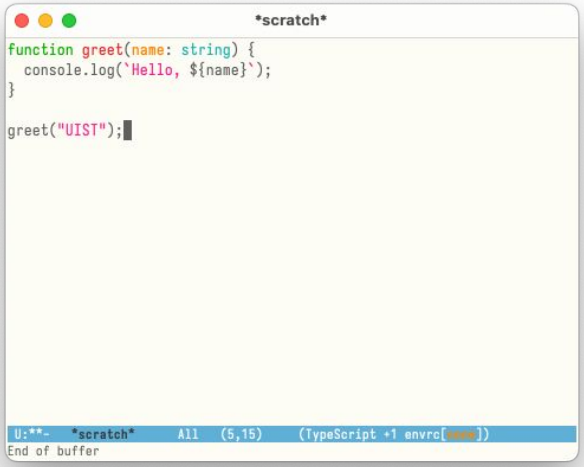
Ragged Blocks: Rendering Structured Text with Style

Sam Cohen and Ravi Chugh

UIST 25, Busan, Korea

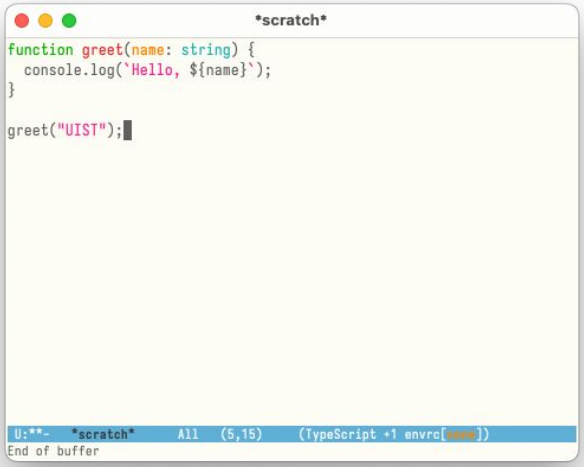
How Do We Display Code?



Text Editors (Flat)

Structure Editors

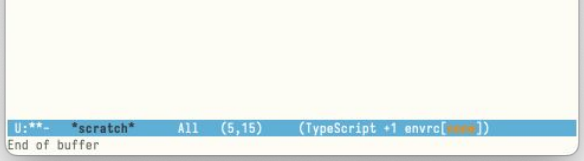
How Do We Display Code?



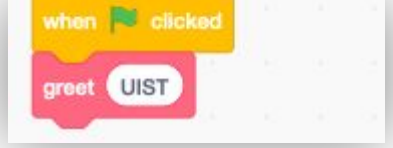
Text Editors (Flat)

Structure Editors

Goal: Blur this line



Text Editors (Flat)



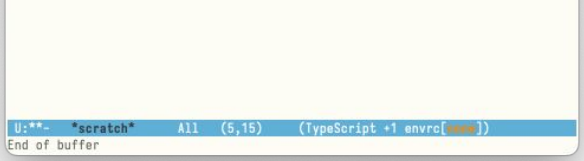
Structure Editors

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

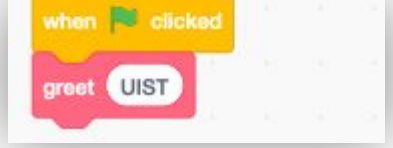
Flat Text

```
const abs = (x) =>
    x < 0 ? -x
          : x
```

Blocks



Text Editors (Flat)

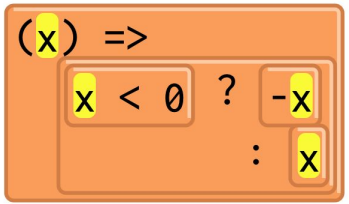


Structure Editors

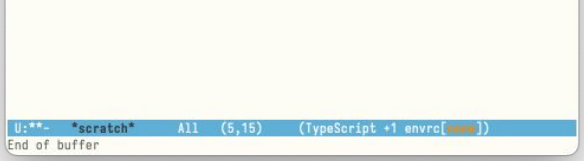
```
const abs = (x) =>
  x < 0 ? -x
        : x
```

Flat Text

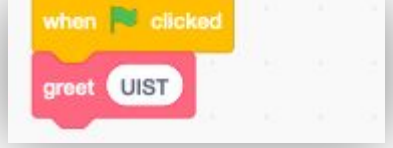
```
const abs =
```



Blocks



Text Editors (Flat)



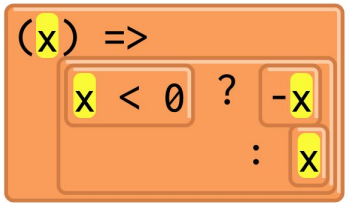
Structure Editors

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

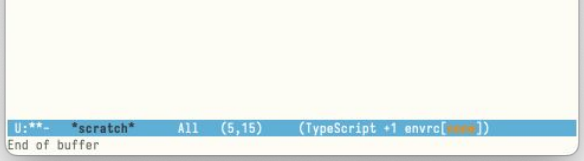
Flat Text

???

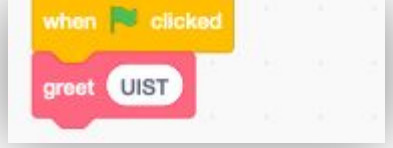
```
const abs =
```



Blocks



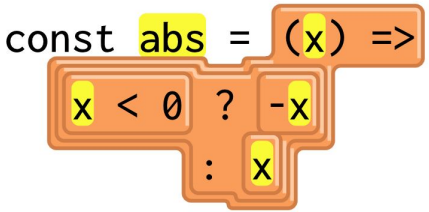
Text Editors (Flat)

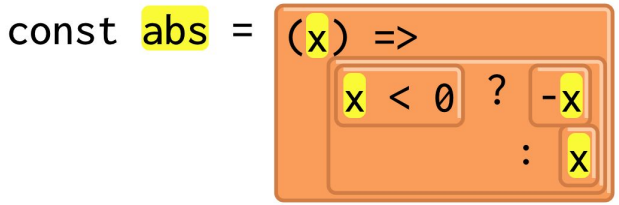


Structure Editors

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

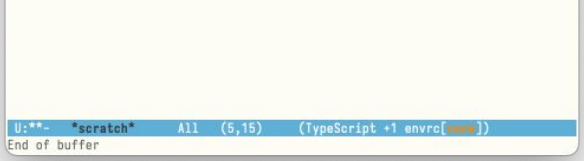
Flat Text



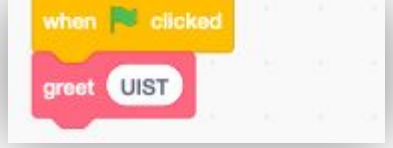


Blocks

Ragged Blocks



Text Editors (Flat)



Structure Editors

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

Flat Text

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

Ragged Blocks (Rocks)

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

Blocks

Ragged Blocks Rock!

```
pairs =
  [ (i, j)
    for i in range(0, 10)
    for j in range(0, 10)
      if i != j ]
```

```
fact 0 = 1
fact n = n * fact n - 1
```

```
function ErrMsgComponent(props) {
  return (
    <p style={{ color:  }}>
      {props.msg}
    </p>
  );
}
```

```
def fib(i):
    def fib▸go(i, n1, n2):
        if i <= 1:
            return 1▸n2
        else:
            return fib▸go(i - 1, n2, n1 + n2) + fib(i - 2)
    return go(i, 1, 1)
```

Ragged Blocks Rock!

```
pairs =
[ (i, j)
  for i in range(0, 10)
  for j in range(0, 10)
  if i != j ]
```

```
fact 0 = 1
fact n = n * fact n - 1
```

```
function ErrMsgComponent(props) {
  return (
    <p style={{ color:  }}>
    {props.msg}
    </p>
  );
}
```

```
def fib(i):
    def fib▸go(i, n1, n2):
        if i <= 1:
            return 1▸n2
        else:
            return fib▸go(i - 1, n2, n1 + n2) + fib(i - 2)
    return go(i, 1, 1)
```

Natural Language Visualizations!

... V-gadgets , which are inserted ... h-gadgets have been inserted ... line can be found by summing the ... of each fragment.

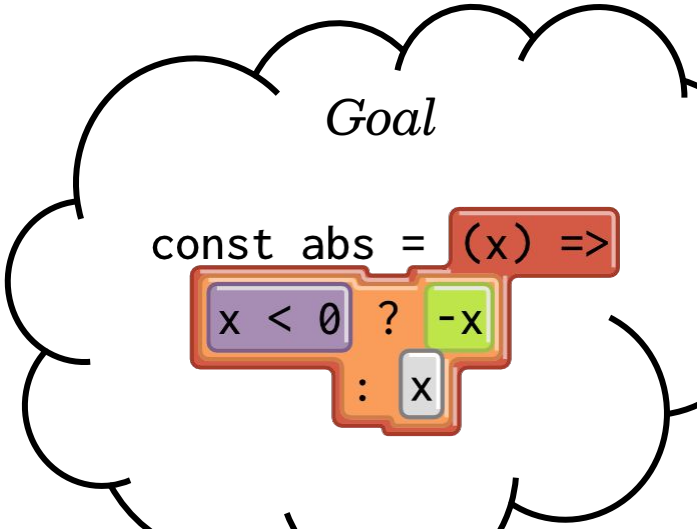
After width resolution, it's time to resolve the height of each line.

So called height resolution is accomplished using V-gadgets , which are analagous to H-gadgets, but exist between lines as opposed to between fragments.

These V-gadgets are inserted based on the horizontal positions of the

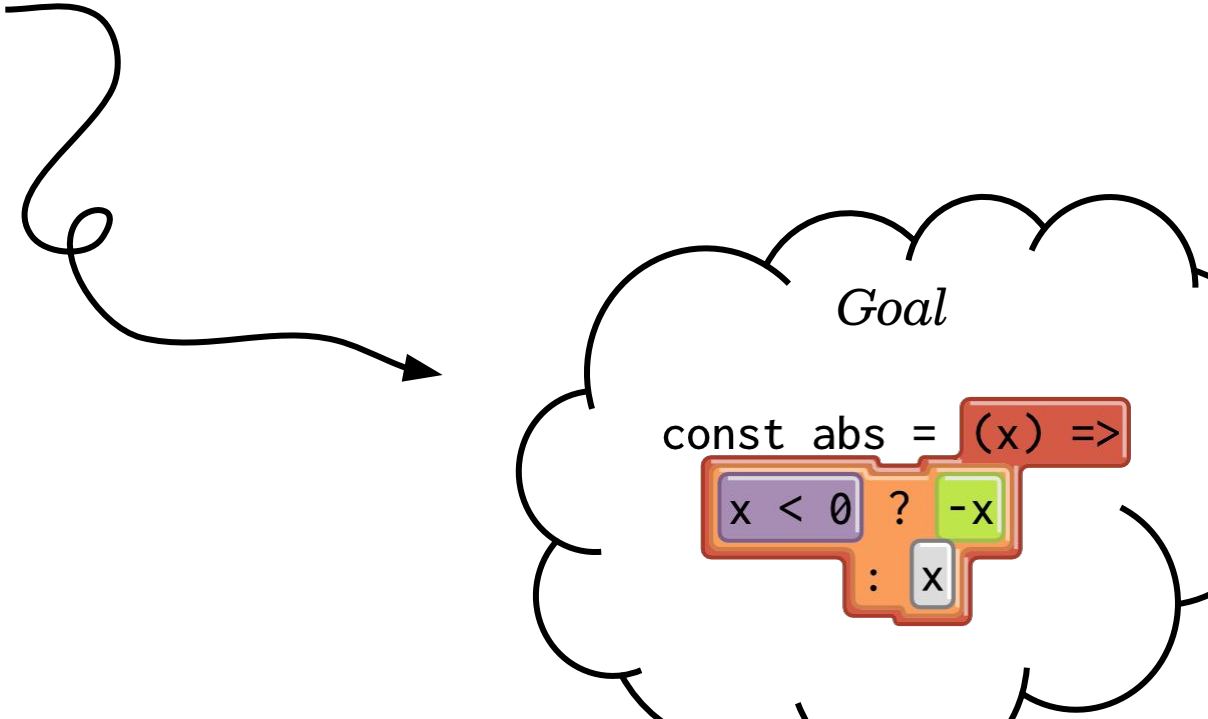
Rocks Layout: Key Operations

```
const abs = (x) =>
  x < 0 ? -x
        : x
```



