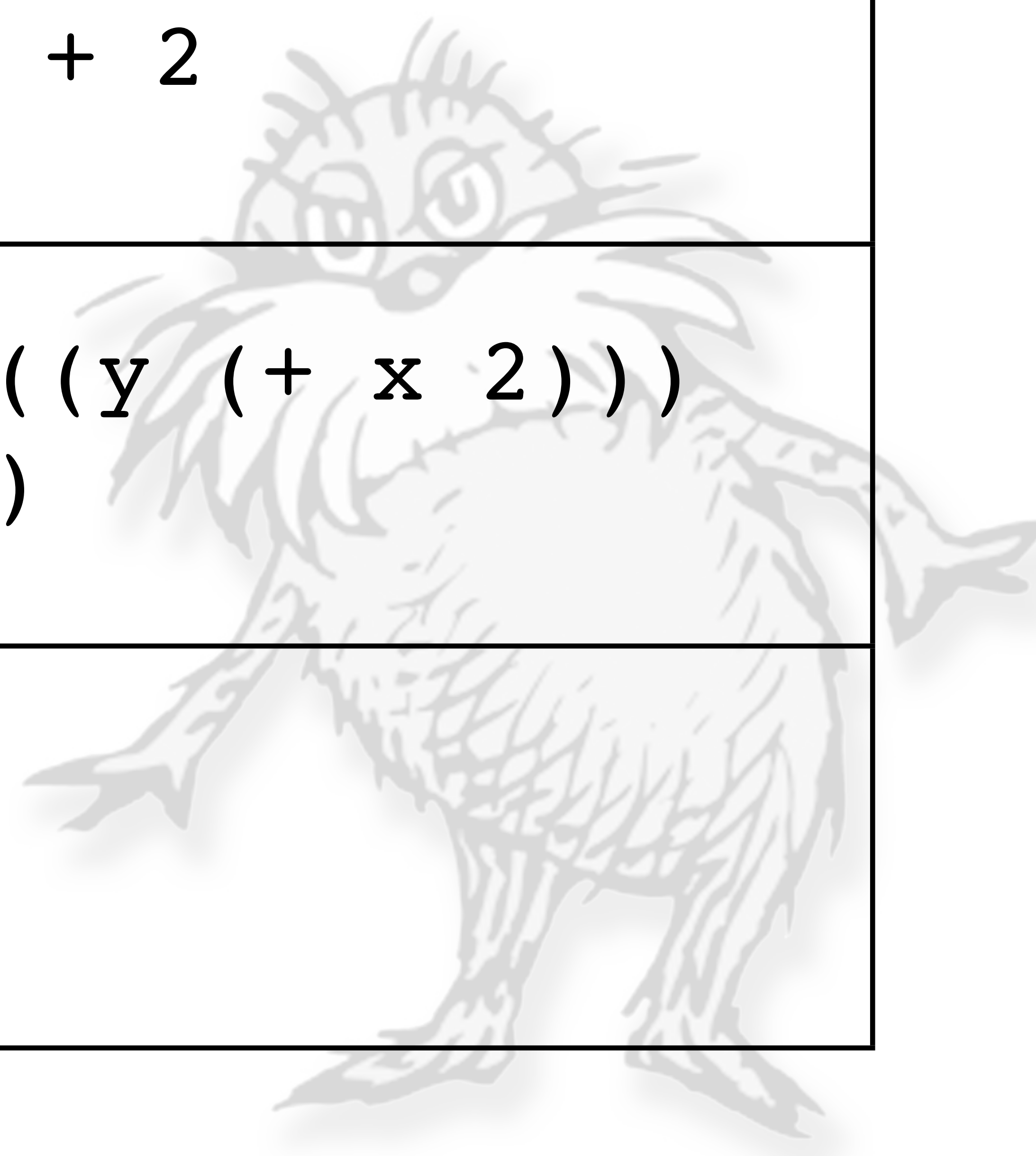
Approaches to Syntax

C, Haskell, …	`y = x + 2`



Approaches to Syntax

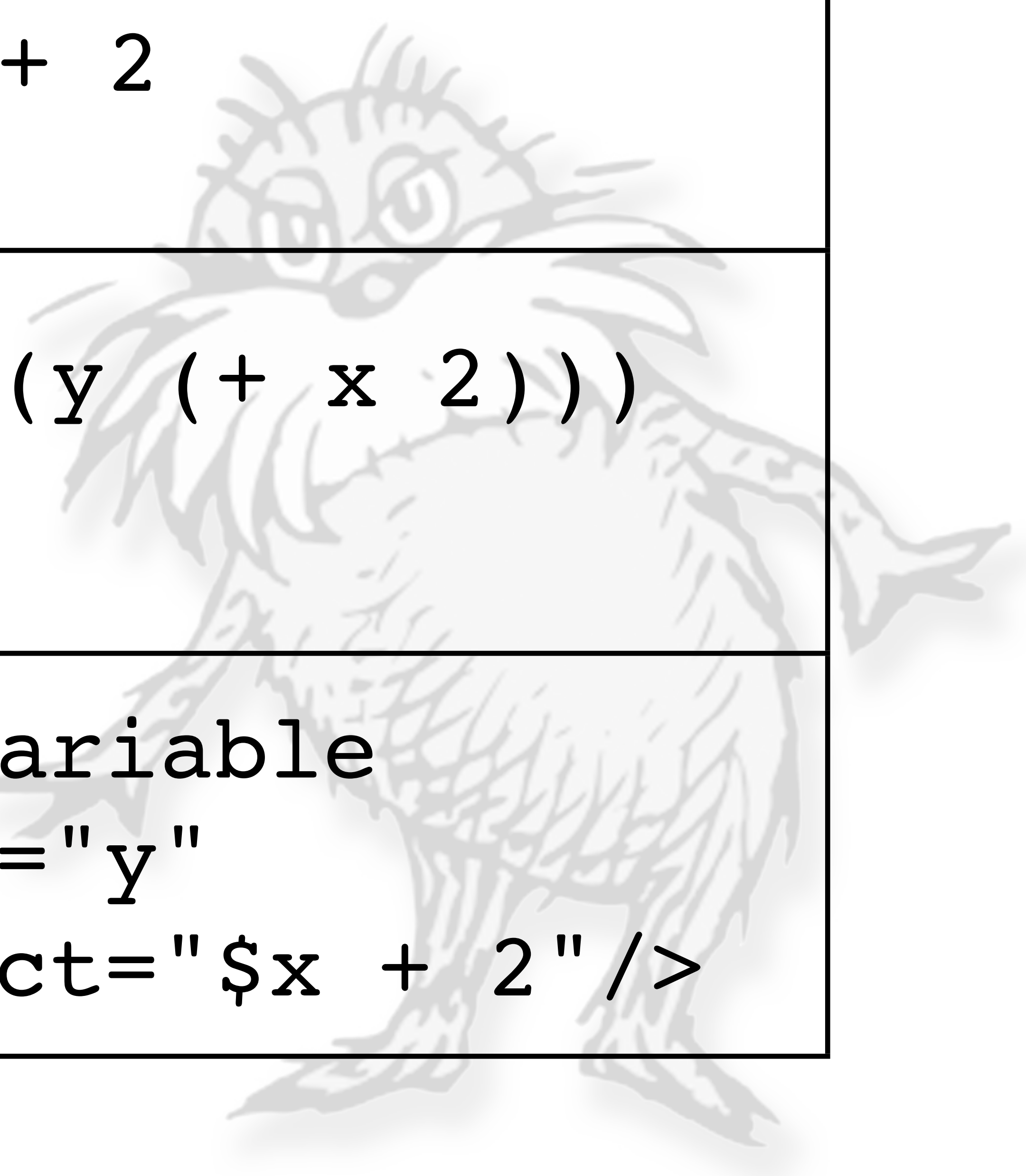
C, Haskell, …	`y = x + 2`
Lisp	`(let ((y (+ x 2)))` `  ...)`



Approaches to Syntax

C, Haskell, …	`y = x + 2`
Lisp	`(let ((y (+ x 2)))` `  ...)`

XSL-T, …	`<xsl:variable`
	`  name="y"`
	`  select="$x + 2"/>`

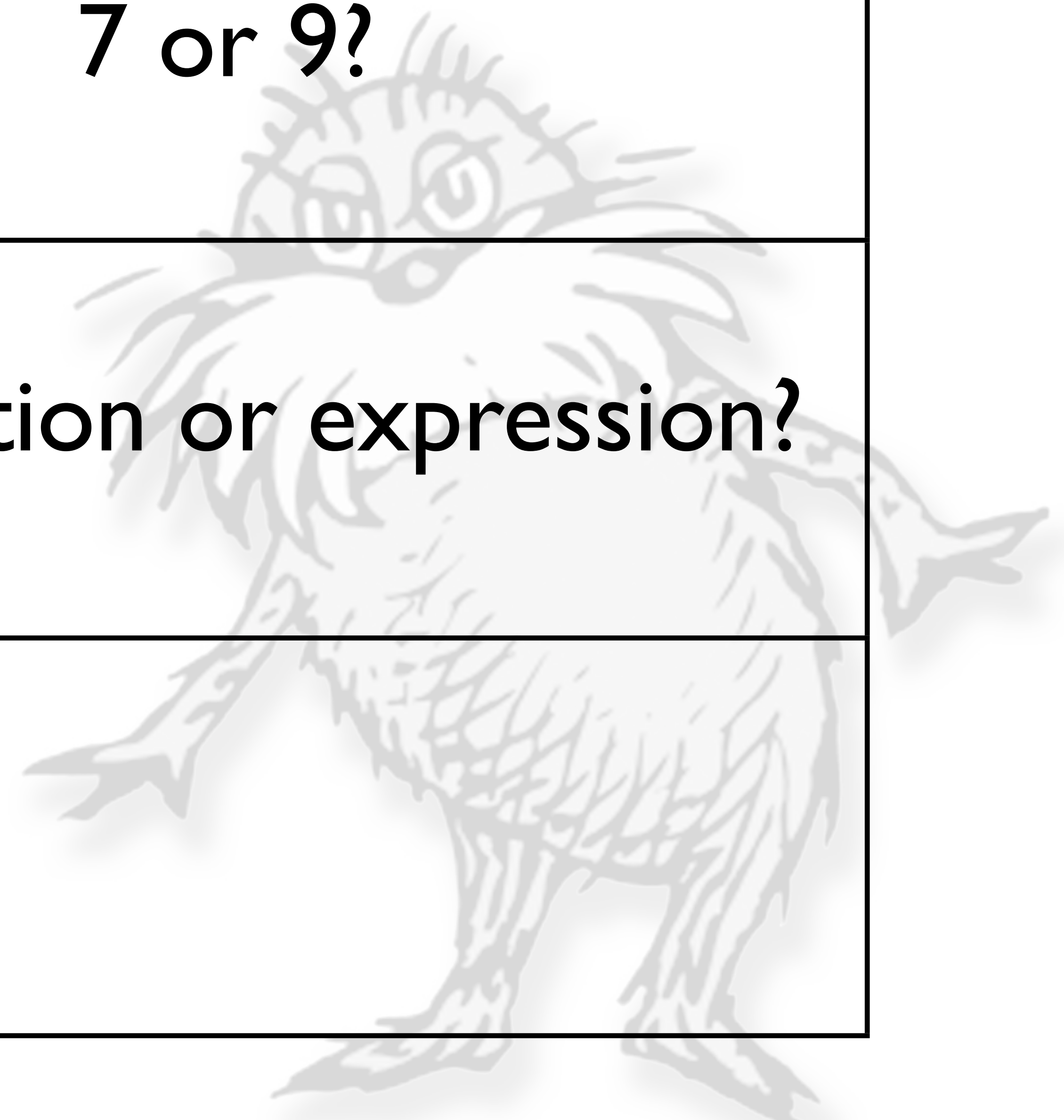


What's hard for text?

`1 + 2 * 3`	7 or 9?

What's hard for text?

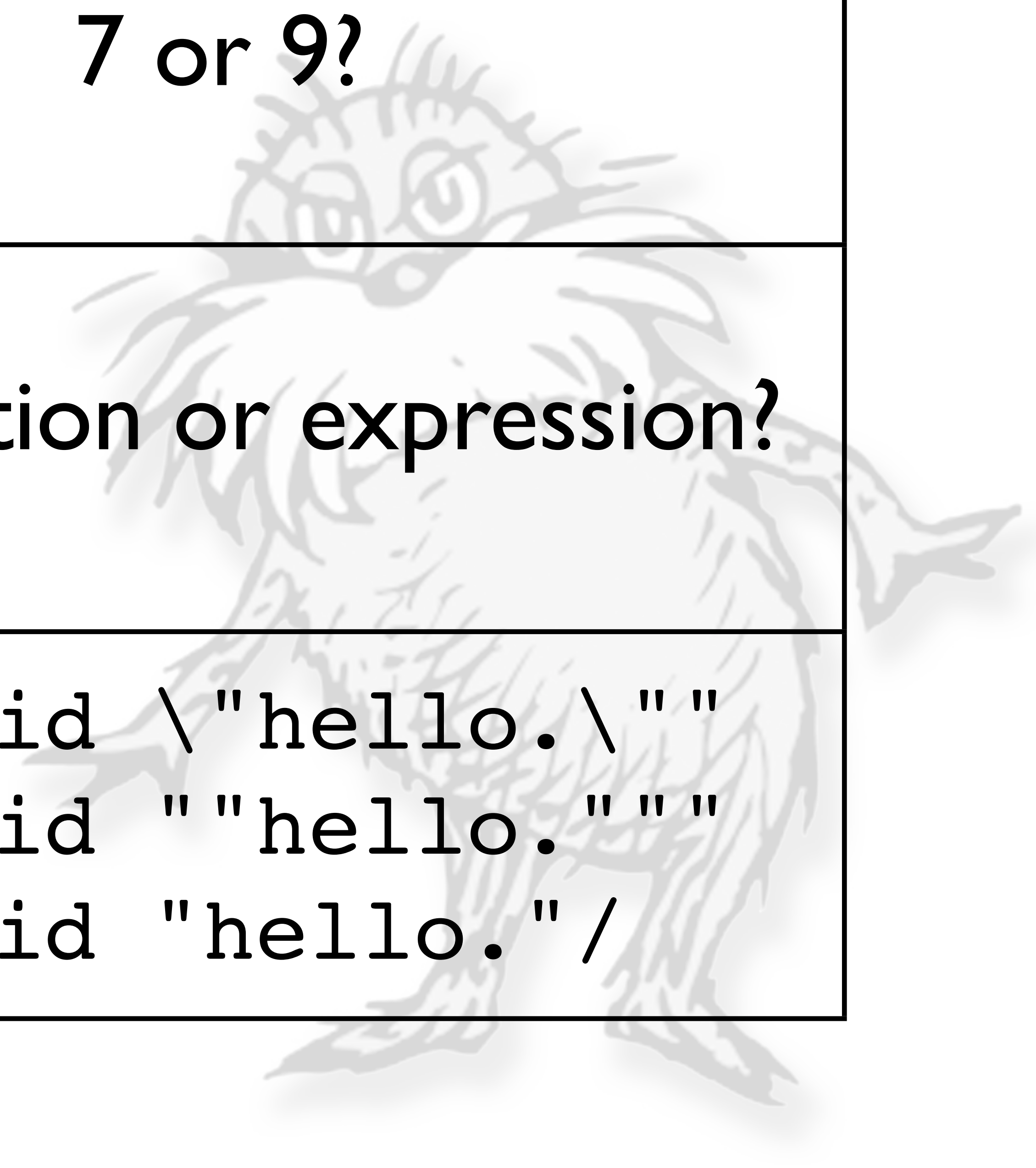
`1 + 2 * 3`	7 or 9?
`foo *ptr;` `a * y;`	Declaration or expression?



What's hard for text?

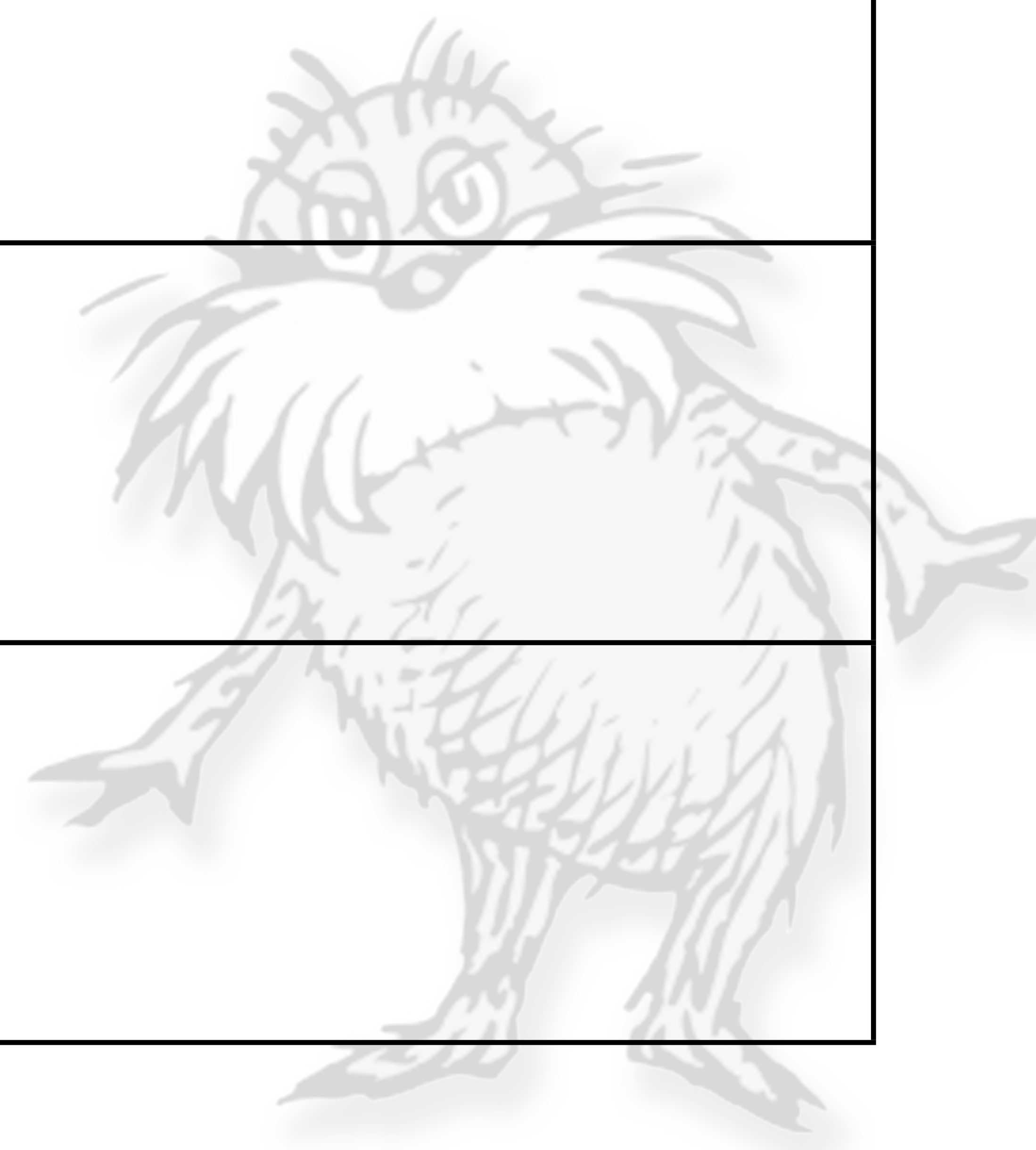
`1 + 2 * 3`	7 or 9?
`foo *ptr;` `a * y;`	Declaration or expression?

She said "hello."	`"She said \"hello.\""`
	`"She said ""hello."""`
	`/She said "hello."/`



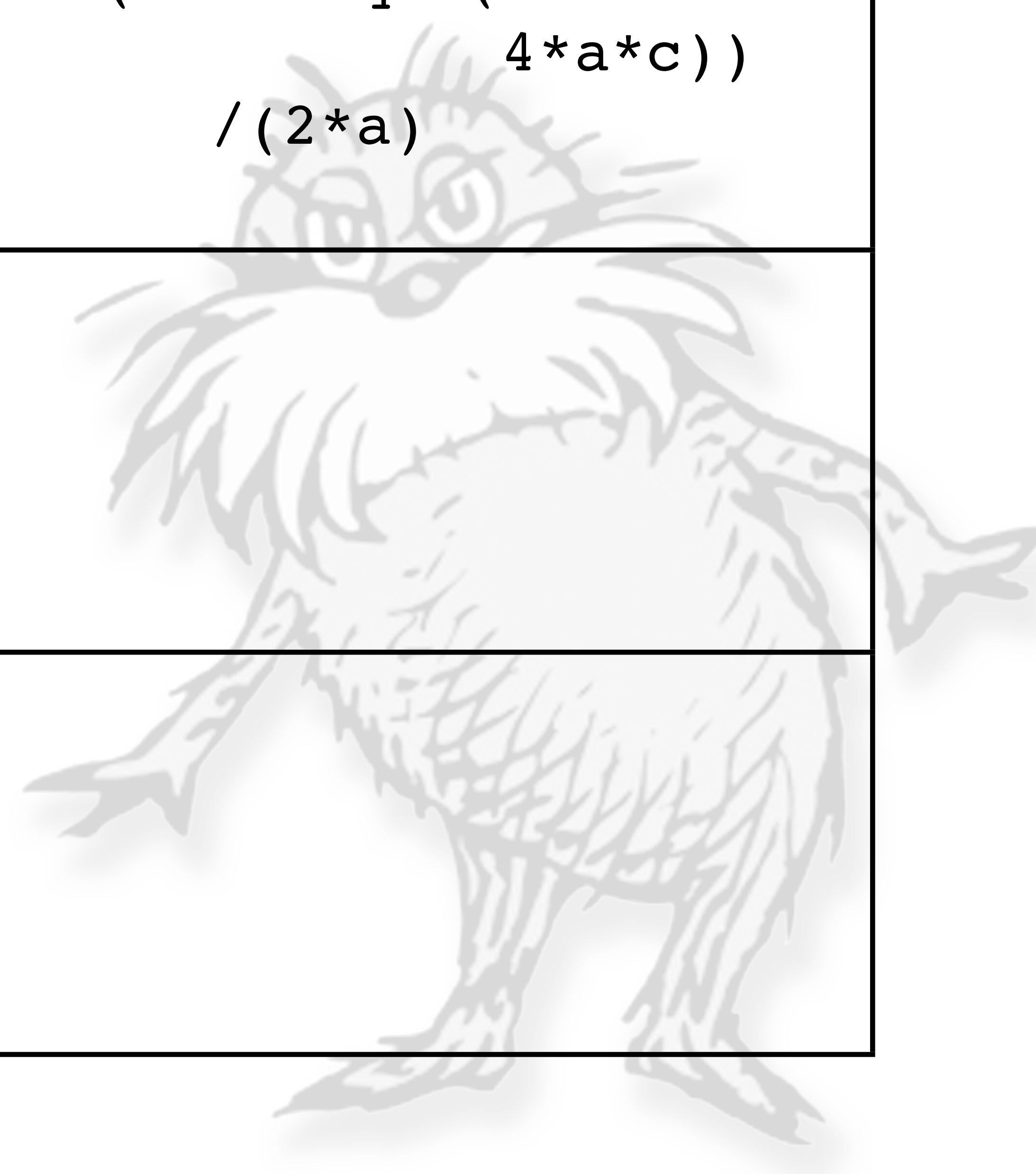
What's not possible?

