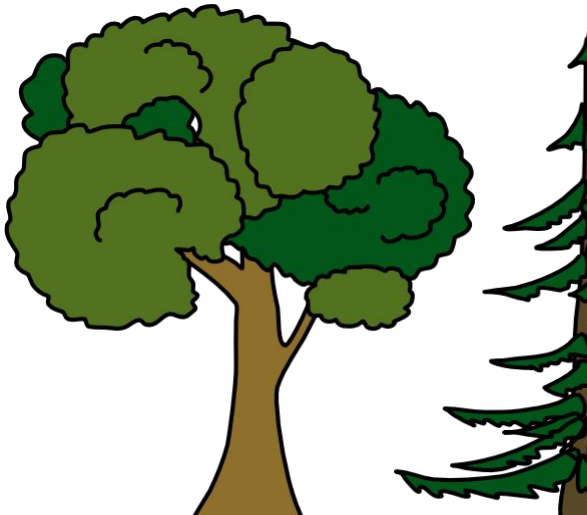
Code Style Sheets: CSS For Code

Sam Cohen and Ravi Chugh

OOPSLA 25, Singapore



Code is Displayed Uniformly, Independent of Task

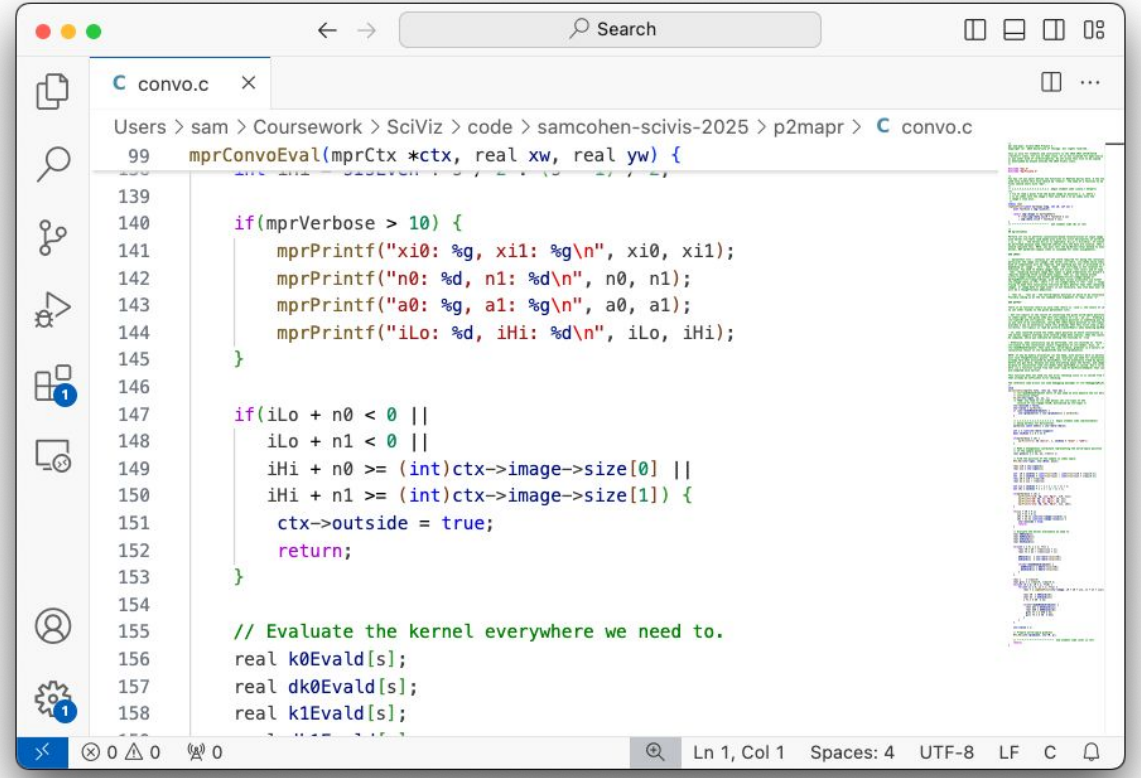
Writing

Debugging

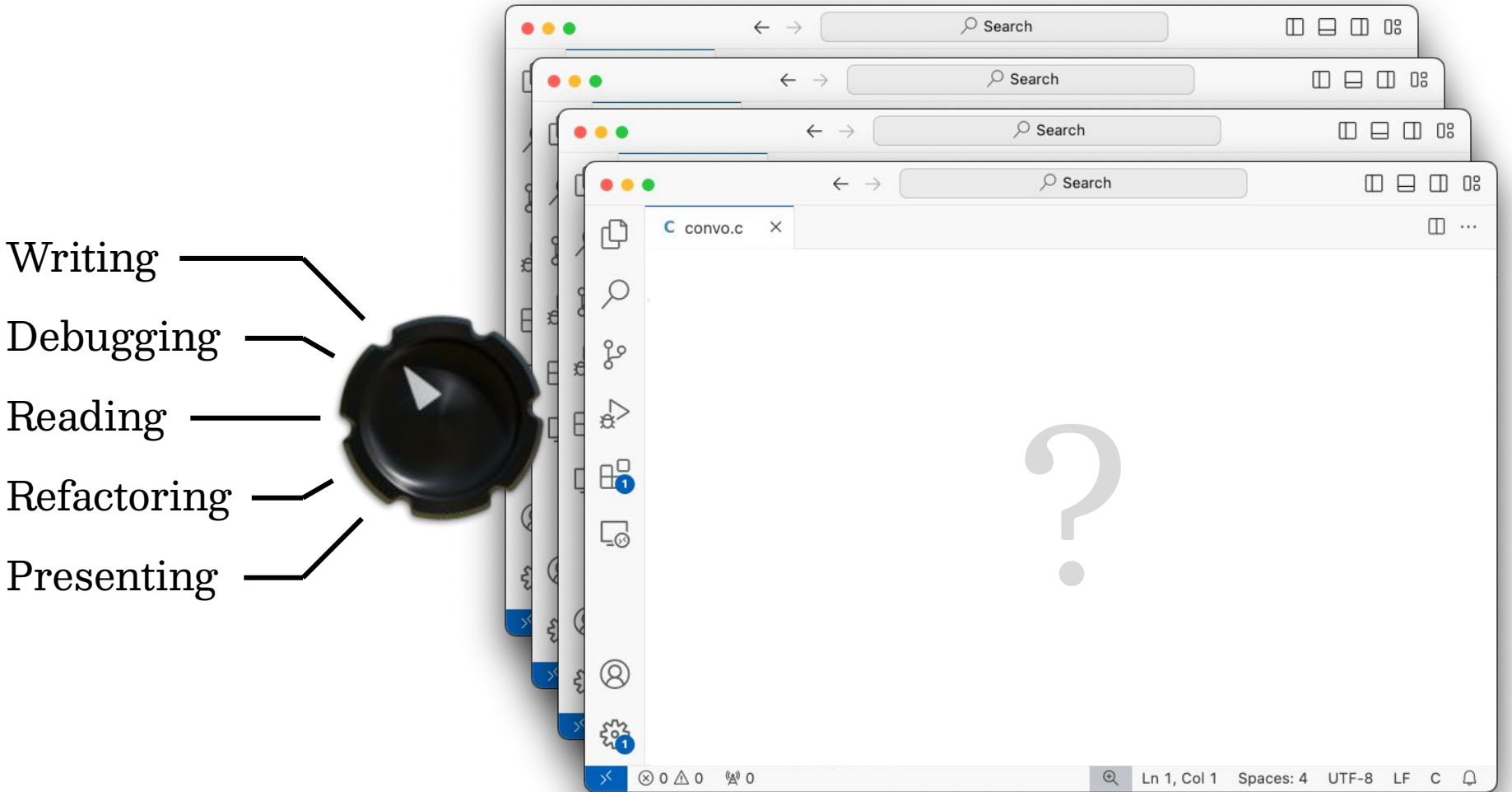
Reading

Refactoring

Presenting



Idea: Code Style Sheets



A Motivating Example

```
main =
  getContents
    >>= print
      . length
      . filter $ isPrefixOf “--”
      . lines
```

```
-- Anonymous Authors
module Main where

-- Print a greeting
main =
  putStrLn “Hello, OOPSLA!”
```

Expected Output:

4

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter $ isPrefixOf “--”
      . lines
```

There’s a type error!

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter $ isPrefixOf "--"
      . lines
```

Color-By-Type

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter $ isPrefixOf “--”
      . lines
```

Seems like `filter` isn't applied to `isPrefixOf`…

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter $ isPrefixOf "--"
      . lines
```

Color-By-Type

Blocks

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (isPrefixOf “--”)
      . lines
```

Dot binds more tightly than $!

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (isPrefixOf "--")
      . lines
```

Color-By-Type

Blocks

Raw Text

```
main =
  getContents
    >>= print
      . length
      . (\ls -> trace (filter (isPrefixOf “--”) ls))
      . lines
```

But the program gives the wrong answer…

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (isPrefixOf "--")
      . lines
```

Color-By-Type

Blocks

Raw Text

```
main =
  getContents
    >>= print
      . length
      . (\ls -> trace (filter (isPrefixOf “--”) ls))
      . lines
```

Filters out the wrong lines!

Styled Text

```
. length
. (\ls -> trace ((filter (isPrefixOf "--")) ls)
```

value

```
[ "-- Anonymous Authors"
, "-- Print a greeting" ]
```

Color-By-Type Blocks **Trace**

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

filter should be called keep

Styled Text

```
>>= print
  . length
  . (\ls -> trace ((filter (isPrefixOf "--")) ls)
```

```
value
[ "-- Anonymous Authors"
, "-- Print a greeting" ]
```

Color-By-Type Blocks **Trace**

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

>>= is read L-to-R, but dot is read R-to-L

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf "--")
      . lines
```

Color-By-Type

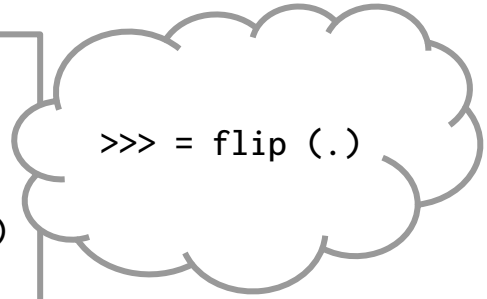
Blocks

Trace

Lint Warnings

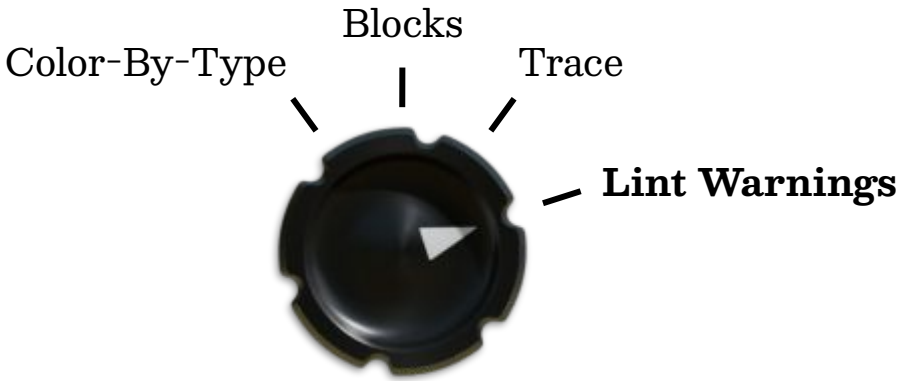
Raw Text

```
main =
  getContents
    >>= lines
    >>> filter (not . isPrefixOf “--”)
    >>> length
    >>> print
```



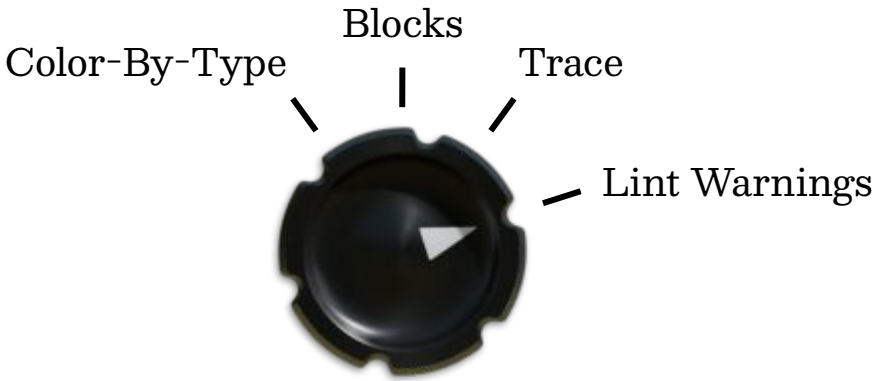
Styled Text

```
main =
  getContents
    >>= lines
    >>> filter (not . isPrefixOf "--")
    >>> length
    >>> print
```



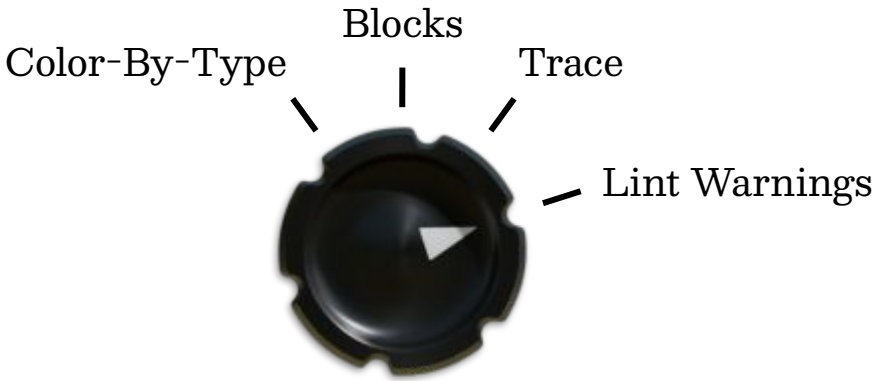
Takeaways

- Views are of the program source code
  (i.e. projections of the program's annotated AST)
- Utilized task-specific views of the same program



Takeaways

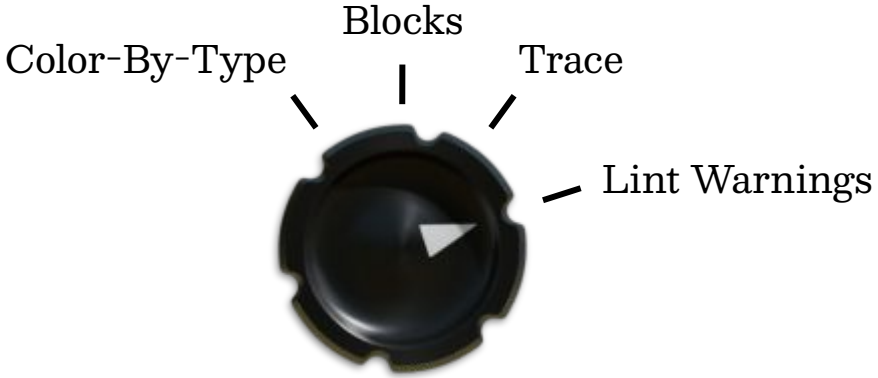
- Views are of the program source code
  (i.e. projections of the program's annotated AST) → **Document**
- Utilized task-specific views of the same program → **Style Sheets**



Takeaways

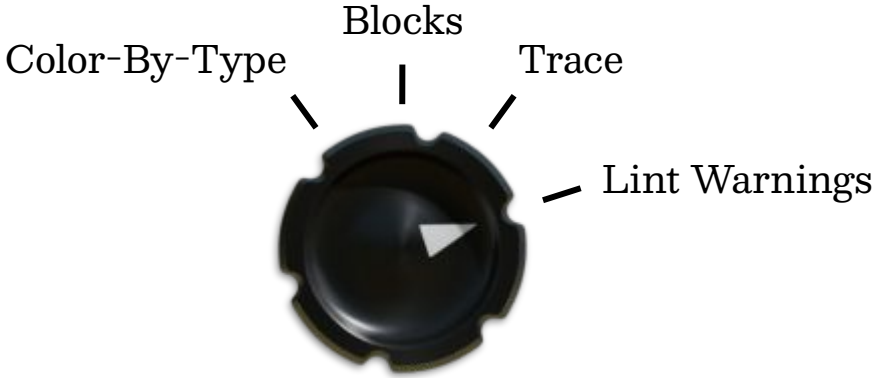
- Views are of the program source code
  (i.e. projections of the program's annotated AST) → **Document**
- Utilized task-specific views of the same program → **Style Sheets**

So we can just use CSS/HTML?



Takeaways

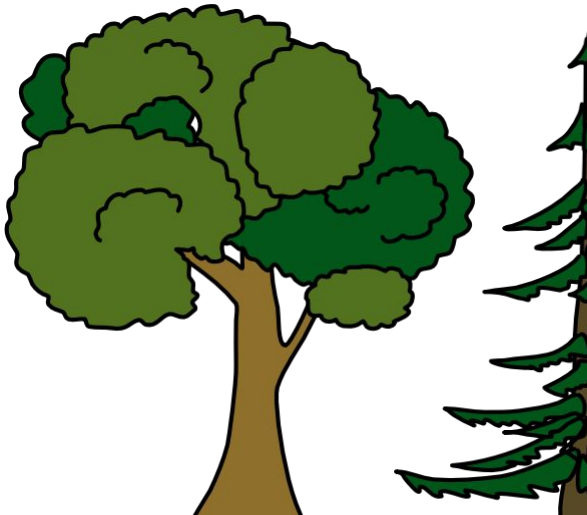
- Views are of the program source code
  (i.e. projections of the program's annotated AST) → **Document**
- Utilized task-specific views of the same program → **Style Sheets**



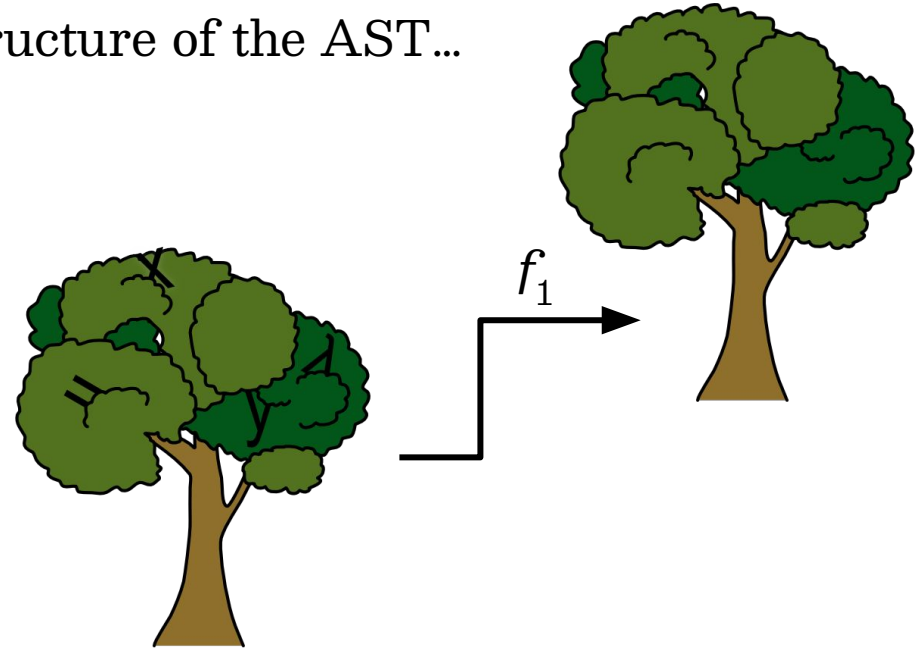
So we can just use CSS/HTML?