Rocks Layout: Key Operations

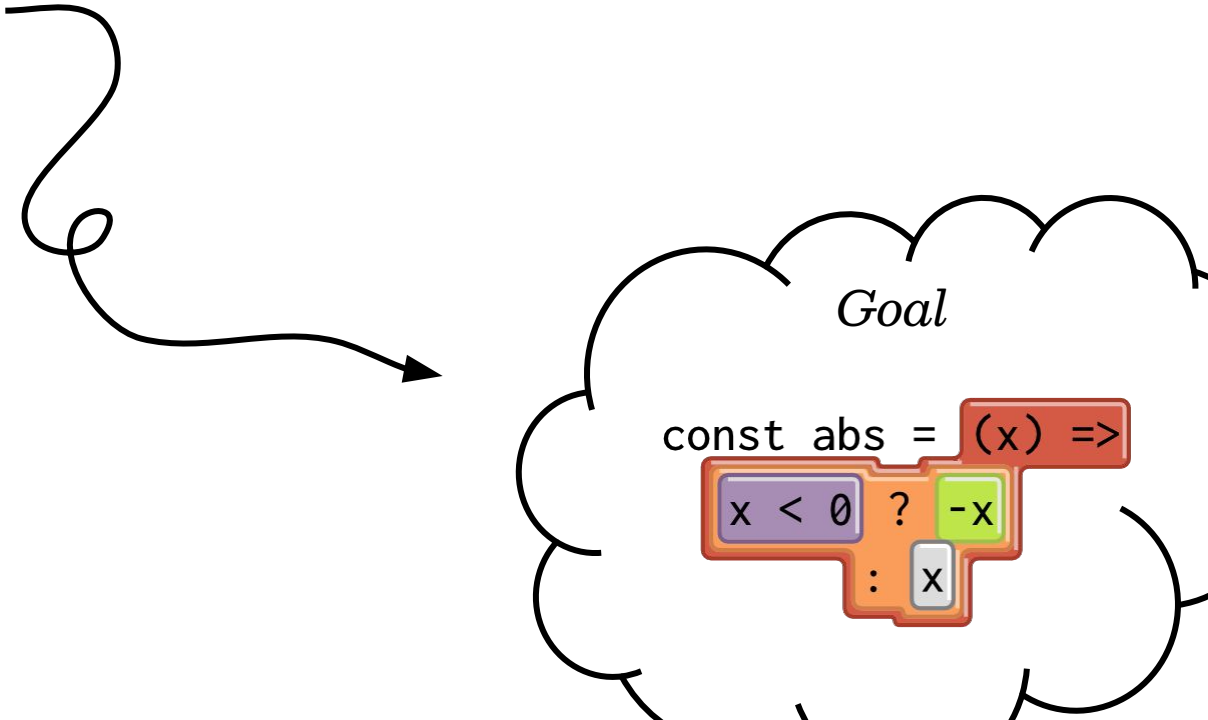
```
const abs = (x) =>
  x < 0 ? -x
        : x;
```



Rocks Layout: Key Operations

```
const abs = (x) =>
  x < 0 ? -x
        : x;
```

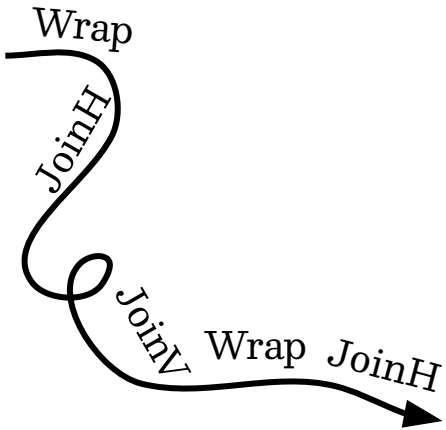
Fragments

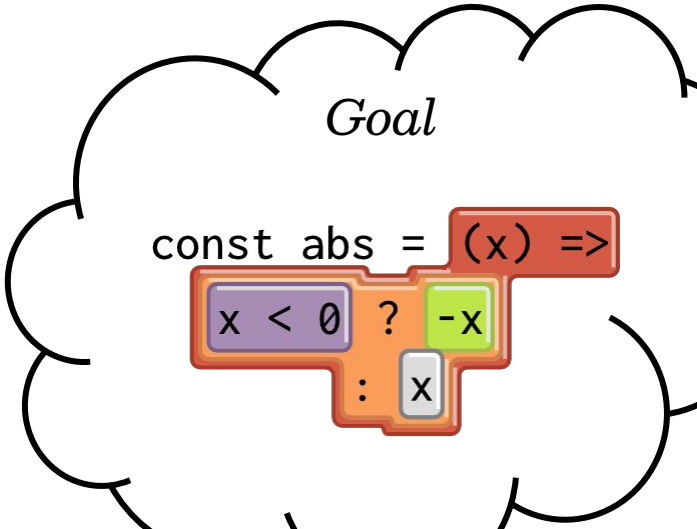


Rocks Layout: Key Operations

```
const abs = (x) =>
  x < 0 ? -x
        : x;
```