$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	



What’s not possible?

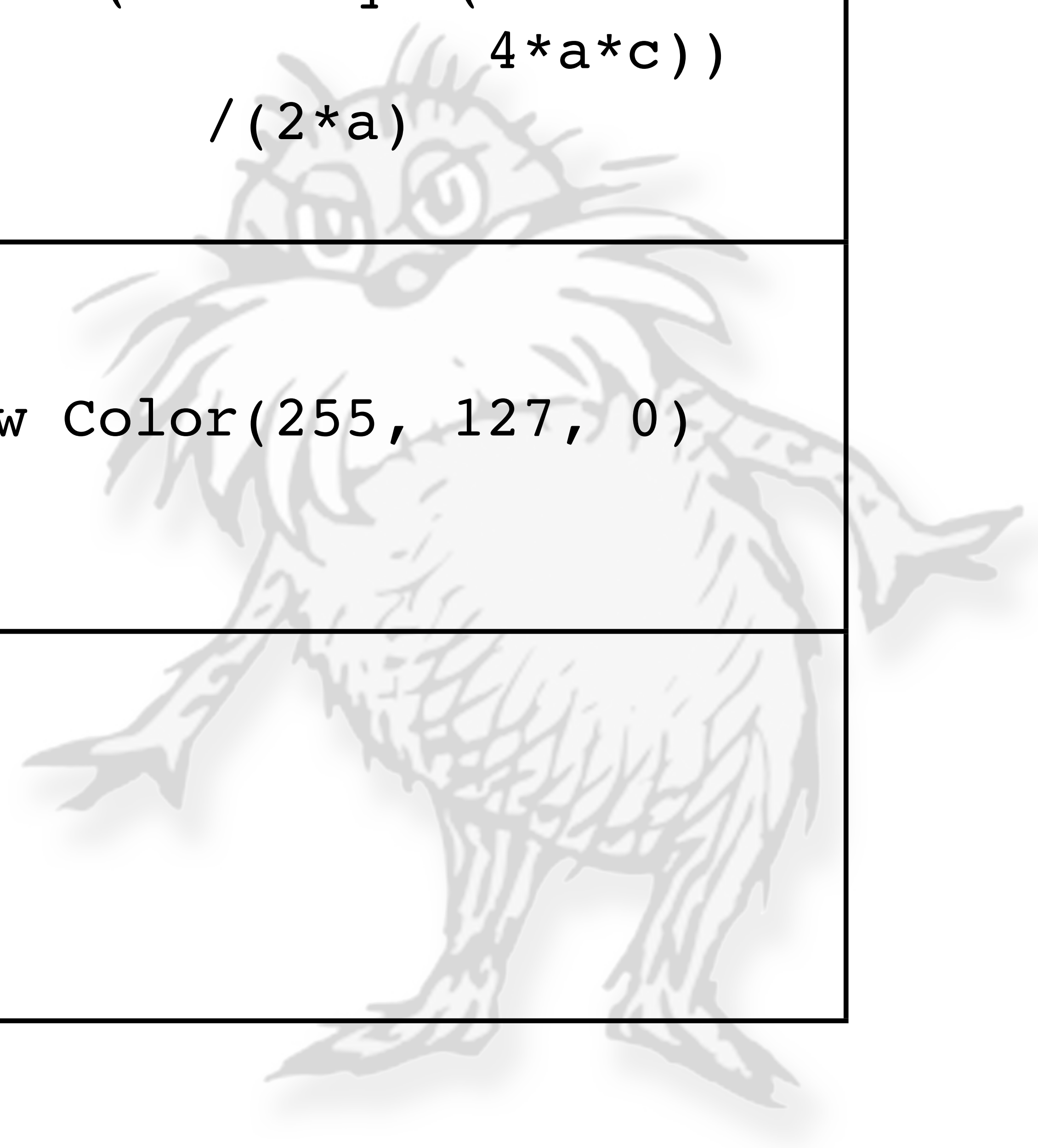
$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	`x1 = (-b + sqrt(b^2 -`
	`4*a*c))`
	`/(2*a)`



What’s not possible?

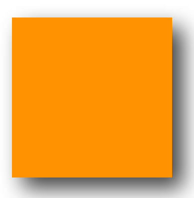
$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	`x1 = (-b + sqrt(b^2 -`
	`4*a*c))`
	`/(2*a)`

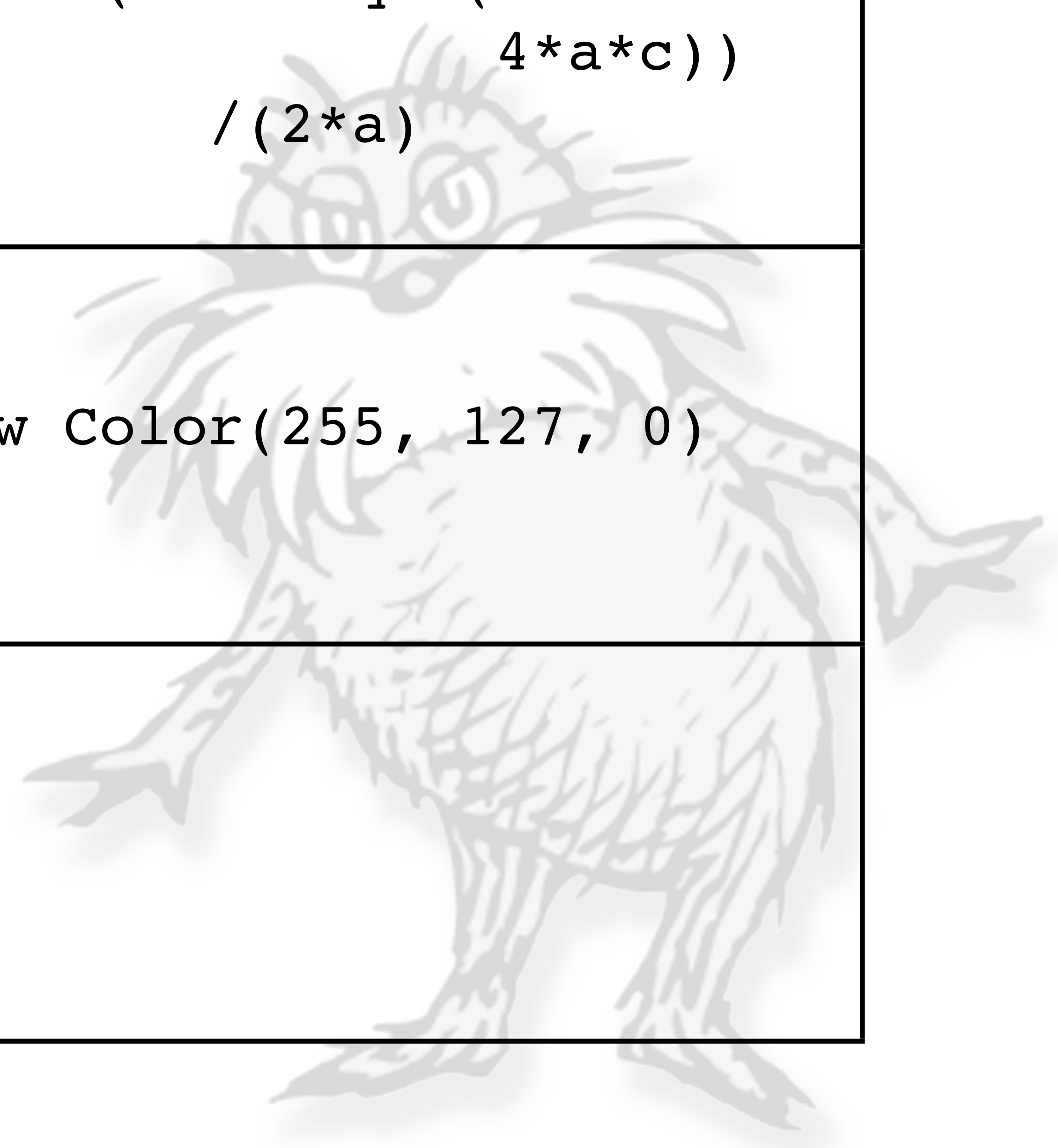
	`new Color(255, 127, 0)`



What's not possible?

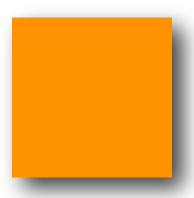
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$	`x1 = (-b + sqrt(b^2 -`
	`4*a*c))`
	`/(2*a)`

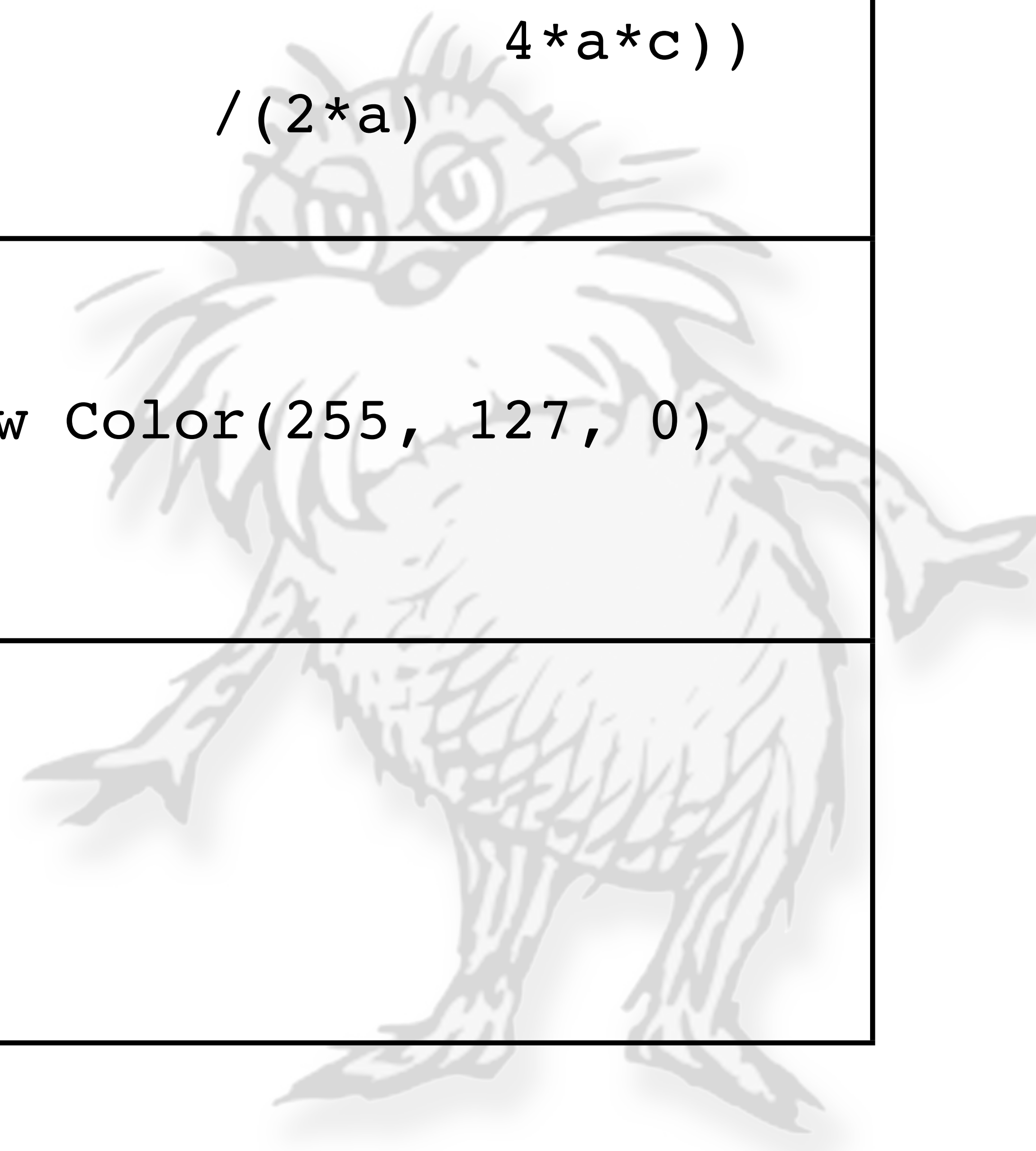
	`new Color(255, 127, 0)`



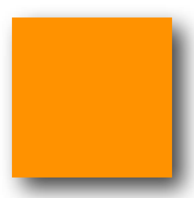
What's not possible?

$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	`x1 = (-b + sqrt(b^2 -`
	`4*a*c))`
	`/(2*a)`

	`new Color(255, 127, 0)`
	



What's not possible?

$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	`x1 = (-b + sqrt(b^2 - 4*a*c)) /(2*a)`
	`new Color(255, 127, 0)`
	`NORTH_NORTH_WEST` `degrees(360-22.5)`

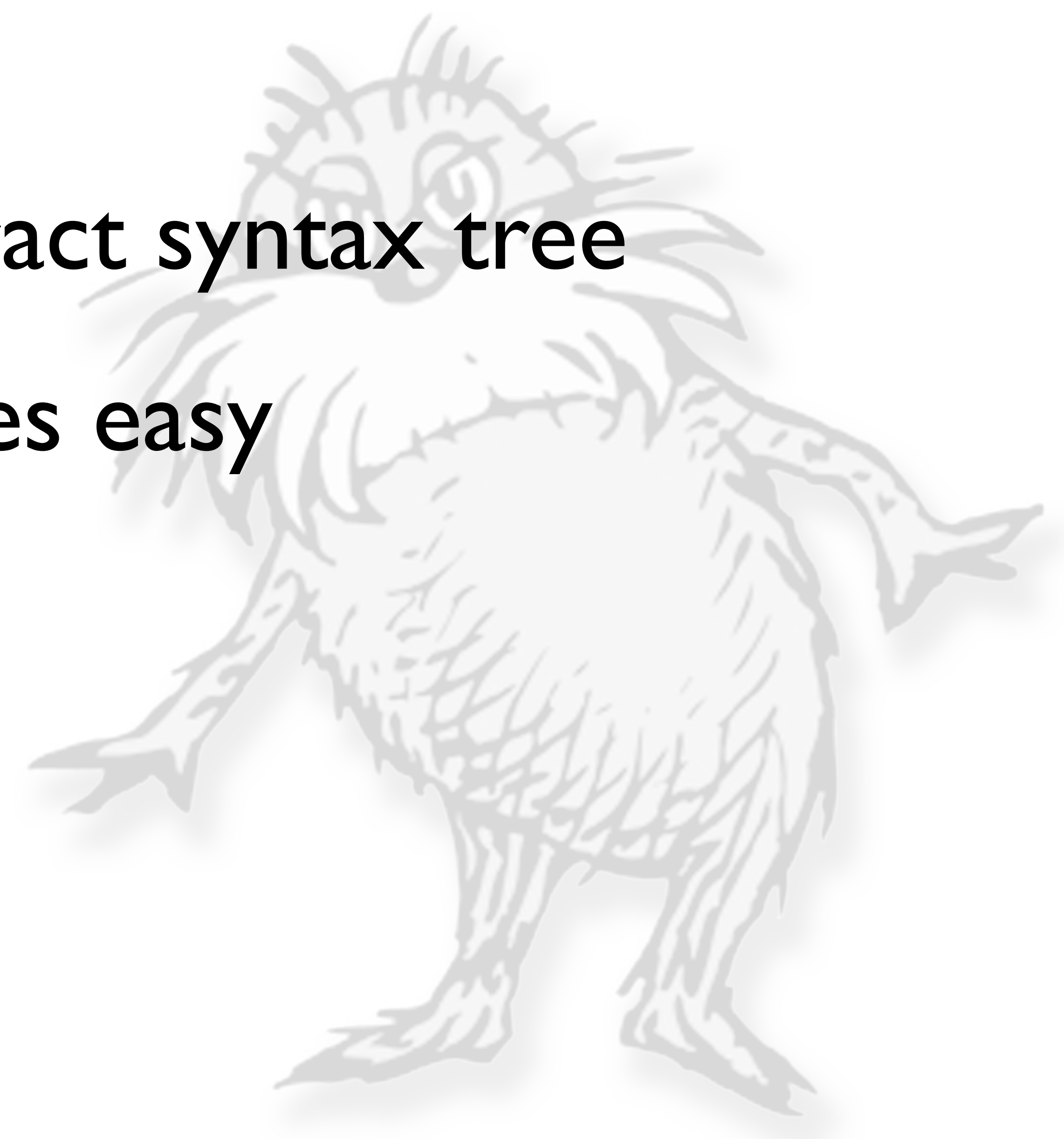
If not text, what?

If not text, what?

obvious alternative: abstract syntax tree

syntax extension becomes easy

how will editing work?



Basic Idea

1. source is an AST
2. reduce new nodes to target language
3. presentation is a target language

Related Work

structure editors

language workbenches

MPS
Meta Programming System

INTENTIONAL®
SOFTWARE

What’s Different

use a kernel meta-language

everything’s a tree

reductions

Lorax

"I am the Lorax.

I speak for the trees."