"Two Trees" Problem
Layout Problem

Two Trees Problem



View can retain the structure of the AST...



e.g. Blocks

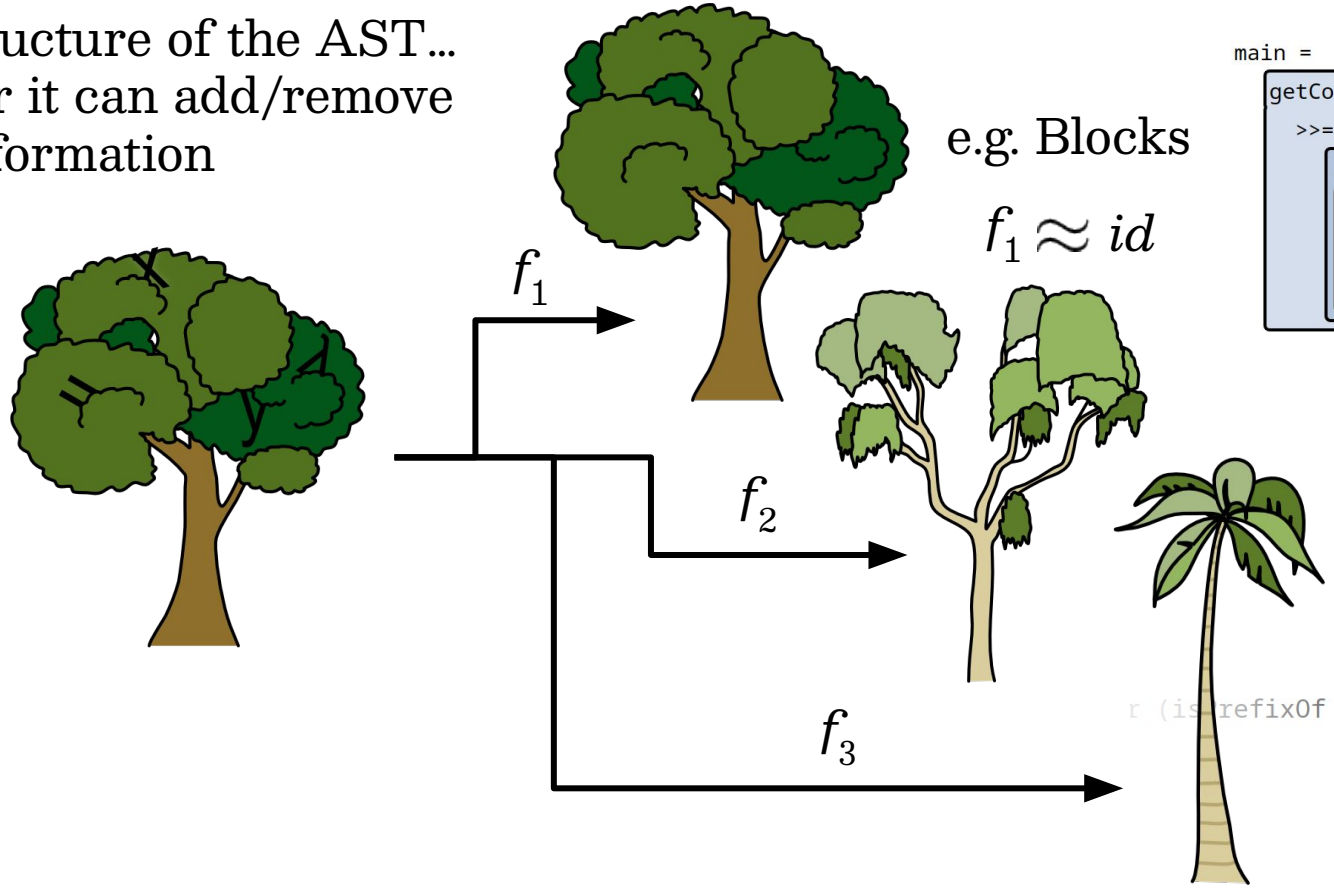
$f_1 \approx id$

```
main =
  getContents
    >>= print
    . length
    . filter (isPrefixOf "--")
    . lines
```

Abstract Syntax Tree

View Trees (Html)

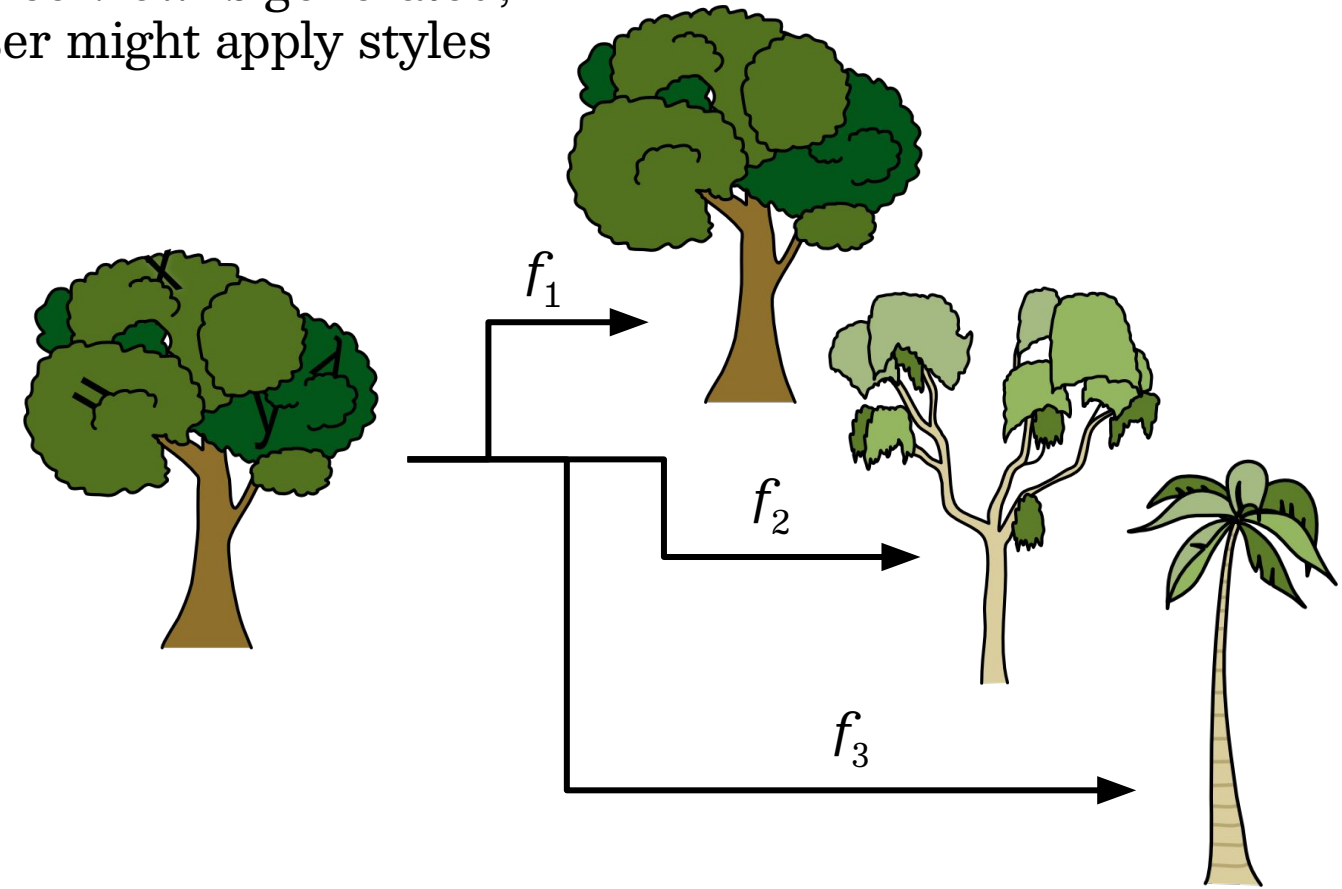
View can retain the structure of the AST...
...or it can add/remove Information



Abstract Syntax Tree

View Trees (Html)

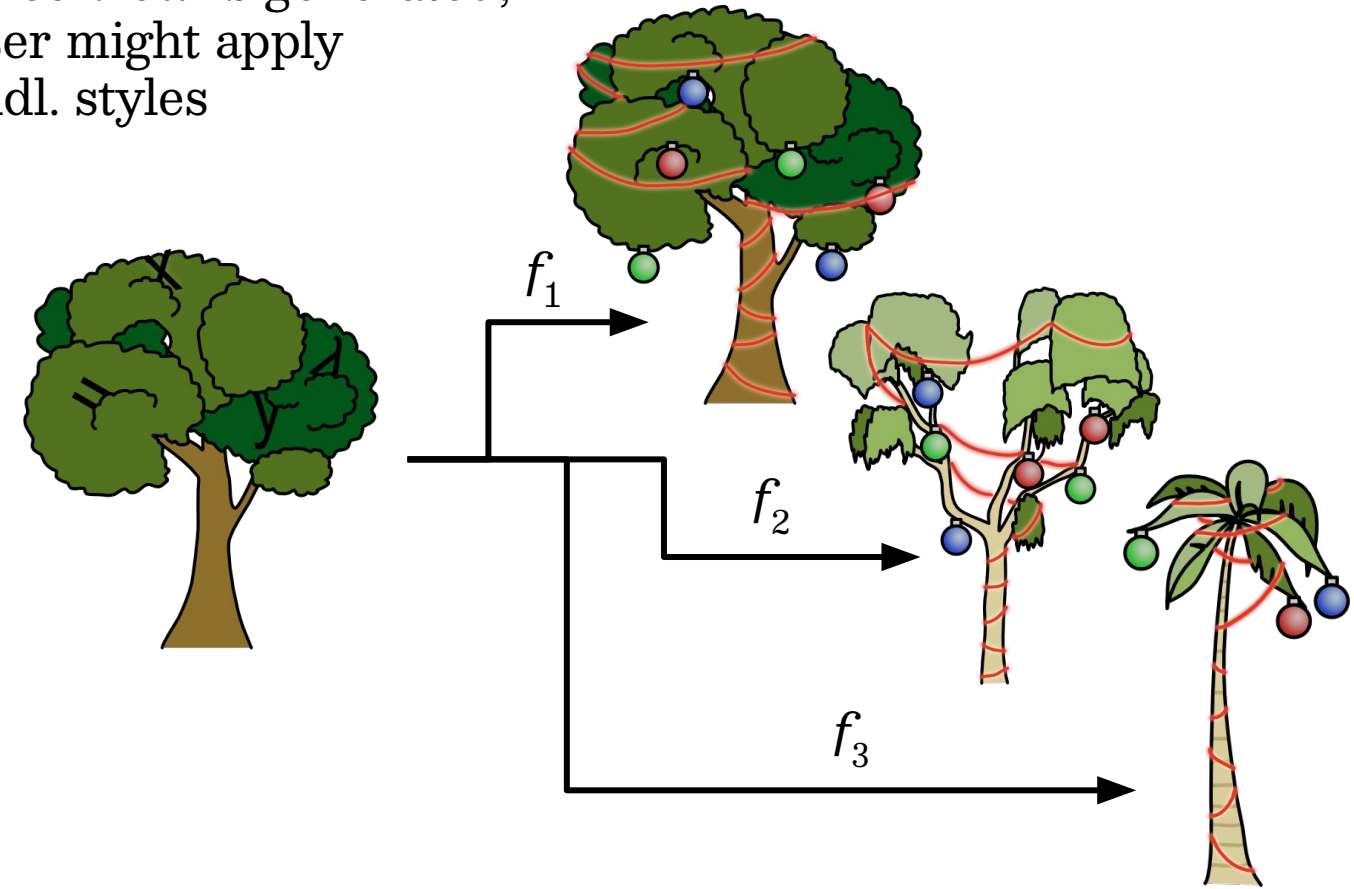
Once view is generated,
user might apply styles



Abstract Syntax Tree

View Trees (Html)

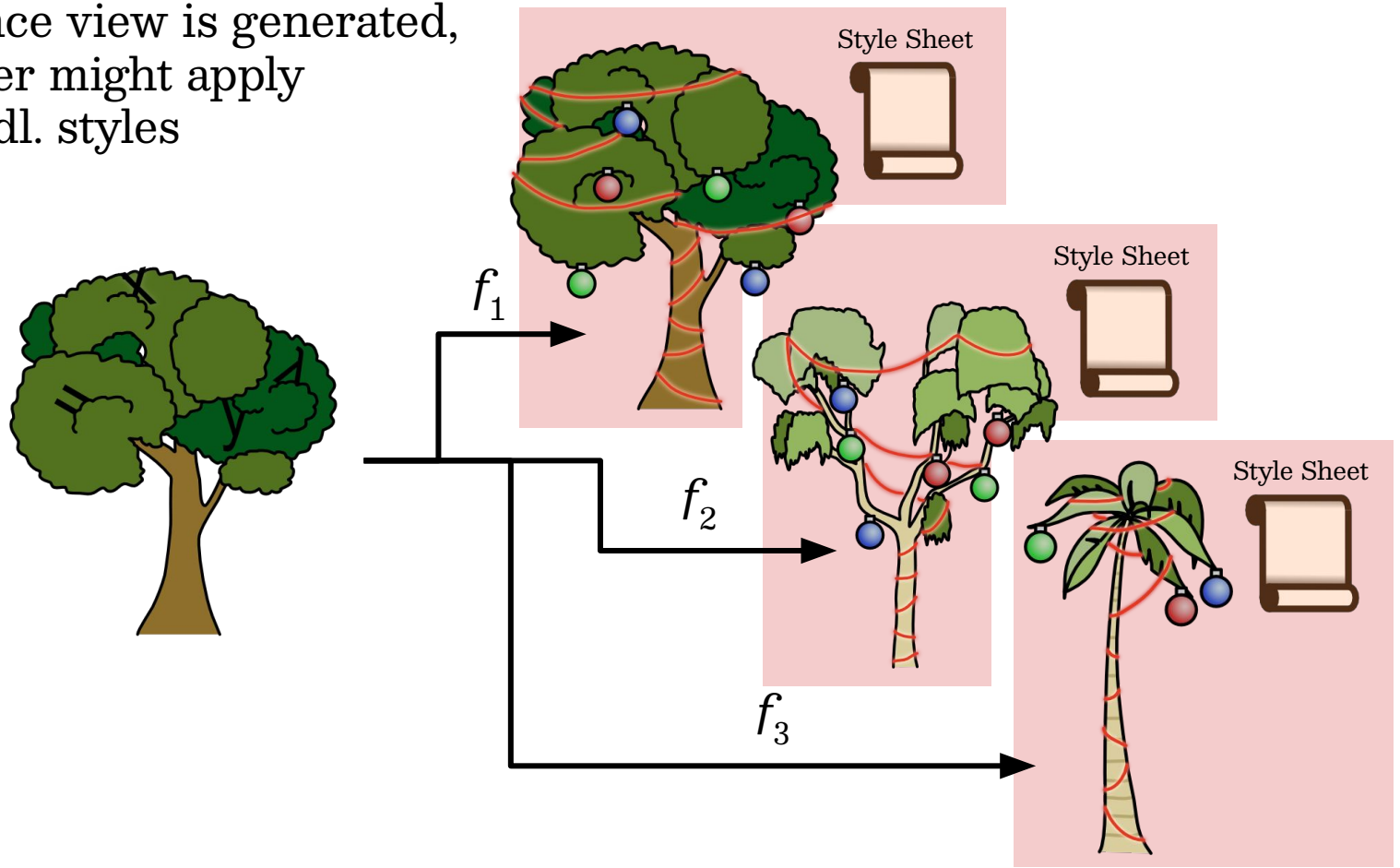
Once view is generated, user might apply addl. styles



Abstract Syntax Tree

View Trees (Html)

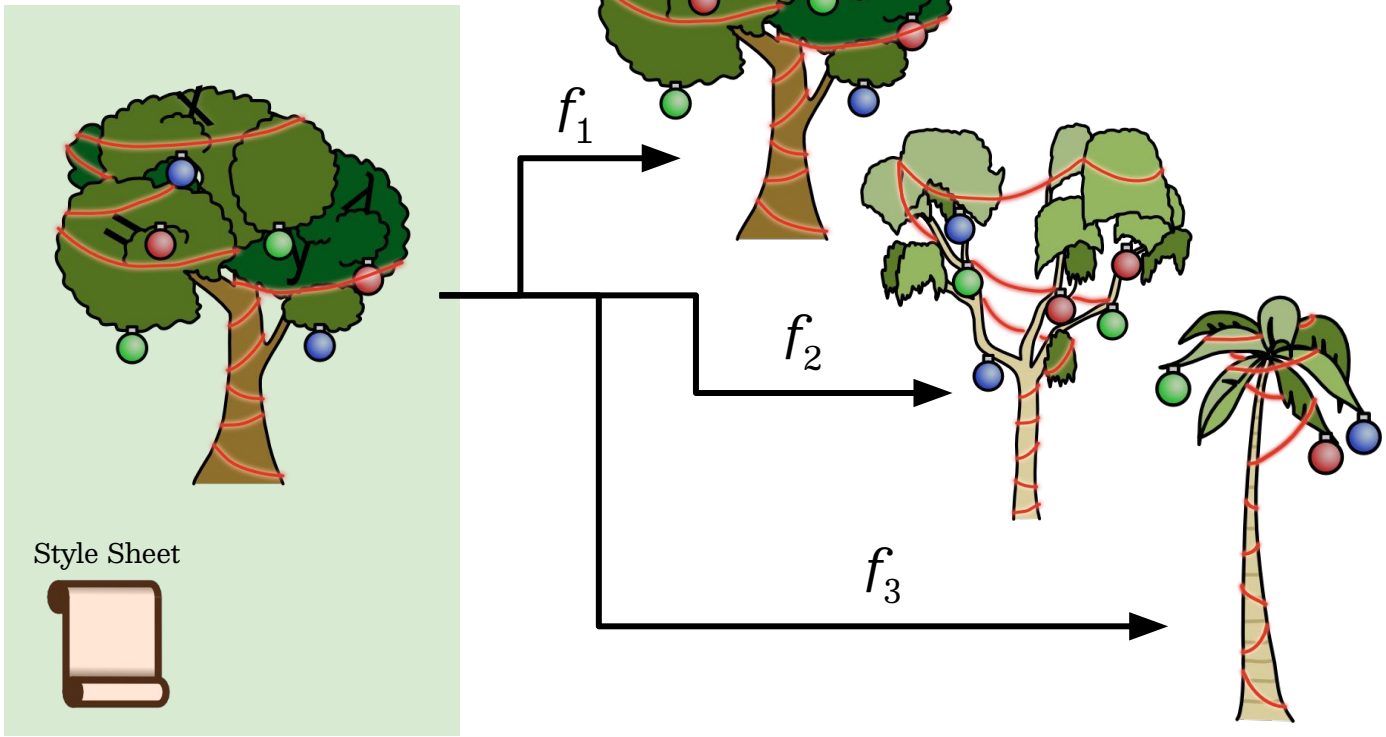
Once view is generated, user might apply addl. styles



Abstract Syntax Tree

View Trees (Html)

Goal: Addl. styles defined w.r.t. ***source AST***.