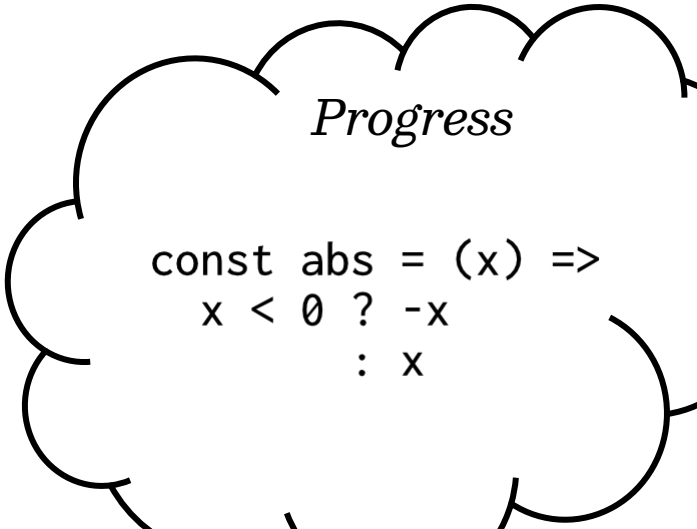
Fragments





Operation 1: Wrap

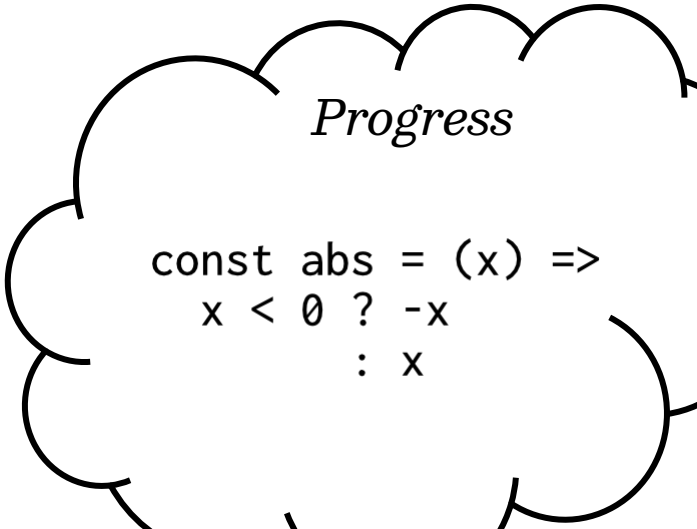
wrap(🔴, 2, x) = 🧱x



Operation 1: Wrap

wrap(🔴, 2, x) = 🟥x

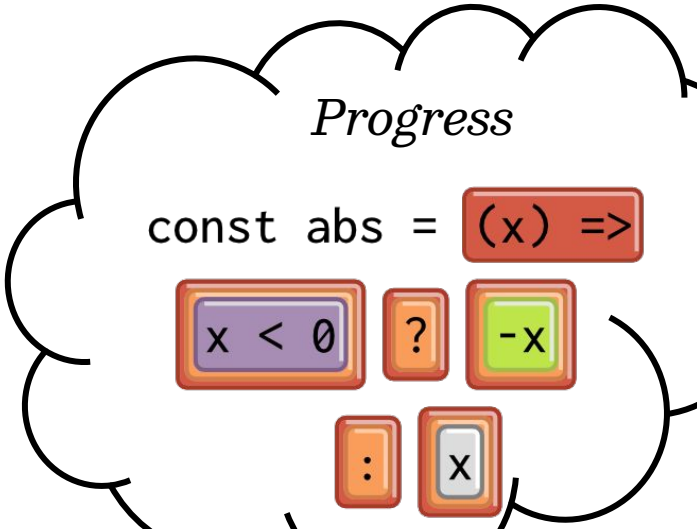
wrap(🟠, 2, 🟥x) = 🟥x



Operation 1: Wrap

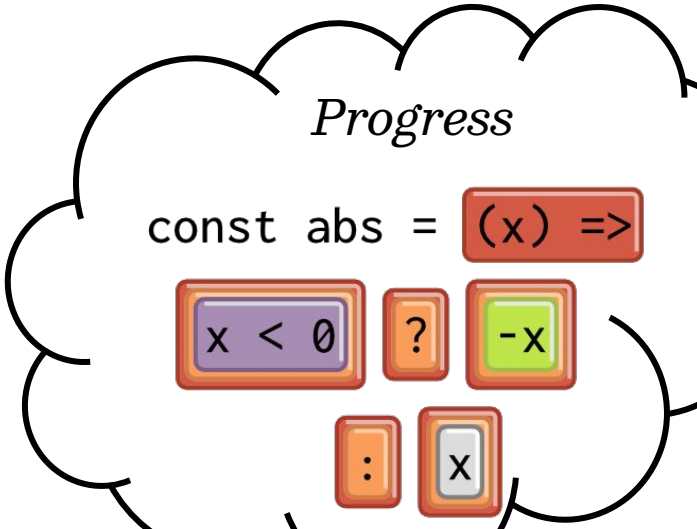
wrap(●, 2, x) = x

wrap(●, 2, x) = x



Operation 2: JoinH

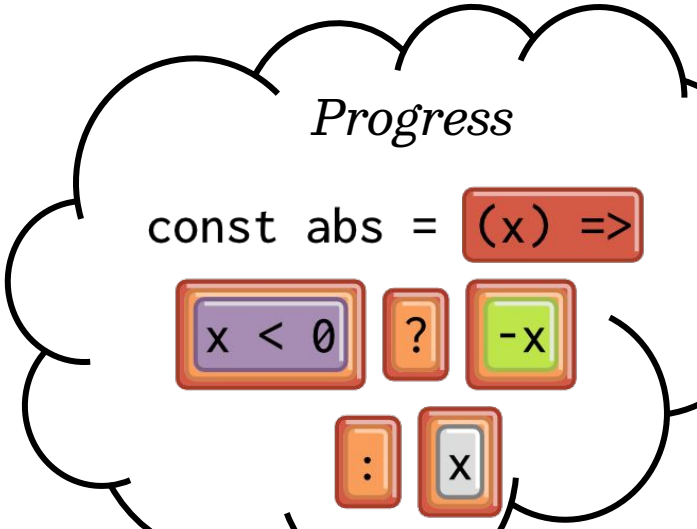
joinH(x, y) = xy



Operation 2: JoinH

joinH(x, y) = xy

joinH(x , y) = xy

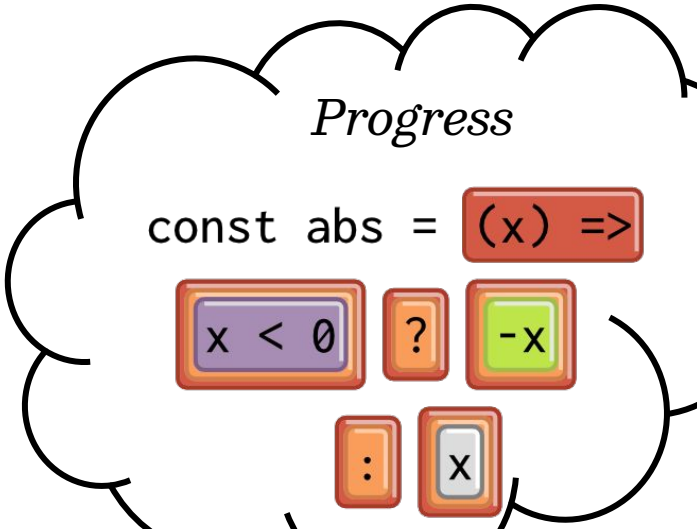


Operation 2: JoinH

joinH(x, y) = xy

joinH(x , y) = xy

joinH(x , y) = x y



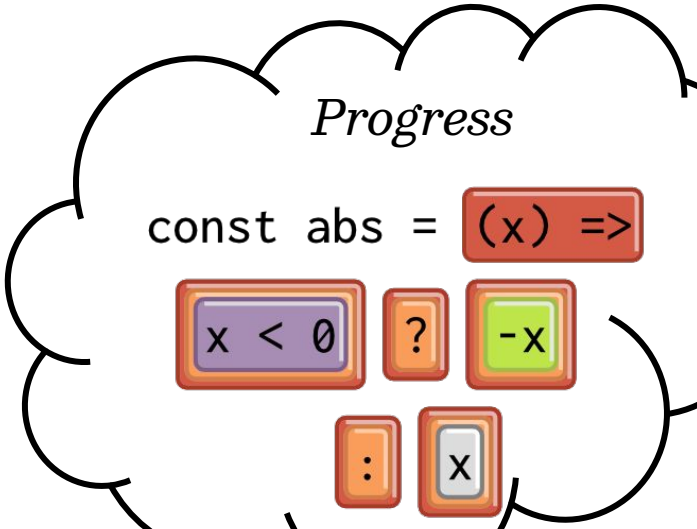
Operation 2: JoinH

joinH(x, y) = xy

joinH(x , y) = xy

joinH(x , y) = x y

joinH(x , y) = x y



Operation 2: JoinH

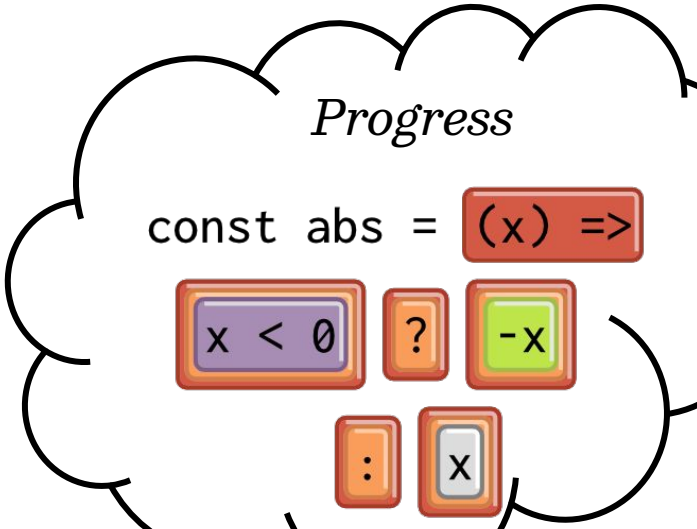
joinH(x, y) = xy

joinH(x , y) = xy

joinH(x , y) = x y

joinH(x , y) = x y

joinH(x , y z) = x y z



Operation 2: JoinH

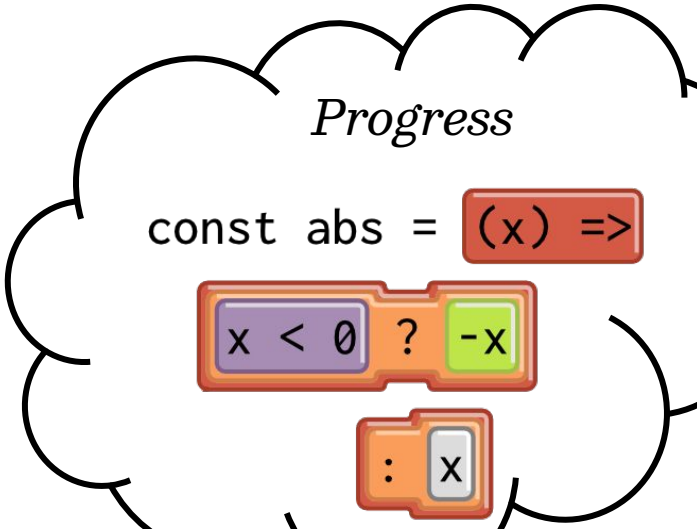
joinH(x, y) = xy

joinH(x , y) = xy

joinH(x , y) = x y

joinH(x , y) = x y

joinH(x , y z) = x y z



Operation 3: JoinV

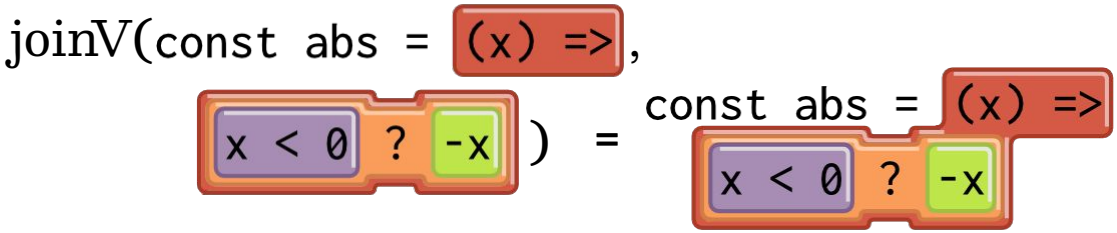
joinV(const abs = (x) => x < 0 ? -x) =

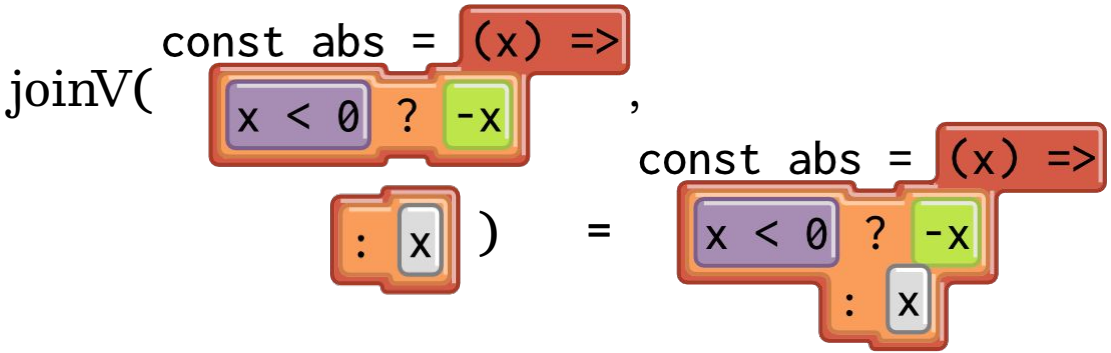
const abs = (x) => x < 0 ? -x

Progress

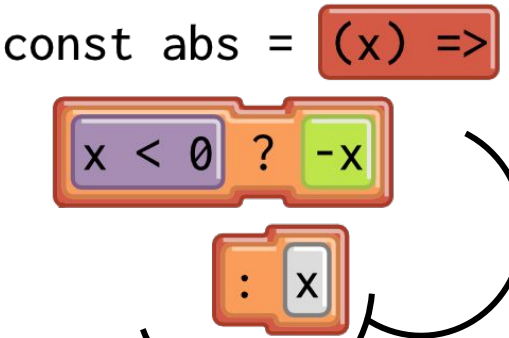
const abs = (x) => x < 0 ? -x : x

Operation 3: JoinV

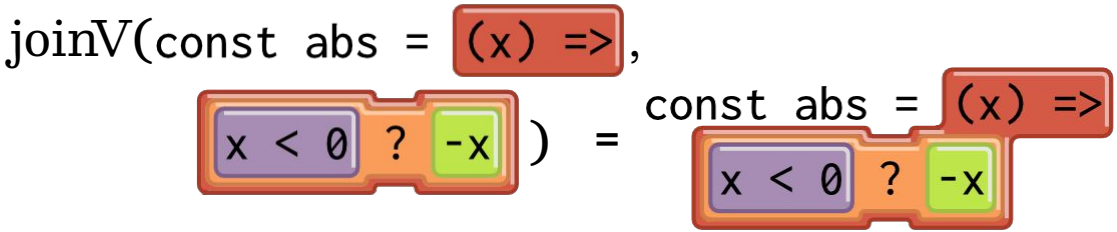


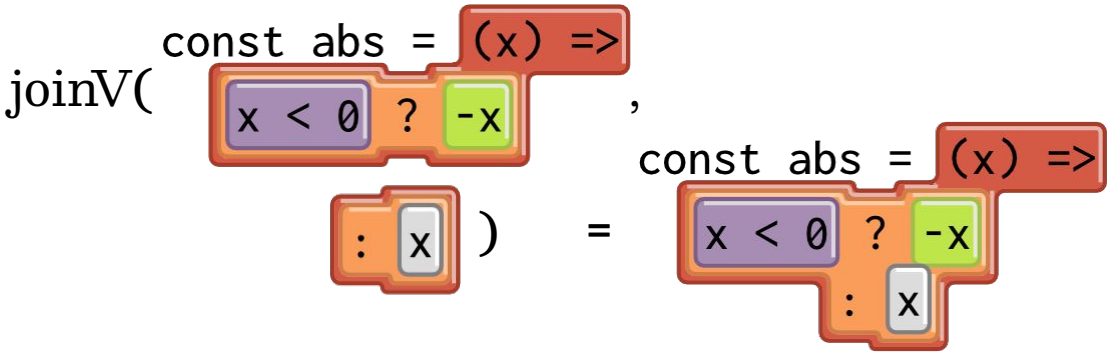


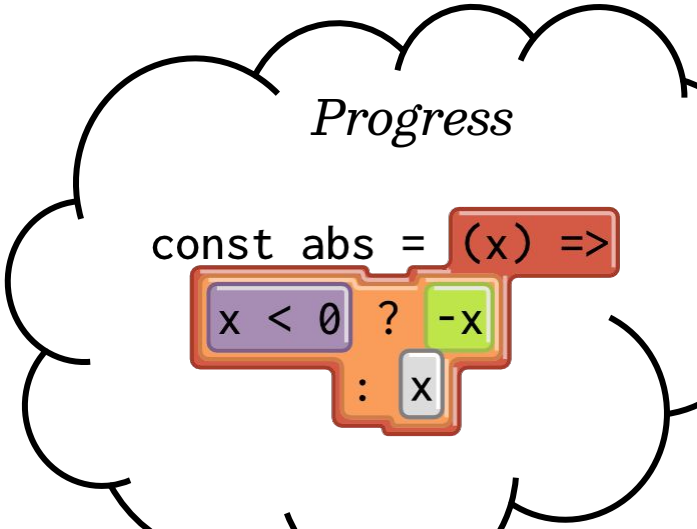
Progress



Operation 3: JoinV







Okay, but how are these operations *implemented*?

joinH(

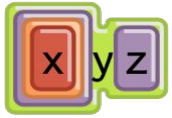
, 

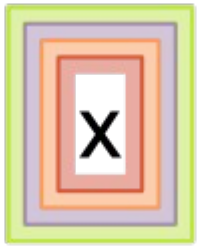
) = 

Okay, but how are these operations *implemented*?