- Dr. Suess

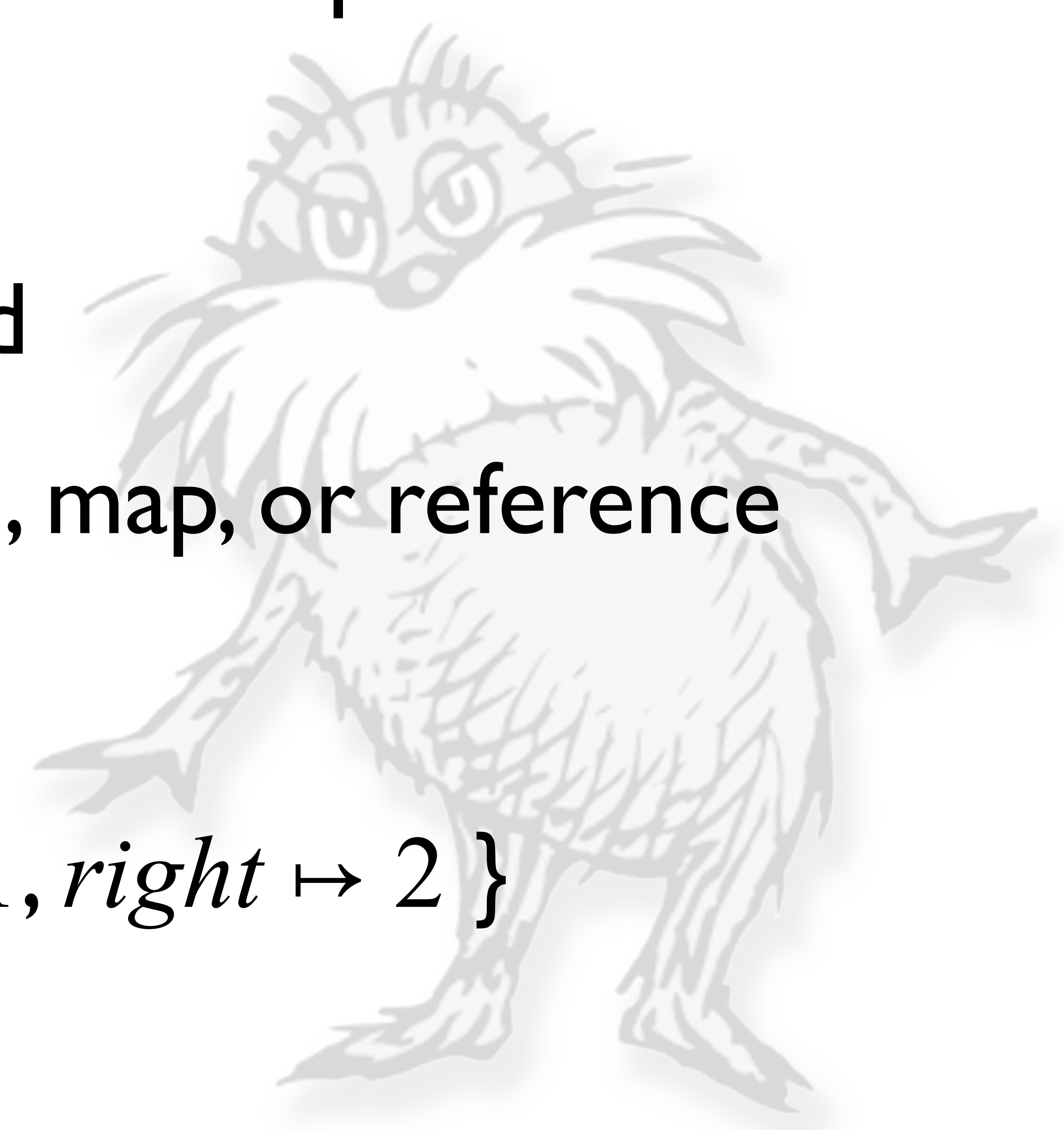
Node

an immutable record with three parts

label: globally unique

type: a name, unrestricted

value: primitive, sequence, map, or reference



$$n_0\text{: plus } \{\ \mathit{left} \mapsto 1, \mathit{right} \mapsto 2\ \}$$

Sequence Node

contains children indexed from 0 to $n - 1$

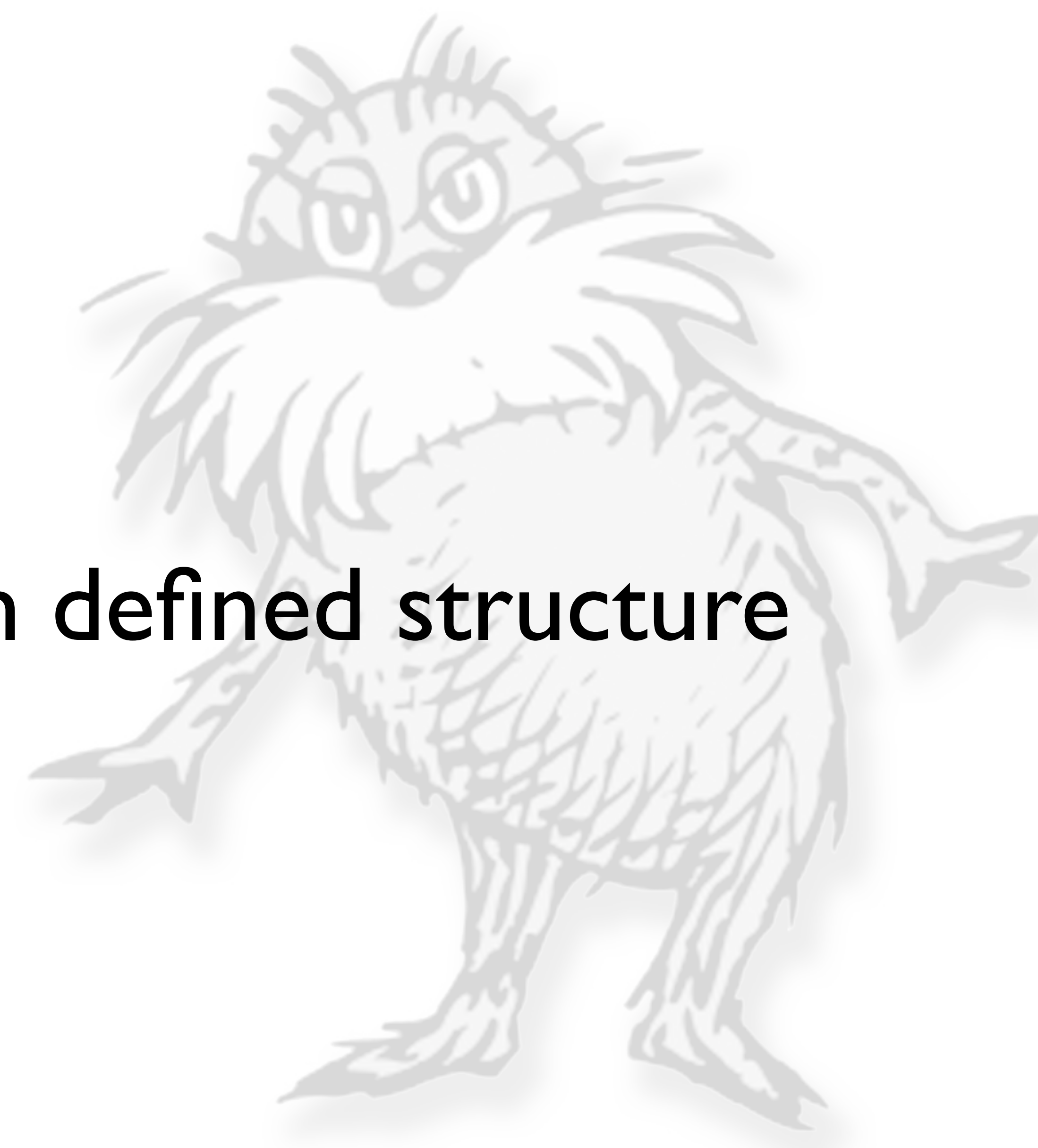
represents any kind of ordered collection

Map Node

contains named children

children *not* ordered

represents any node with defined structure



Reference Node

refers to another node

otherwise opaque

uses label internally

Well-formed Trees

The nodes of a *well-formed* tree satisfy certain constraints:

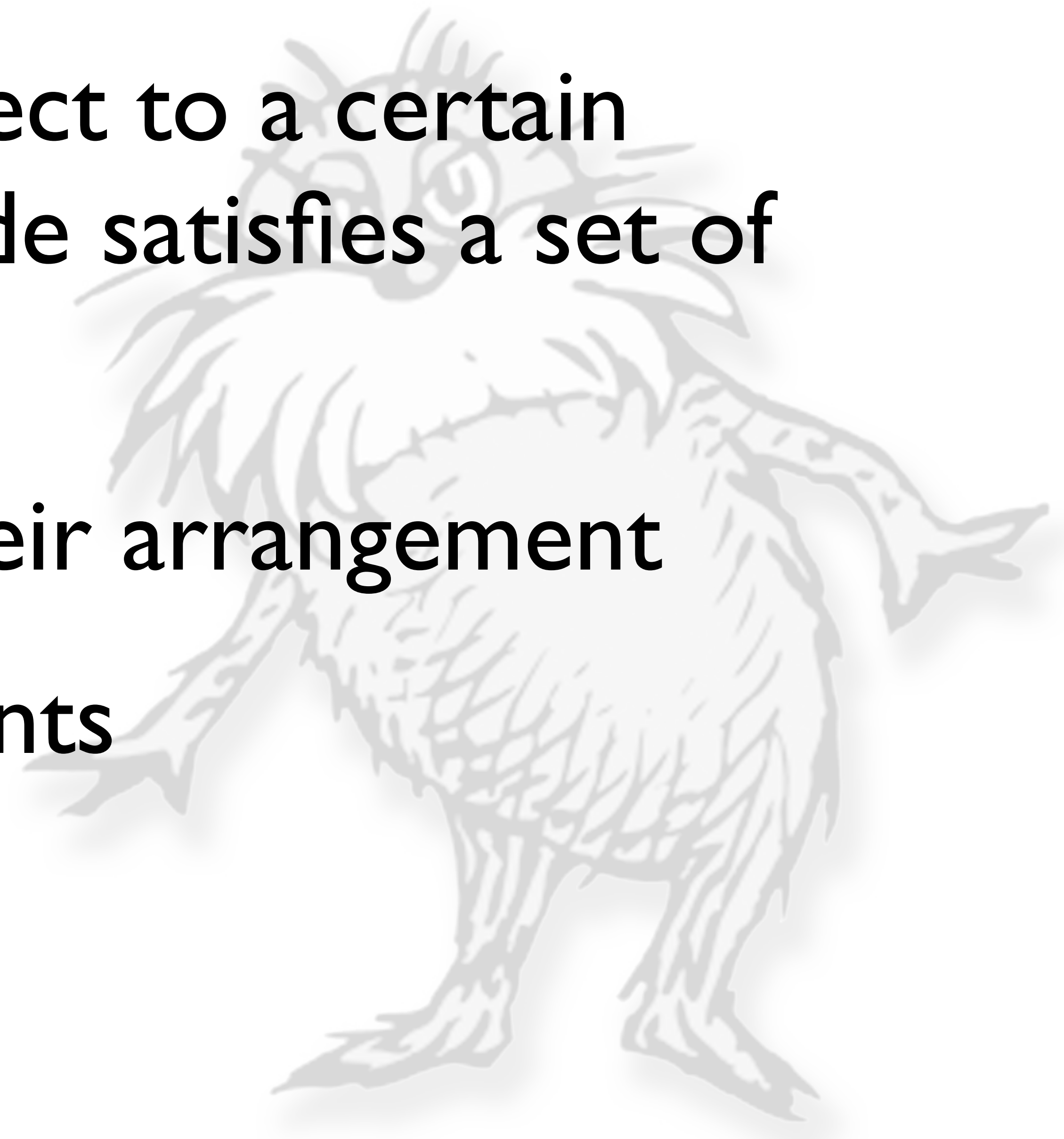
1. labels are unique
2. each node appears only once
3. referenced nodes appear in tree

Specifications

A node is *valid* with respect to a certain specification iff every node satisfies a set of constraints.

...on types, values, and their arrangement

local vs. *non-local* constraints

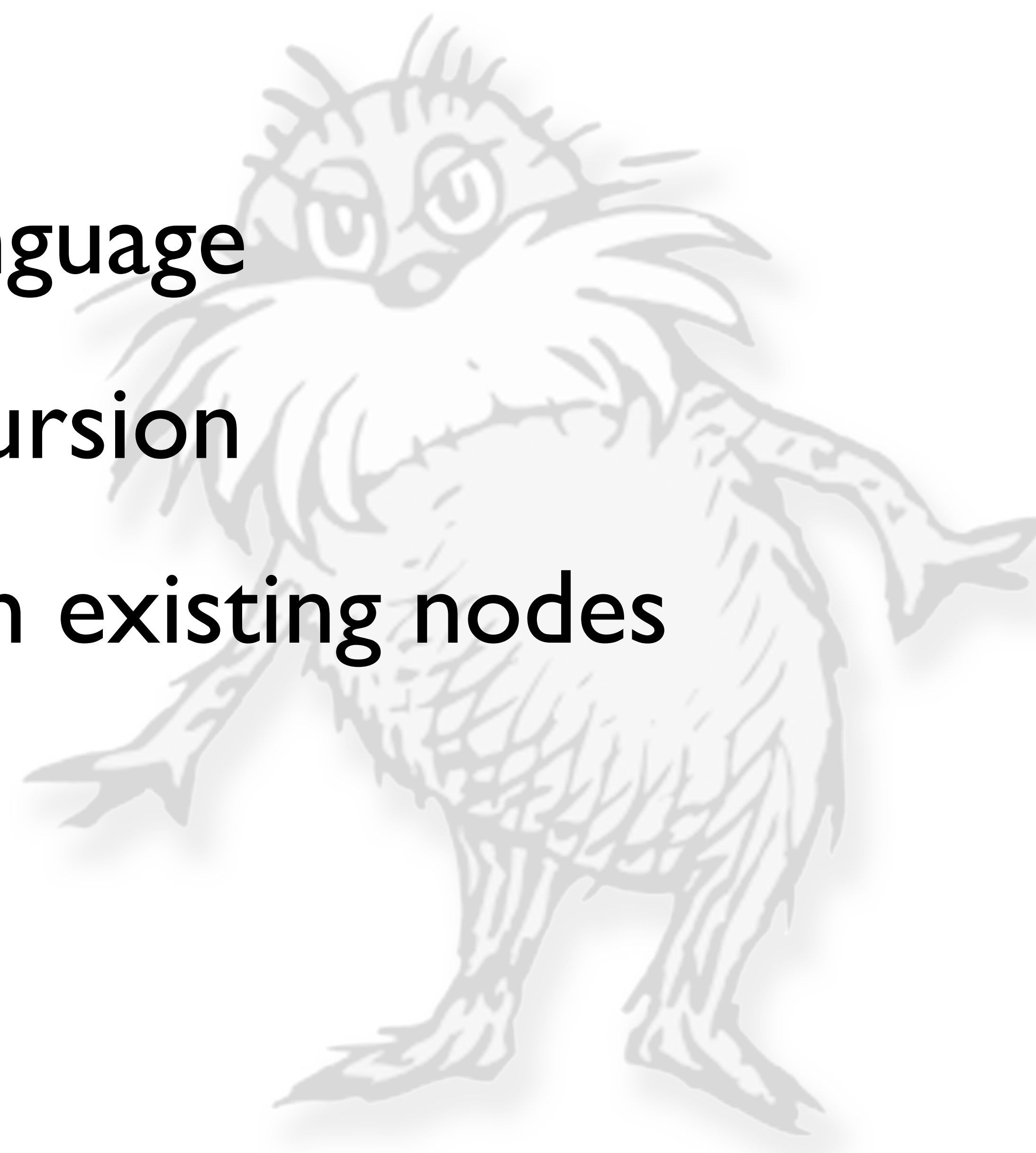


Reductions

from source to target language

top-down, structural recursion

constructs new tree from existing nodes



Languages

"…Truffula Trees are what everyone needs."

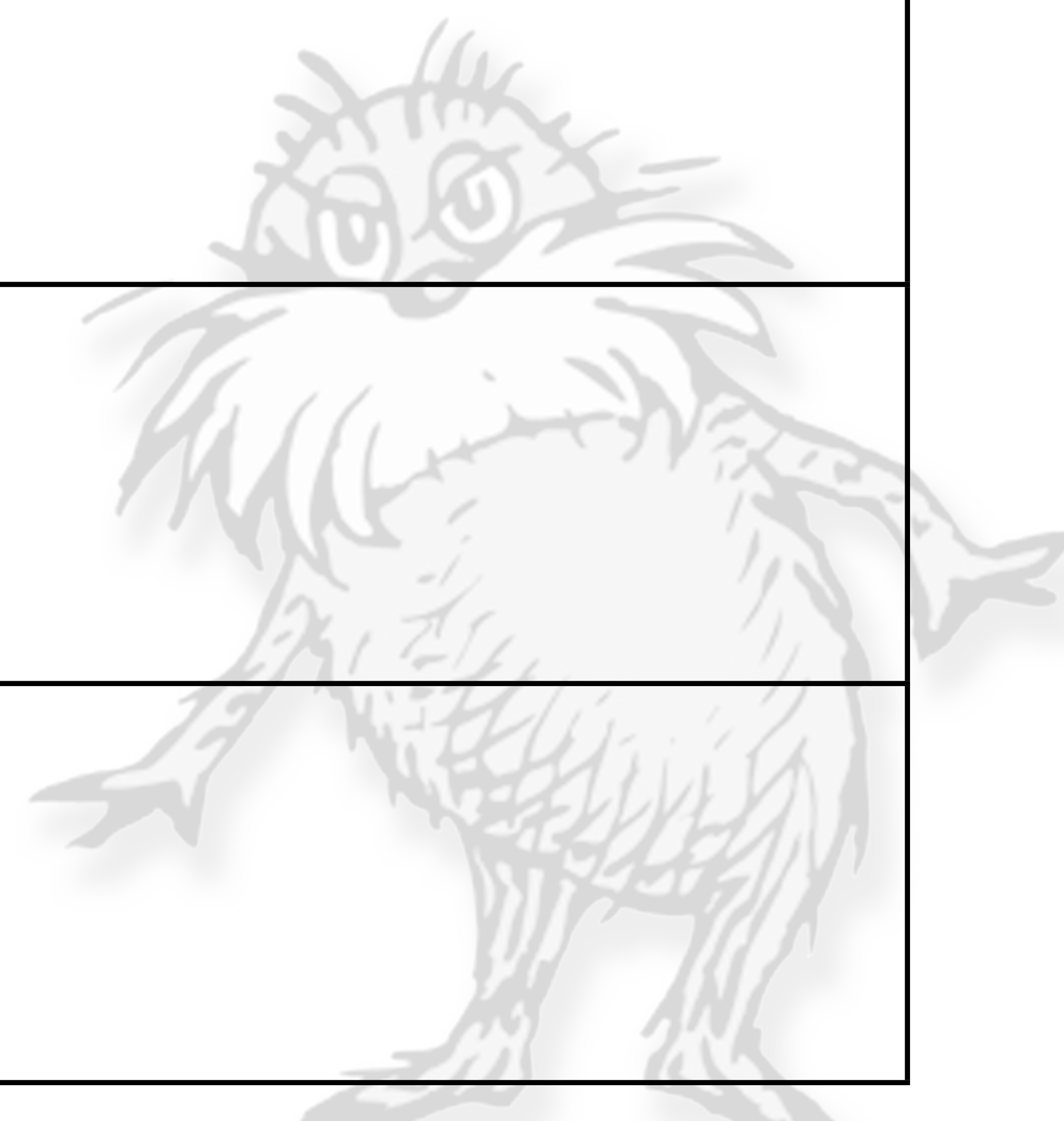
- Dr. Suess

Defining a Language

abstract syntax	

Defining a Language

abstract syntax	
semantics	

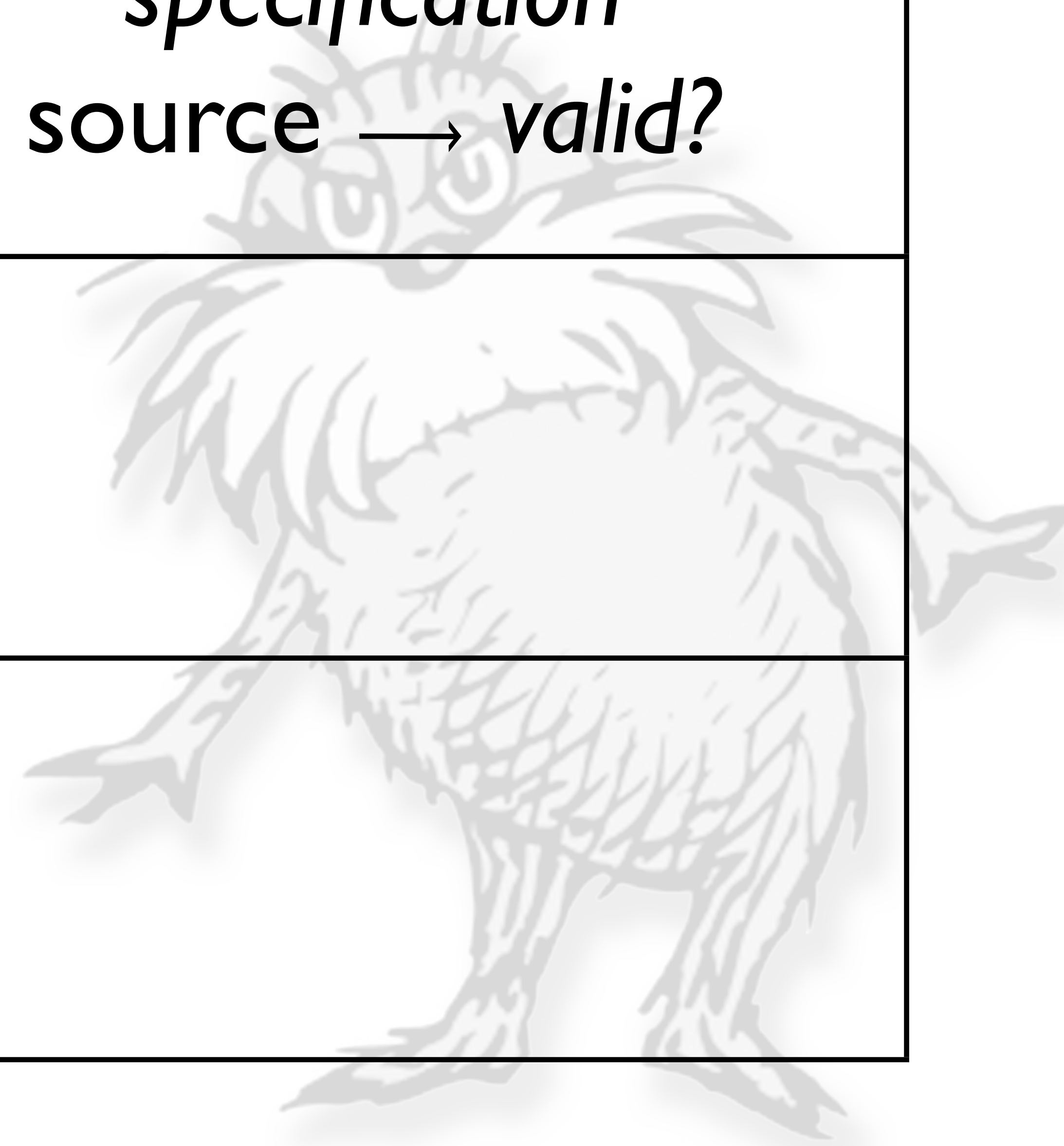


Defining a Language

abstract syntax	
semantics	
concrete syntax	

Defining a Language

abstract syntax	*specification* source $\longrightarrow$ *valid?*
semantics	
concrete syntax	



Defining a Language

abstract syntax	*specification* source $\longrightarrow$ *valid?*
semantics	*reduction* source $\longrightarrow$ kernel
concrete syntax	