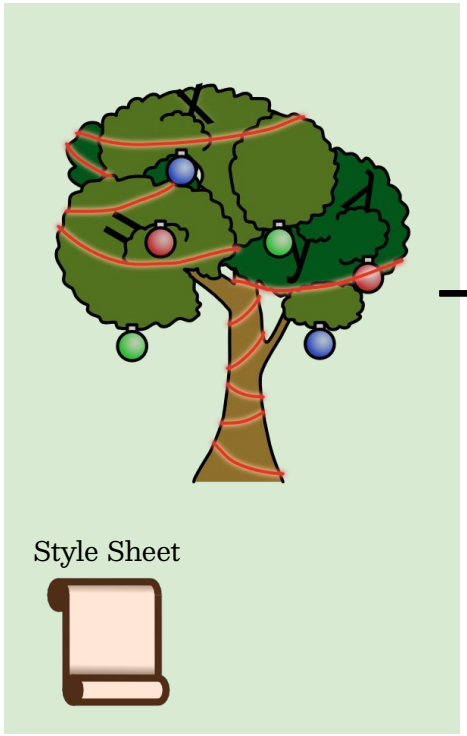


Abstract Syntax Tree

View Trees (Html)

Goal: Addl. styles defined w.r.t. ***source AST***.

One style sheet, many views



f_1

f_2

f_3

```
main =
  getContents
    >>= print
    . length
    . filter (isPrefixOf "--")
    . lines
```

```
r (isPrefixOf "--")) ls)
```

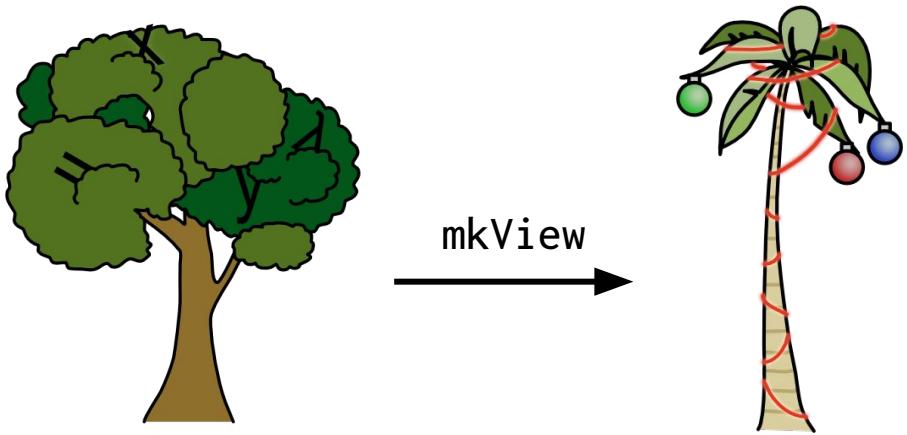
```
value
[ "-- Anonymous Aut
, "-- Print a greet
```

Abstract Syntax Tree

View Trees (Html)

Example: Tally Marks

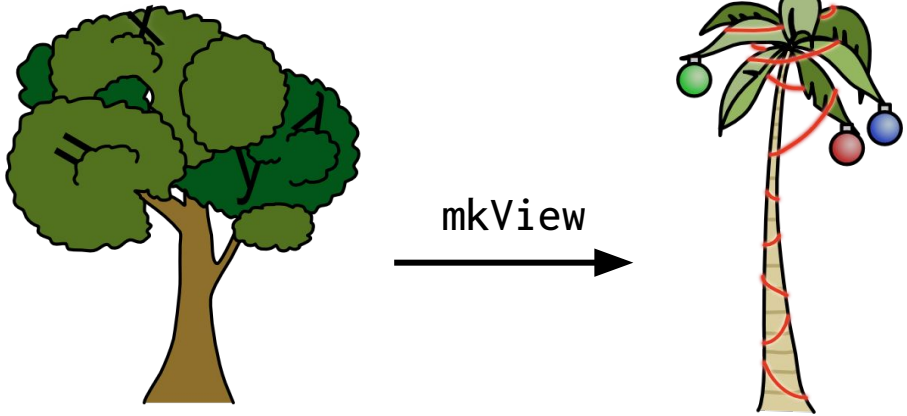
```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```



Example: Tally Marks

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```

1 | 2 || 3 ||| 4 ||||



Example: Tally Marks

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast = …
```

Example: Tally Marks

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast = …
```

```
<div>
  <span>1</span>
  <span>|</span>
  <span>2</span>
  <span>||</span>
  …
</div>
```

Color `Left` numbers orange

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast = …
```

```
<div>
  <span>1</span>
  <span>|</span>
  <span>2</span>
  <span>||</span>
  …
</div>
```

Color `Left` numbers orange

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast =
  … case e of {
    Left i -> …;
    Right i -> …;
  } …
```

```
<div>
  <span class="left">1</span>
  <span class="left">|</span>
  <span class="right">2</span>
  <span class="right">||</span>
  …
</div>
```

Box Left Square Numbers

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast =
  … case e of {
    Left i -> …;
    Right i -> …;
  } …
```

```
<div>
  <span class="left">1</span>
  <span class="left">|</span>
  <span class="right">2</span>
  <span class="right">||</span>
  …
</div>
```

Box Left Square Numbers

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```


```
mkView :: AST -> Html
mkView ast =
  … case e of {
    Left i -> …;
    Left i | square i -> …;
    Right i -> …;
  } …
```

```
<div>
  <span class="left square">1</span>
  <span class="left">|</span>
  <span class="right">2</span>
  <span class="right">||</span>
  …
</div>
```

Box Left Square Numbers

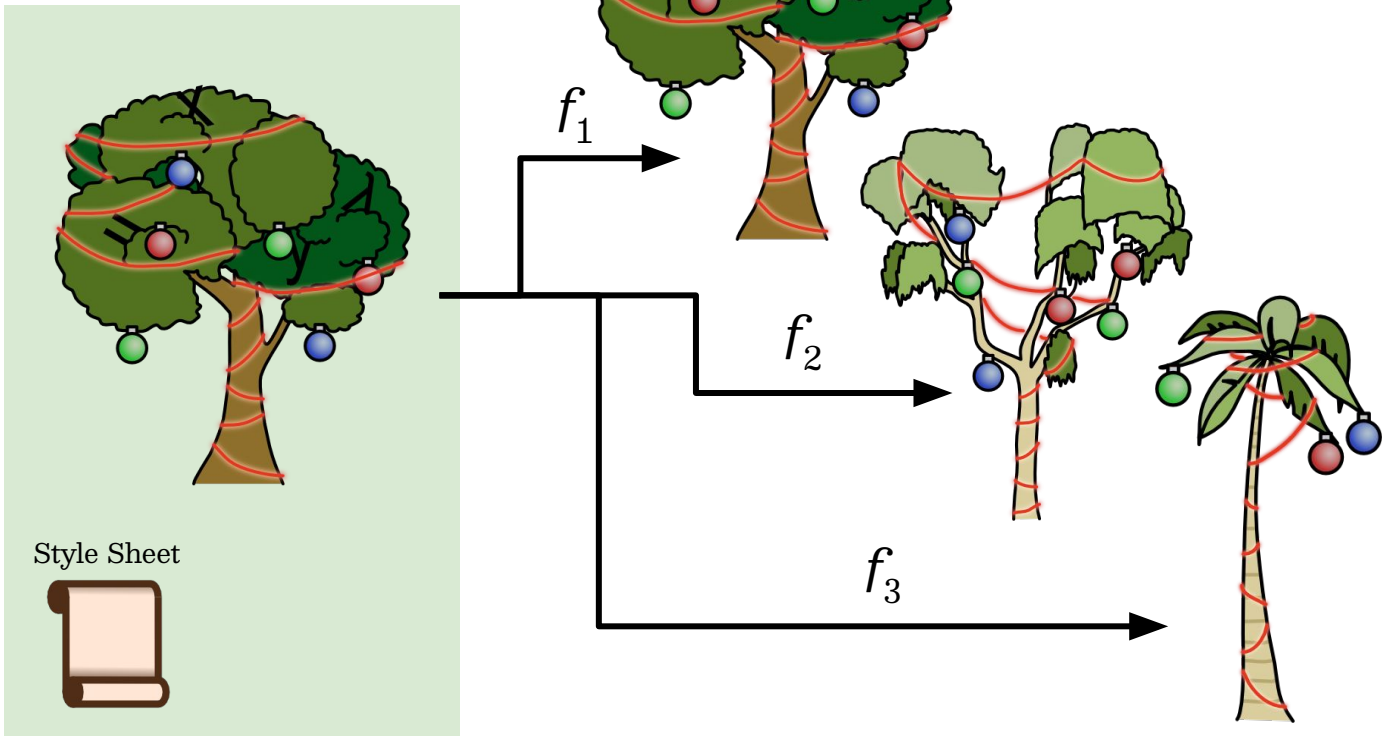
```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]
```

Style is decided in `mkView`, not in style sheet

```
mkView :: AST -> Html
mkView ast =
  … case e of {
    Left i -> …;
    Left i | square i -> …;
    Right i -> …;
  } …
```

```
<div>
  <span class="left square">1</span>
  <span class="left">|</span>
  <span class="right">2</span>
  <span class="right">||</span>
  …
</div>
```

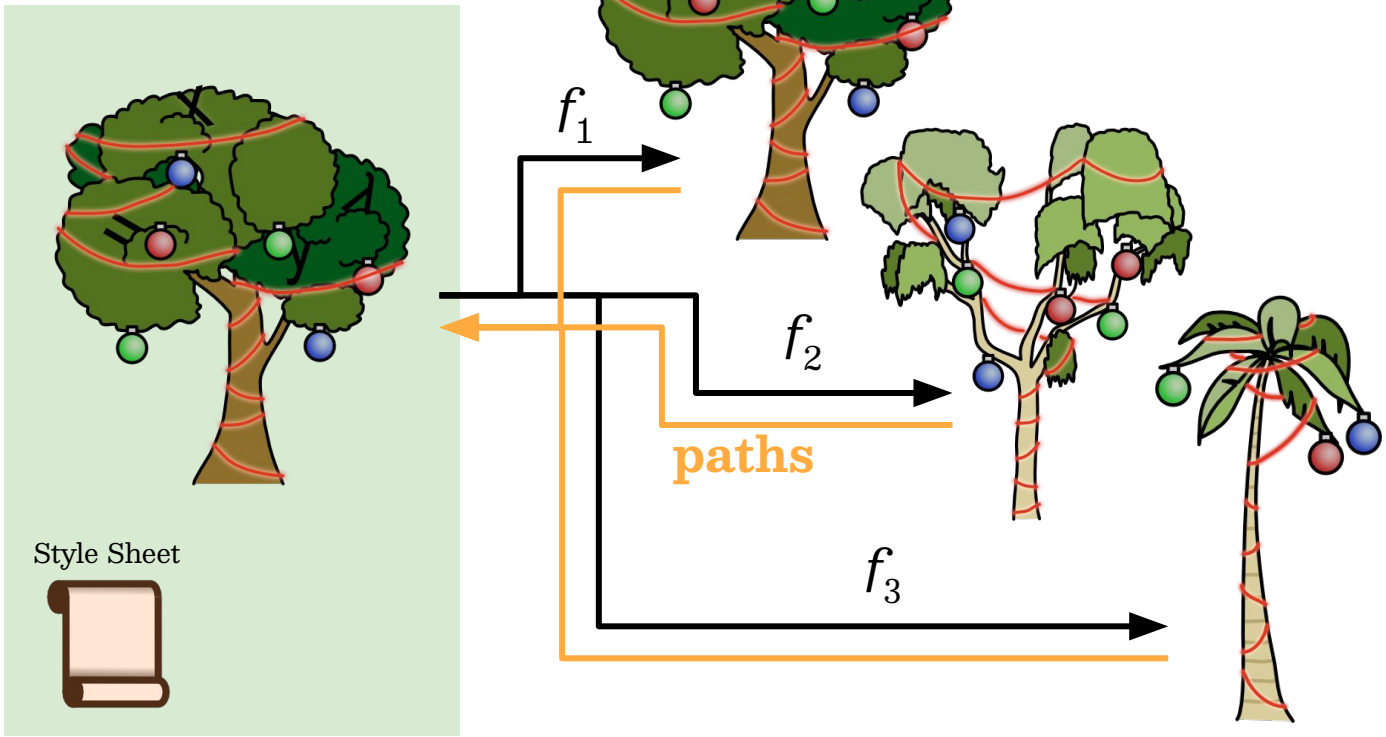
Goal: Addl. styles defined w.r.t. ***source AST***.



Abstract Syntax Tree

View Trees (Html)

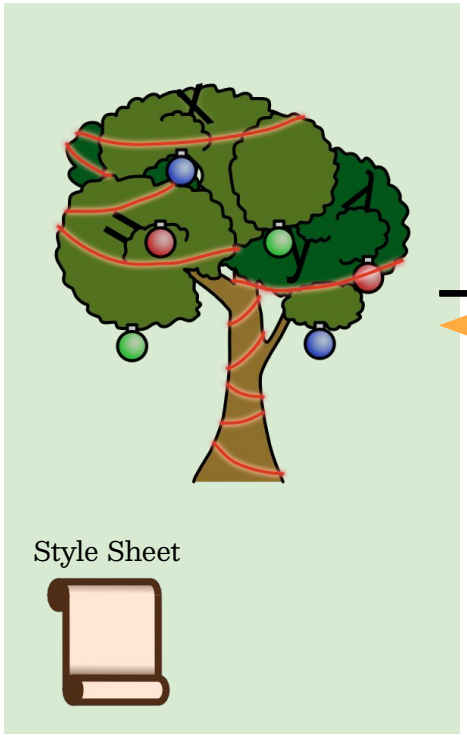
Goal: Addl. styles defined w.r.t. ***source AST***.

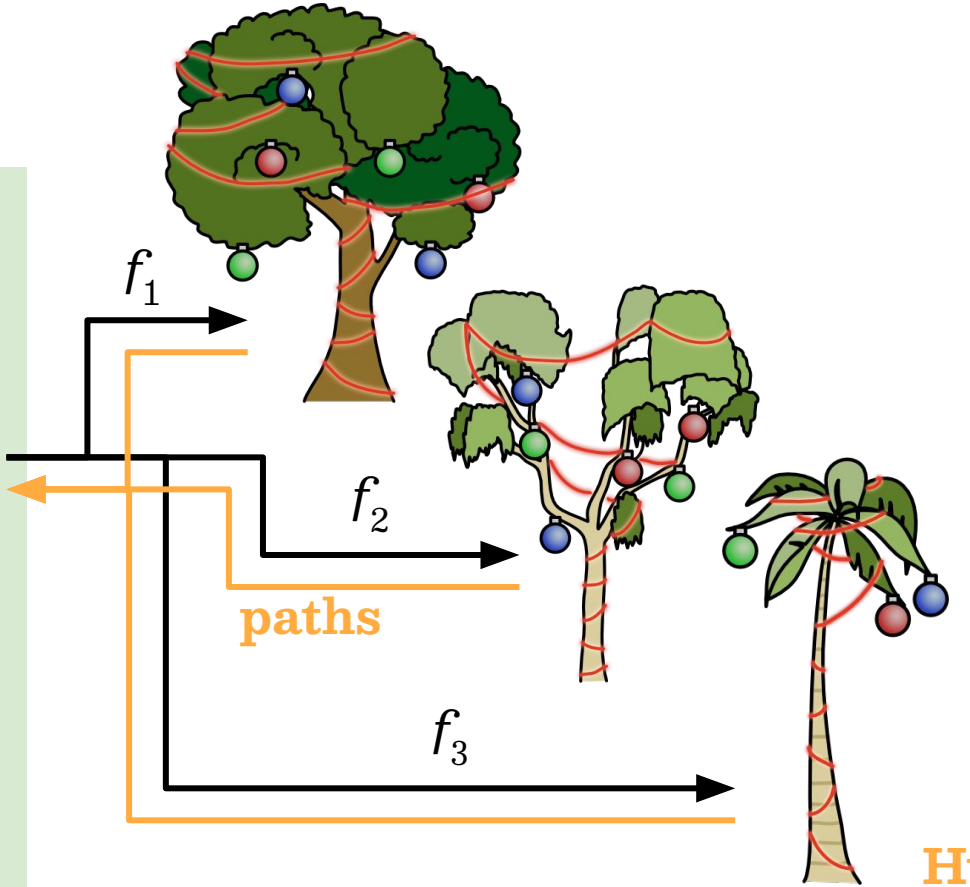


Abstract Syntax Tree

View Trees (Html)

Goal: Addl. styles defined w.r.t. ***source AST***.