joinH(

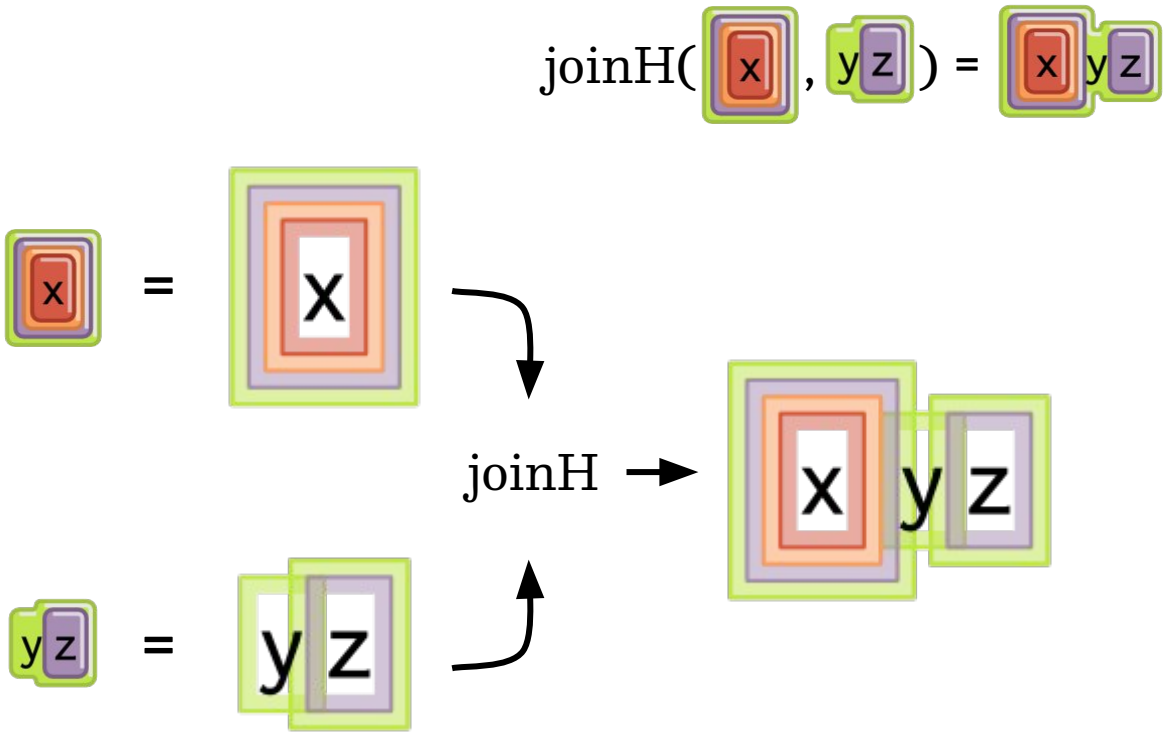
, 

) = 

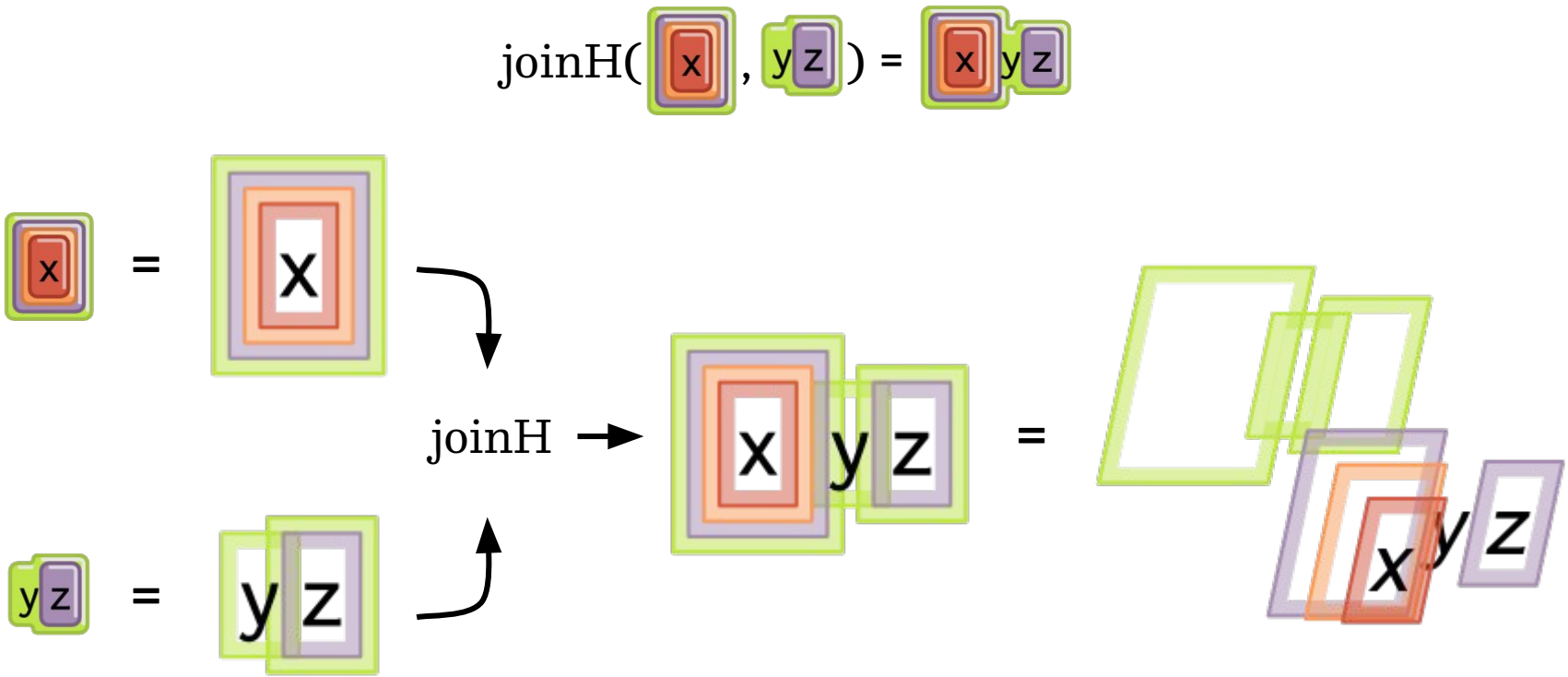
=


=

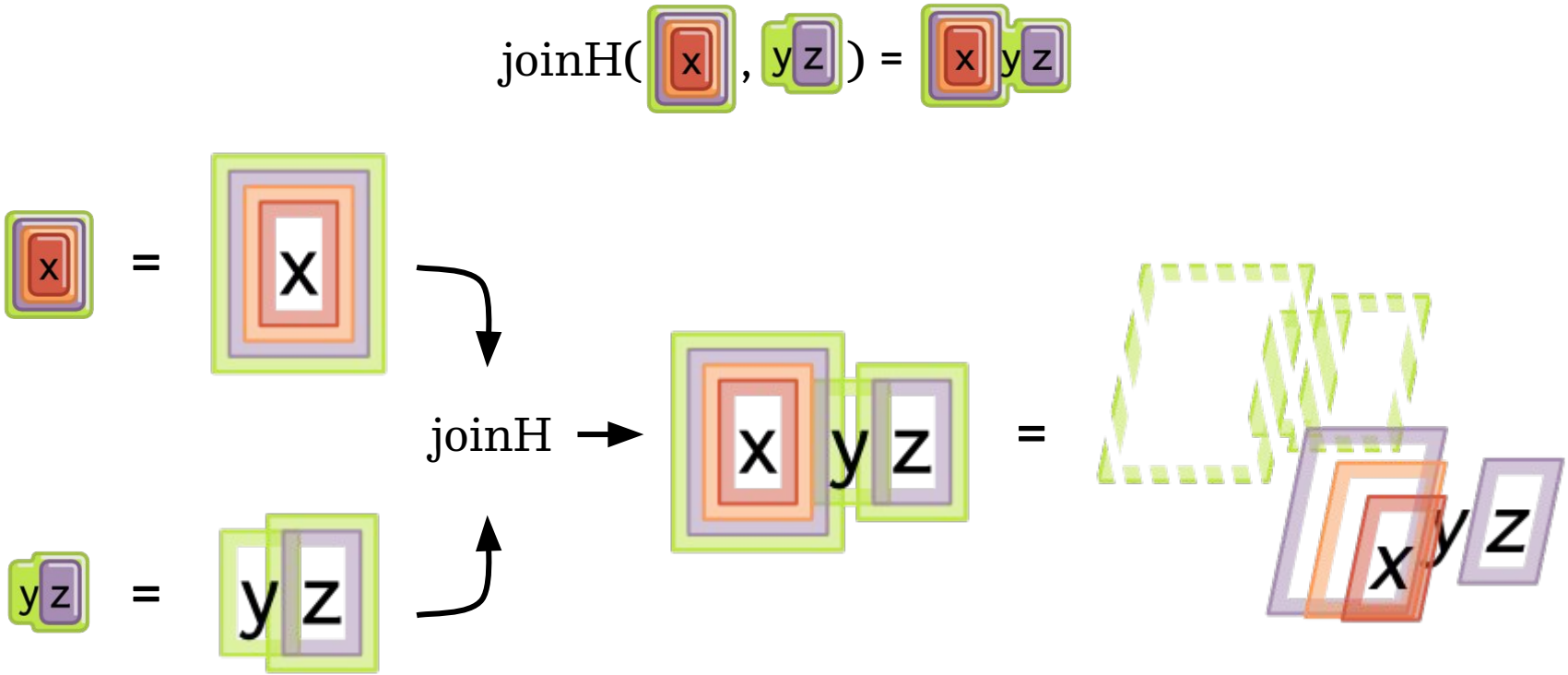

Okay, but how are these operations *implemented*?



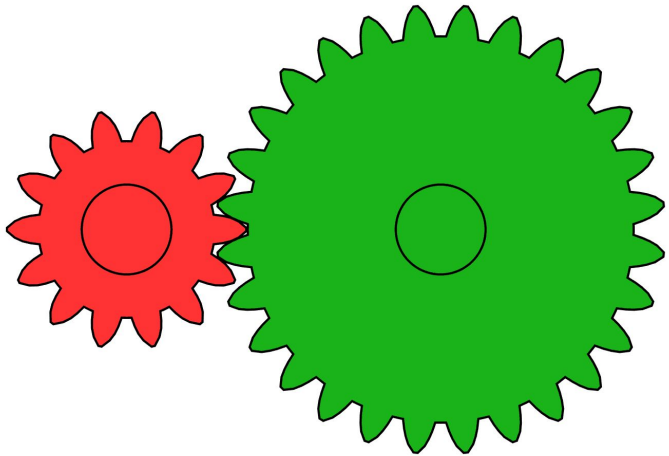
Okay, but how are these operations *implemented*?



Okay, but how are these operations *implemented*?

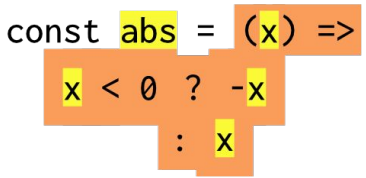


Changing gears... Evaluation



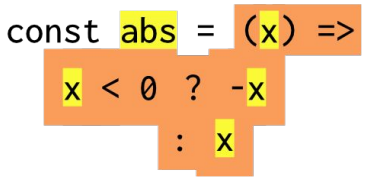
Implementation (5k LOC of TypeScript)

Base Algorithm

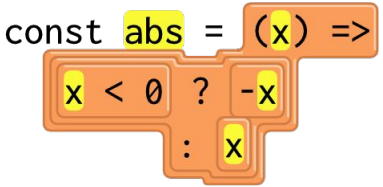


Implementation (5k LOC of TypeScript)

Base Algorithm

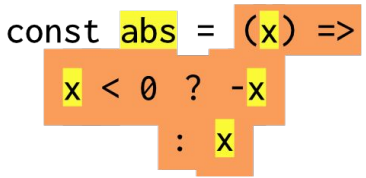


Optimized Algorithm

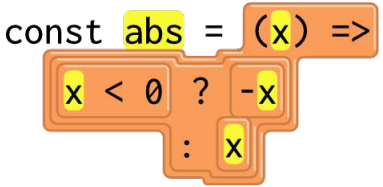


Implementation (5k LOC of TypeScript)

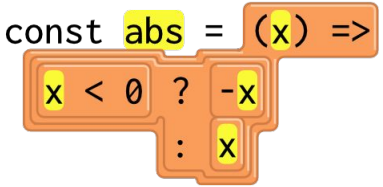
Base Algorithm



Optimized Algorithm



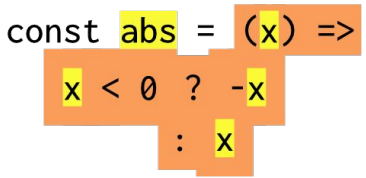
Extension: Tab Stops



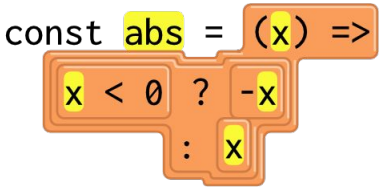
Align ? and : characters

Implementation (5k LOC of TypeScript)

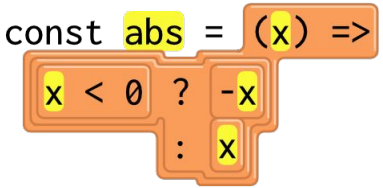
Base Algorithm



Optimized Algorithm



Extension: Tab Stops



Align ? and : characters

Extension: Automatically Insert Newlines

These H-gadgets are inserted between each fragment on a line (as opposed to V-gadgets, which are inserted between lines). Once the h-gadgets have been inserted the length of each line can be found by summing the width of each fragment.

After width resolution, it's time to resolve the height of each line.

So called height resolution is accomplished using V-gadgets, which are analagous to H-gadgets, but exist between lines as opposed to between fragments.

These V-gadgets are inserted based on the horizontal positions of the

Evaluation – What?

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

```
const abs = (x) =>
  x < 0 ? -x
        : x
```

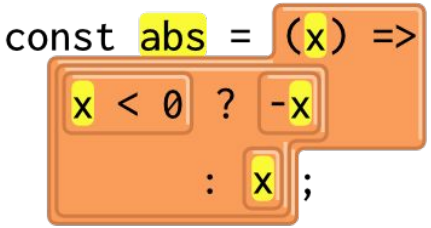
```
const abs = (x) =>
  x < 0 ? -x
        : x
```

Flat Text

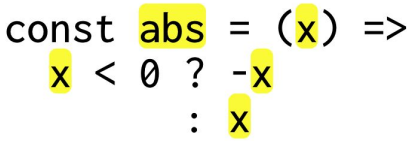
Rocks

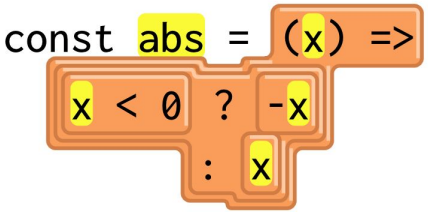
Blocks

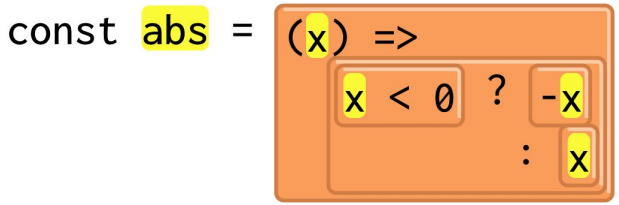
Evaluation – What?