Defining a Language

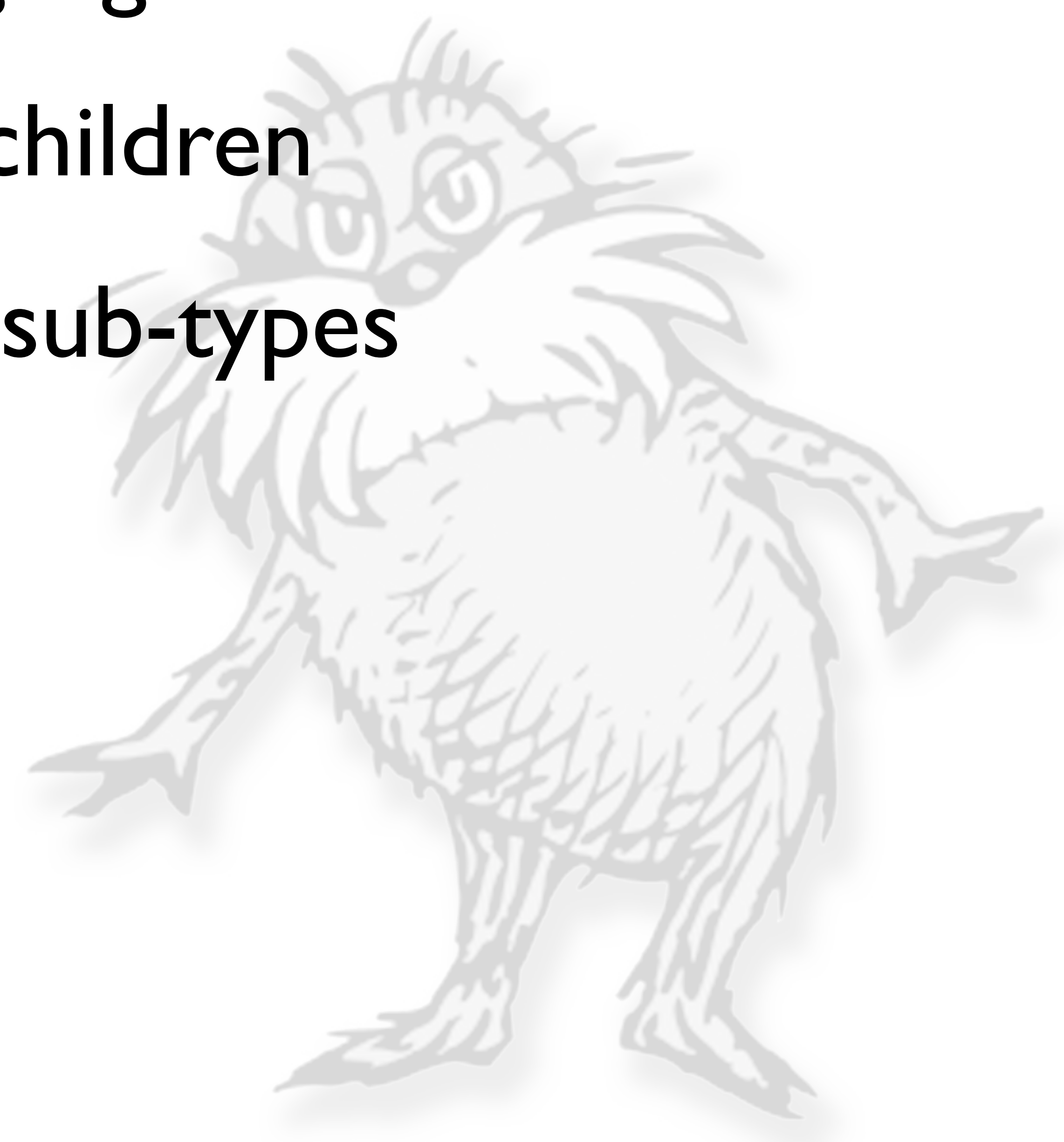
abstract syntax	*specification* source $\longrightarrow$ *valid?*
semantics	*reduction* source $\longrightarrow$ kernel
concrete syntax	*reduction* source $\longrightarrow$ view

grammar

language for defining languages

declaration of expected children

node types: abstract and sub-types



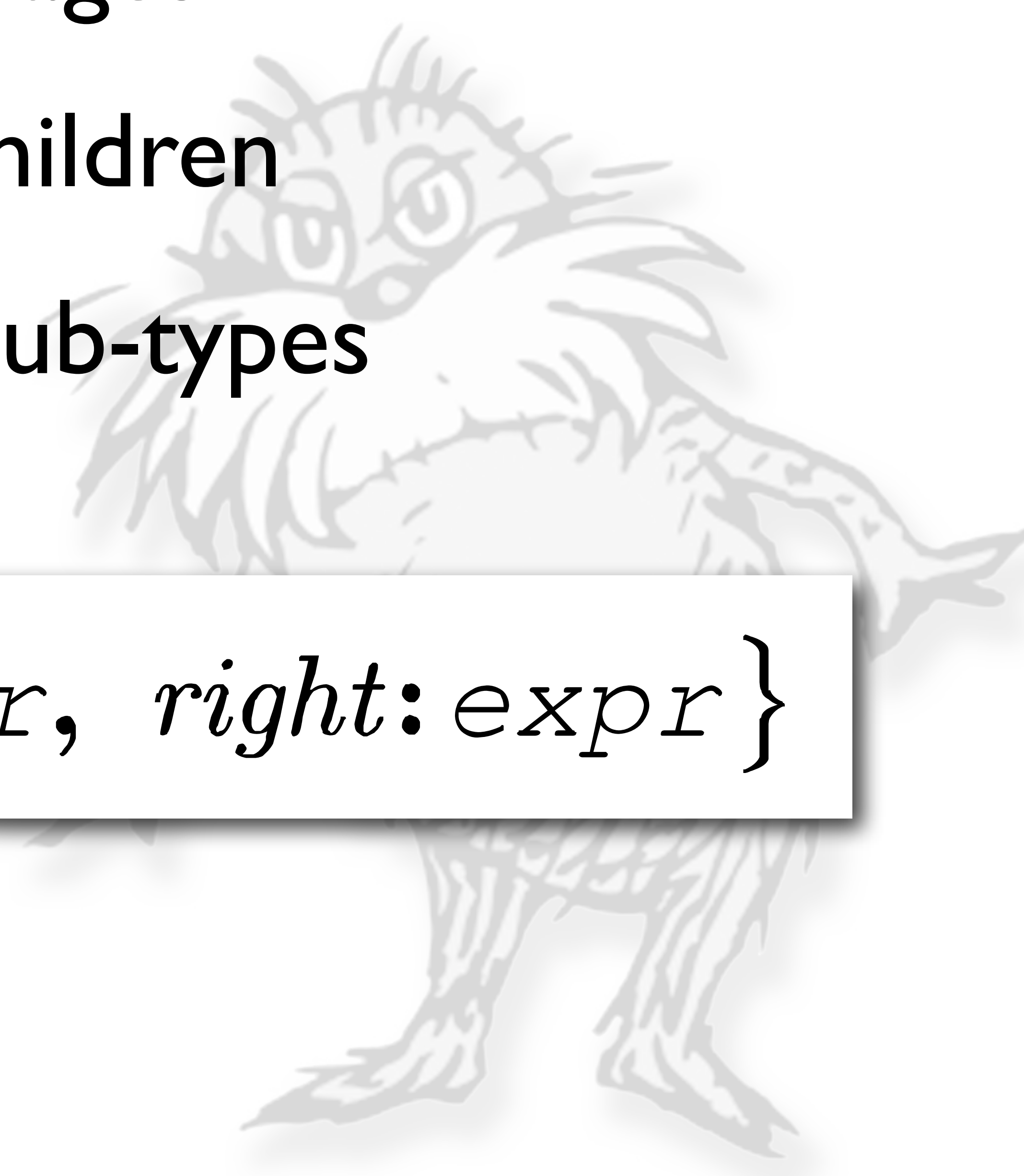
grammar

language for defining languages

declaration of expected children

node types: abstract and sub-types

$$\mathtt{expr} \leftarrow \mathtt{or}\ \{\mathit{left}\colon \mathtt{expr},\ \mathit{right}\colon \mathtt{expr}\}$$

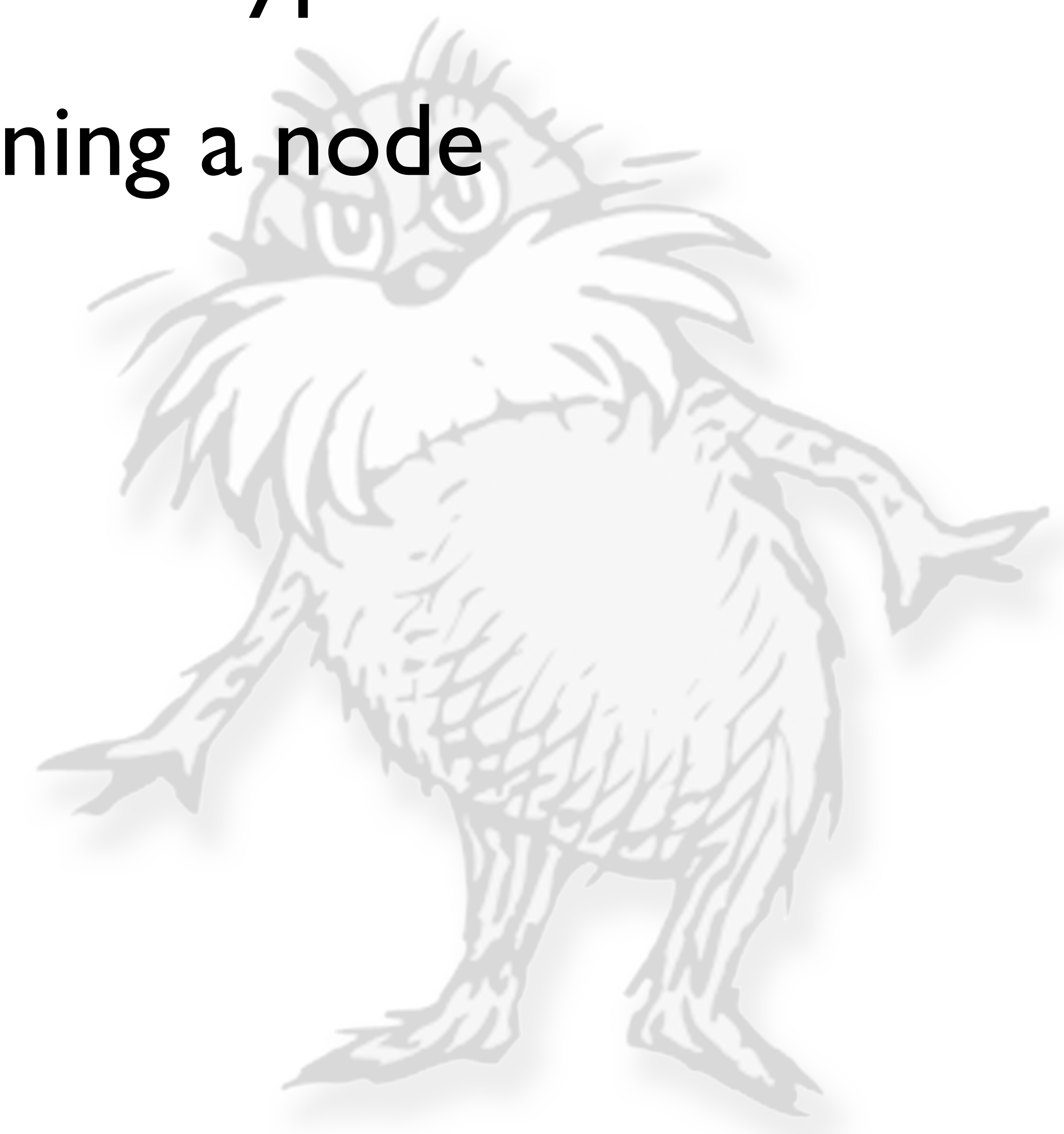


“Display” Reduction

defines concrete syntax for a type of node

a fragment of code returning a node

typically quasi-quoted



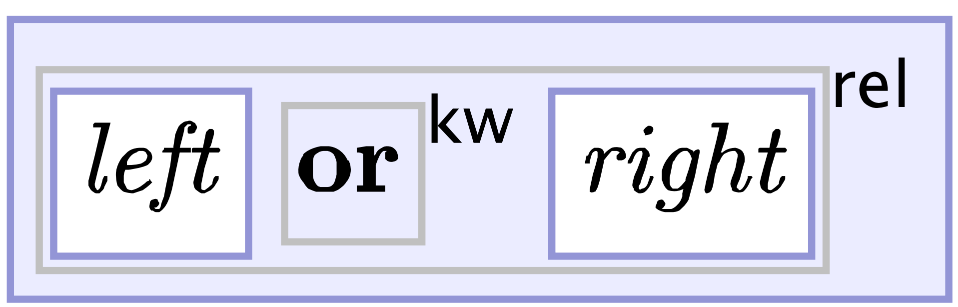
"Display" Reduction

defines concrete syntax for a type of node

a fragment of code returning a node

typically quasi-quoted

$$\mathtt{expr} \leftarrow \mathtt{or}\ \{\mathit{left}\colon \mathtt{expr},\ \mathit{right}\colon \mathtt{expr}\}$$

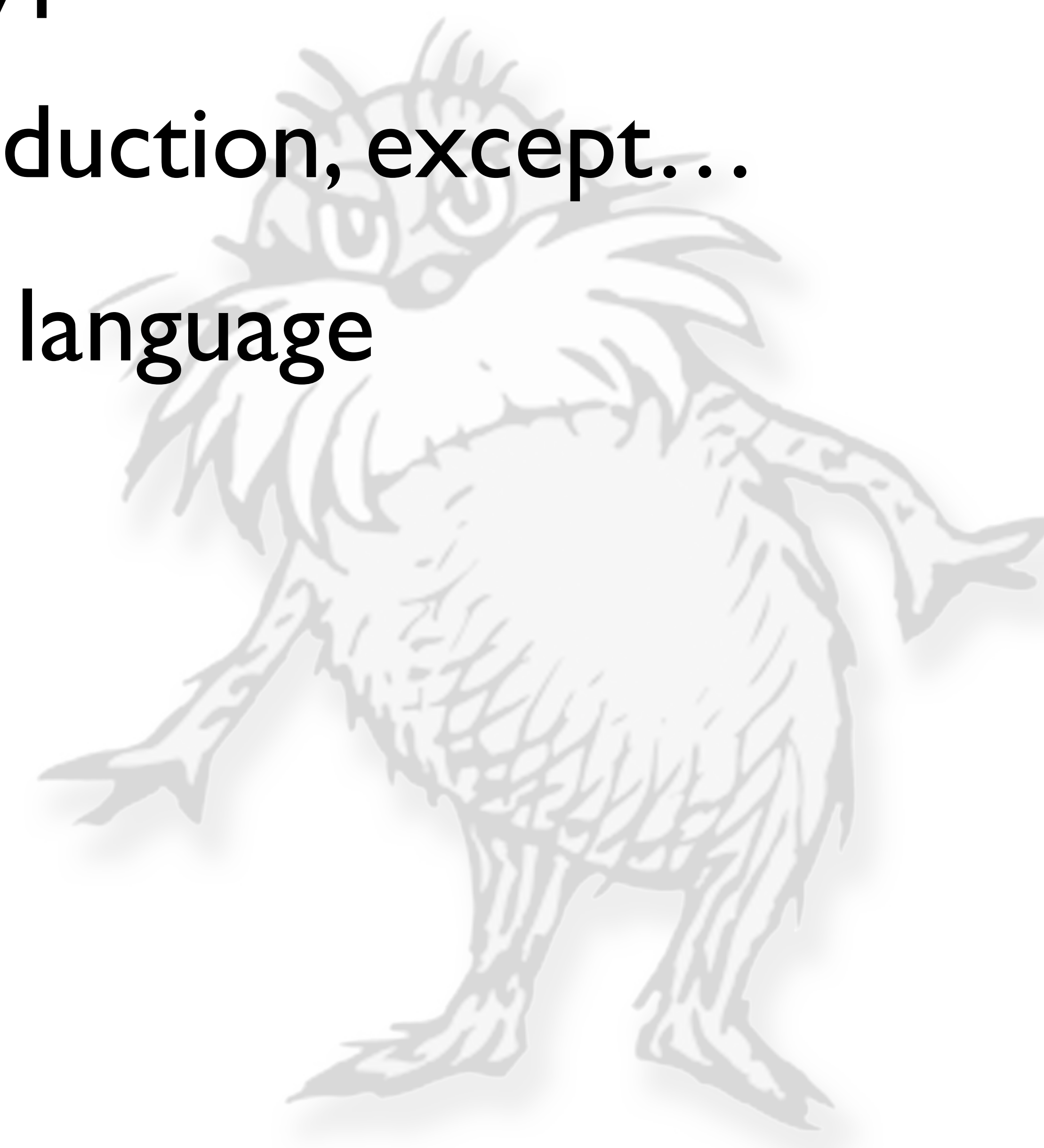


“Expand” Reduction

defines semantics for a type of node

exactly like a “display” reduction, except…

result is a node in target language



“Expand” Reduction

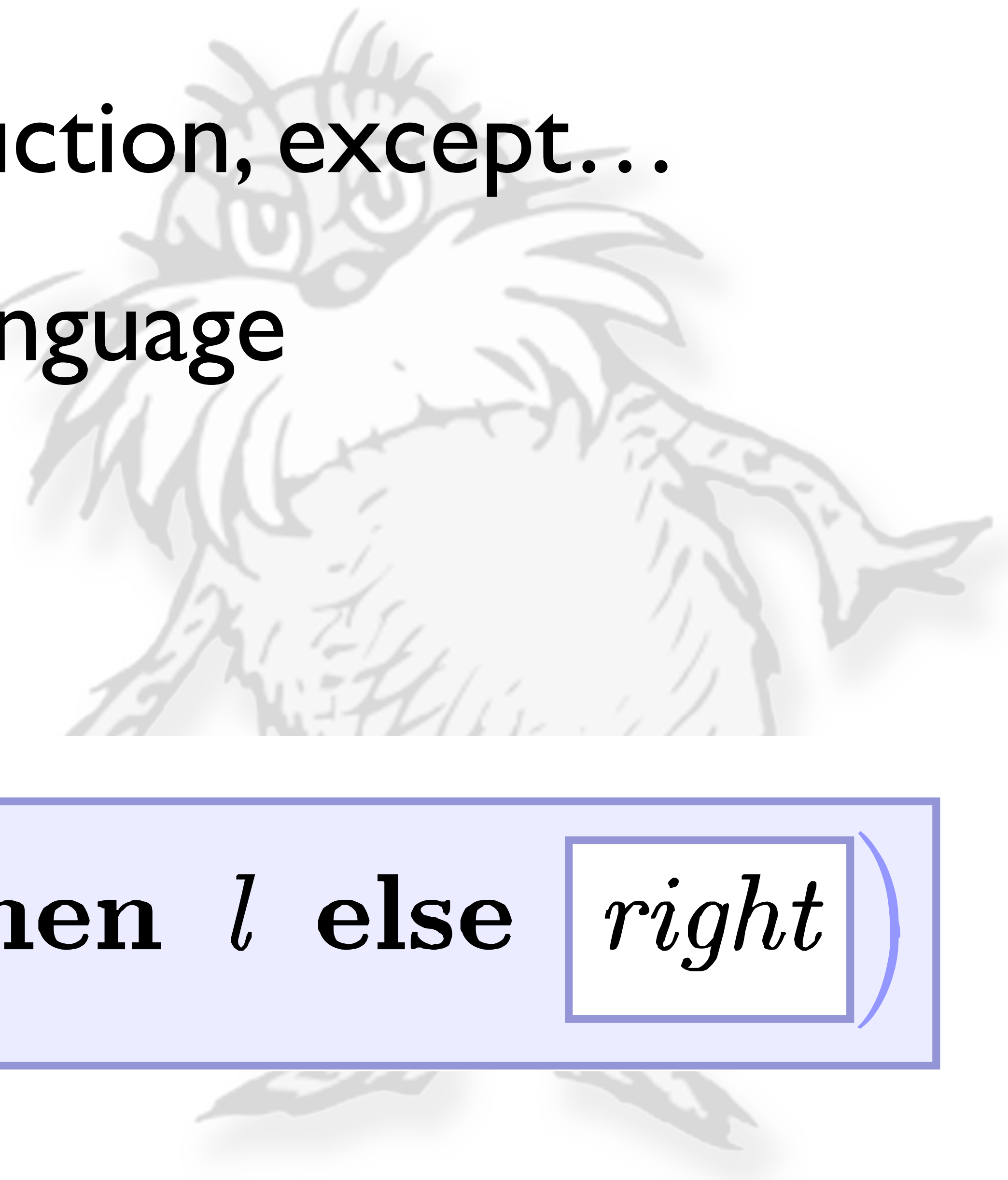
defines semantics for a type of node

exactly like a “display” reduction, except…

result is a node in target language

$$\boxed{\boxed{\mathit{left}}\ \boxed{\mathbf{or}}^{\mathsf{kw}}\ \boxed{\mathit{right}}}^{\mathsf{rel}} \rightarrow$$

$$\mathbf{let}\ l = \boxed{\mathit{left}}\ \mathbf{in}\ \left(\mathbf{if}\ l\ \mathbf{then}\ l\ \mathbf{else}\ \boxed{\mathit{right}}\right)$$