Abstract Syntax Tree

View Trees (**HtmlP**)

HtmlP = Html + Provenance

Box Left Square Numbers

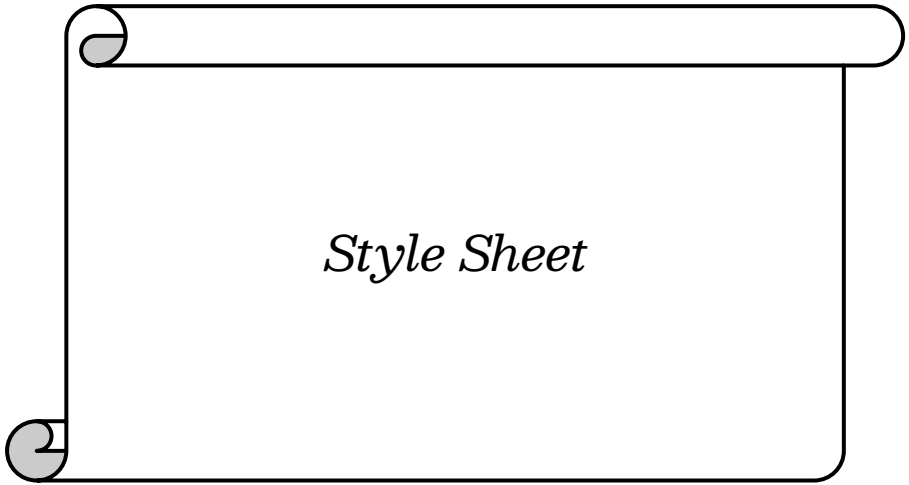
```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]

mkView :: AST -> Html HtmlP
mkView ast = …
```


```
data HtmlP
 = Text String
 | Node {
   tag      :: String,
   attrs    :: …,
   children :: [HtmlP],
   path     :: Maybe [Int]
 }
```

Color `Left` numbers orange

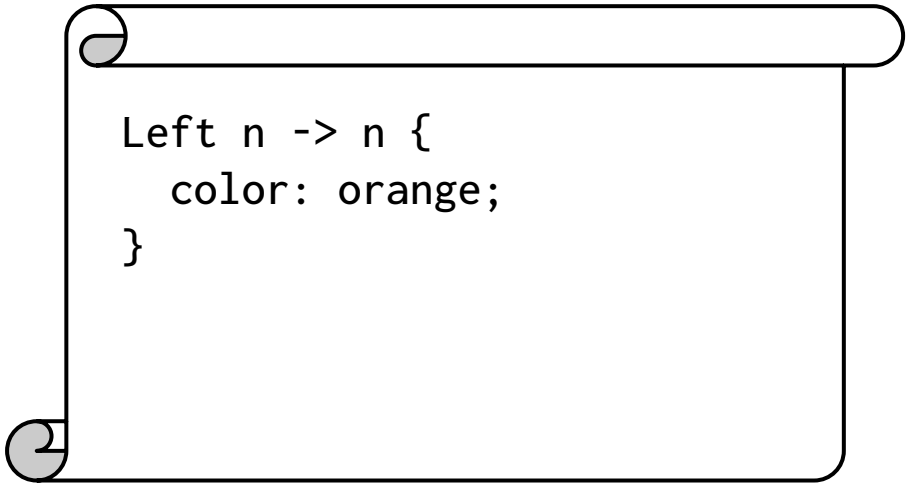
```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]

mkView :: AST -> HtmlP
mkView ast = …
```

1 | 2 || 3 ||| 4 ||||



Color `Left` numbers orange

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]

mkView :: AST -> HtmlP
mkView ast = …
```



Box Left Square Numbers

```
type AST = [Either Int Int]
myAst :: AST = [
  Left 1,
  Right 2,
  Left 3,
  Left 4
]

mkView :: AST -> HtmlP
mkView ast = …
```


```
Left n -> n {
  color: orange;
}
Left n if square n -> n {
  border: 3px solid orange;
}
```

Style Sheet Semantics

Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```

Style Sheet Semantics

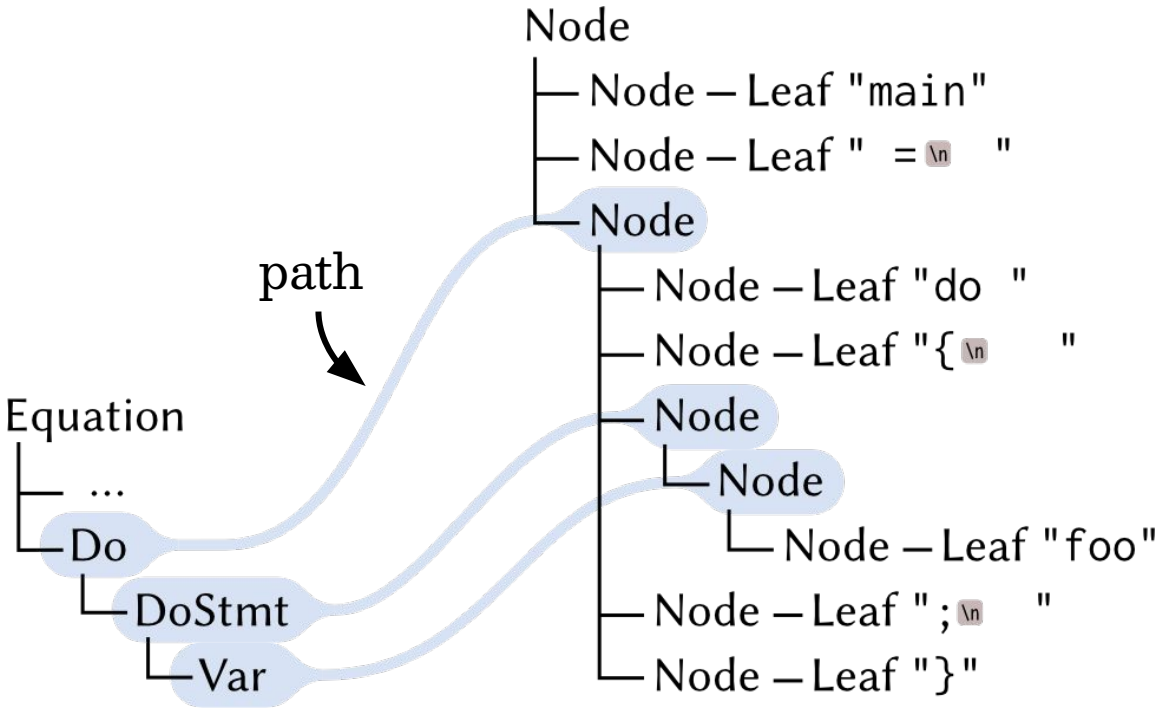
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



Style Sheet Semantics

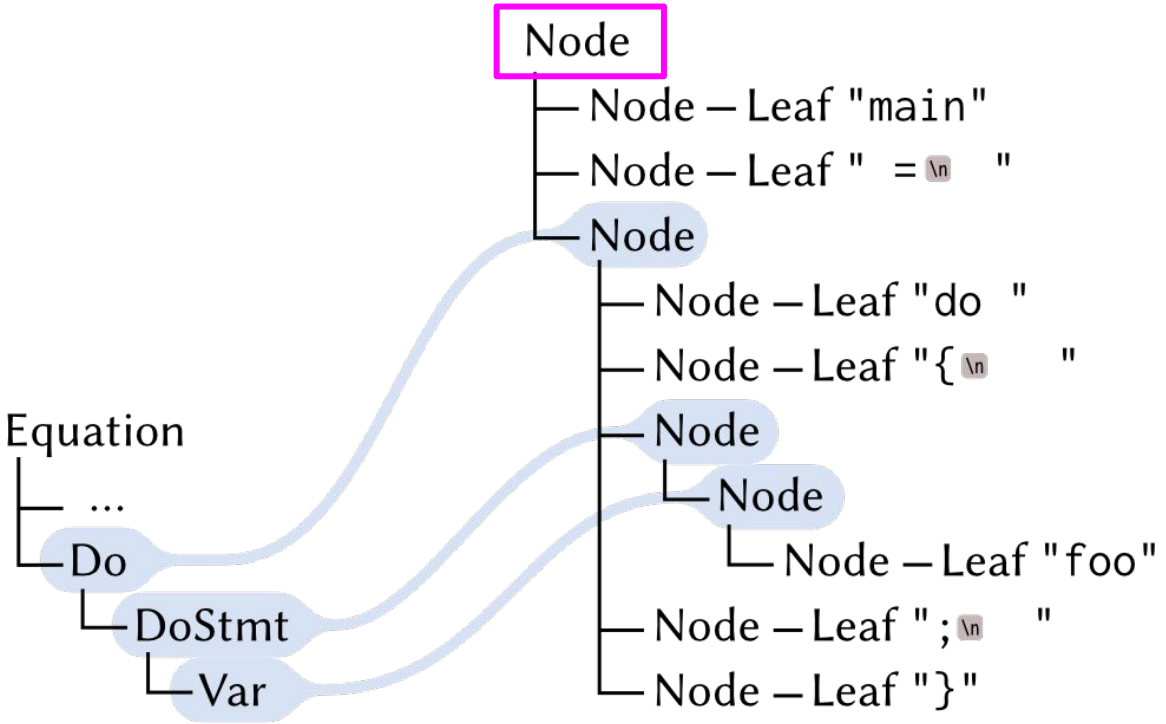
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

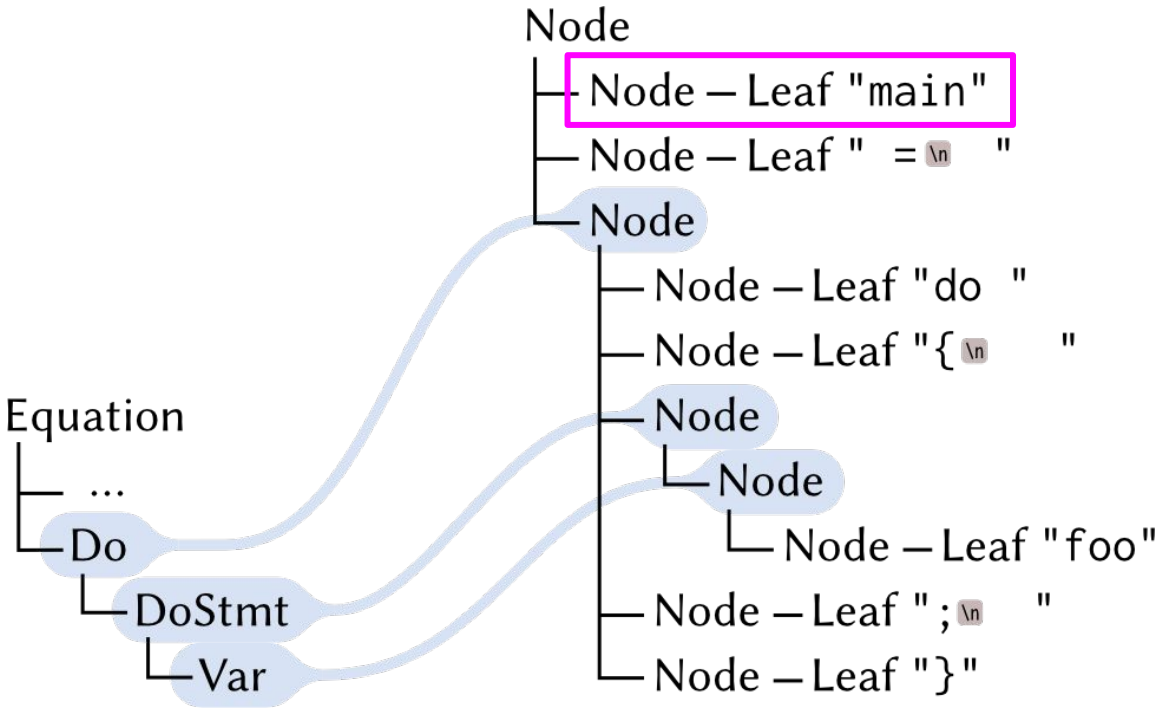
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

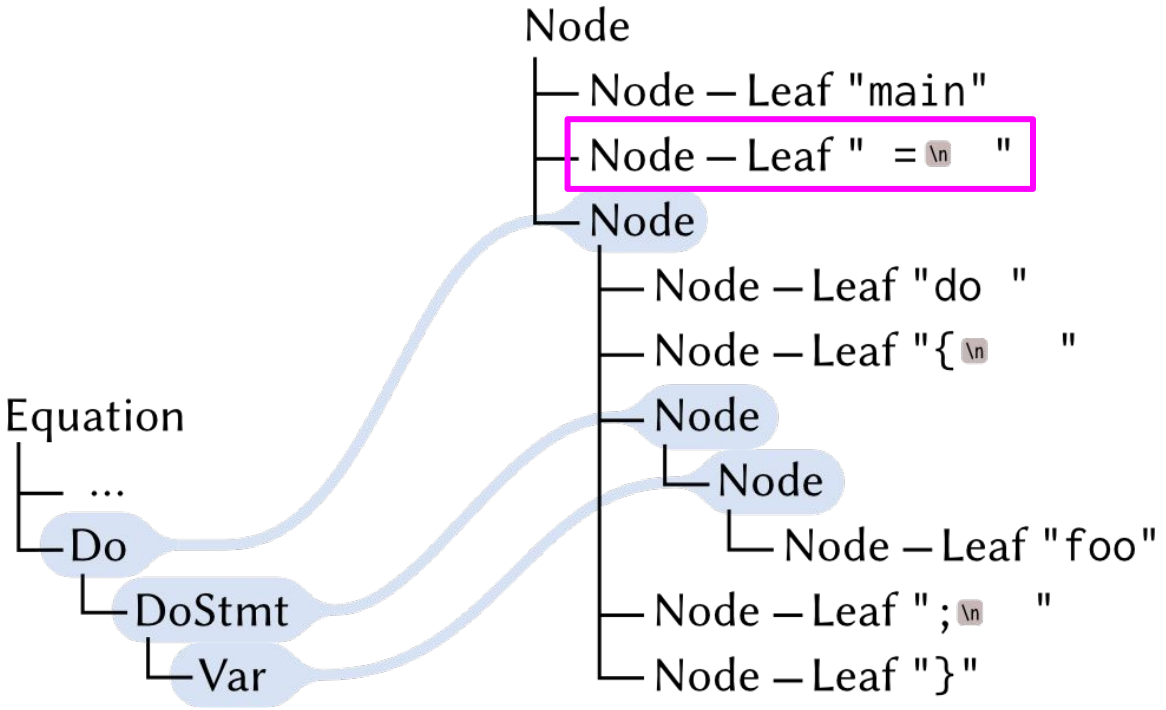
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



Style Sheet Semantics

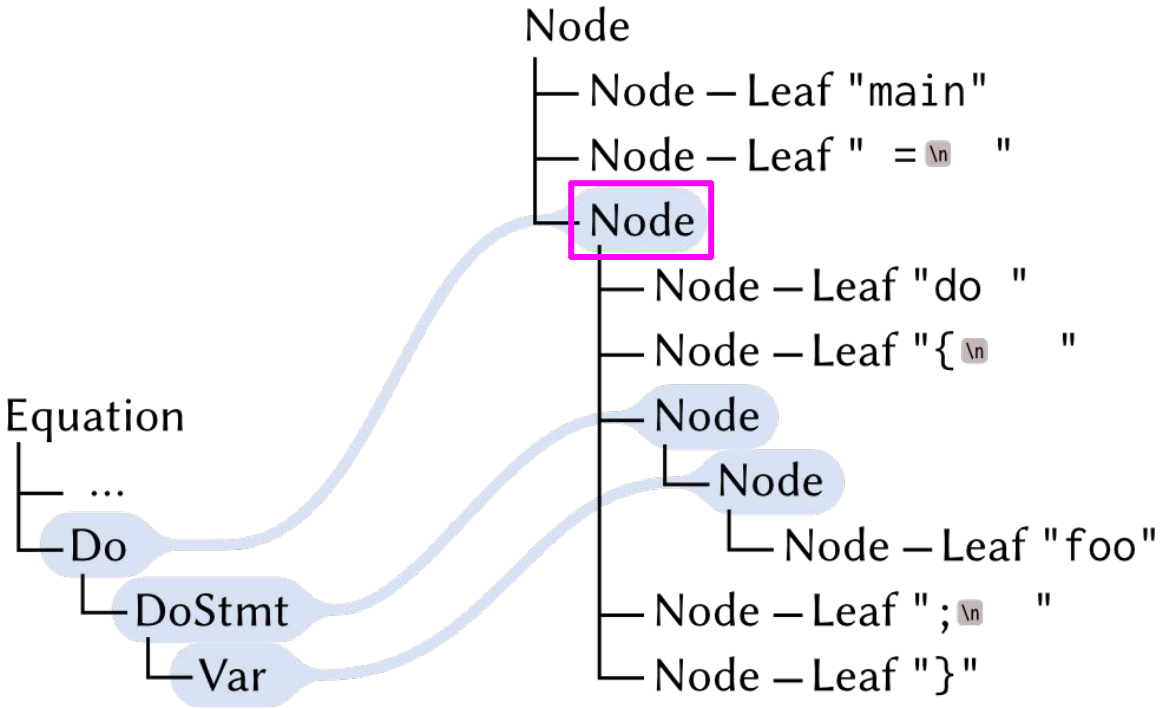
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

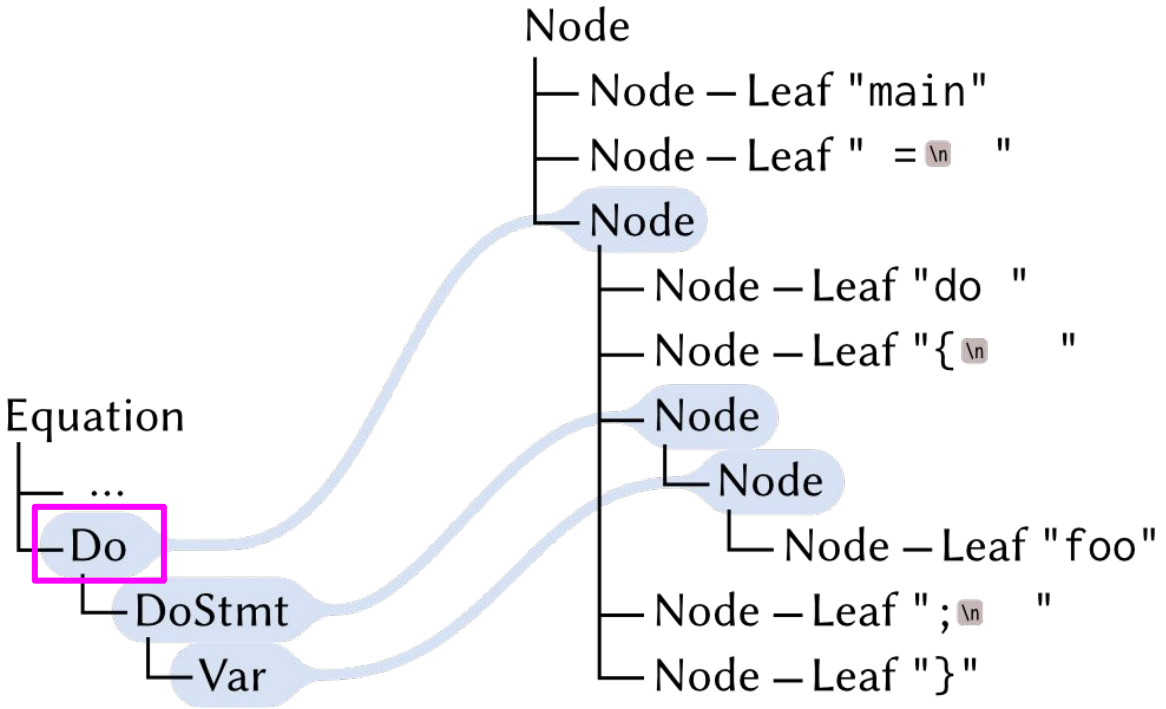
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

Raw Text

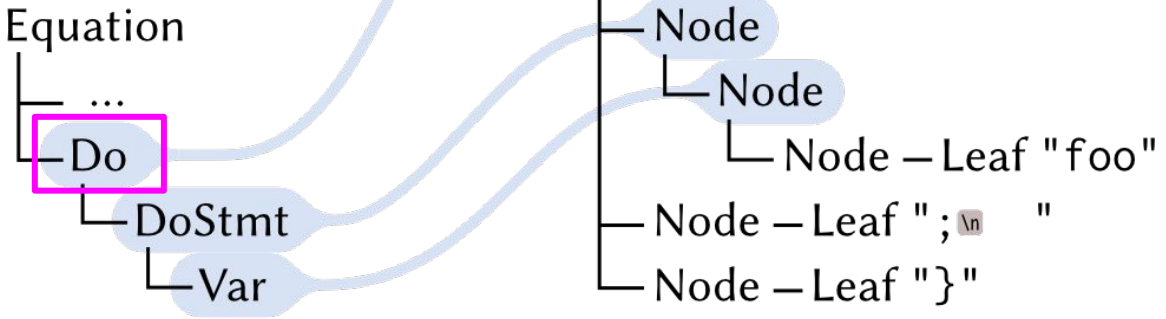
```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