*Prior work, presented in the future (OOPSLA 25)







Flat Text

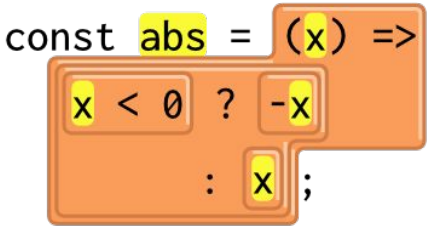
Rocks

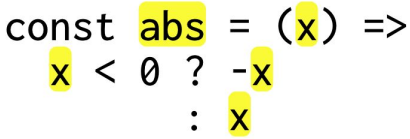
S-Blocks*

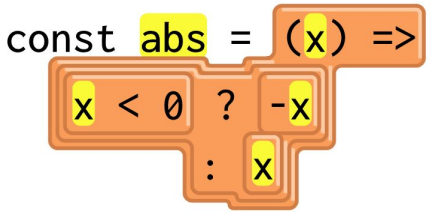
Blocks

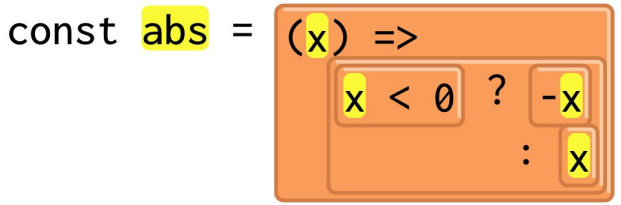
Evaluation – How?

1. Measure layout size









Flat Text

Rocks

S-Blocks

Blocks

Evaluation – How?

1. Measure layout size (Line Width)

TypeScript, Python, and Haskell examples	**Mean Line Width (*px* (%))**			
	Unstyled	L1_{S}	S-BLOCKS	BOXES_{NS}
core.ts				
diff-objs.ts				
functional.py				
simplex.py				
solve.hs				
layout.hs				

Evaluation – How?

1. Measure layout size (Line Width)

TypeScript, Python, and Haskell examples	Mean Line Width (*px* (%))			
	Unstyled	$L1_S$	S-Blocks	$Boxes_{NS}$
core.ts				
λ diff-objs.ts				
functional.py				
simplex.py				
λ solve.hs				
λ layout.hs				

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)

TypeScript, Python, and Haskell examples	Mean Line Width (*px* (%))			
	Unstyled	$L1_S$	S-Blocks	$Boxes_{NS}$
core.ts	260 (0.78)			
λ diff-objs.ts	280 (0.38)			
functional.py	180 (0.97)			
simplex.py	180 (1.3)			
λ solve.hs	330 (0.78)			
λ layout.hs	270 (0.73)			

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)

TypeScript, Python, and Haskell examples	Mean Line Width (*px* (%))			
	Unstyled	$L1_S$	S-Blocks	$Boxes_{NS}$
core.ts	260 (0.78)			340 (1.0)
λ diff-objs.ts	280 (0.38)			740 (1.0)
functional.py	180 (0.97)			180 (1.0)
simplex.py	180 (1.3)			140 (1.0)
λ solve.hs	330 (0.78)			430 (1.0)
λ layout.hs	270 (0.73)			360 (1.0)

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)

TypeScript, Python, and Haskell examples	Mean Line Width (*px* (%))			
	Unstyled	L1$_S$	S-Blocks	Boxes$_{NS}$
core.ts	260 (0.78)	310 (0.91)	310 (0.91)	340 (1.0)
λ diff-objs.ts	280 (0.38)	340 (0.46)	340 (0.46)	740 (1.0)
functional.py	180 (0.97)	210 (1.1)	210 (1.1)	180 (1.0)
simplex.py	180 (1.3)	210 (1.5)	210 (1.5)	140 (1.0)
λ solve.hs	330 (0.78)	370 (0.88)	370 (0.88)	430 (1.0)
λ layout.hs	270 (0.73)	300 (0.84)	300 (0.84)	360 (1.0)

λ = Functional Style

Reduced Line Width

Evaluation – How?

1. Measure layout size (Line Width)

“Optimal” Algorithm:

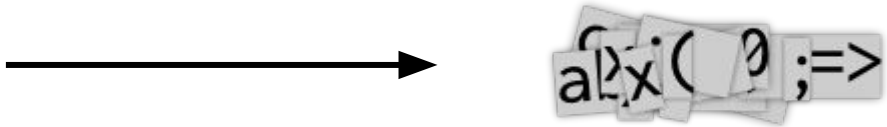
```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```

Evaluation – How?

1. Measure layout size (Line Width)

“Optimal” Algorithm:

```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```



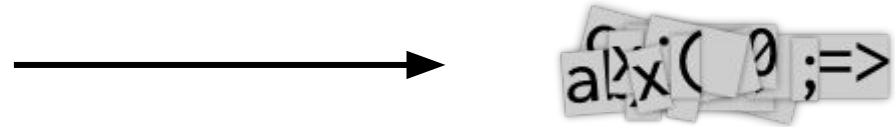
Evaluation – How?

1. Measure layout size (Line Width)

“Optimal” Algorithm:

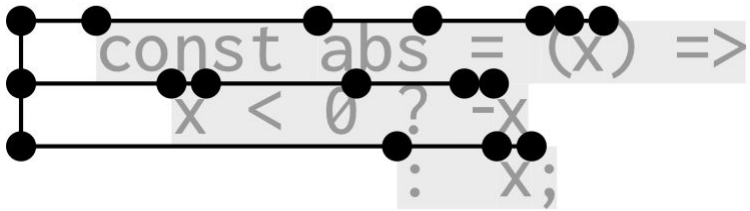
Rule: Fragments should be arranged left-to-right, top-to-bottom

```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```



Evaluation – How?

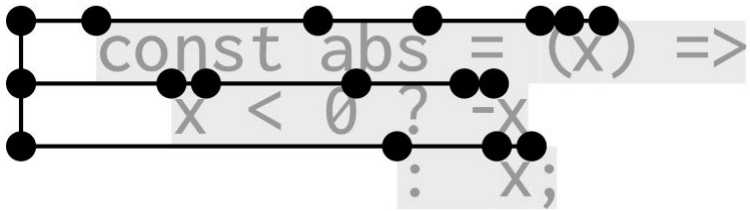
1. Measure layout size (Line Width)
2. Measure layout error (Mesh Distance)



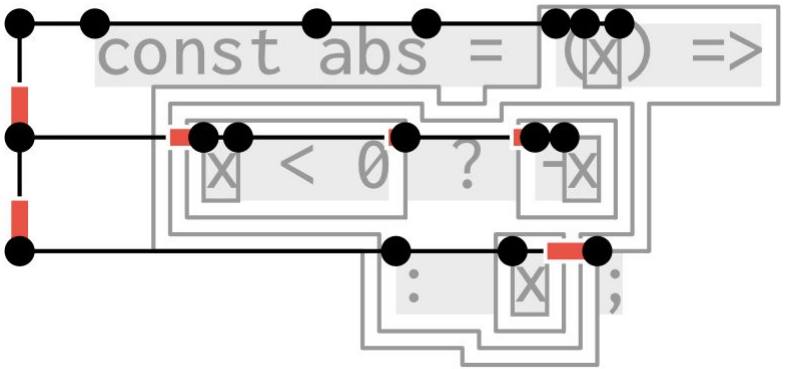
Reference Layout
(Plain Text)

Evaluation – How?

1. Measure layout size (Line Width)
2. Measure layout error (Mesh Distance)



Reference Layout
(Plain Text)



Test Layout

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)
2. Measure layout error (Mesh Distance)

TypeScript, Python, and Haskell examples	Mean Horizontal Mesh Distance w.r.t. Unstyled (*px* (%)) BOXES$_{NS}$	Mean Vertical Mesh Distance w.r.t. Unstyled (*px* (%)) BOXES$_{NS}$
core.ts	13 (1.0)	18 (1.0)
λ diff-objs.ts	41 (1.0)	26 (1.0)
functional.py	22 (1.0)	9.7 (1.0)
simplex.py	8.0 (1.0)	9.7 (1.0)
λ solve.hs	20 (1.0)	20 (1.0)
λ layout.hs	12 (1.0)	20 (1.0)

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)
2. Measure layout error (Mesh Distance)

TypeScript, Python, and Haskell examples	Mean Horizontal Mesh Distance w.r.t. Unstyled (*px* (%))		Mean Vertical Mesh Distance w.r.t. Unstyled (*px* (%))	
	S-Blocks	Boxes$_{NS}$	S-Blocks	Boxes$_{NS}$
core.ts	4.8 (0.36)	13 (1.0)	21 (1.2)	18 (1.0)
λ diff-objs.ts	5.4 (0.13)	41 (1.0)	32 (1.2)	26 (1.0)
functional.py	7.4 (0.33)	22 (1.0)	11 (1.1)	9.7 (1.0)
simplex.py	7.1 (0.88)	8.0 (1.0)	11 (1.1)	9.7 (1.0)
λ solve.hs	4.9 (0.24)	20 (1.0)	22 (1.1)	20 (1.0)
λ layout.hs	3.8 (0.33)	12 (1.0)	23 (1.1)	20 (1.0)

Reduced horizontal error

Exacerbated vertical error

λ = Functional Style

Evaluation – How?

1. Measure layout size (Line Width)
2. Measure layout error (Mesh Distance)

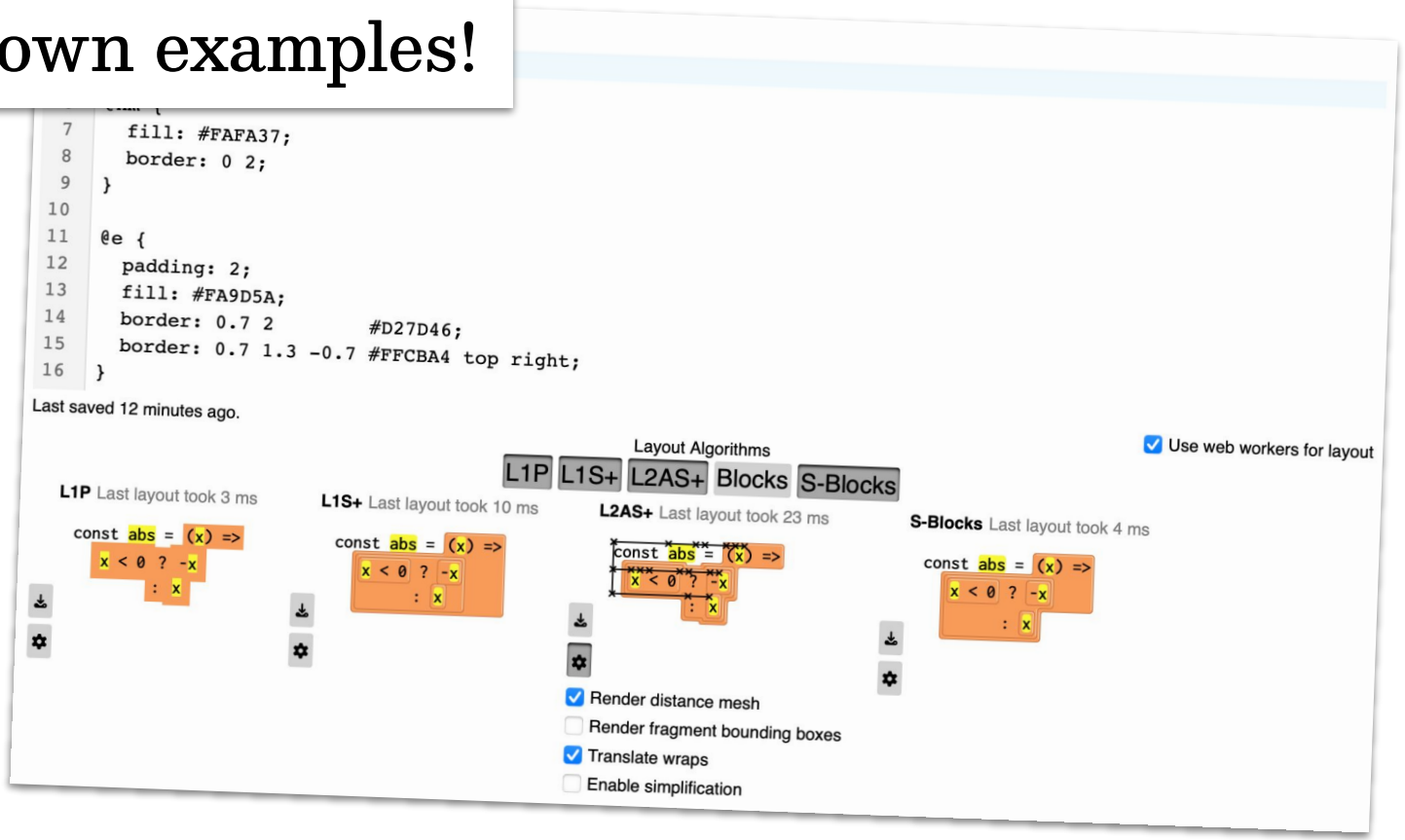
TypeScript, Python, and Haskell examples	Mean Horizontal Mesh Distance w.r.t. Unstyled (*px* (%))			Mean Vertical Mesh Distance w.r.t. Unstyled (*px* (%))		
	L1$_S$	S-Blocks	Boxes$_{NS}$	L1$_S$	S-Blocks	Boxes$_{NS}$
core.ts	4.8 (0.36)	4.8 (0.36)	13 (1.0)	9.2 (0.52)	21 (1.2)	18 (1.0)
λ diff-objs.ts	5.4 (0.13)	5.4 (0.13)	41 (1.0)	22 (0.85)	32 (1.2)	26 (1.0)
functional.py	7.4 (0.33)	7.4 (0.33)	22 (1.0)	6.2 (0.64)	11 (1.1)	9.7 (1.0)
simplex.py	7.1 (0.88)	7.1 (0.88)	8.0 (1.0)	7.7 (0.79)	11 (1.1)	9.7 (1.0)
λ solve.hs	4.9 (0.24)	4.9 (0.24)	20 (1.0)	18 (0.87)	22 (1.1)	20 (1.0)
λ layout.hs	3.8 (0.33)	3.8 (0.33)	12 (1.0)	19 (0.94)	23 (1.1)	20 (1.0)

Identical horizontal error c.f. S-Blocks

Reduced vertical error

https://sbcohen2000.github.io/ragged-blocks/

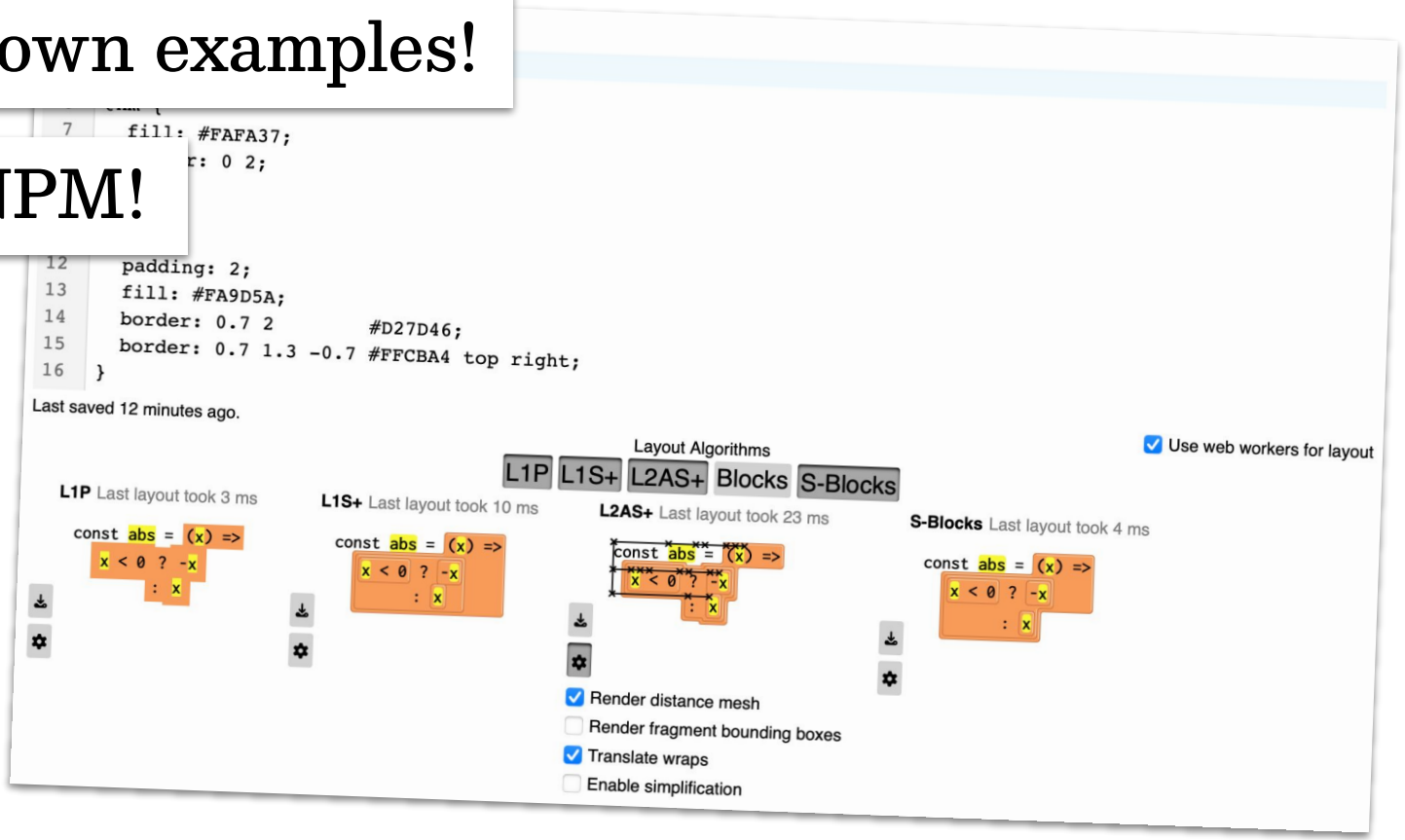
Try your own examples!



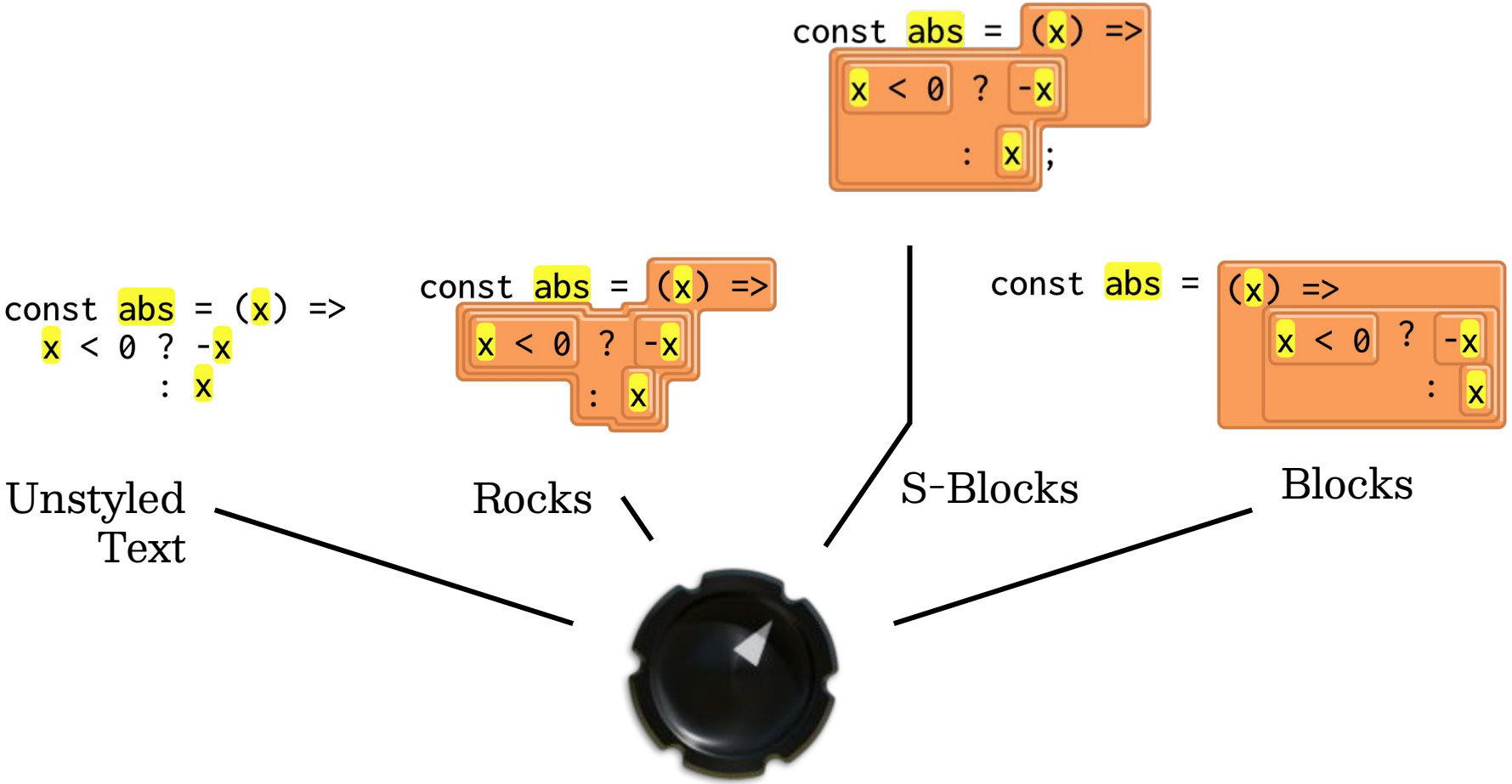
https://sbcohen2000.github.io/ragged-blocks/

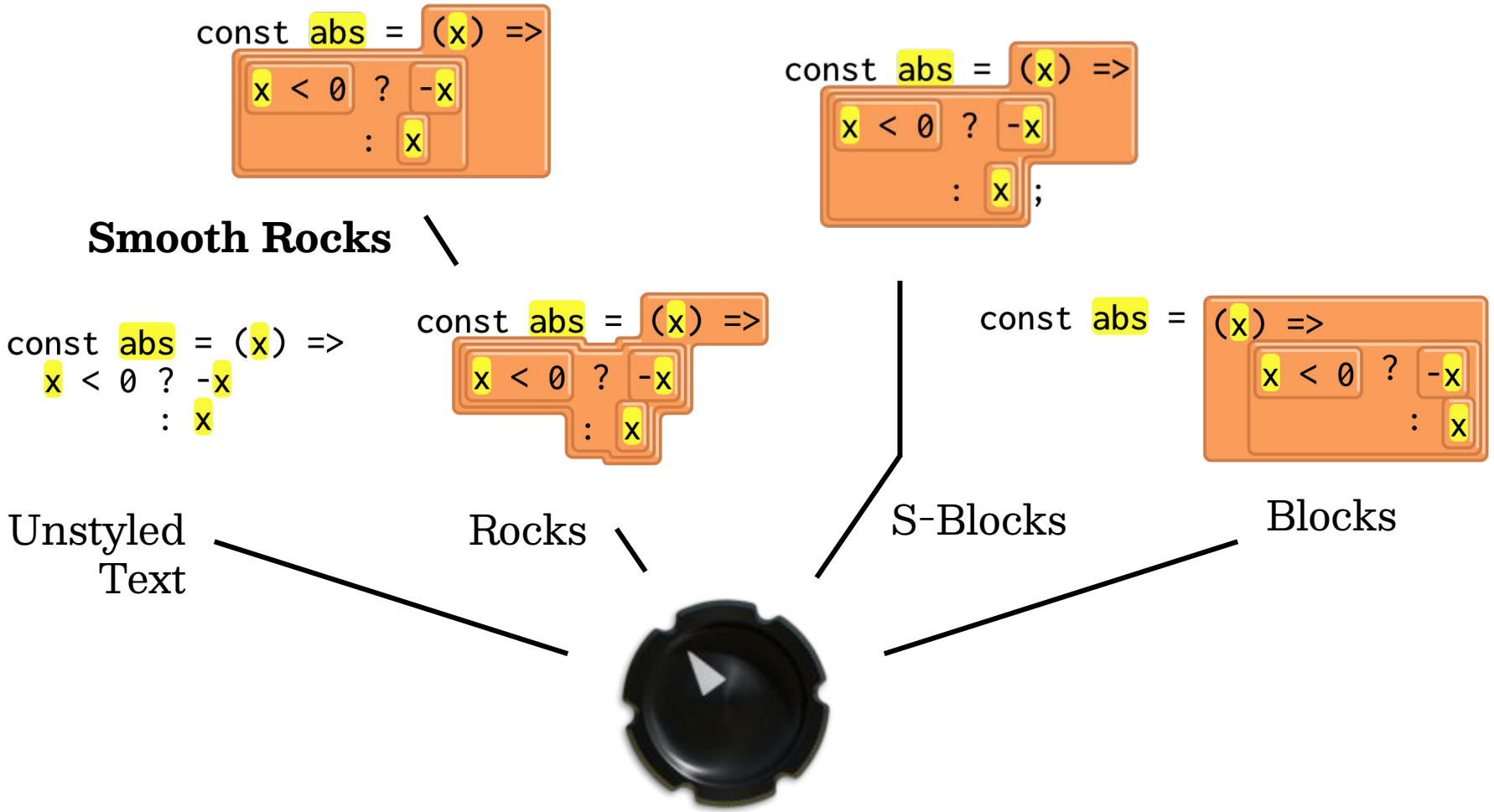
Try your own examples!

Also on NPM!



Extra Slides


const abs = (x) =>
x < 0 ? -x
: x
const abs = (x) =>
x < 0 ? -x
: x
const abs = (x) =>
x < 0 ? -x
: x ;
const abs = (x) =>
x < 0 ? -x
: x
Unstyled Text
Rocks
S-Blocks
Blocks


const abs = (x) =>
x < 0 ? -x
: x
Smooth Rocks
const abs = (x) =>
x < 0 ? -x
: x;
const abs = (x) =>
x < 0 ? -x
: x
const abs = (x) =>
x < 0 ? -x
: x
const abs = (x) =>
x < 0 ? -x
: x
Unstyled Text
Rocks
S-Blocks
Blocks

Have