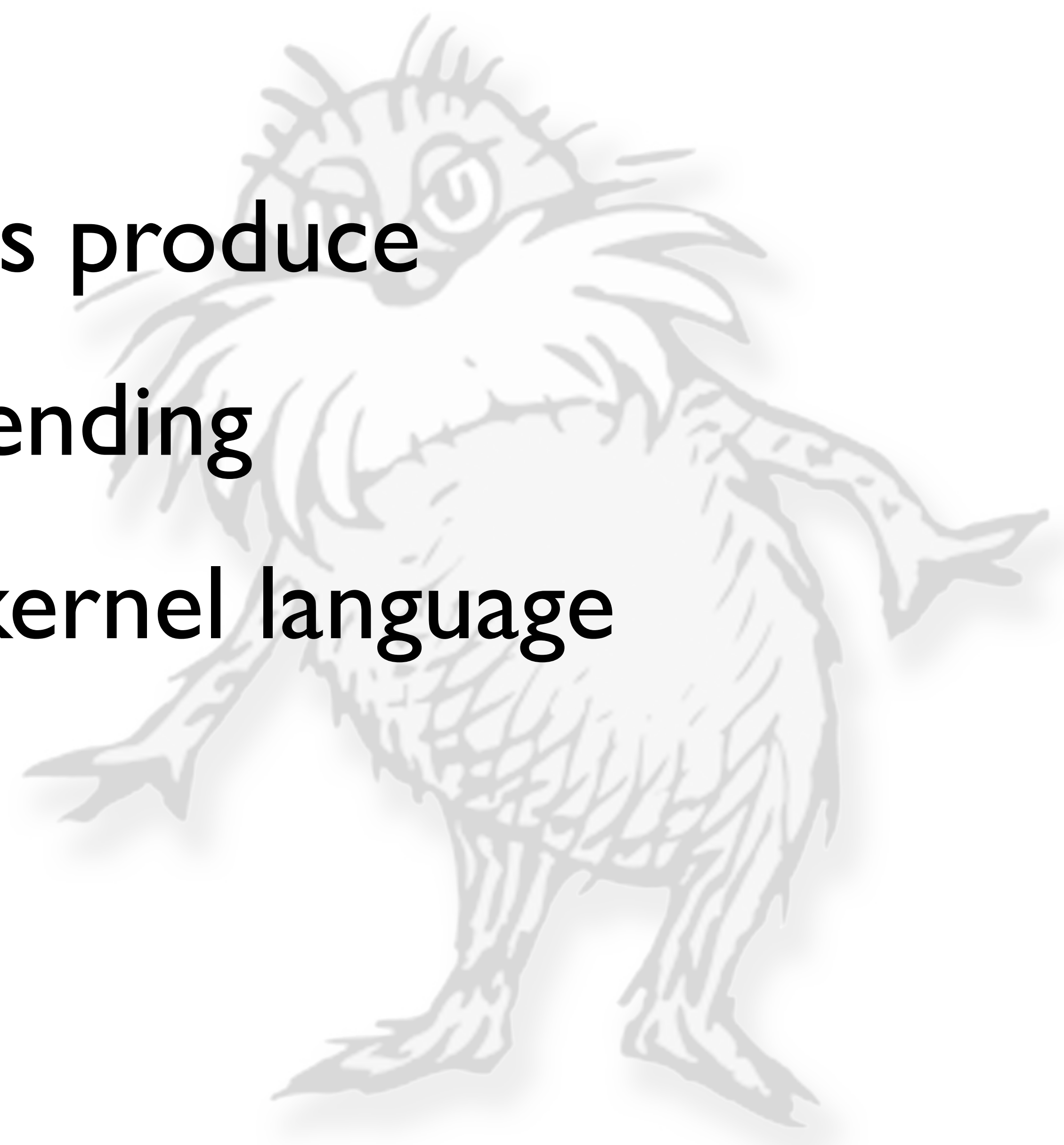


Kernel Language

what "expand" reductions produce

the language you are extending

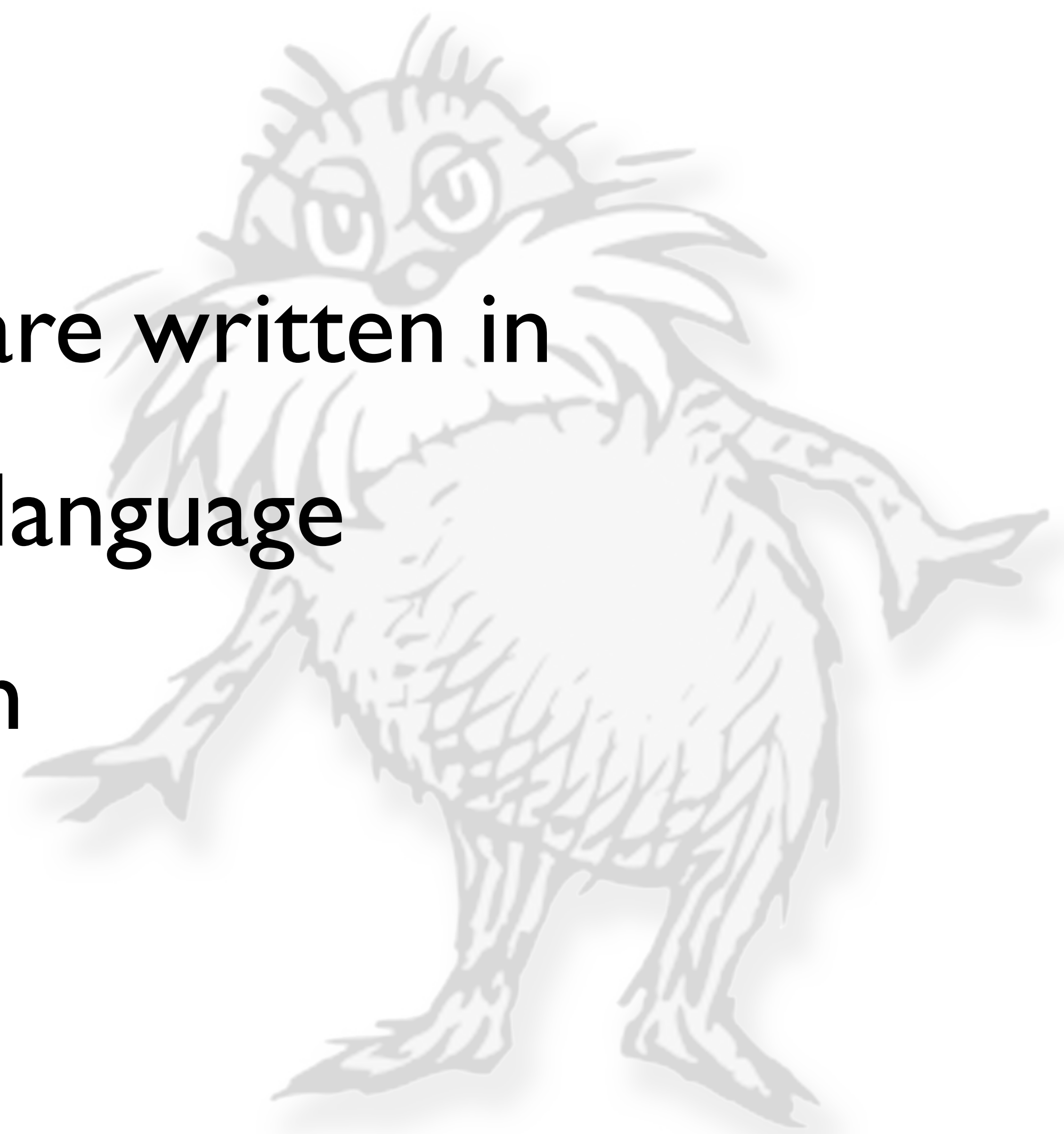
any language could be a kernel language



Host Kernel Language

Is...

1. the language reductions are written in
2. the most obvious target language
3. a test case for the system

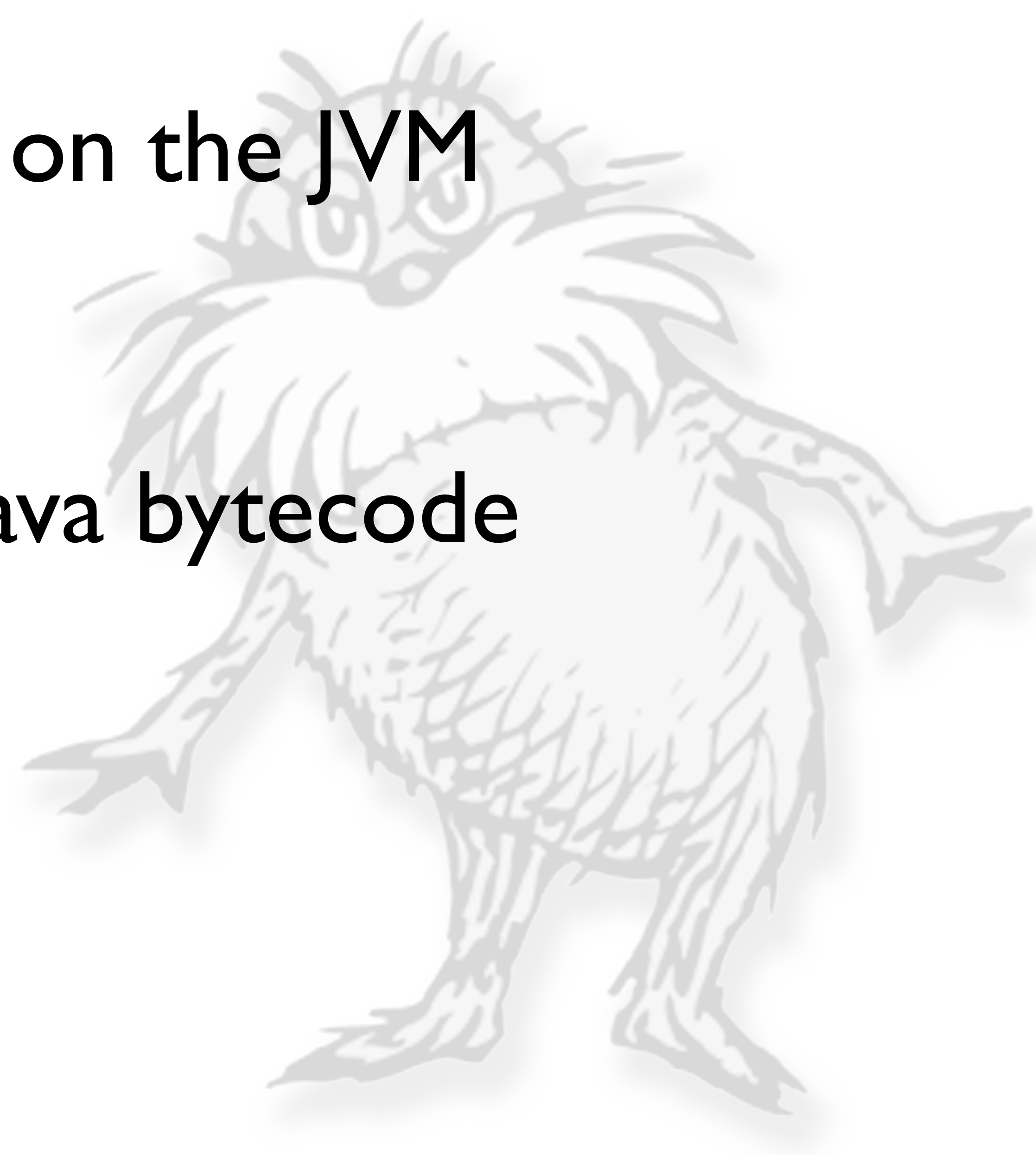


Host Kernel Language

Designed to be...

1. functional
2. meta
3. minimal
4. easy to implement

Clojure



a relatively new language on the JVM

a Lisp

compiled at runtime to Java bytecode

easy access to Java APIs

Lorax Kernel Language

a subset of the “special forms” of Clojure

LC + primitives + efficiency

`meta-compile` ⟶ s-expressions

`eval` ⟶ Clojure/Java value

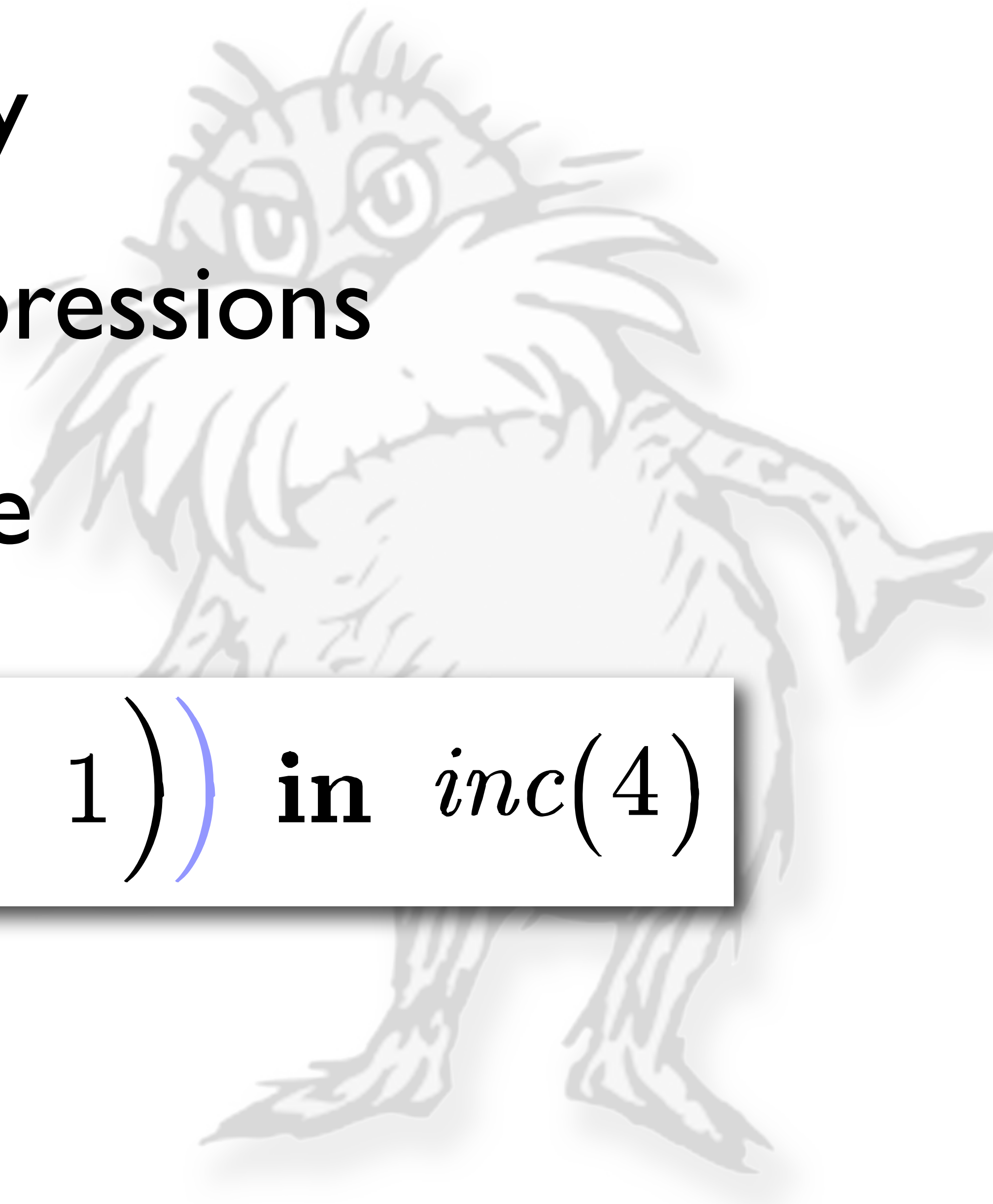
Lorax Kernel Language

a subset of the "special forms" of Clojure

LC + primitives + efficiency

`meta-compile` $\longrightarrow$ s-expressions

`eval` $\longrightarrow$ Clojure/Java value



$$\mathbf{let}\ \ inc\ =\ \left(\mathbf{fn}\ \ n' \ \to\ +\left(n',\ 1\right)\right)\ \ \mathbf{in}\ \ inc(4)$$

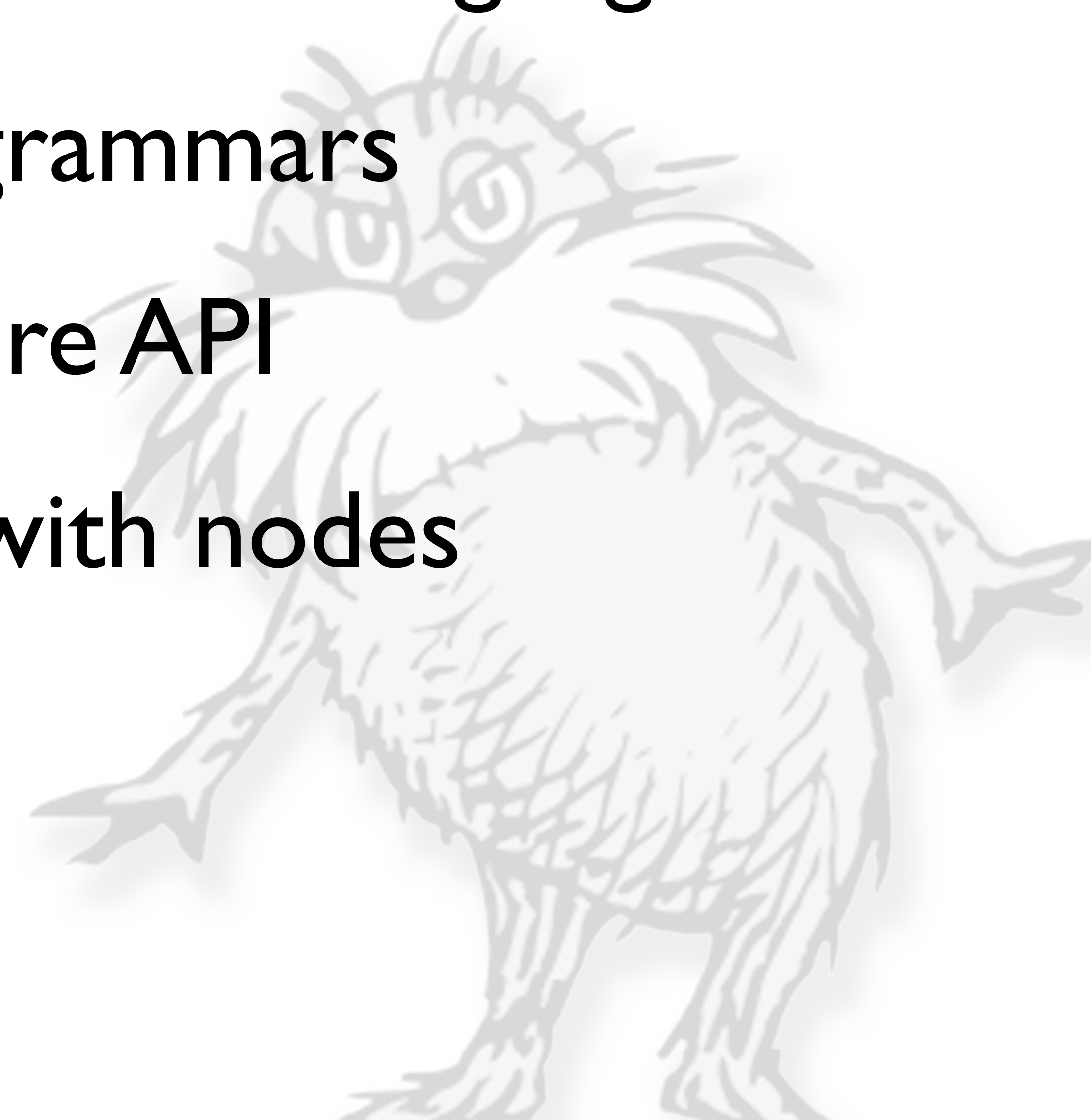
Lorax Core Language

built on (reduces to) the kernel language

implemented as several grammars

subset of the Clojure Core API

plus syntax for working with nodes



Presentation Languages

what "display" reductions produce

meant to be generic across source languages

different languages for different needs

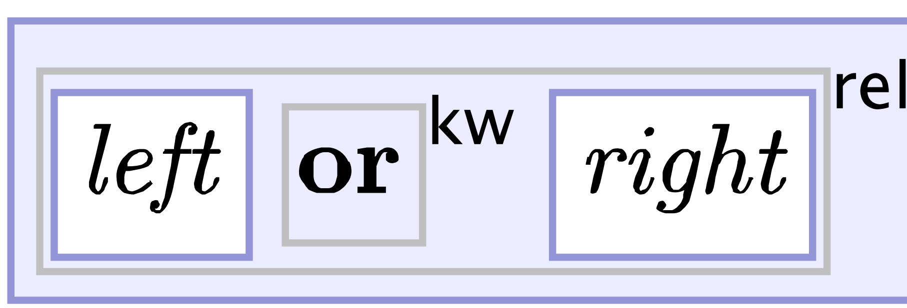
Presentation Languages

what "display" reductions produce

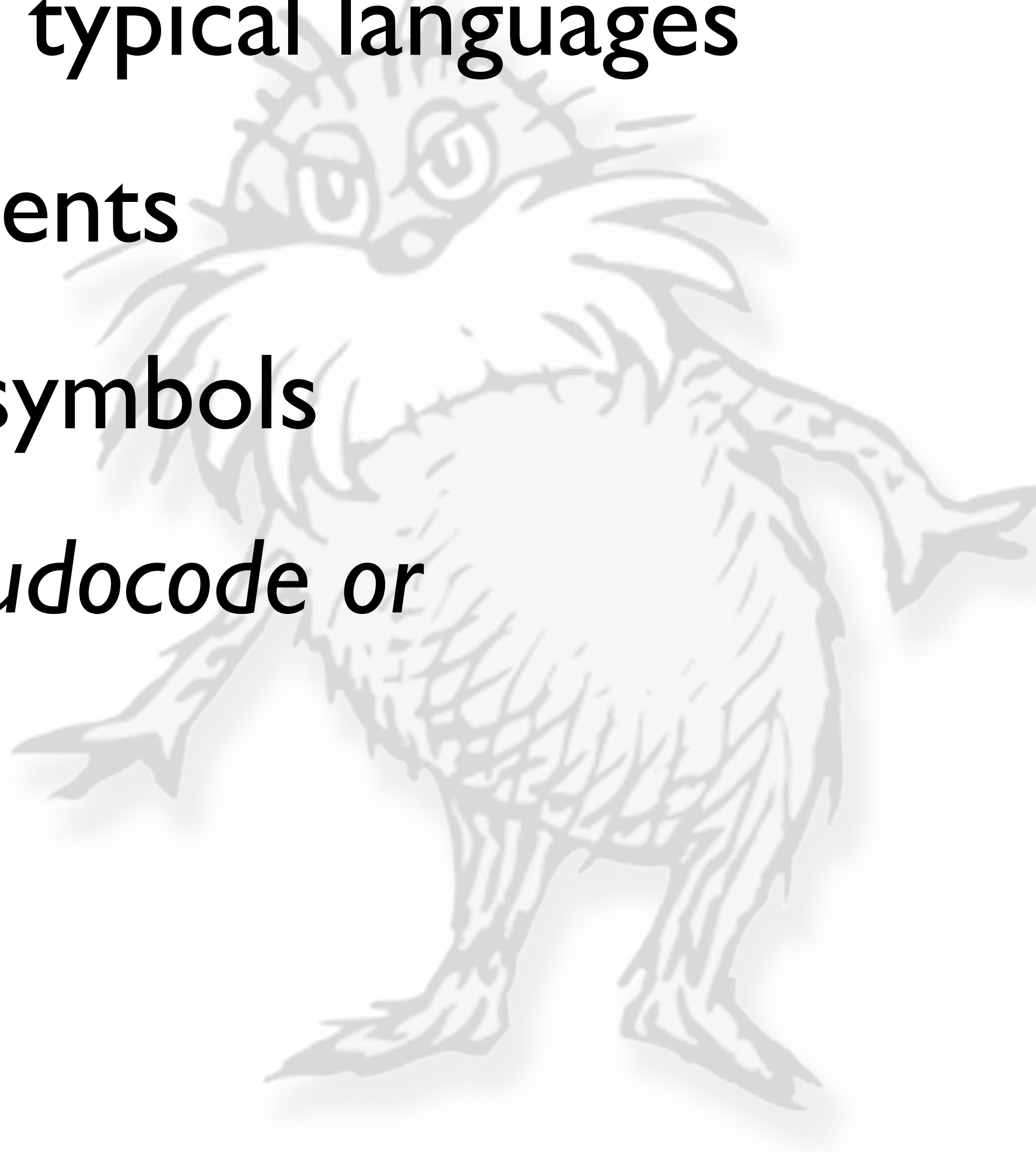
meant to be generic across source languages

different languages for different needs

$$expr \leftarrow or \left\{ left\colon expr,\ right\colon expr \right\}$$



Lorax Pres. Lang. (LPL)