✗

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

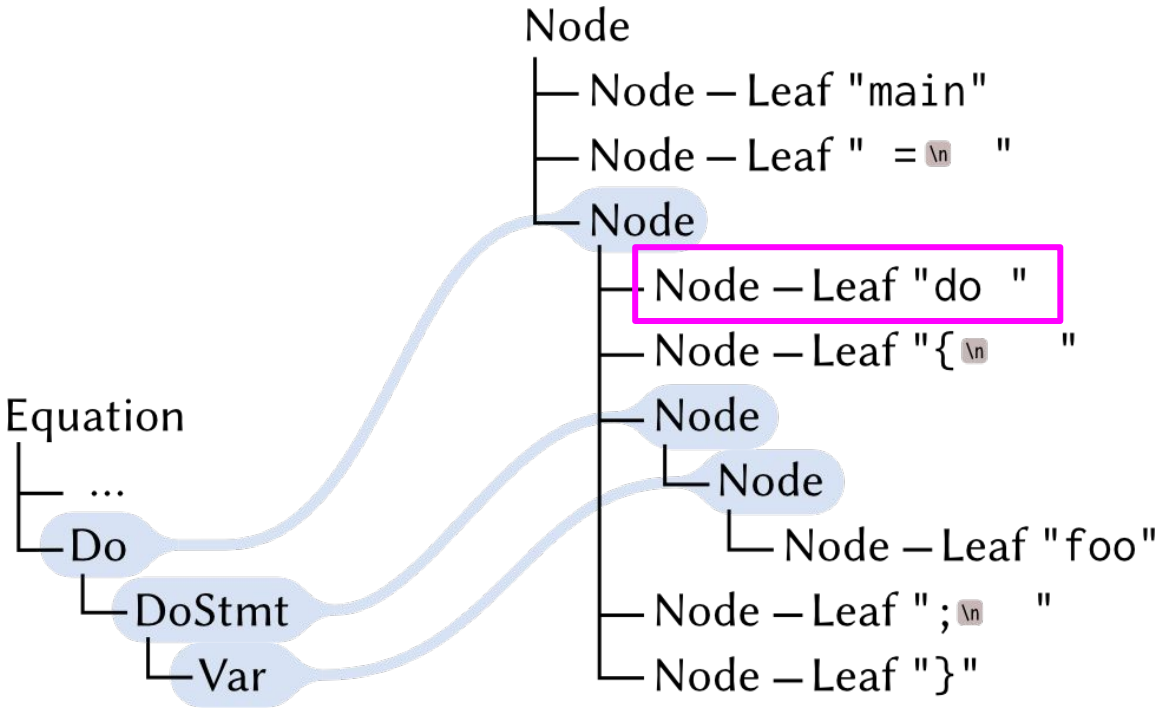
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

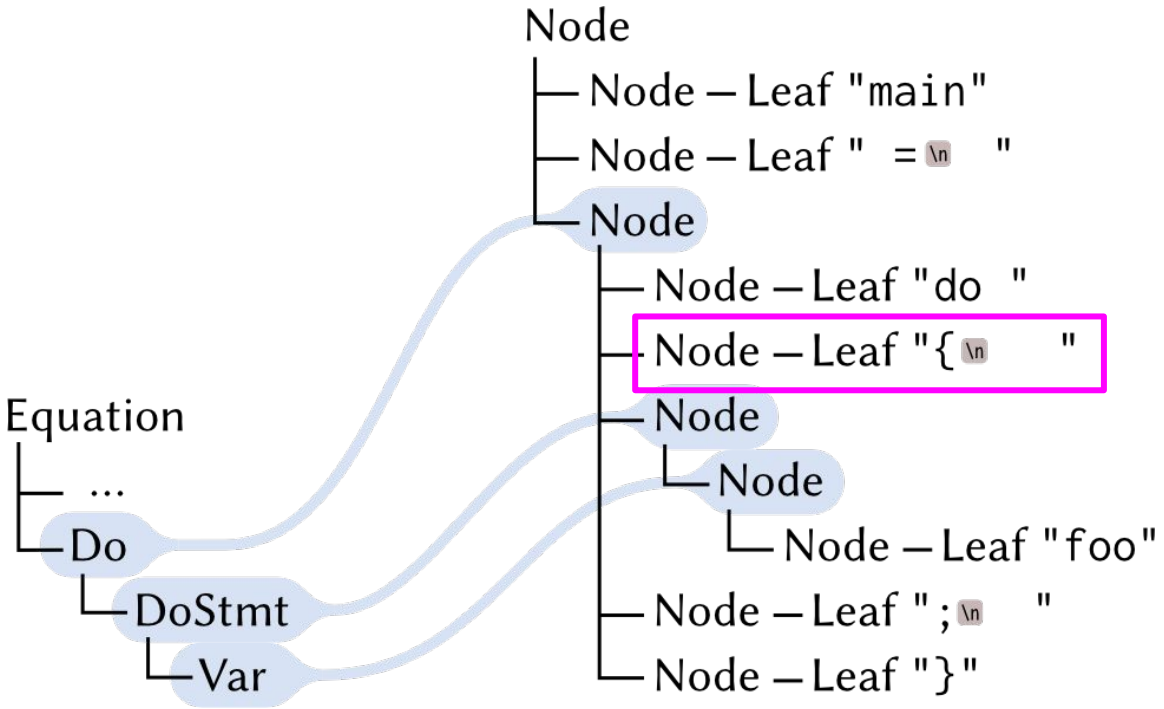
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



Style Sheet Semantics

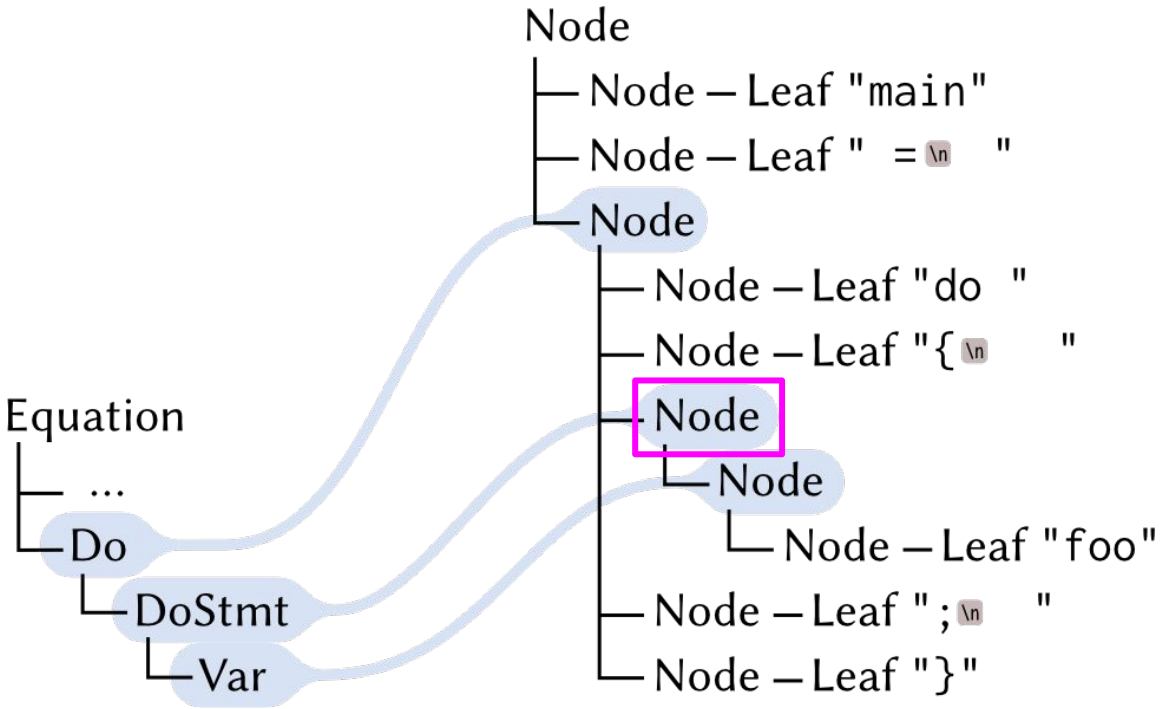
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

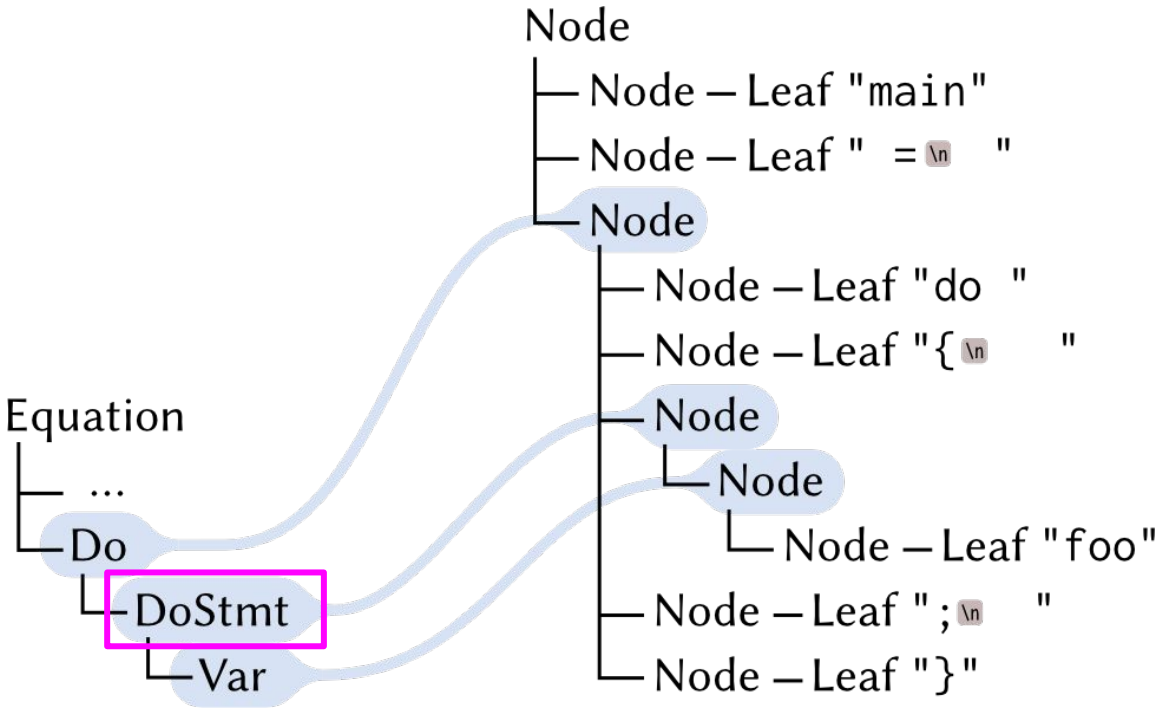
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



Style Sheet Semantics

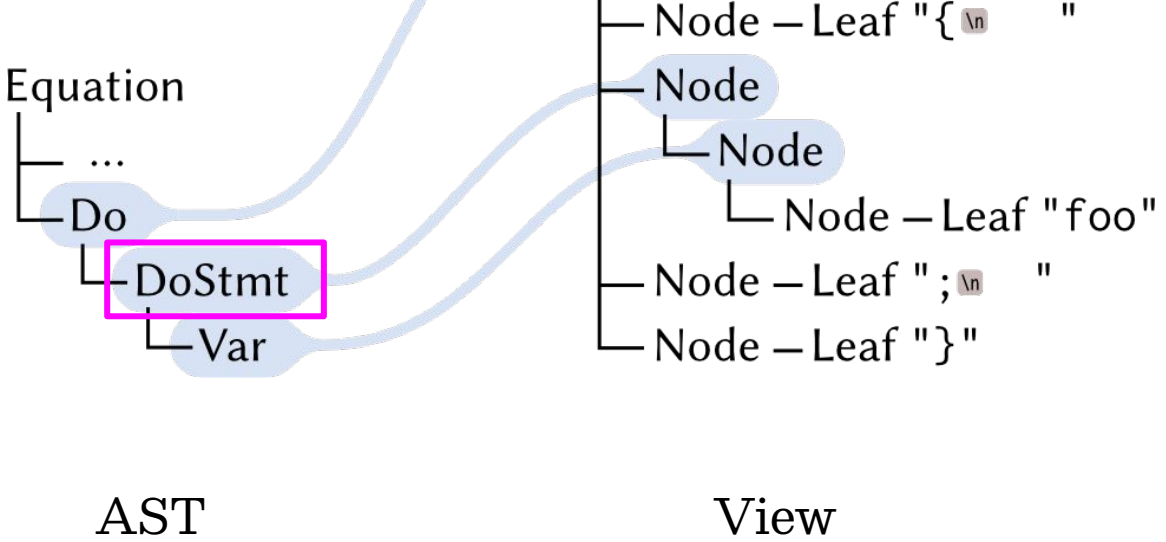
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

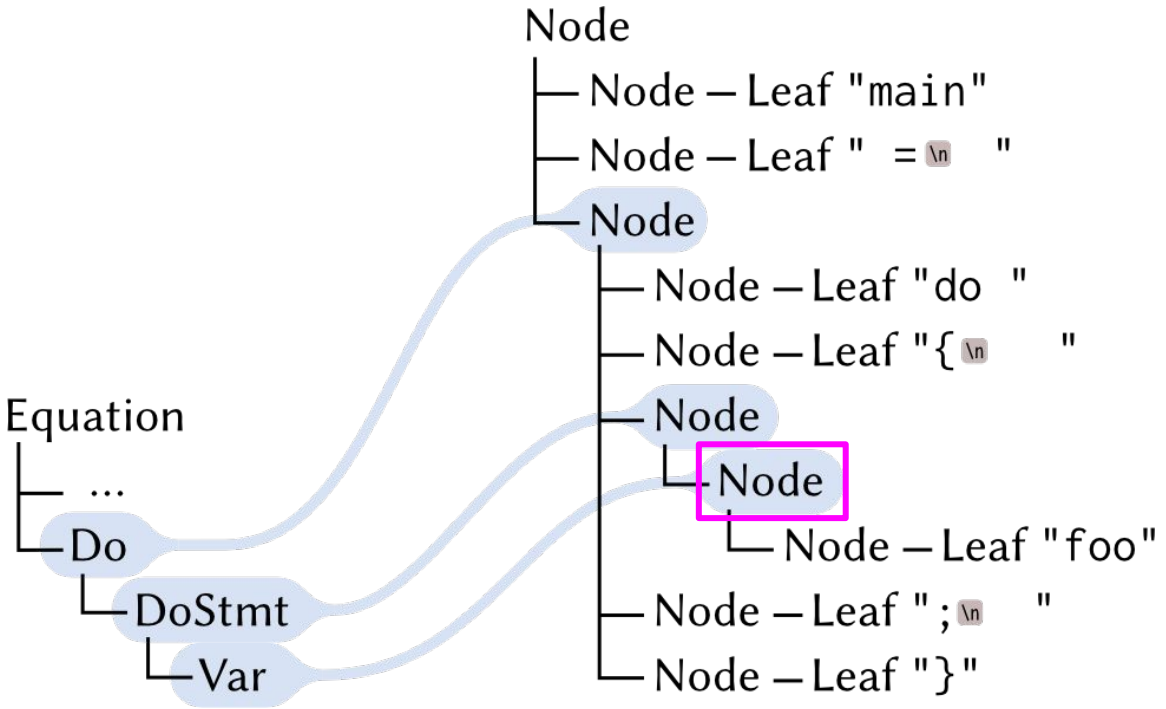
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