```
const abs = (x) =>
              x < 0 ? -x
                    : x;
```

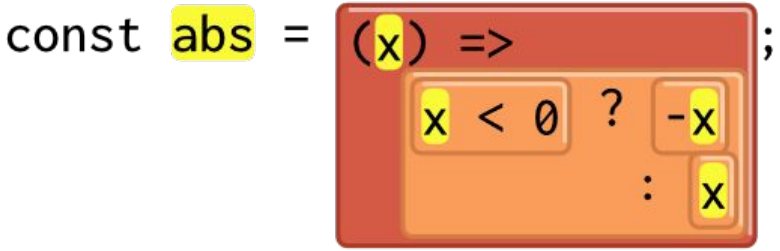
Plain Text

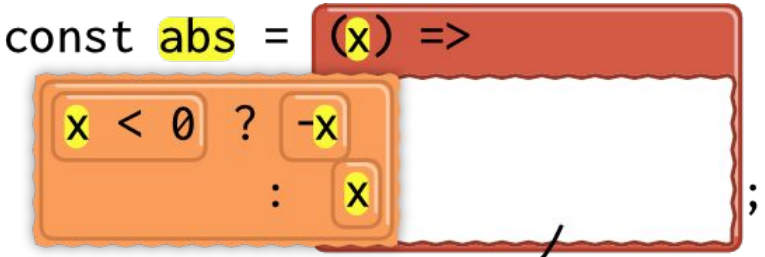
```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```

Can we just use boxes?

Proof by Intimidation

Have





Child expression not contained in parent

Plain Text

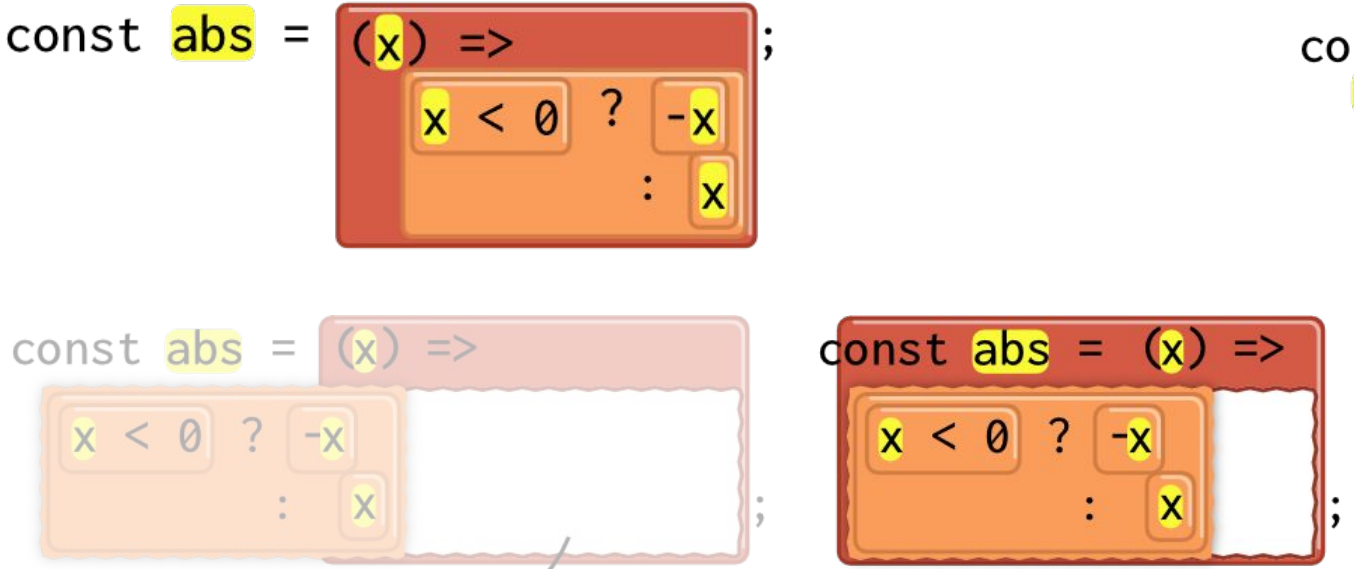
```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```

Rearrange to match *Plain Text*

Have

Plain Text

```
const abs = (x) =>
  x < 0 ? -x
        :  x;
```



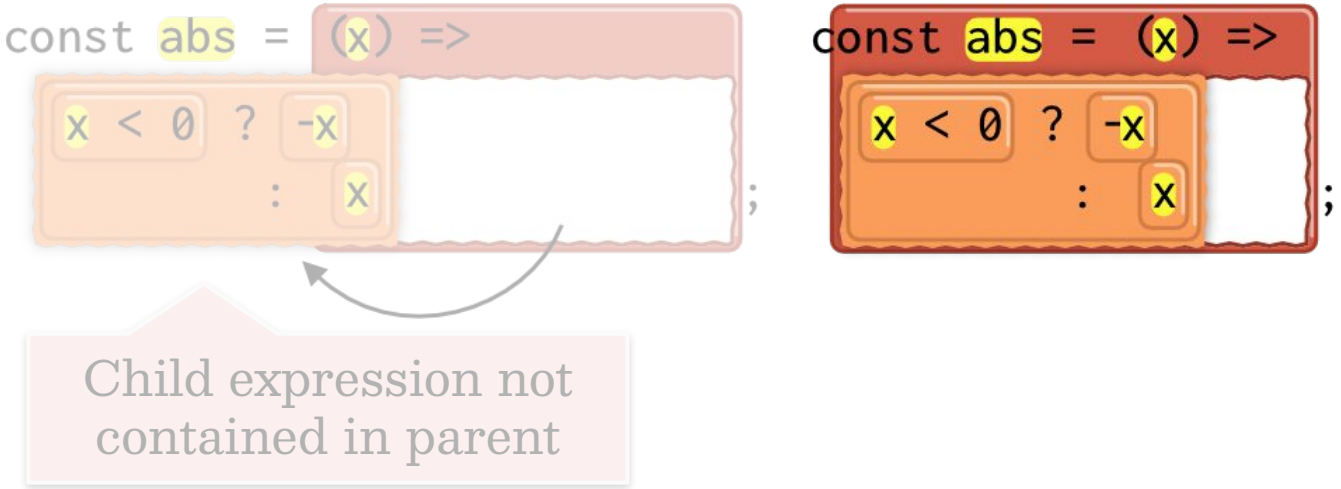
Stretch parent boundary to contain child

Have

Plain Text

```
const abs = (x) =>
  x < 0 ? -x
        : x;
```

Child box overlaps parent's text



Stretch parent boundary to contain child

Have

Plain Text

Make order clearer, maybe arc from plain text on left and "have" on right (same as knob)

```
const abs = (x) =>
  x < 0 ? -x
        : x;
```

Child box overlaps parent's text

Child expression not contained in parent

Cut child boundary to avoid parent text

Have

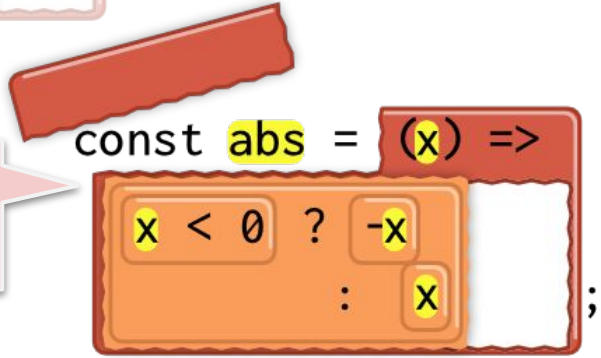
Plain Text