presentation language for typical languages

common syntactical elements

plus additional signs and symbols

as good as publishable pseudocode or

a typical algebra textbook

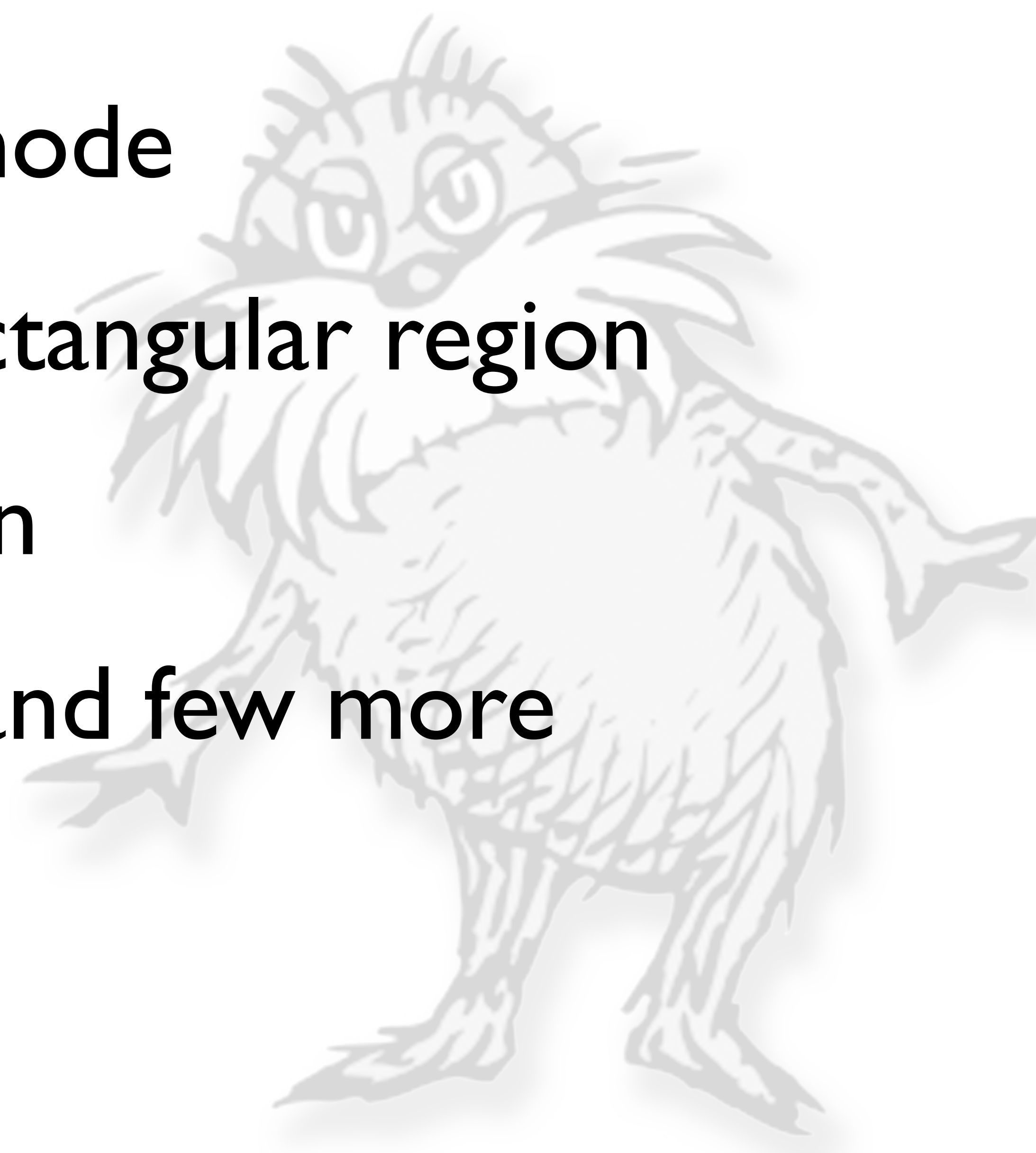
LPL Concepts

inspired by TEX’s math mode

each node occupies a rectangular region

parents surround children

atoms, *sequences*, *groups*, and few more



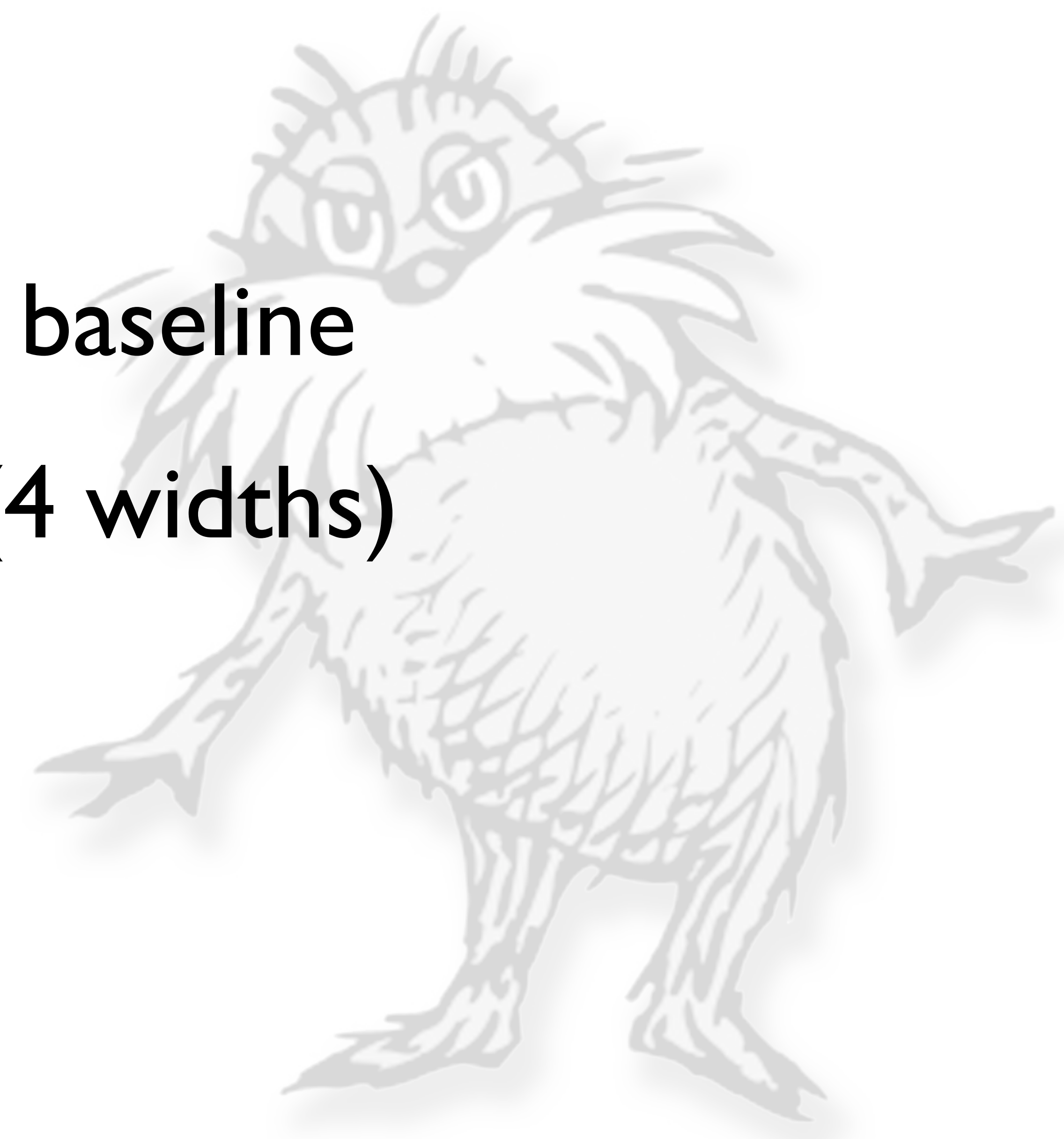
Atoms

type	examples
keyword	**if** **then** **else**
var	x g fib
num	1 2.0 3,000
string	abc Hello,␣world
symbol	$\longrightarrow$ $\in$ Σ

Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)

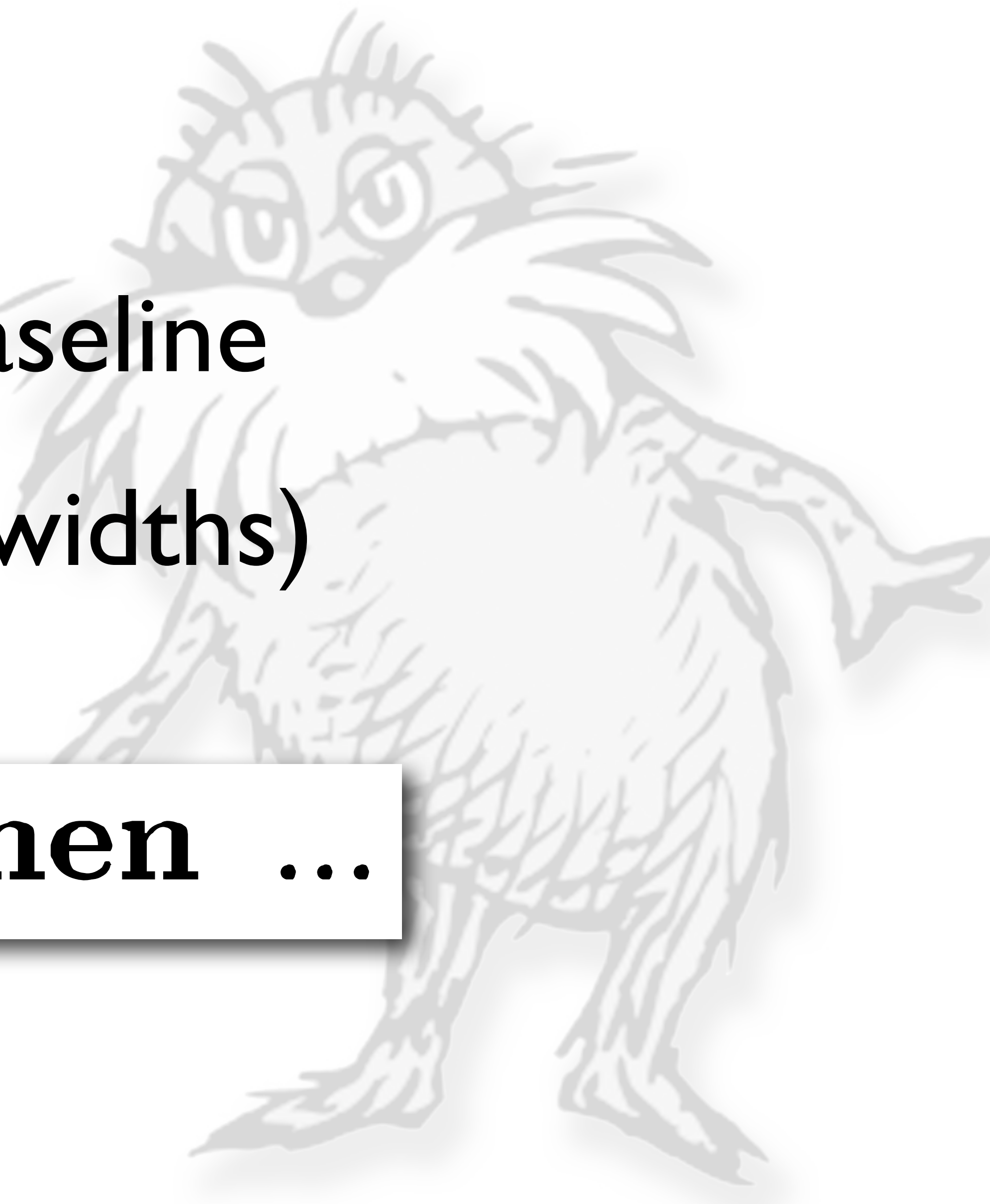


Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)

$$\textbf{if}\ \ x < \ y + 4!\ \ \textbf{then}\ \ \ldots$$

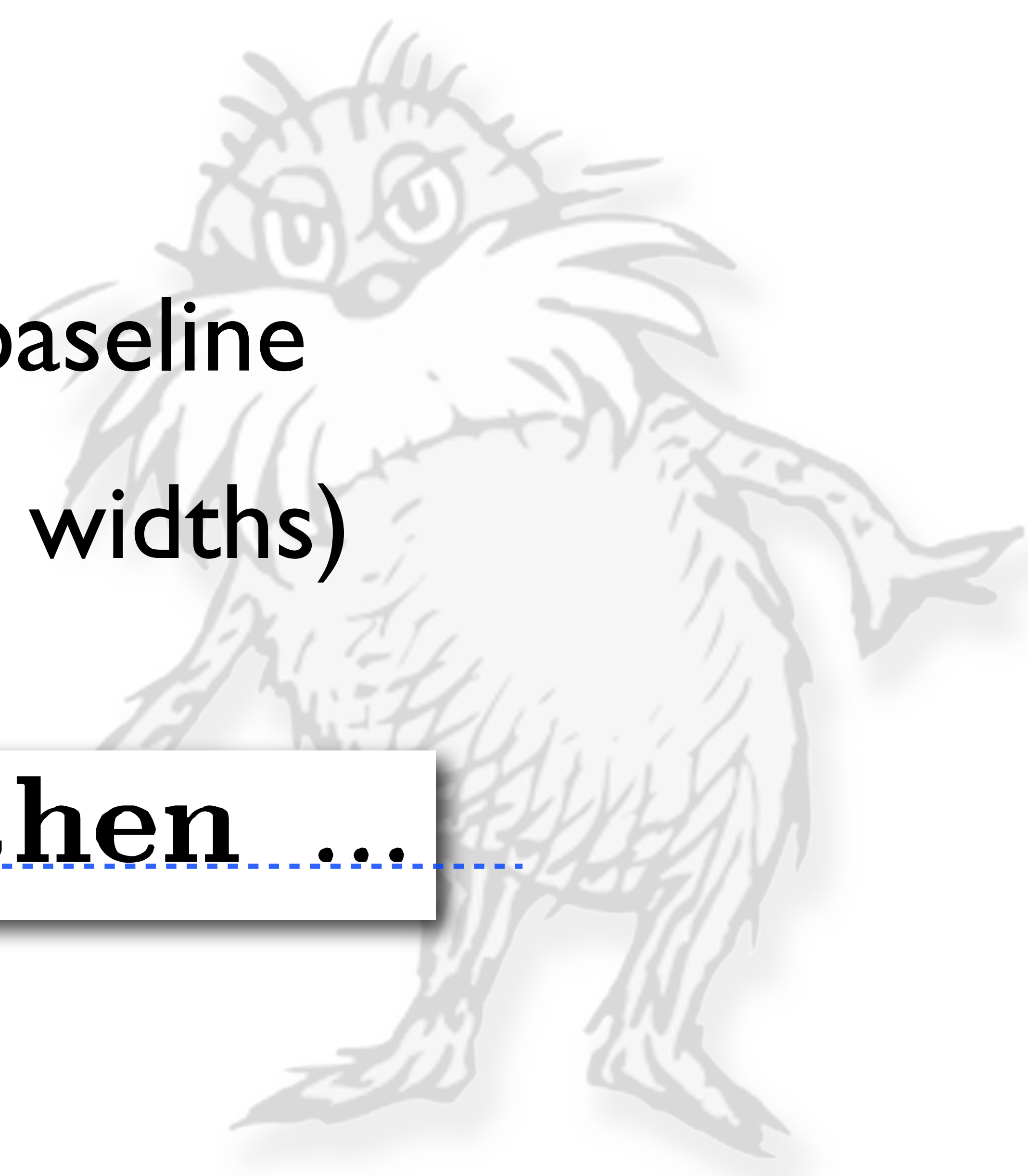


Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)

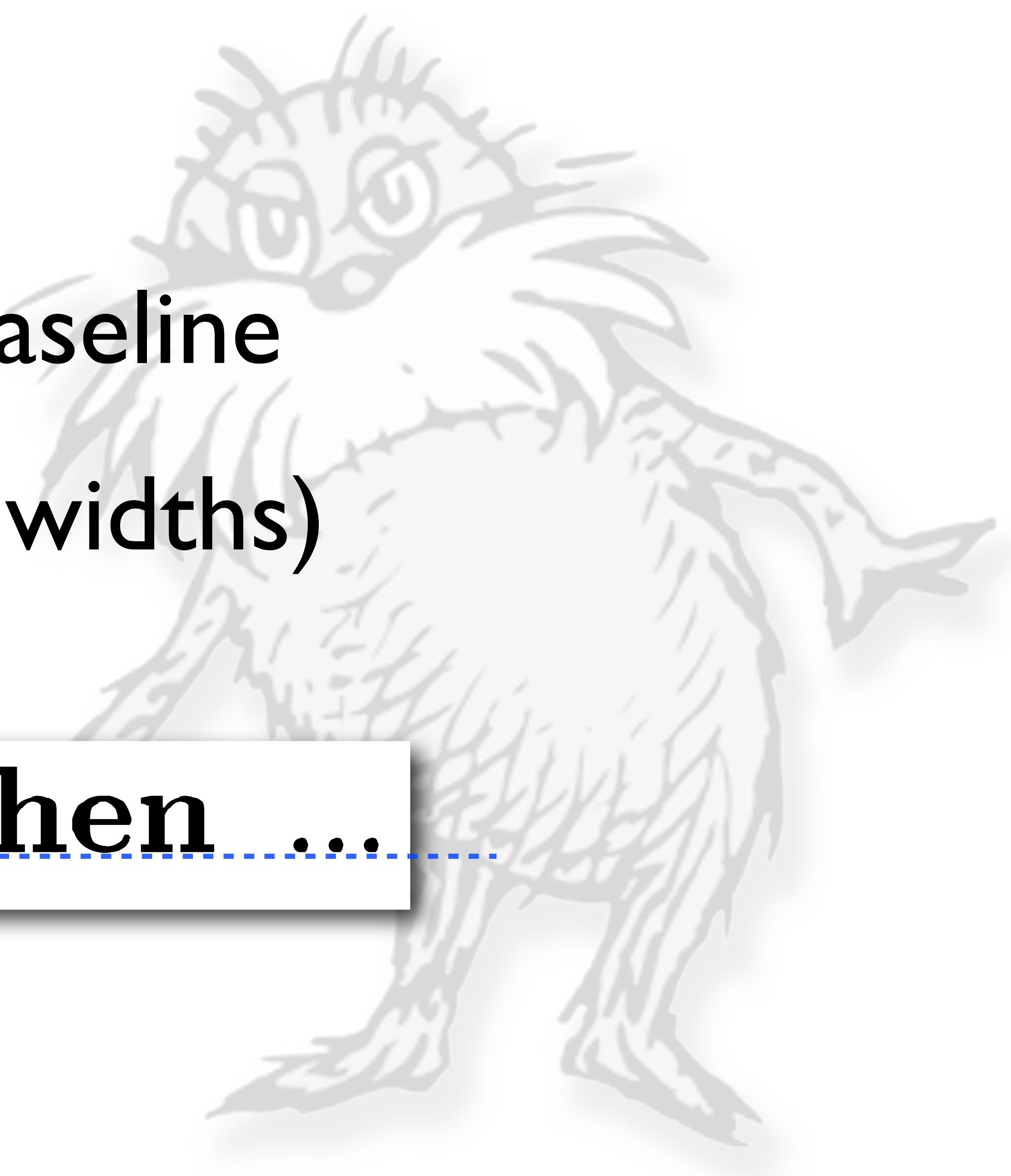
$$\mathbf{if}\ \ x < \ y + 4!\ \ \mathbf{then}\ \ \ldots$$



Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)



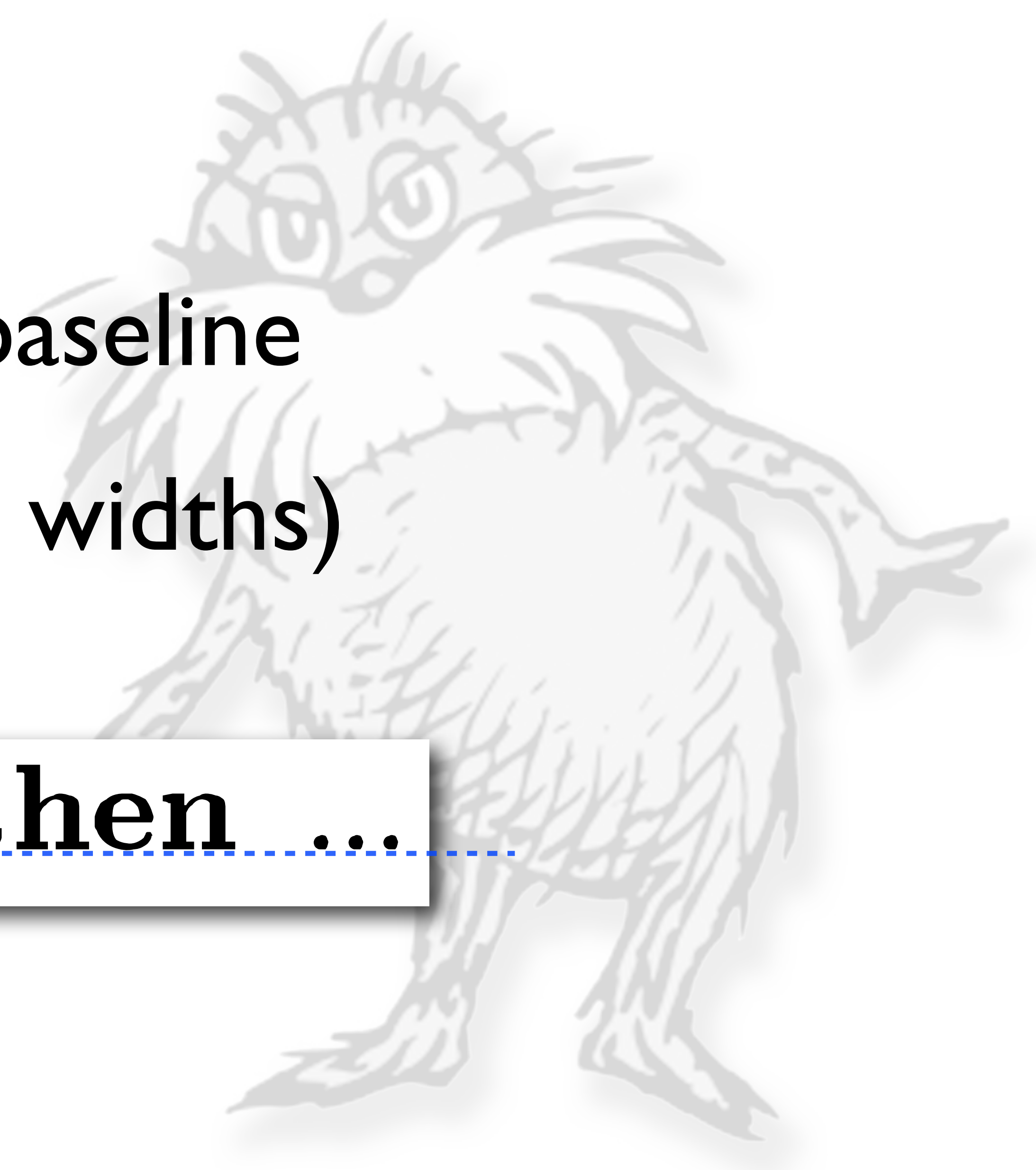
$$\mathbf{if}\ \ x < \boxed{y}\,\boxed{}\,\boxed{4!}\ \ \mathbf{then}\ \ \ldots$$

Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)

$$\mathbf{if}\ \ \boxed{x}\ \ \boxed{<}\ \ \boxed{\boxed{y}\boxed{\boxplus}\boxed{4!}}\ \ \mathbf{then}\ \ \ldots$$



Sequences

arrange child nodes:

- in a horizontal line
- aligned to a common baseline
- separated by spaces (4 widths)

$$\mathbf{if}\ \ x < y + 4!\ \ \mathbf{then}\ \ \ldots$$

Others

√, fractions, superscripts

( ) [ ] { } ⌊ ⌋ ⌈ ⌉ ⟨ ⟩ |

Others

√, fractions, superscripts

( ) [ ] { } ⌊ ⌋ ⌈ ⌉ ⟨ ⟩ |

$$\frac{1}{3} \times \sqrt{3^2}$$

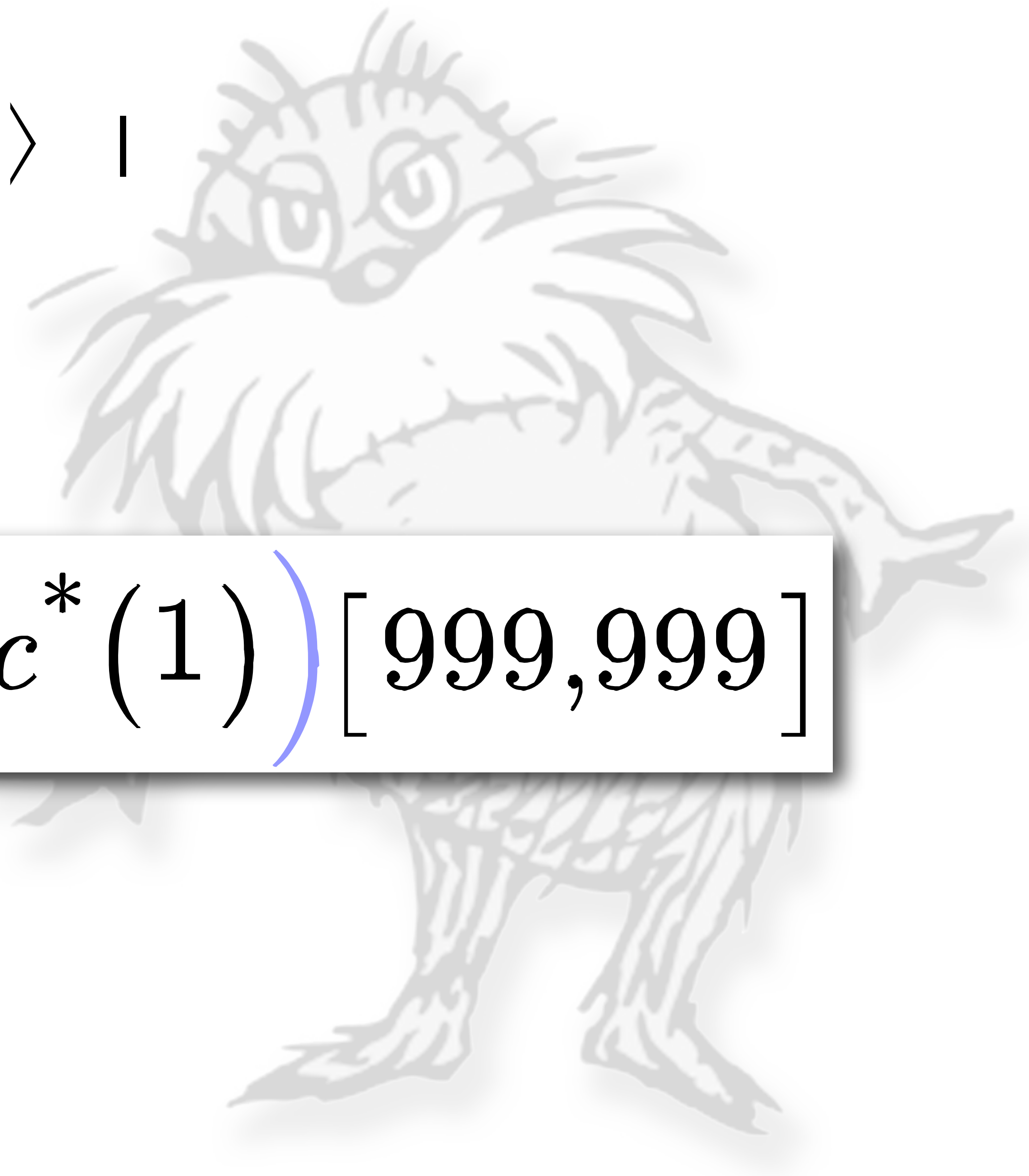
Others

$\surd$, fractions, superscripts

$(\ ) \ [\ ] \ \{ \ \} \ \lfloor \ \rfloor \ \lceil \ \rceil \ \langle \ \rangle \ |$

$$\frac{1}{3} \times \sqrt{3^2}$$

$$\left(inc^*(1) \right) [999{,}999]$$



Editor

UI for viewing/editing Lorax programs

implemented in Clojure, runs on JVM

uses “display” reductions from grammars

can also evaluate expressions

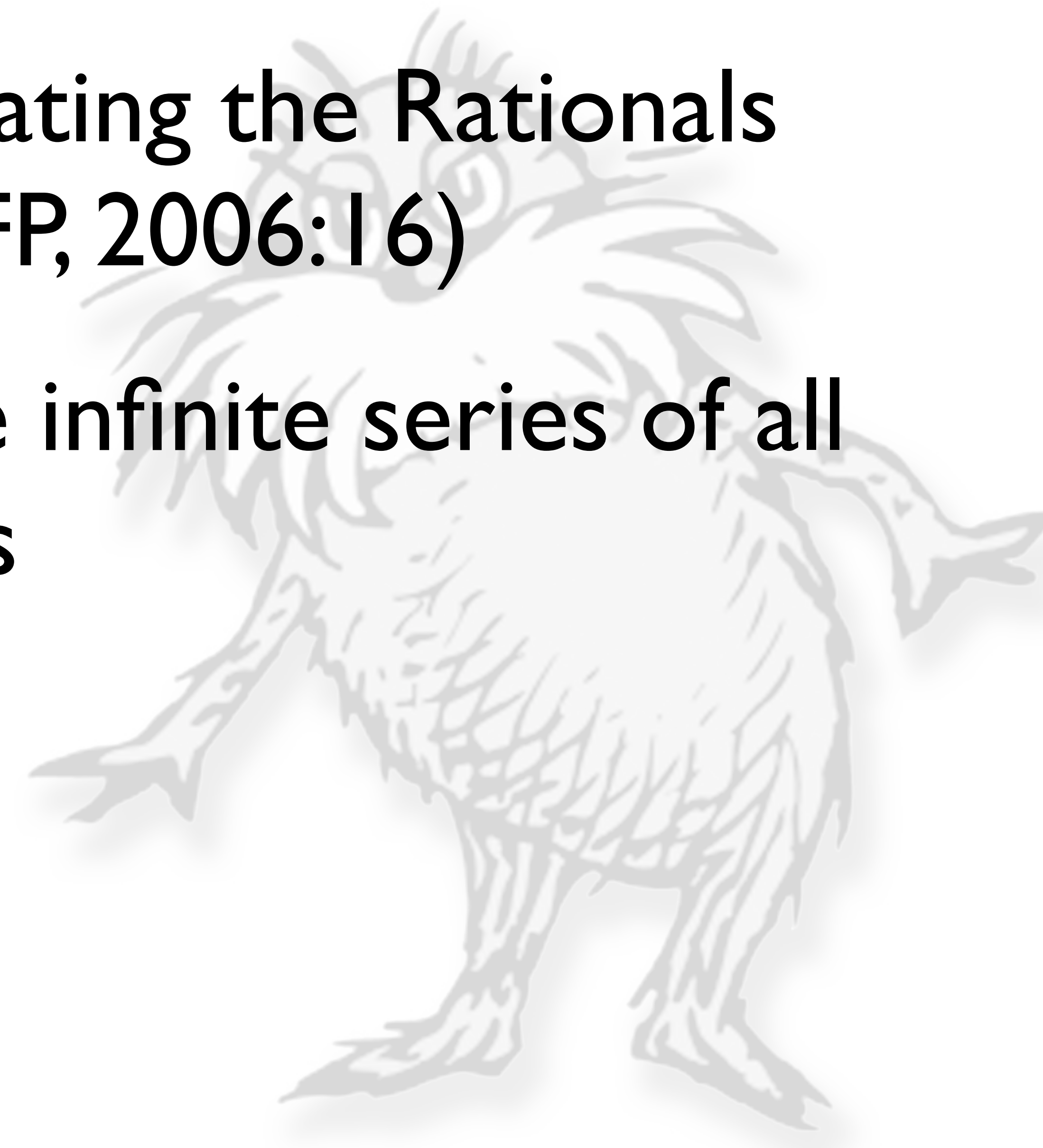
basic editing, but not yet an integrated tool

Demo

Rationals

Functional Pearl: Enumerating the Rationals (Jeremy Gibbons, et al., JFP, 2006:16)

write a generator for the infinite series of all positive rational numbers



Matrix Traversal

$$
\begin{array}{cccccc}
{}^{1}\!/_{1} & {}^{2}\!/_{1} & {}^{3}\!/_{1} & \cdots & {}^{m}\!/_{1} & \cdots \\
{}^{1}\!/_{2} & {}^{2}\!/_{2} & {}^{3}\!/_{2} & \cdots & {}^{m}\!/_{2} & \cdots \\
\vdots & & & & & \\
{}^{1}\!/_{n} & {}^{2}\!/_{n} & {}^{3}\!/_{n} & \cdots & {}^{m}\!/_{n} & \cdots \\
\vdots & & & & &
\end{array}
$$

Matrix Traversal

$$
\begin{array}{cccccc}
1/1 & 2/1 & 3/1 & \cdots & m/1 & \cdots \\
1/2 & 2/2 & 3/2 & \cdots & m/2 & \cdots \\
\vdots & & & & & \\
1/n & 2/n & 3/n & \cdots & m/n & \cdots \\
\vdots & & & & &
\end{array}
$$

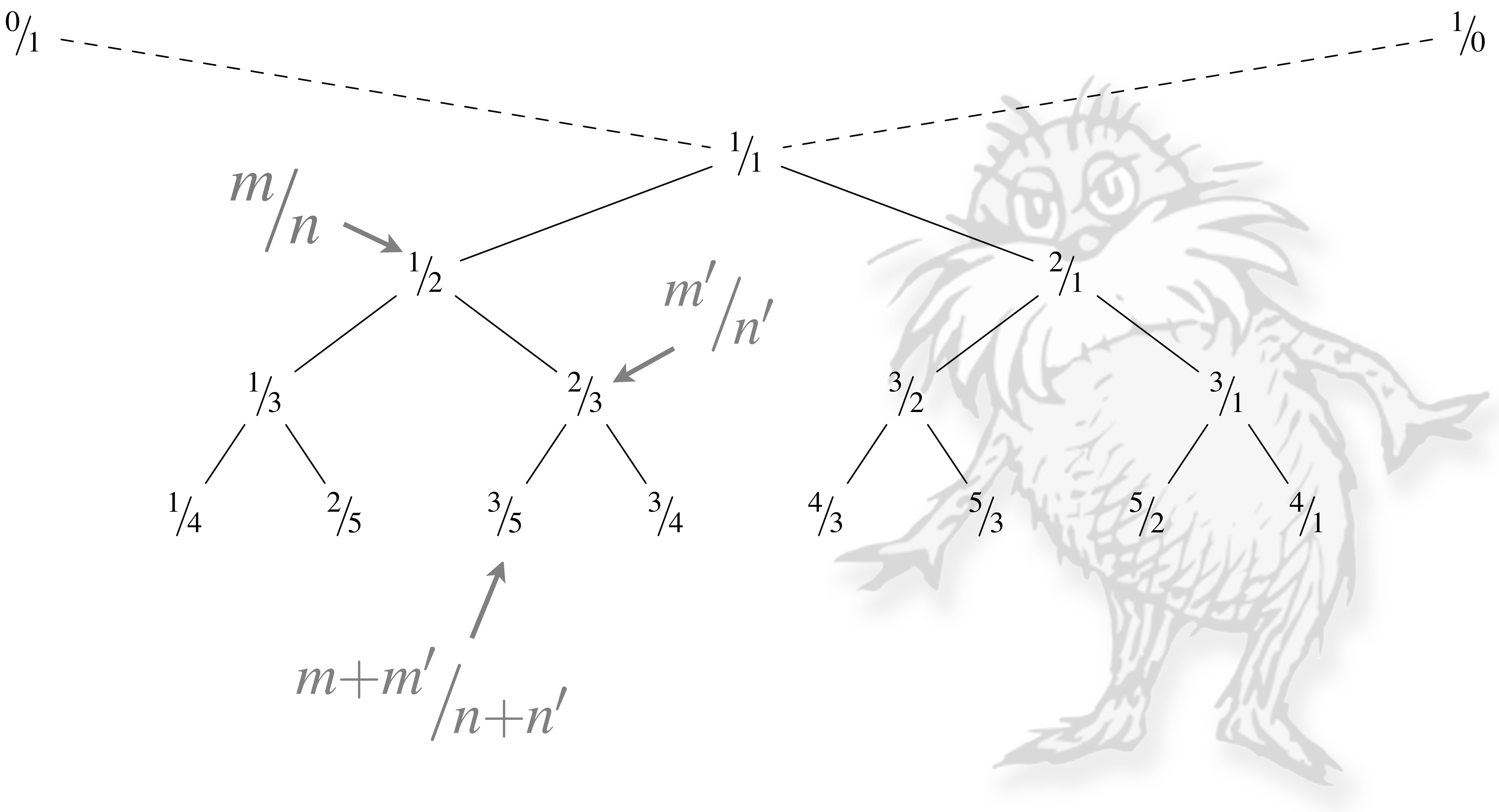
Matrix Traversal

$$\begin{array}{cccccc}
{}^{1}/_{1} & {}^{2}/_{1} & {}^{3}/_{1} & \cdots & {}^{m}/_{1} & \cdots \\
{}^{1}/_{2} & {}^{2}/_{2} & {}^{3}/_{2} & \cdots & {}^{m}/_{2} & \cdots \\
\vdots & & & & & \\
{}^{1}/_{n} & {}^{2}/_{n} & {}^{3}/_{n} & \cdots & {}^{m}/_{n} & \cdots \\
\vdots & & & & &
\end{array}$$

Matrix Traversal

$$\begin{array}{cccccc}
{}^{1}/_{1} & {}^{2}/_{1} & {}^{3}/_{1} & \cdots & {}^{m}/_{1} & \cdots \\
{}^{1}/_{2} & {}^{2}/_{2} & {}^{3}/_{2} & \cdots & {}^{m}/_{2} & \cdots \\
 & \vdots & & & & \\
{}^{1}/_{n} & {}^{2}/_{n} & {}^{3}/_{n} & \cdots & {}^{m}/_{n} & \cdots \\
 & \vdots & & & &
\end{array}$$

Stern-Brocot Tree



A Formula

$$x' = 1/(\lfloor x \rfloor + 1 - \{x\})$$

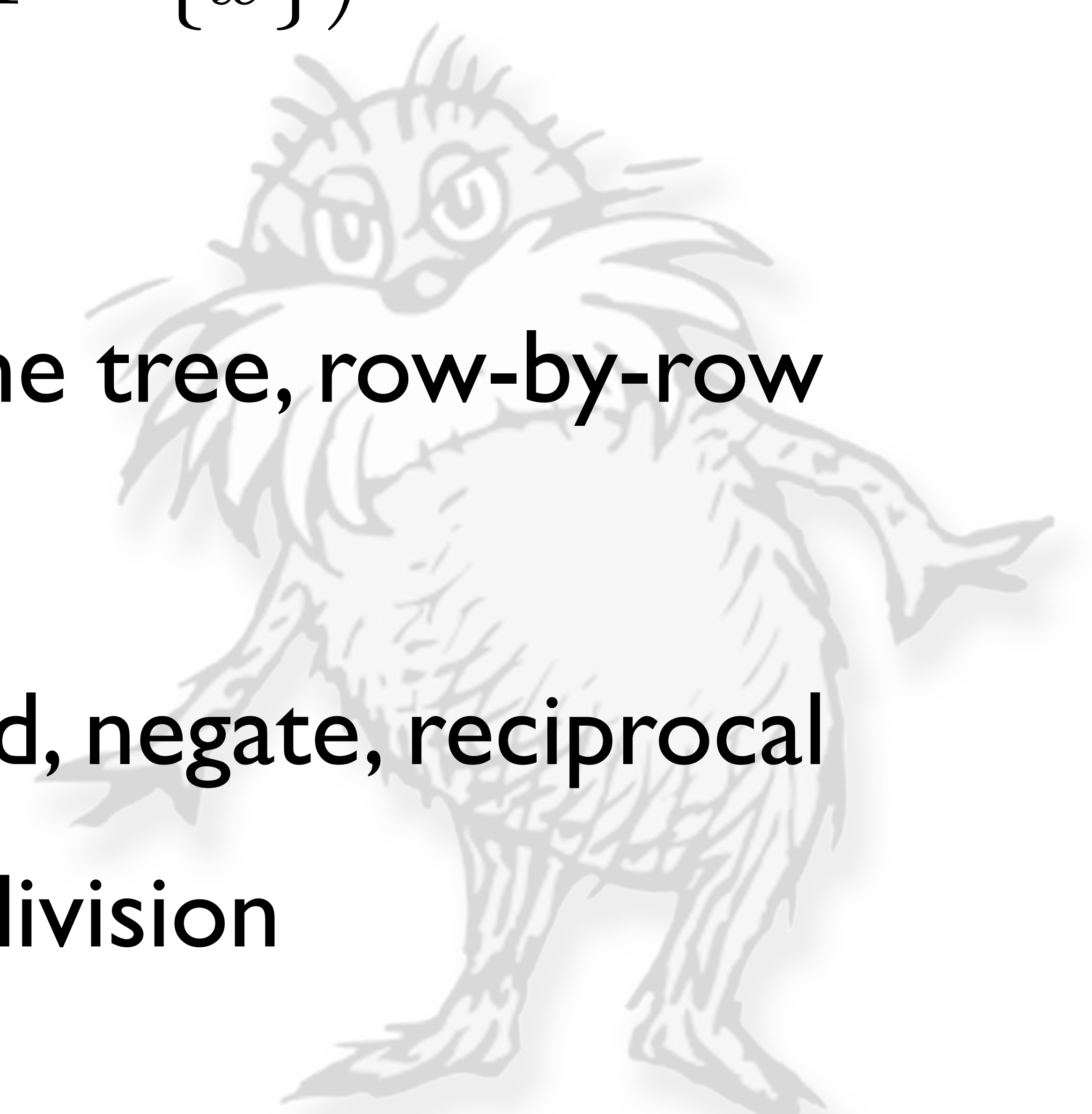
start with $x = 1$

generates the ratios in the tree, row-by-row

$1, \tfrac{1}{2}, 2, \tfrac{1}{3}, \ldots$

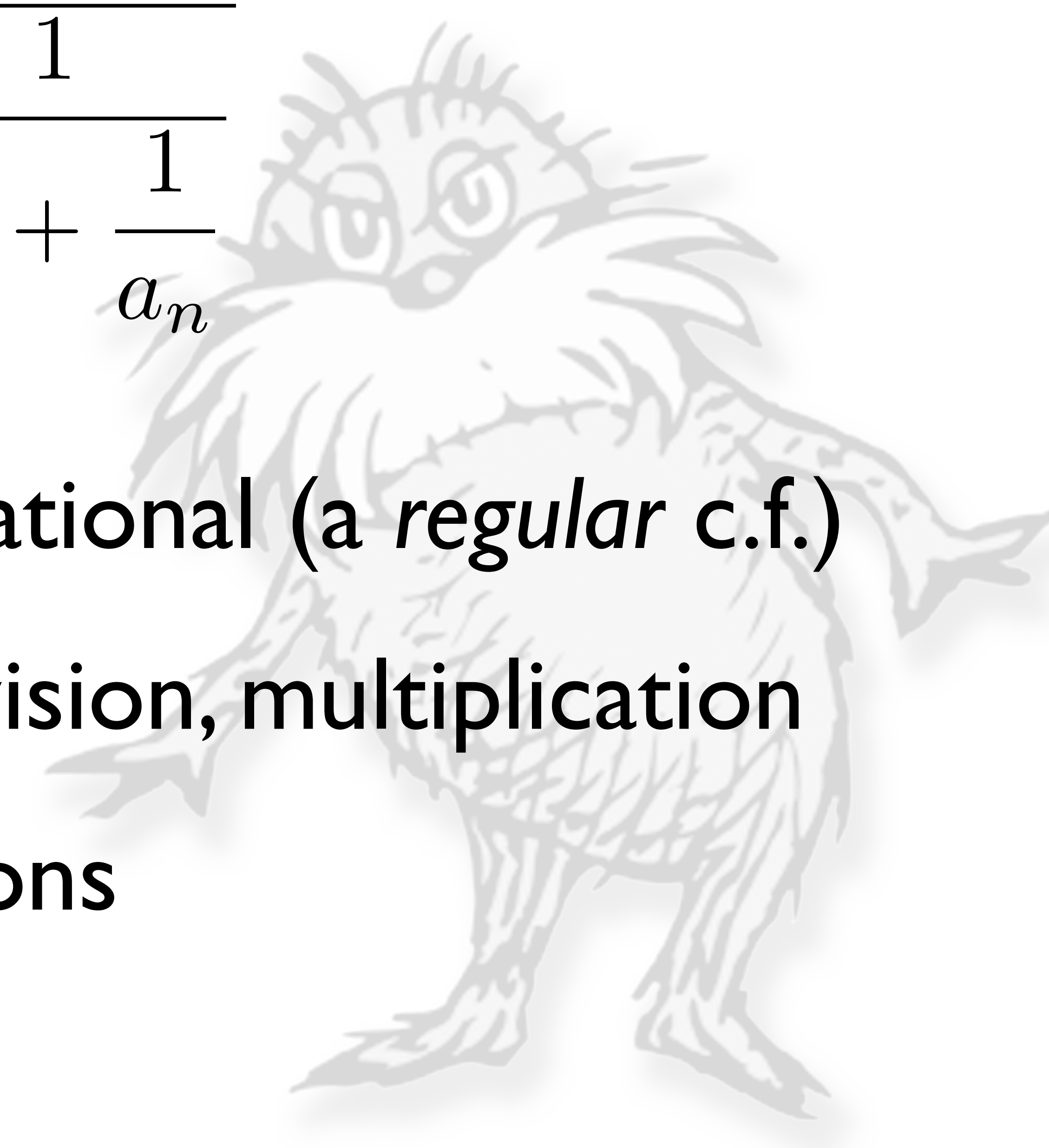
int. part, fraction part, add, negate, reciprocal

not cheap: $\lfloor x \rfloor$ involves division



Continued Fraction

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\cdots + \cfrac{1}{a_n}}}$$



a unique repr. for each rational (a *regular* c.f.)