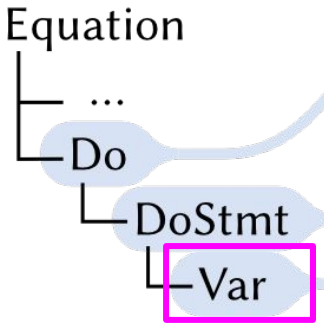
Raw Text

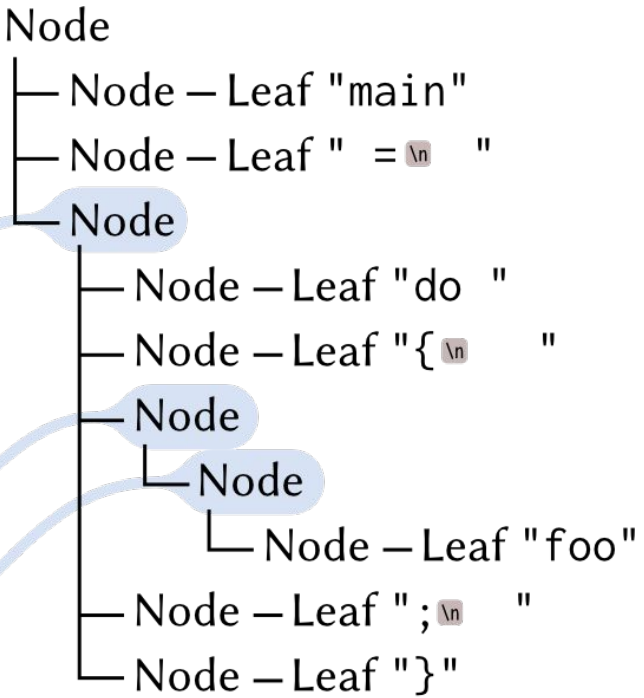
```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```

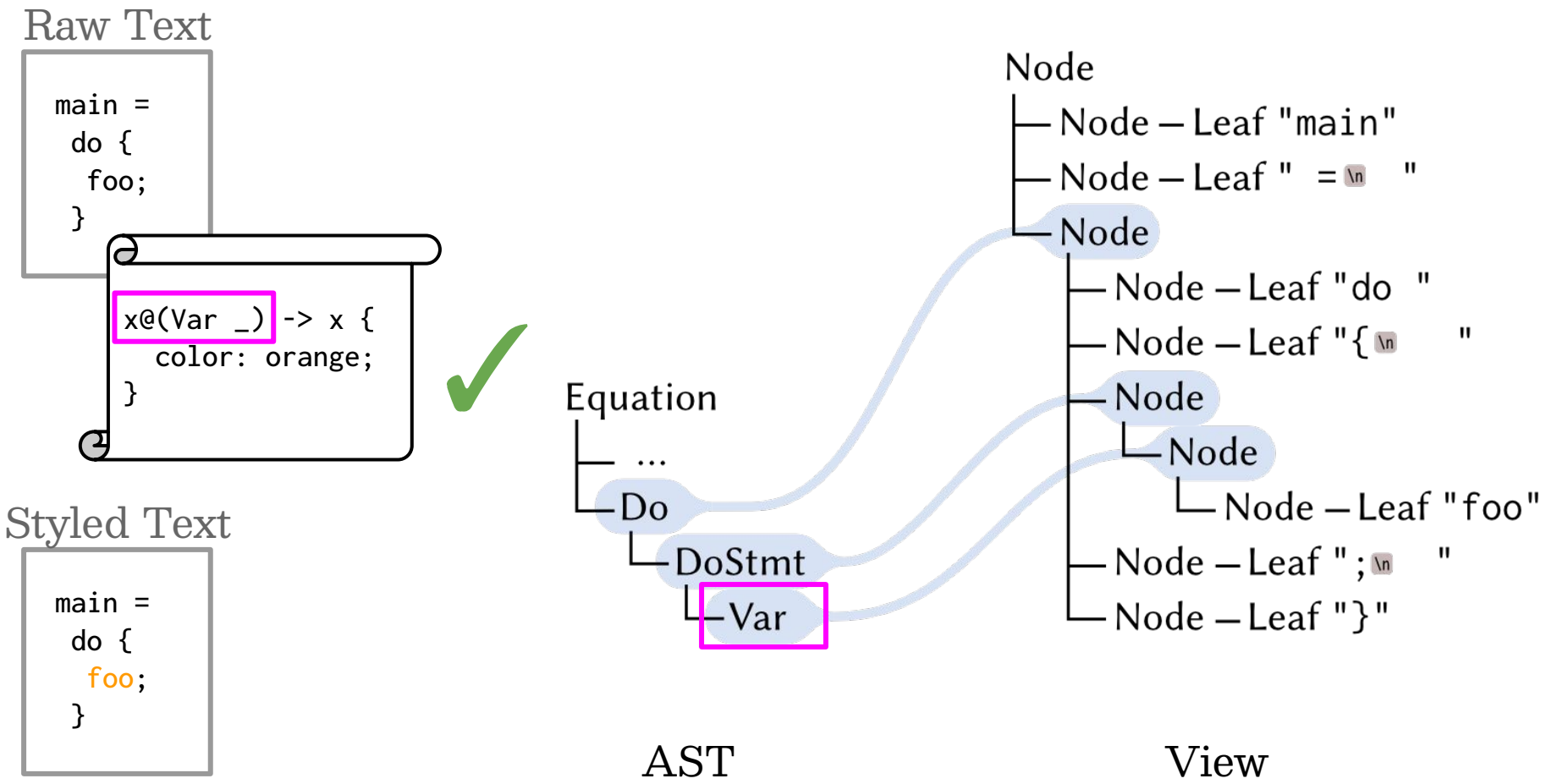




AST

View

Style Sheet Semantics



Style Sheet Semantics

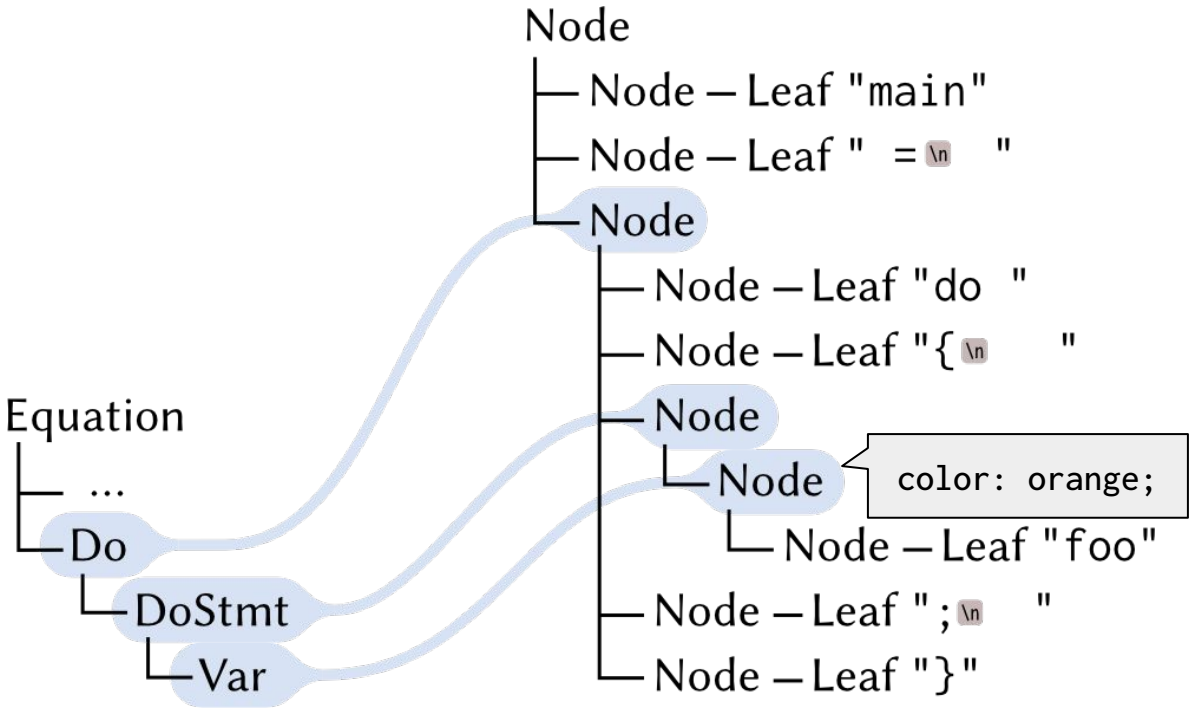
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

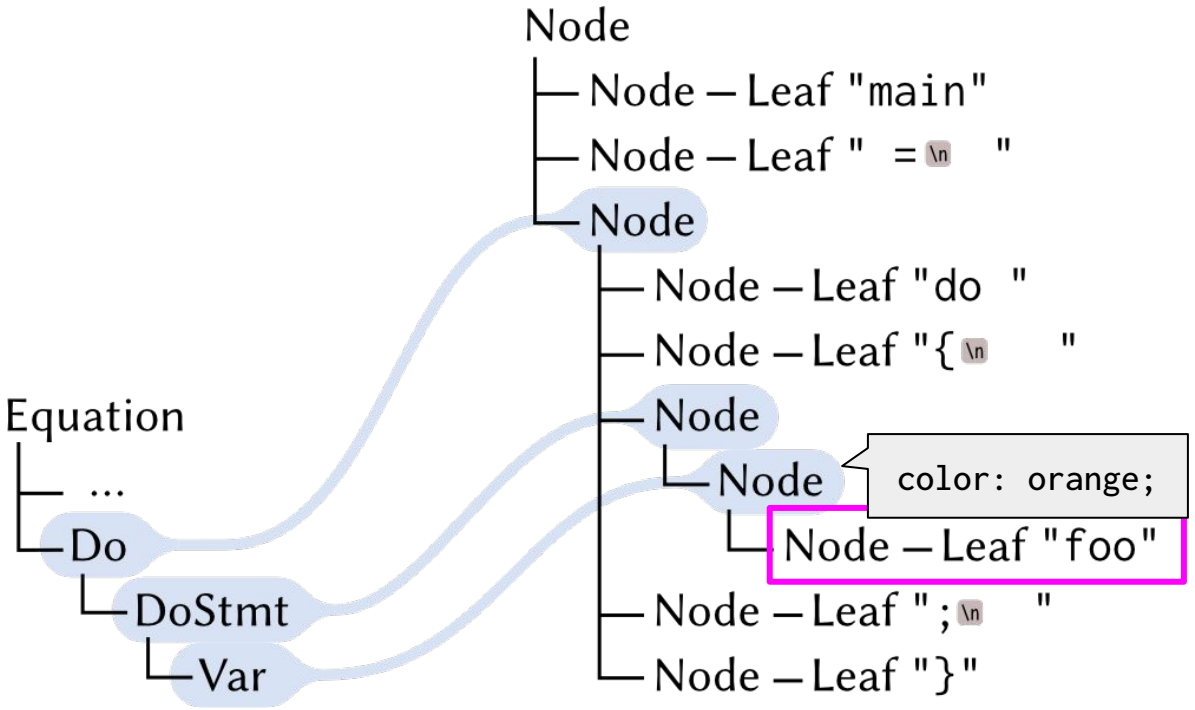
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

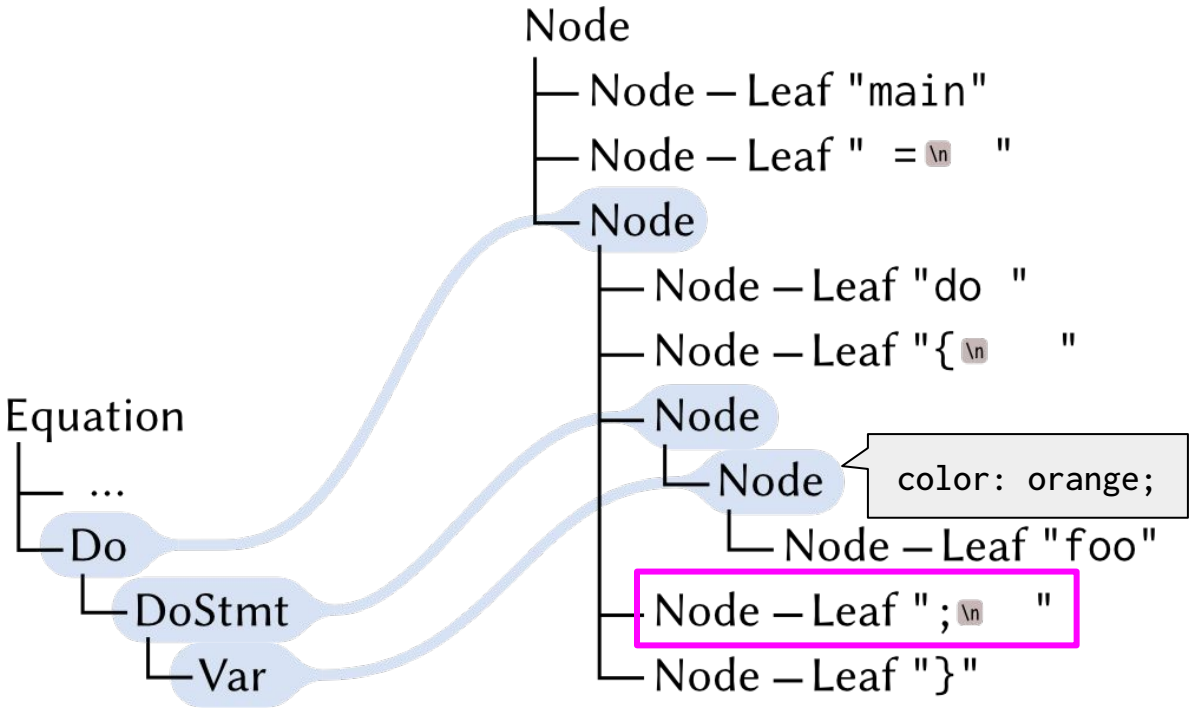
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

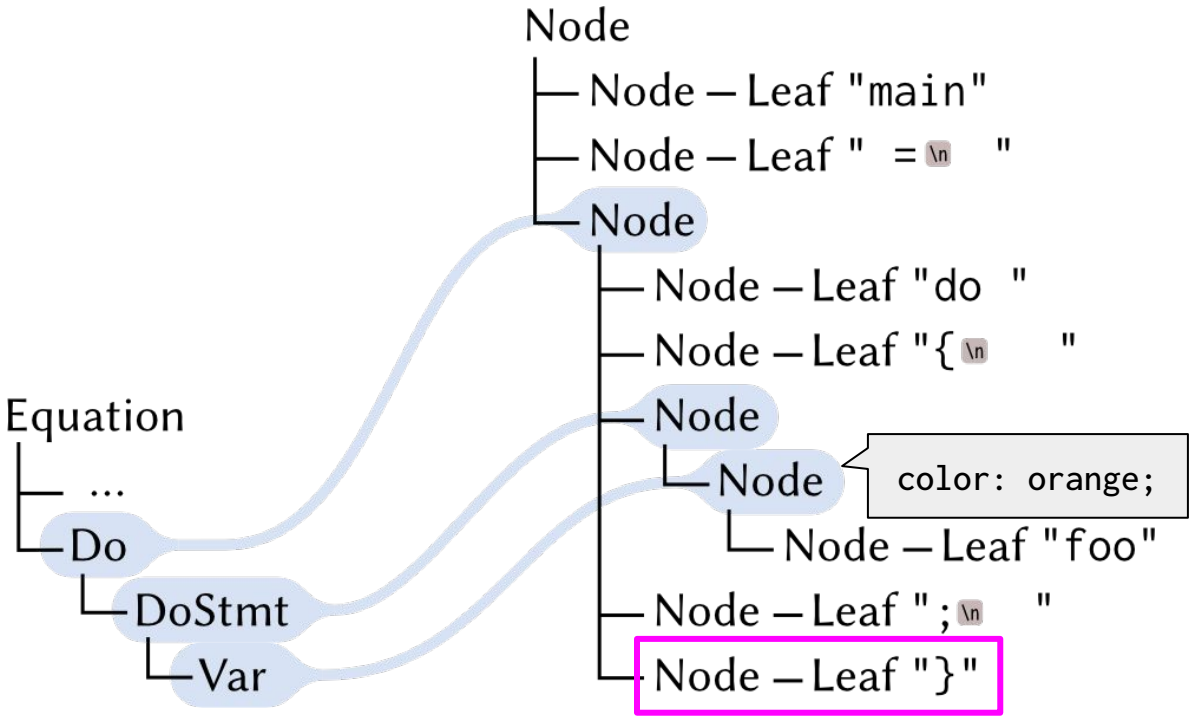
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

Style Sheet Semantics

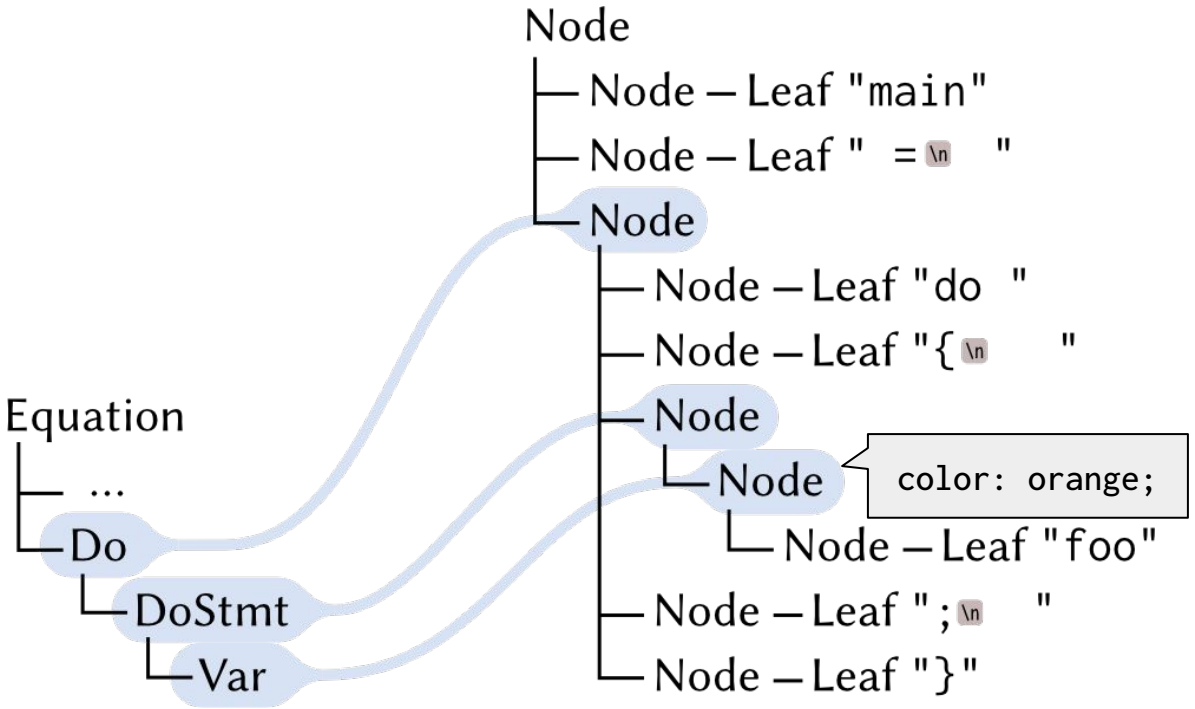
Raw Text

```
main =
 do {
  foo;
 }
```

```
x@(Var _) -> x {
  color: orange;
}
```

Styled Text

```
main =
 do {
  foo;
 }
```



AST

View

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;

}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;

}
```

… additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: block; /* implicit */
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: block; /* implicit */
}
```

… additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: inline-block;
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: inline-block;
}
```

... additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

Styled Text

```
main =
getContents
>>= print
  . length
  . filter (not . isPrefixOf "--")
  . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: inline-block;
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: inline-block;
}
```

... additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
```

Program text moved!

Styled Text

```
main =
getContents
>>= print
  . length
    . filter (not . isPrefixOf "--")
      . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: inline;
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: inline;
}
```

... additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf "--")
      . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: inline;
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: inline;
}
```

… additional rules

Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
```

Styled Text doesn’t show structure!

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf "--")
      . lines
```

```
x@(EBinop (_, Op op1, y@(EBinop (_, Op op2, _))))
    if isDirUnEq op1 op2 ->
x {
  border: 1px solid indigo; border-radius 3;
  padding: 2; margin: 2;
  background-color: lavender;
  display: s-block;
}
y {
  border: 1px solid orange; border-radius: 3;
  padding: 2; margin: 2;
  background-color: papayawhip;
  display: s-block;
}
```

... additional rules

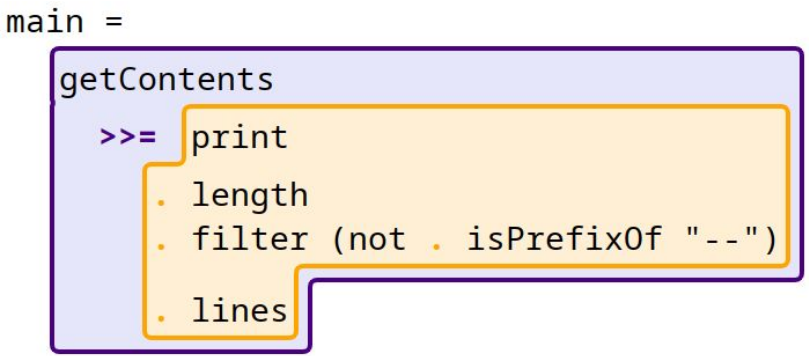
Raw Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf “--”)
      . lines
```

Styled Text

```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf "--")
      . lines
```

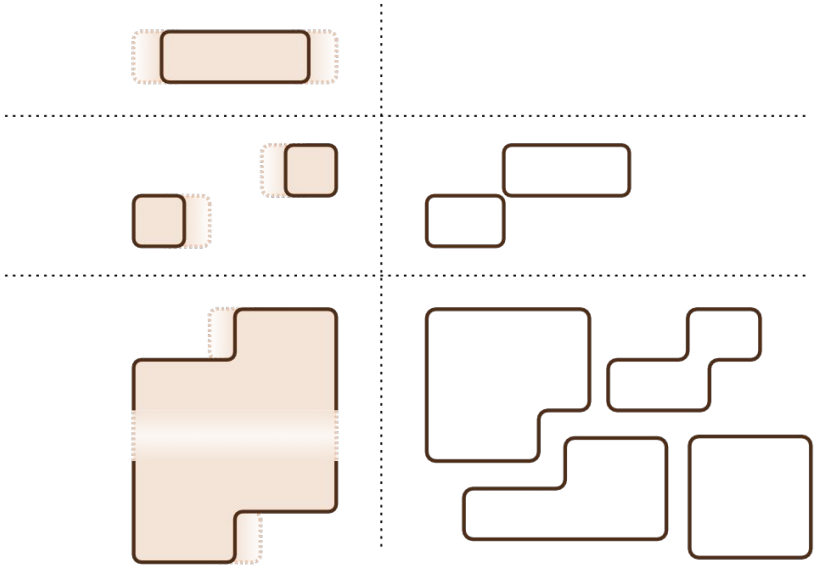
Layout Problem



S-Block Layout: *How?*

Idea:

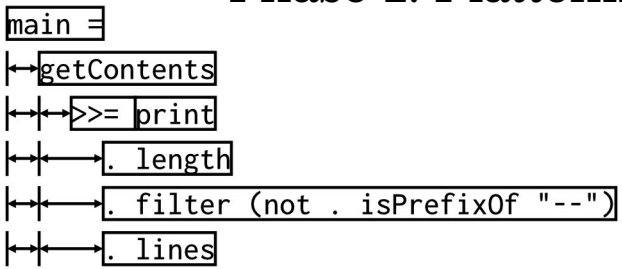
Surround each text span with a nestable 8-sided shape (S-Block)