```
const abs = (x) =>
  x < 0 ? -x
        : x;
```

Child box overlaps parent's text

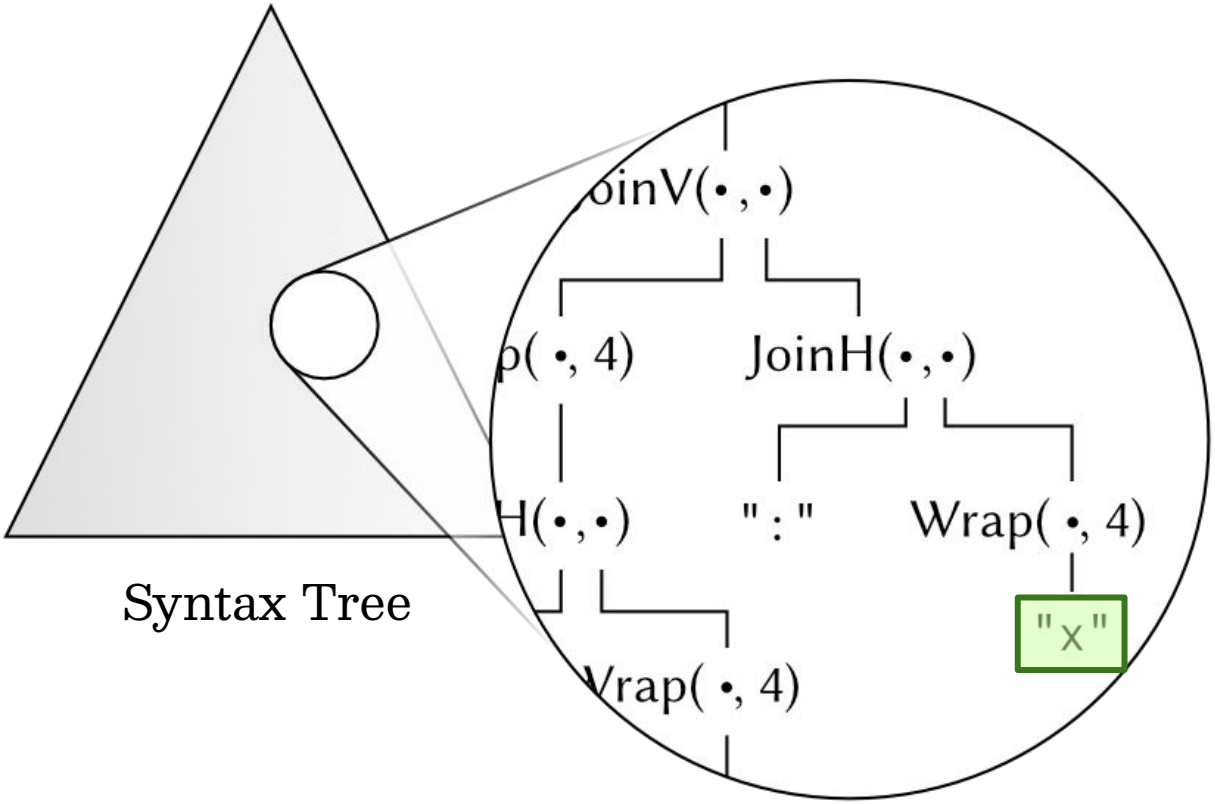
Child expression not contained in parent

Not a box!

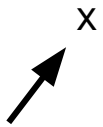


Cut child boundary to avoid parent text

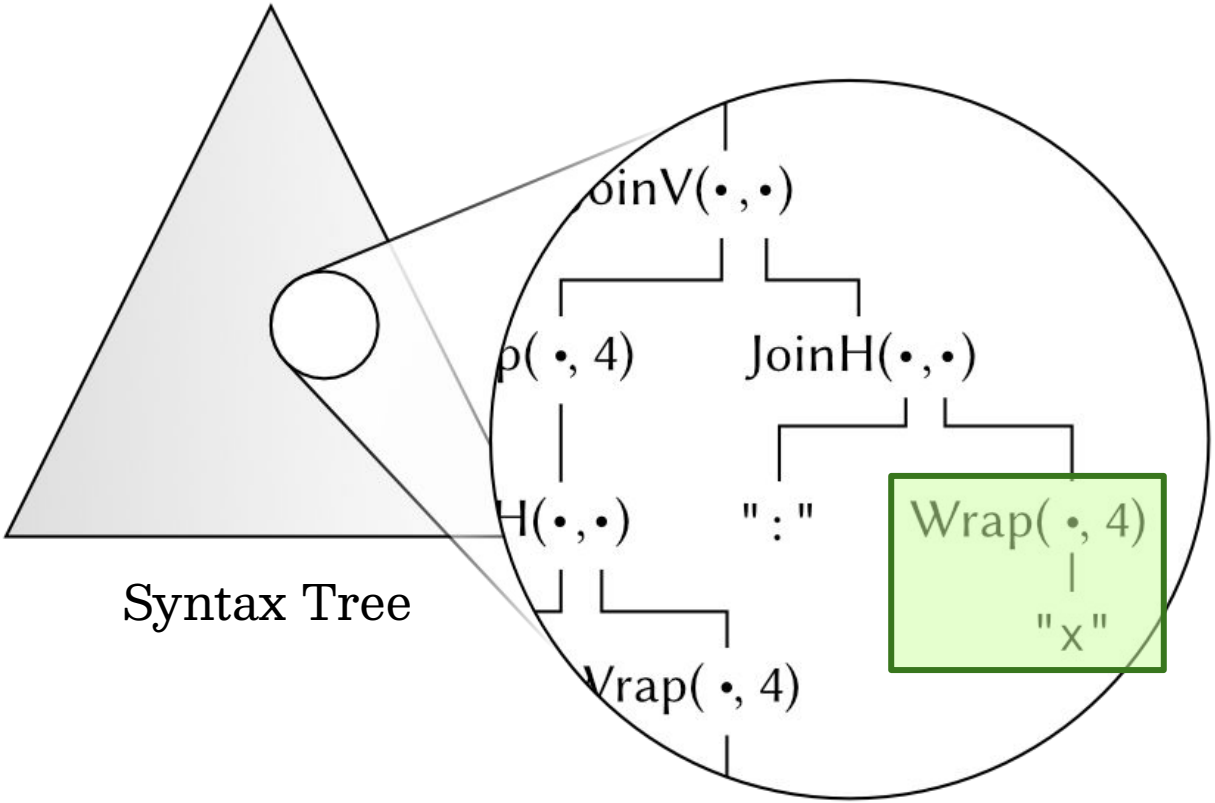
How do we know which operation to apply?



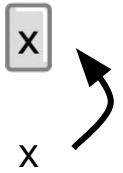
Syntax Tree



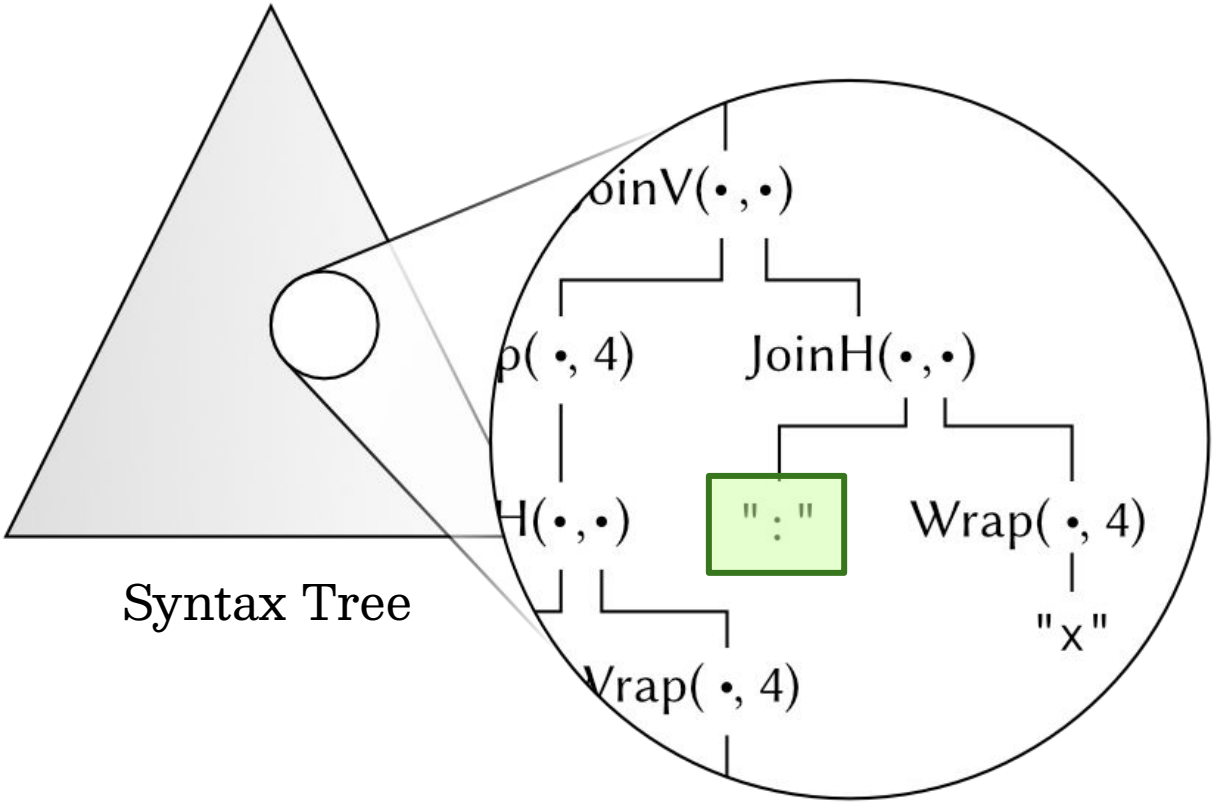
How do we know which operation to apply?

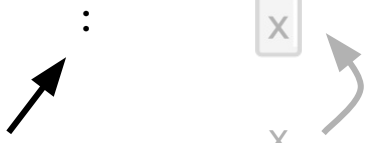


Syntax Tree

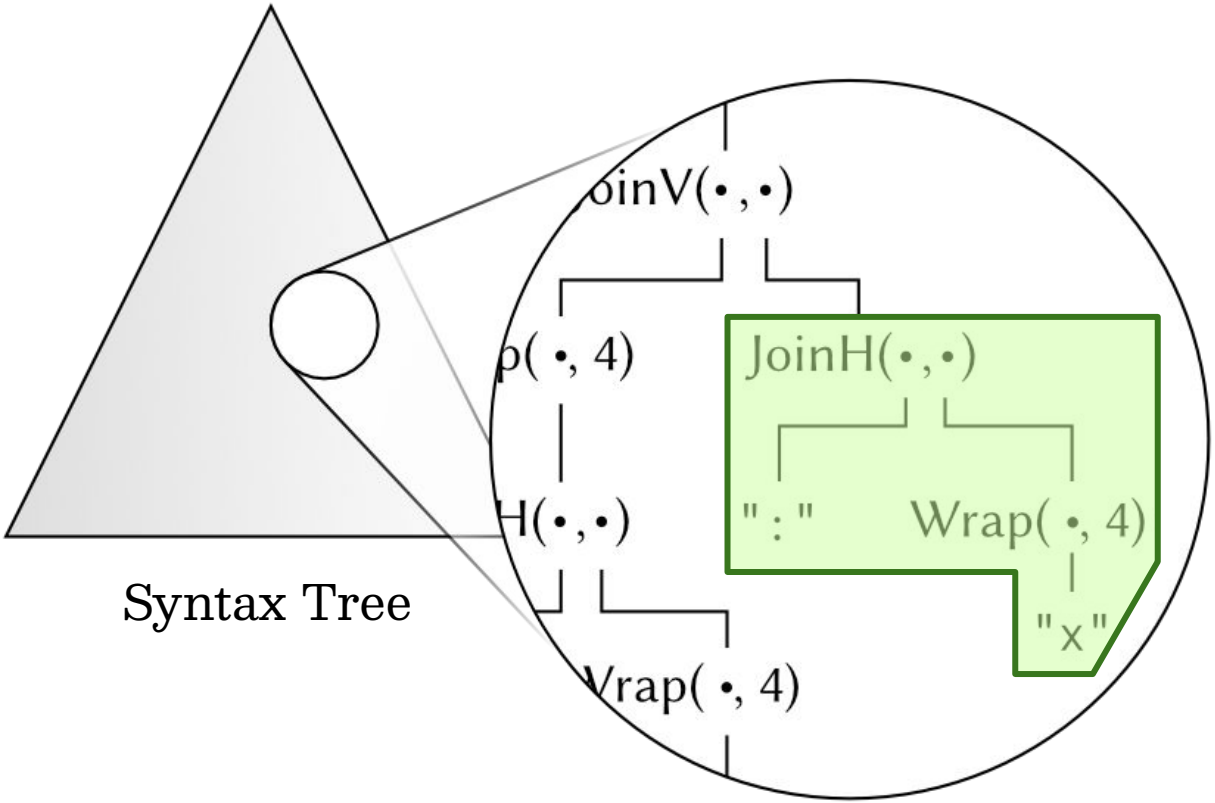


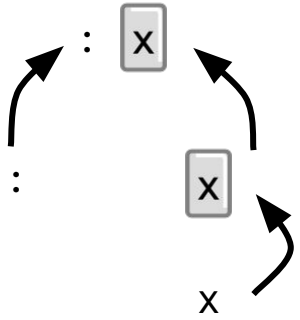
How do we know which operation to apply?



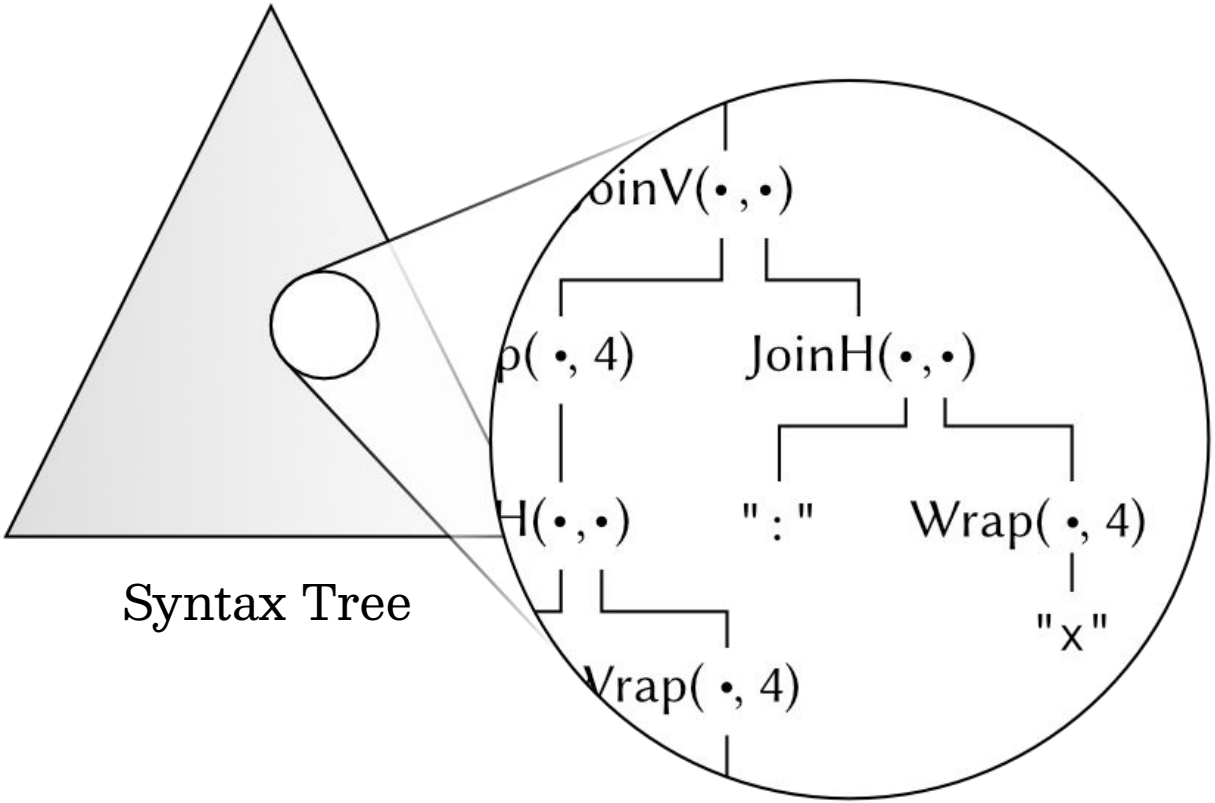


How do we know which operation to apply?





How do we know which operation to apply?



Syntax Tree