5 operations with no division, multiplication

$O(1)$ additions/subtractions

Cont. Fractions: Haskell

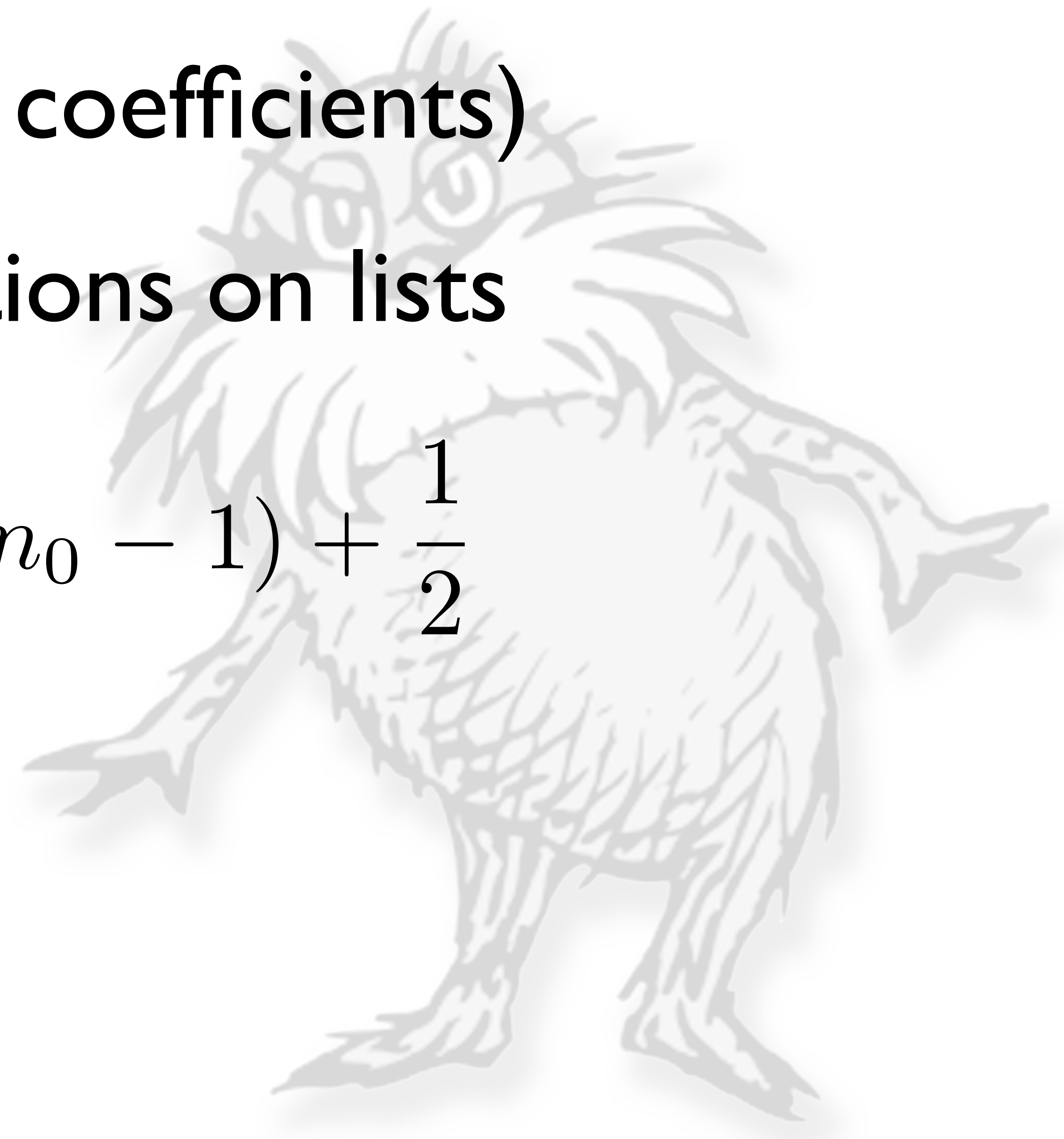
need a new kind of runtime value

use a list of integers (the coefficients)

write operations as functions on lists

ex.:

$$-\left(n_0 + \frac{1}{2}\right) = (-n_0 - 1) + \frac{1}{2}$$



Cont. Fractions: Haskell

need a new kind of runtime value

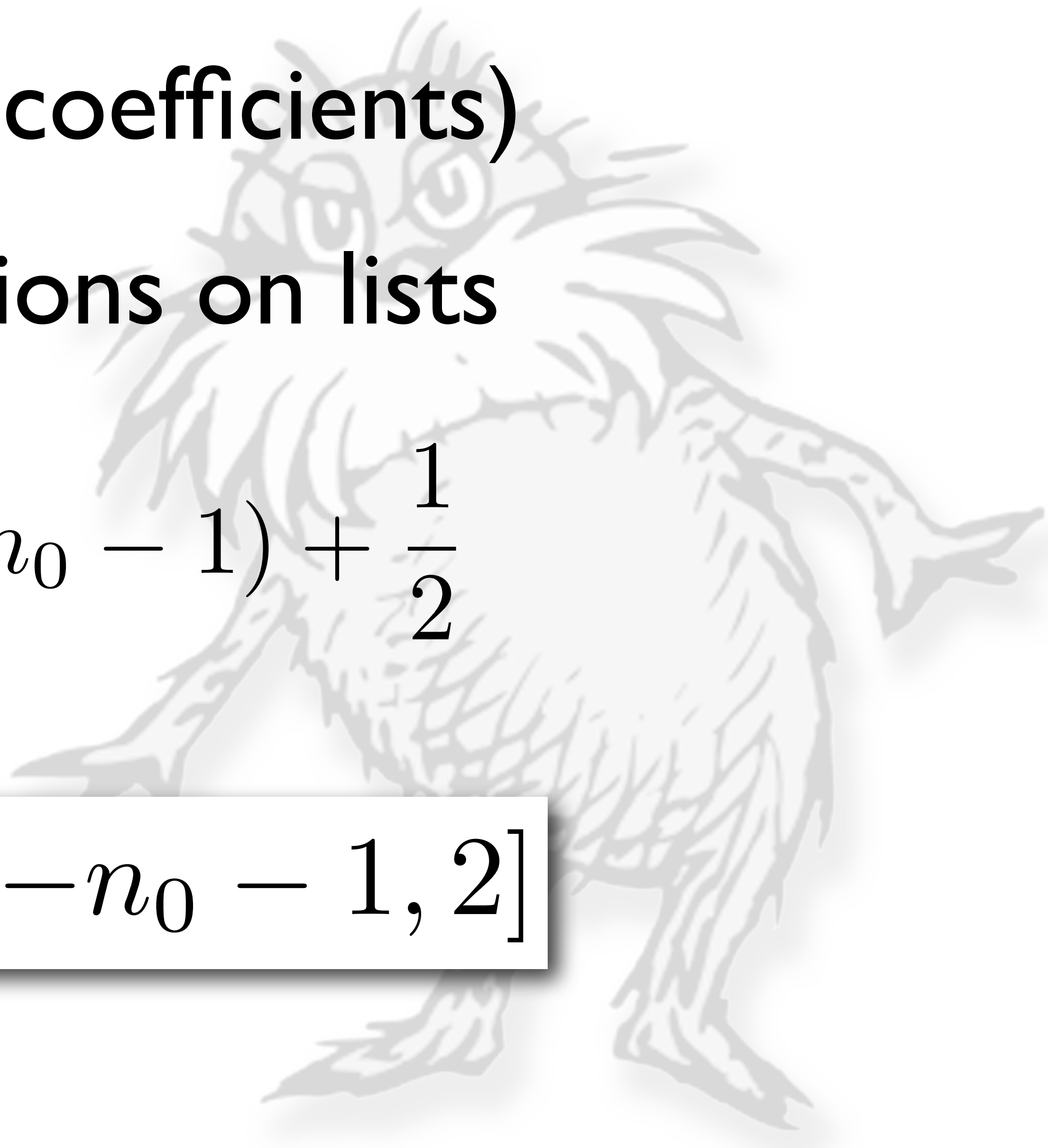
use a list of integers (the coefficients)

write operations as functions on lists

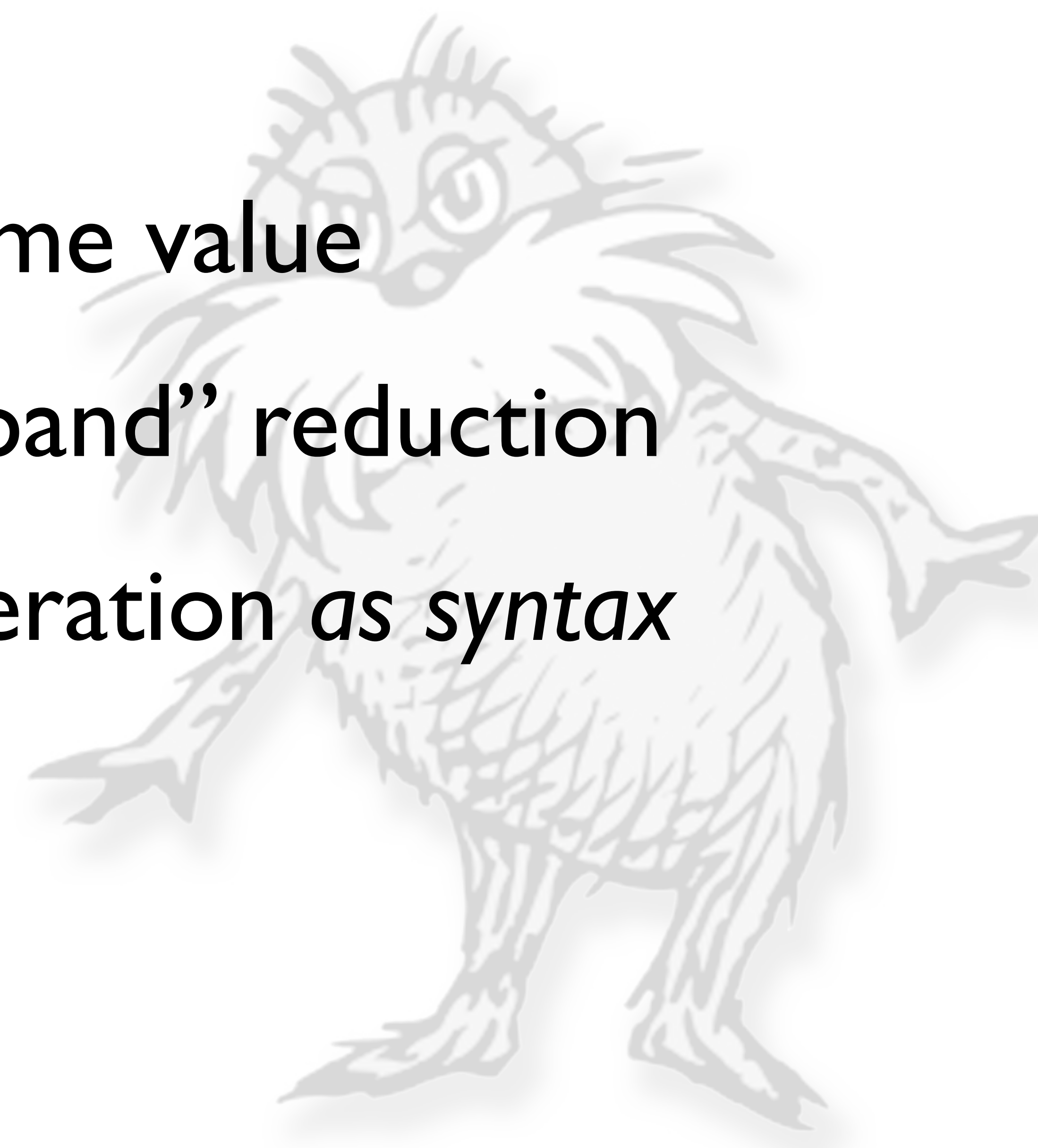
ex.:

$$-\left(n_0 + \frac{1}{2}\right) = (-n_0 - 1) + \frac{1}{2}$$

$$\boxed{negatecf\,[n_0, 2] = [-n_0 - 1, 2]}$$



Cont. Fractions: Lorax



need a new kind of runtime value

can do that with the "expand" reduction

then implement each operation *as syntax*

Operations

$\texttt{expr} \leftarrow \texttt{ip}\ \{\mathit{expr}\colon \texttt{expr}\}$

$$\boxed{\left\lfloor \boxed{\mathit{expr}} \right\rfloor} \rightarrow \boxed{\begin{array}{l} \textbf{match}\ \boxed{\mathit{expr}} \\ \quad n + \frac{1}{d} \rightarrow n \\ \quad \textbf{else} \rightarrow \textbf{fail} \end{array}}$$

$\texttt{expr} \leftarrow \texttt{fp}\ \left\{\mathit{expr}'\colon \texttt{expr}\right\}$

$$\boxed{\left\{ \boxed{\mathit{expr}'} \right\}} \rightarrow \boxed{\begin{array}{l} \textbf{match}\ \boxed{\mathit{expr}'} \\ \quad n' + \frac{1}{d'} \rightarrow 0 + \frac{1}{d'} \\ \quad \textbf{else} \rightarrow \textbf{fail} \end{array}}$$

$\texttt{expr} \leftarrow \texttt{plus}\ \{\mathit{int}\colon \texttt{expr},\ \mathit{cf}\colon \texttt{expr}\}$

$$\boxed{\boxed{\boxed{\mathit{int}}\ \boxed{+}^{\text{sym}}\ \boxed{\mathit{cf}}}^{\text{rel}}} \rightarrow \boxed{\begin{array}{l} \textbf{match}\ \boxed{\mathit{cf}} \\ \quad n'' + \frac{1}{d'''} \rightarrow \left(\boxed{\mathit{int}} + n''\right) + \frac{1}{d'''} \\ \quad \textbf{else} \rightarrow \textbf{fail} \end{array}}$$

Operations

$\mathtt{expr} \leftarrow \mathtt{negate}\ \{expr\!:\mathtt{expr}\}$

$\boxed{\boxed{-}\,\text{sym}\ \boxed{expr}}\,\text{juxt} \;\rightarrow$

$$
\begin{array}{l}
\textbf{match}\ \ c \\
\quad n + \circ \ \rightarrow\ -n + \circ \\
\quad \textbf{else}\ \rightarrow\ \textbf{match}\ \ c \\
\qquad\qquad n' + \dfrac{1}{2+\circ} \ \rightarrow\ \left(-n' - 1\right) + \dfrac{1}{2+\circ} \\
\qquad\qquad \textbf{else}\ \rightarrow\ \textbf{match}\ \ c \\
\qquad\qquad\qquad n'' + \dfrac{1}{1 + \dfrac{1}{l + \frac{1}{d}}} \ \rightarrow\ \left(-n'' - 1\right) + \dfrac{1}{\left(l + 1\right) + \frac{1}{d}} \\
\qquad\qquad\qquad \textbf{else}\ \rightarrow\ \textbf{match}\ \ c \\
\qquad\qquad\qquad\qquad n''' + \dfrac{1}{m + \frac{1}{d'}} \ \rightarrow\ \left(-n''' - 1\right) + \dfrac{1}{1 + \dfrac{1}{\left(m - 1\right) + \frac{1}{d'}}} \\
\qquad\qquad\qquad\qquad \textbf{else}\ \rightarrow\ \textbf{fail} \\
\quad \textbf{where} \\
\qquad c = \boxed{expr}
\end{array}
$$

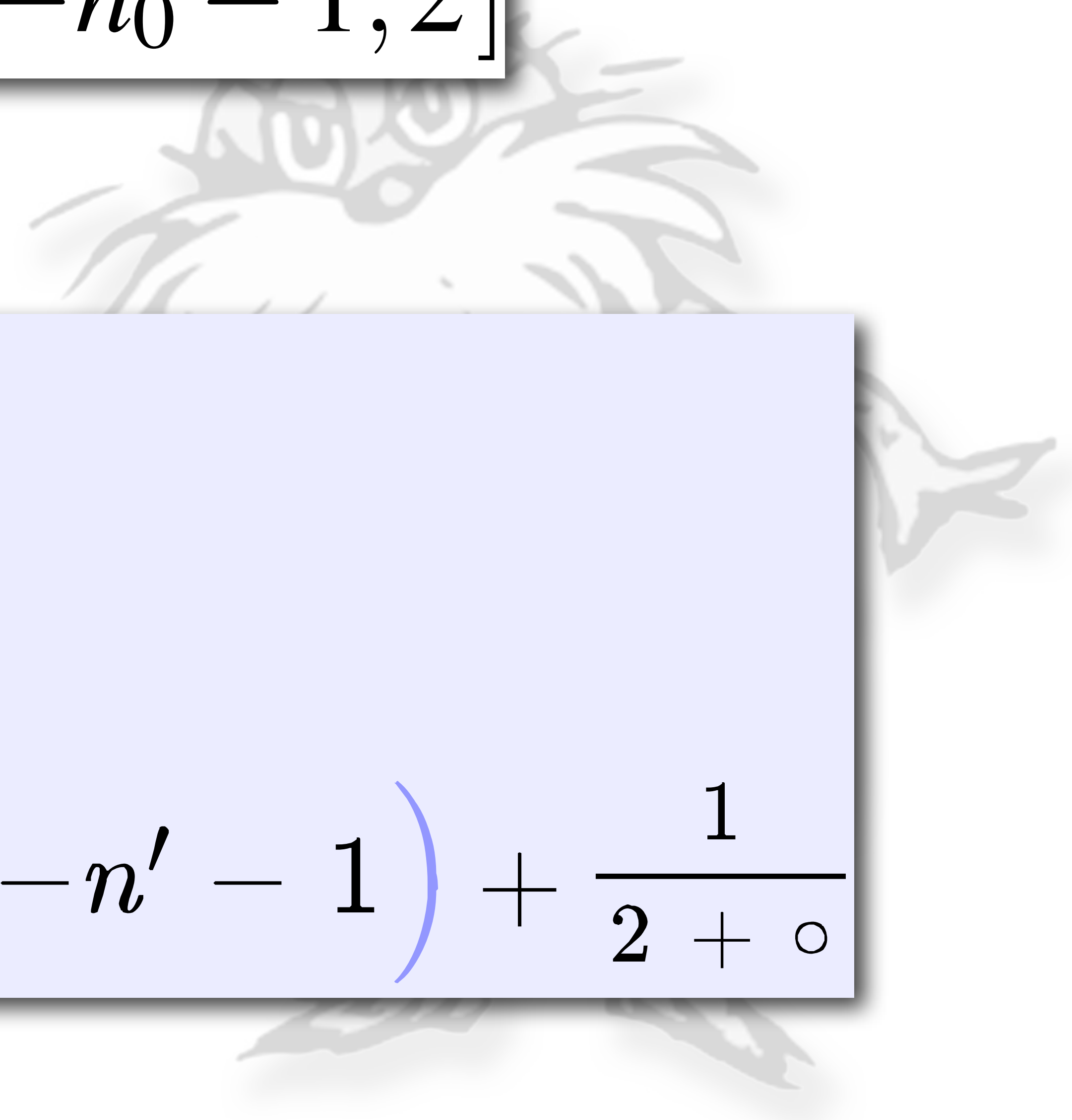
Haskell vs. Lorax

$$
\begin{aligned}
\textit{negatecf}\ [n_0] &= [-n_0] \\
\textit{negatecf}\ [n_0, 2] &= [-n_0 - 1, 2]
\end{aligned}
$$

Haskell vs. Lorax

$$
\begin{aligned}
&\mathit{negatecf}\ [n_0] && = [-n_0] \\
&\mathit{negatecf}\ [n_0, 2] && = [-n_0 - 1, 2]
\end{aligned}
$$

$$
\begin{array}{l}
\textbf{match}\ \ c \\
\quad n + \circ \ \ \rightarrow \ \ -n + \circ \\
\quad \textbf{else} \ \ \rightarrow \ \ \textbf{match}\ \ c \\
\qquad\qquad\qquad n' + \dfrac{1}{2 + \circ} \ \ \rightarrow \ \ \left(-n' - 1\right) + \dfrac{1}{2 + \circ}
\end{array}
$$



Conclusion

editor supports extended syntax equally

lower barrier for language creation

syntax freed from limitations of text

Questions?