```
main =
  getContents
    >>= print
      . length
      . filter (not . isPrefixOf "--")
      . lines
```

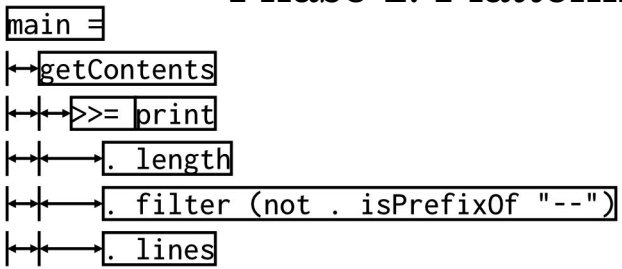
Layout Algorithm Overview/Phases

Phase 1: Flattening

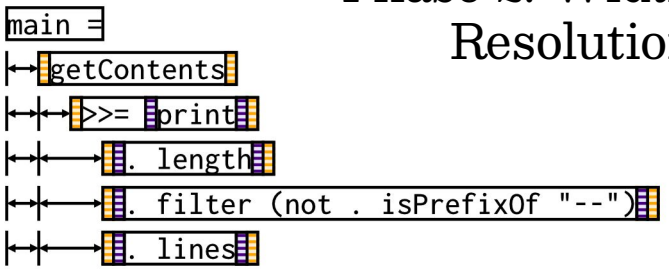


Layout Algorithm Overview/Phases

Phase 1: Flattening

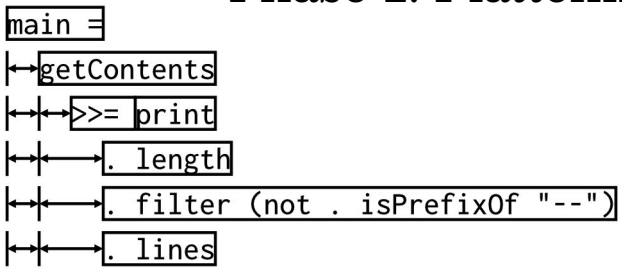


Phase 2: Width Resolution

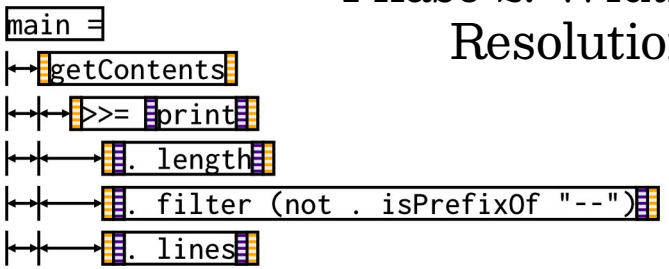


Layout Algorithm Overview/Phases

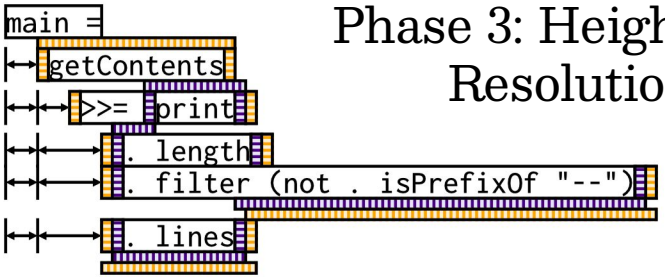
Phase 1: Flattening



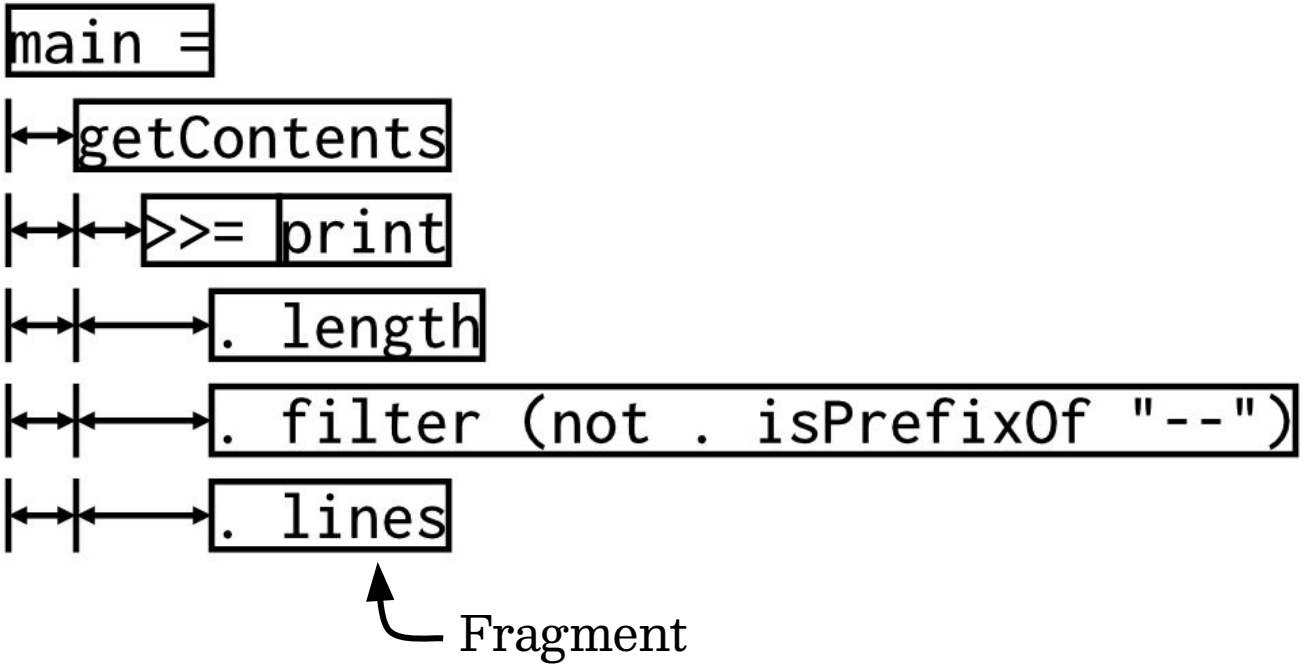
Phase 2: Width Resolution



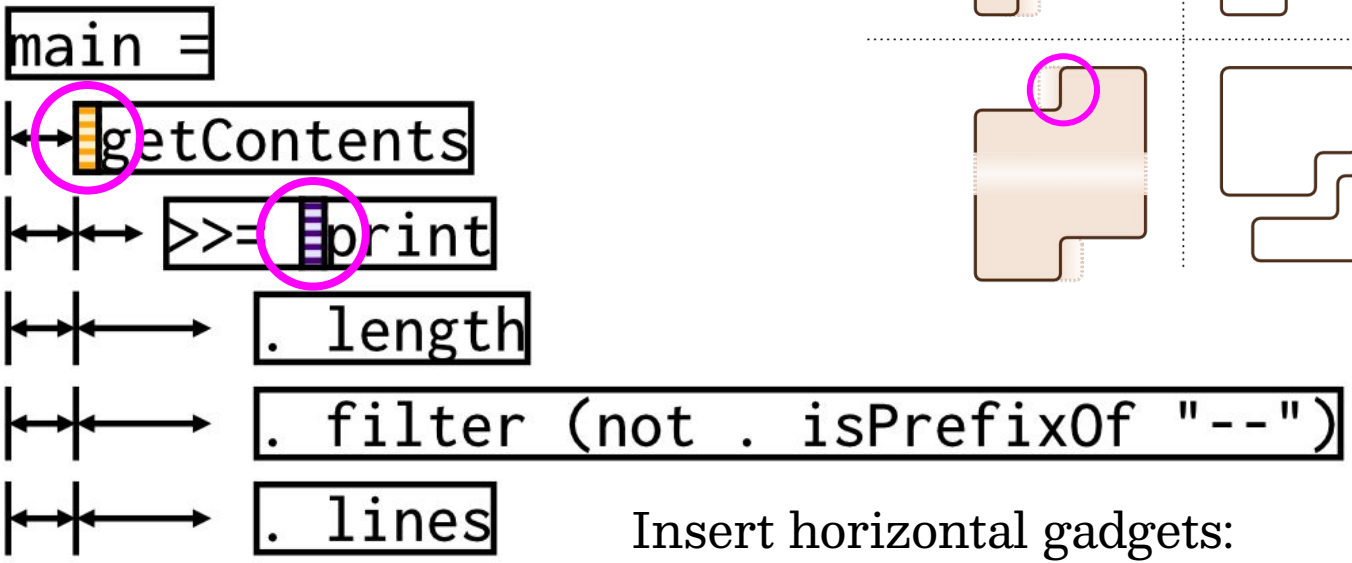
Phase 3: Height Resolution

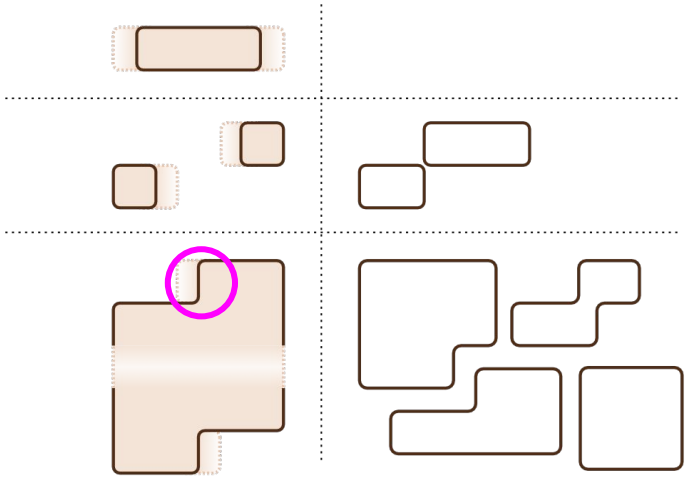


Phase 1: Flattening



Phase 2: Width Resolution

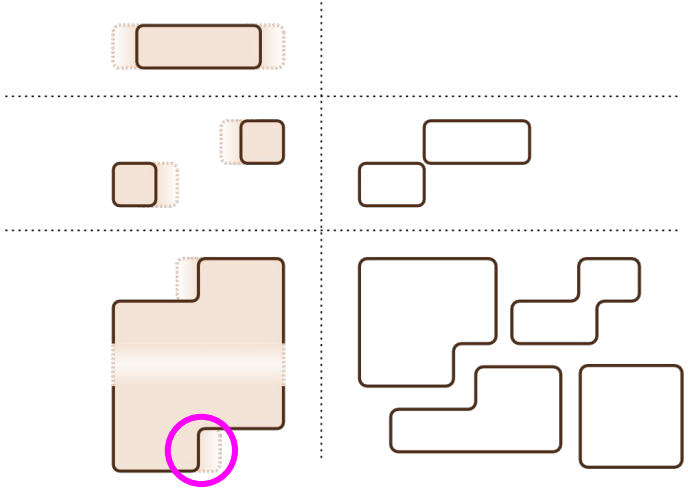


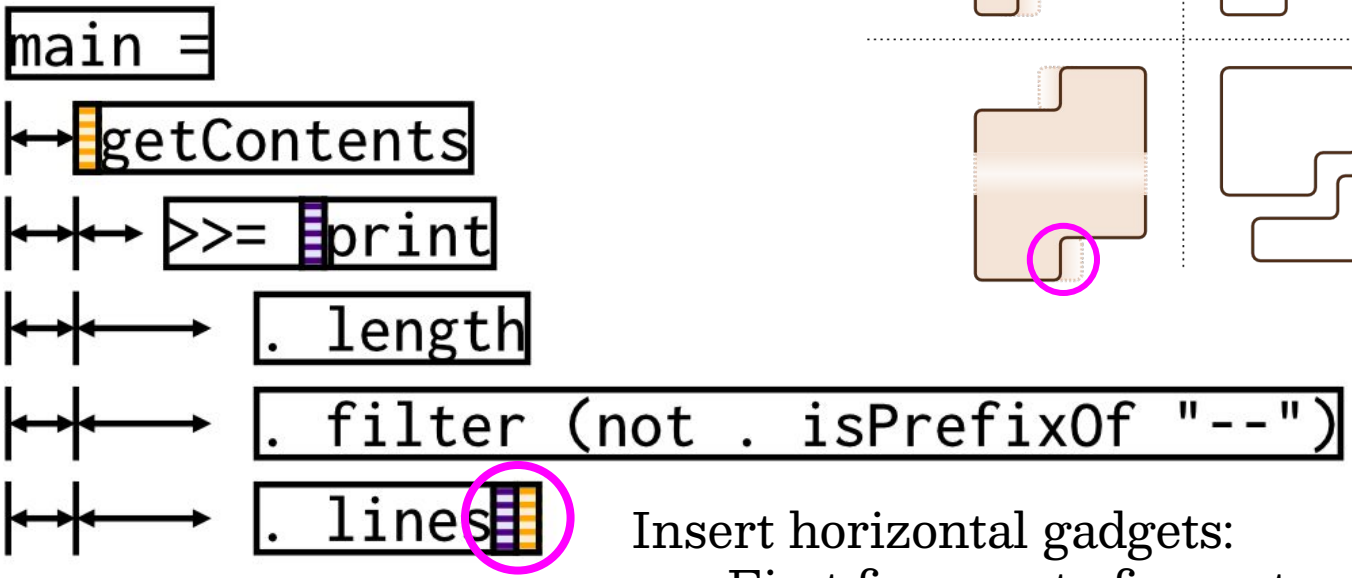


Insert horizontal gadgets:

- First fragment of a syntax tree node

Phase 2: Width Resolution

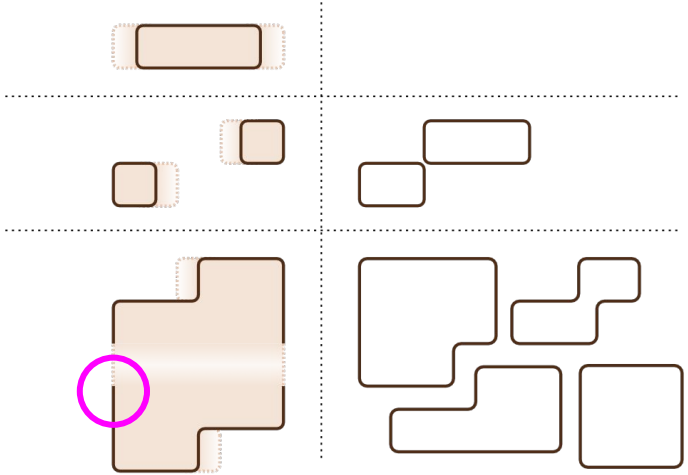


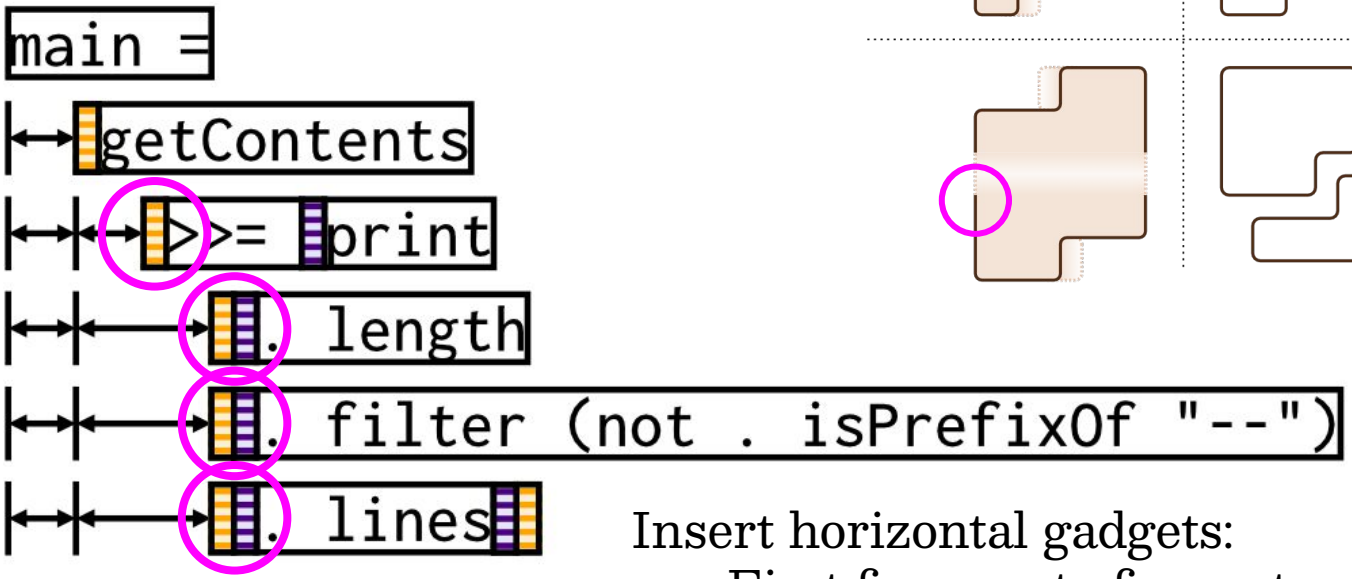


Insert horizontal gadgets:

- First fragment of a syntax tree node
- Last fragment of a syntax tree node

Phase 2: Width Resolution

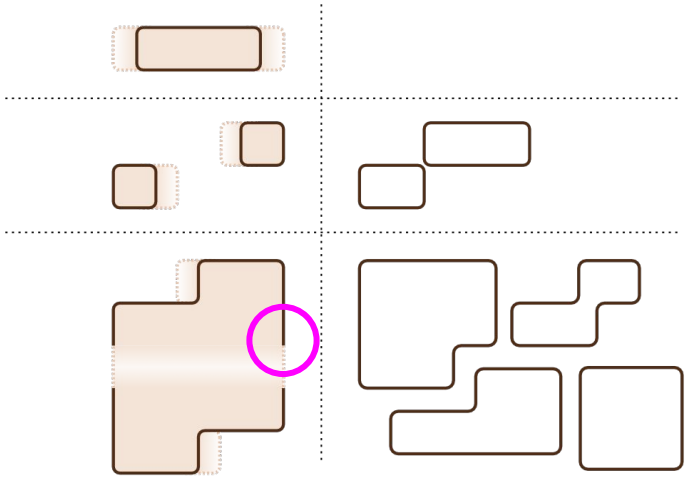


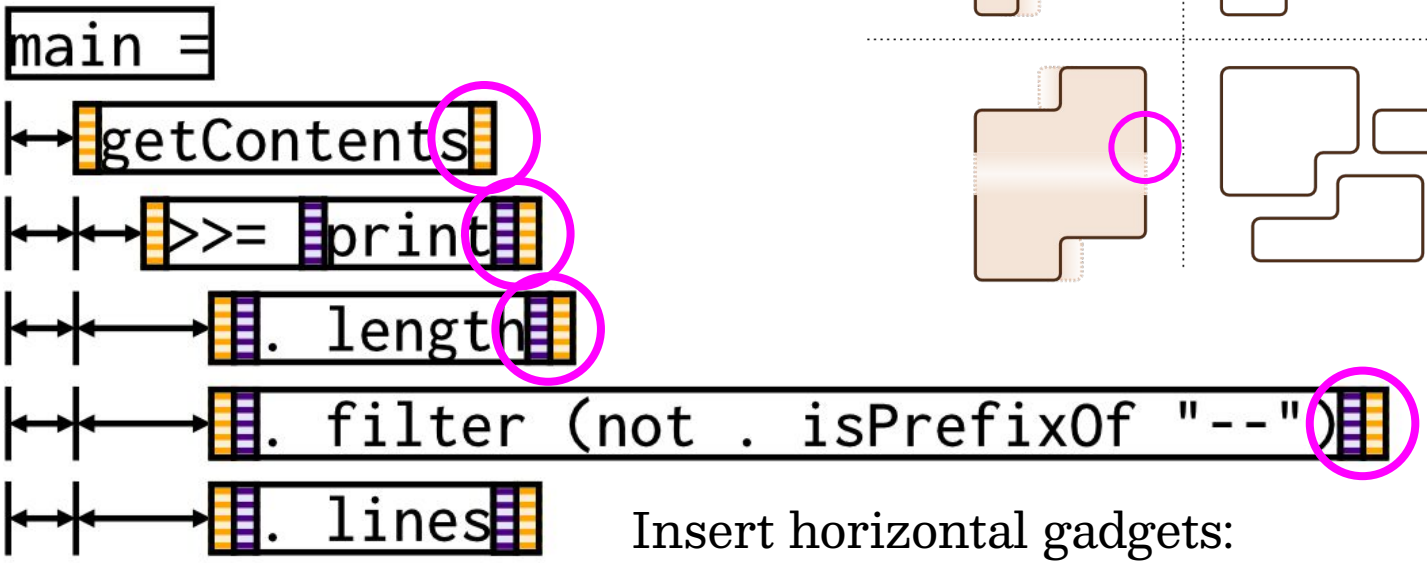


Insert horizontal gadgets:

- First fragment of a syntax tree node
- Last fragment of a syntax tree node
- First (non ws) fragment on a line

Phase 2: Width Resolution

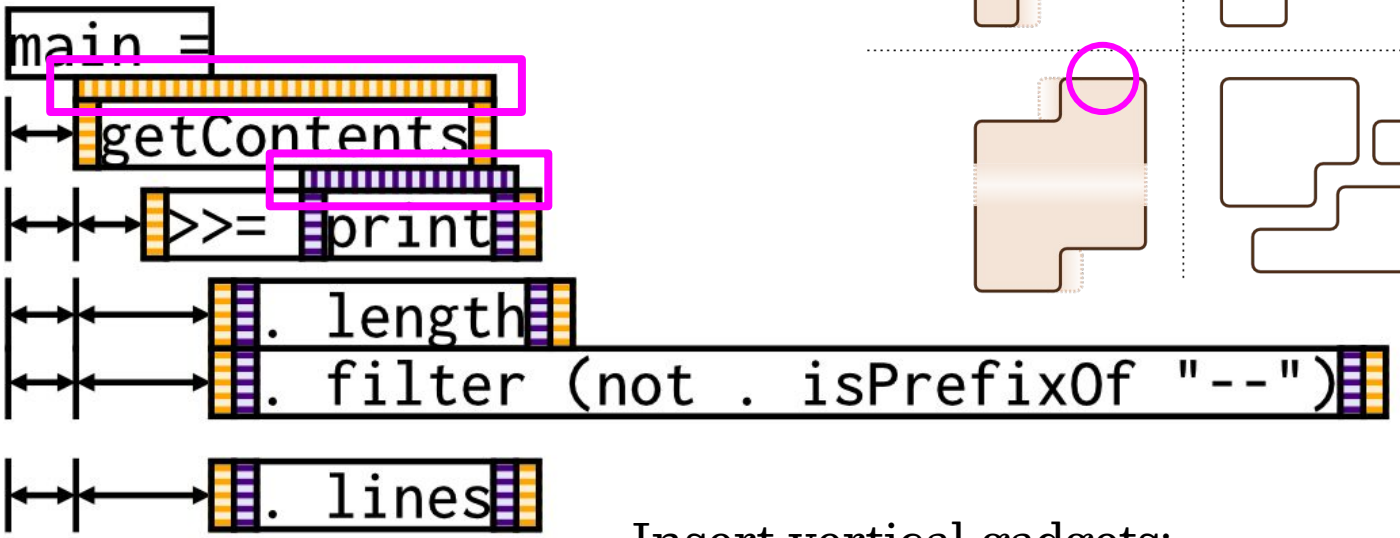


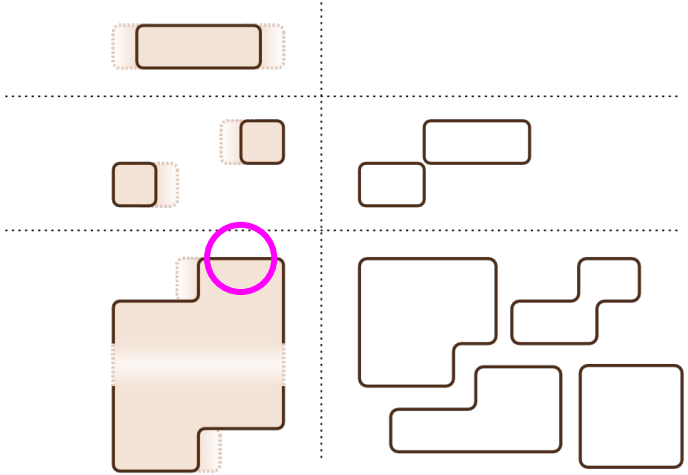


Insert horizontal gadgets:

- First fragment of a syntax tree node
- Last fragment of a syntax tree node
- First (non ws) fragment on a line
- Last fragment on a line

Phase 3: Height Resolution

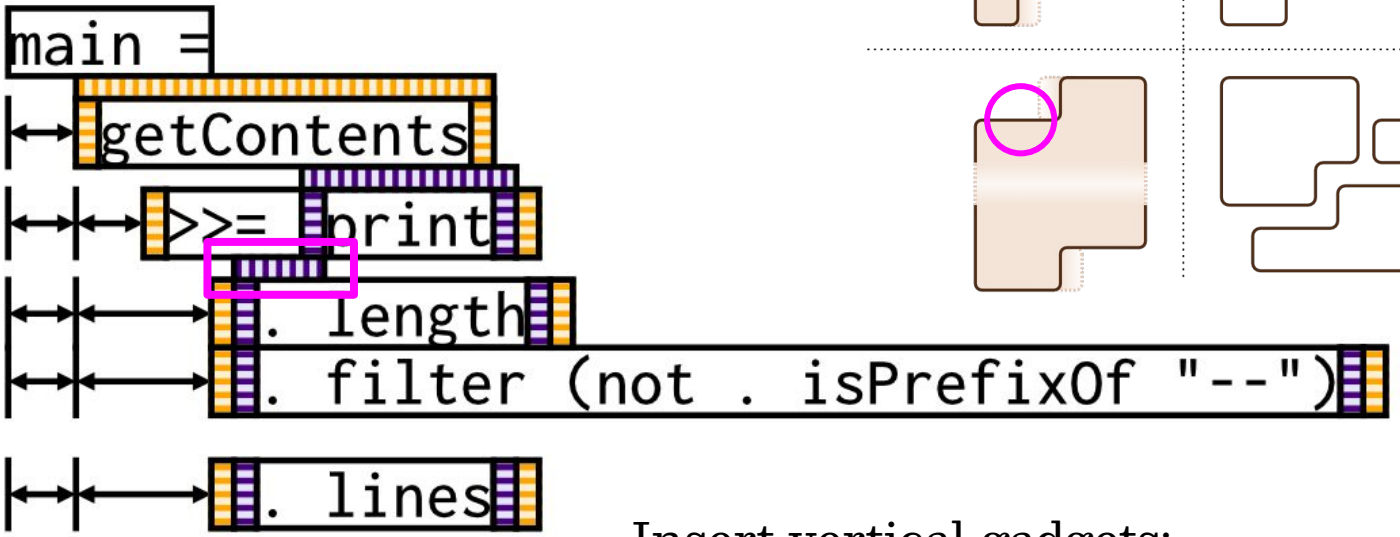


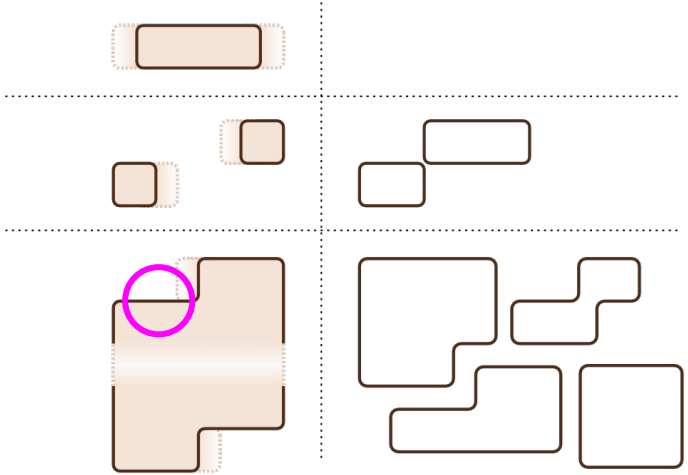


Insert vertical gadgets:

- Above the first line

Phase 3: Height Resolution

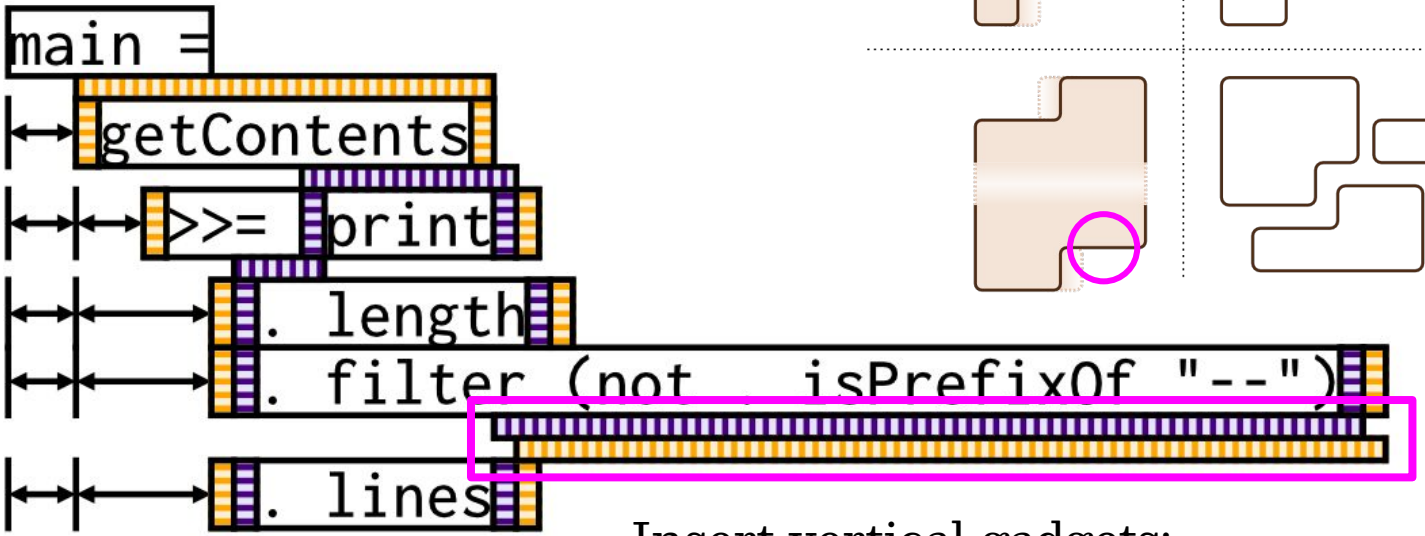


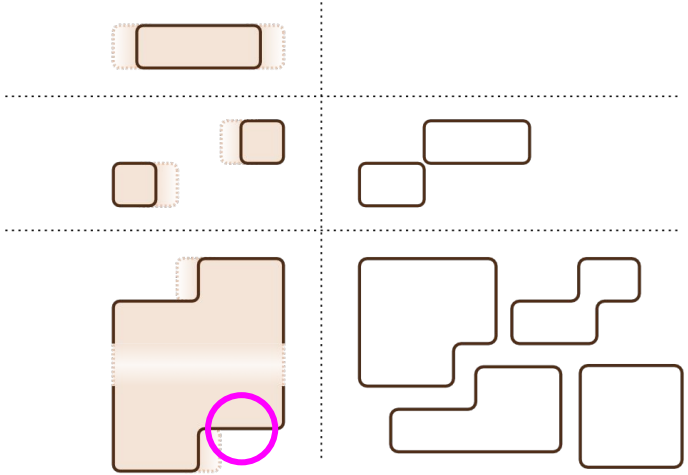


Insert vertical gadgets:

- Above the first line
- Above the second line

Phase 3: Height Resolution

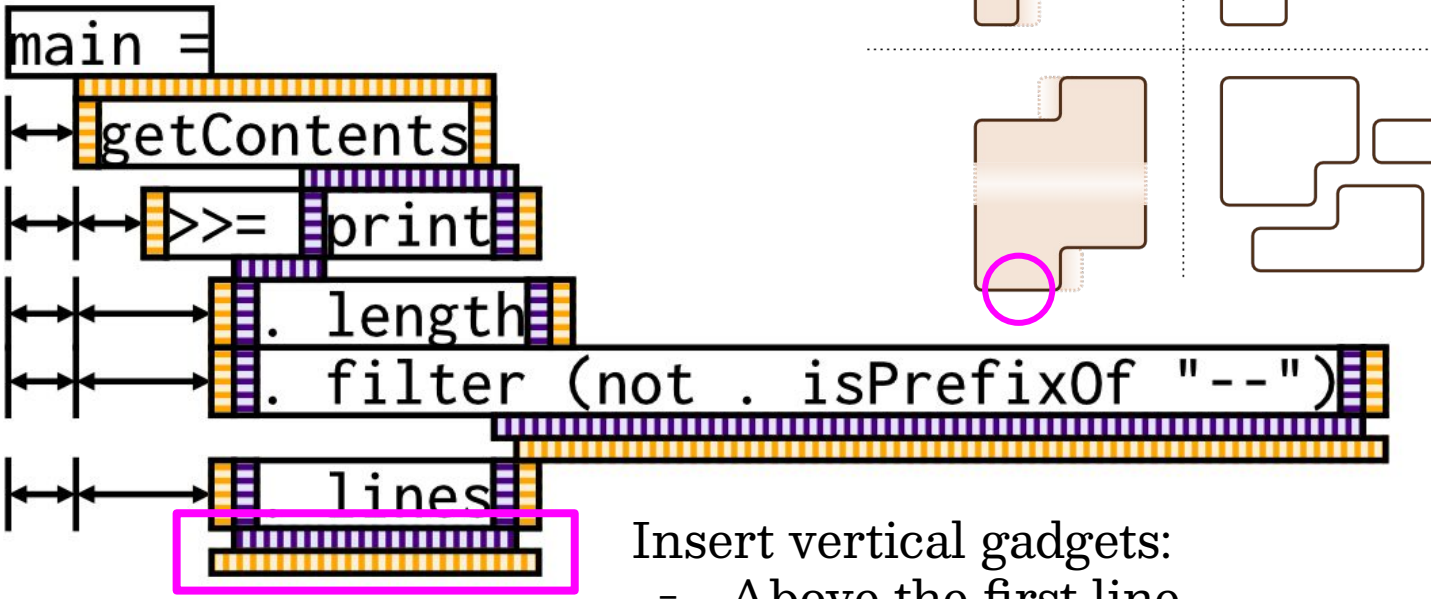




Insert vertical gadgets:

- Above the first line
- Above the second line
- Below the second-to-last line

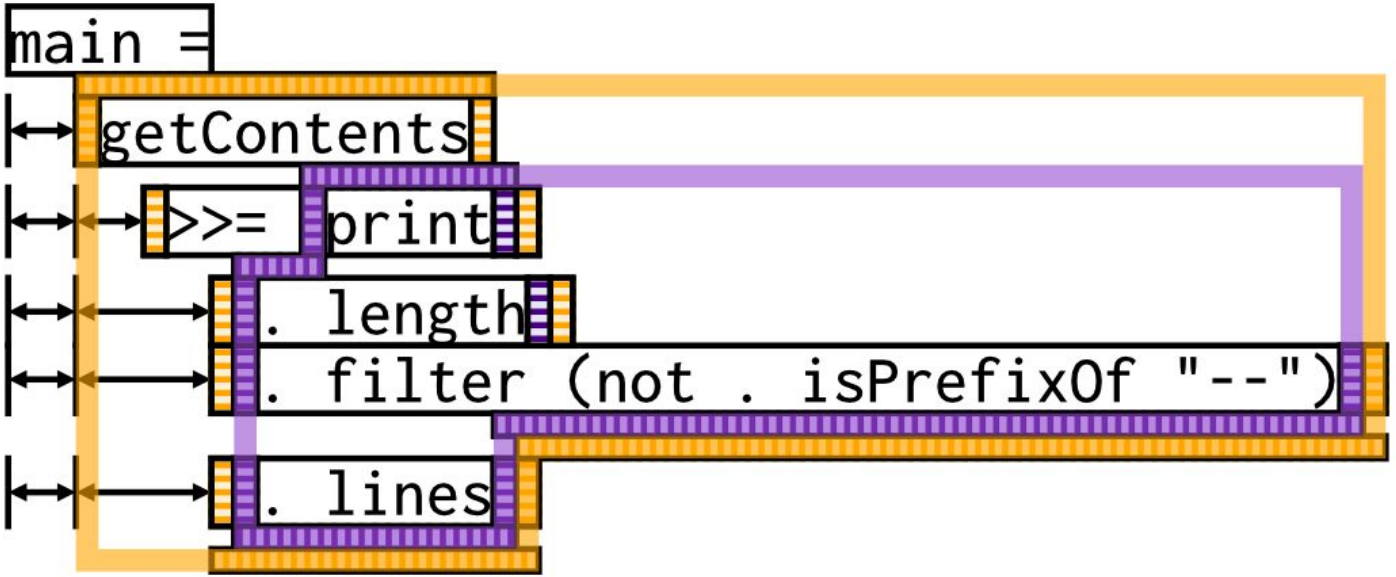
Phase 3: Height Resolution



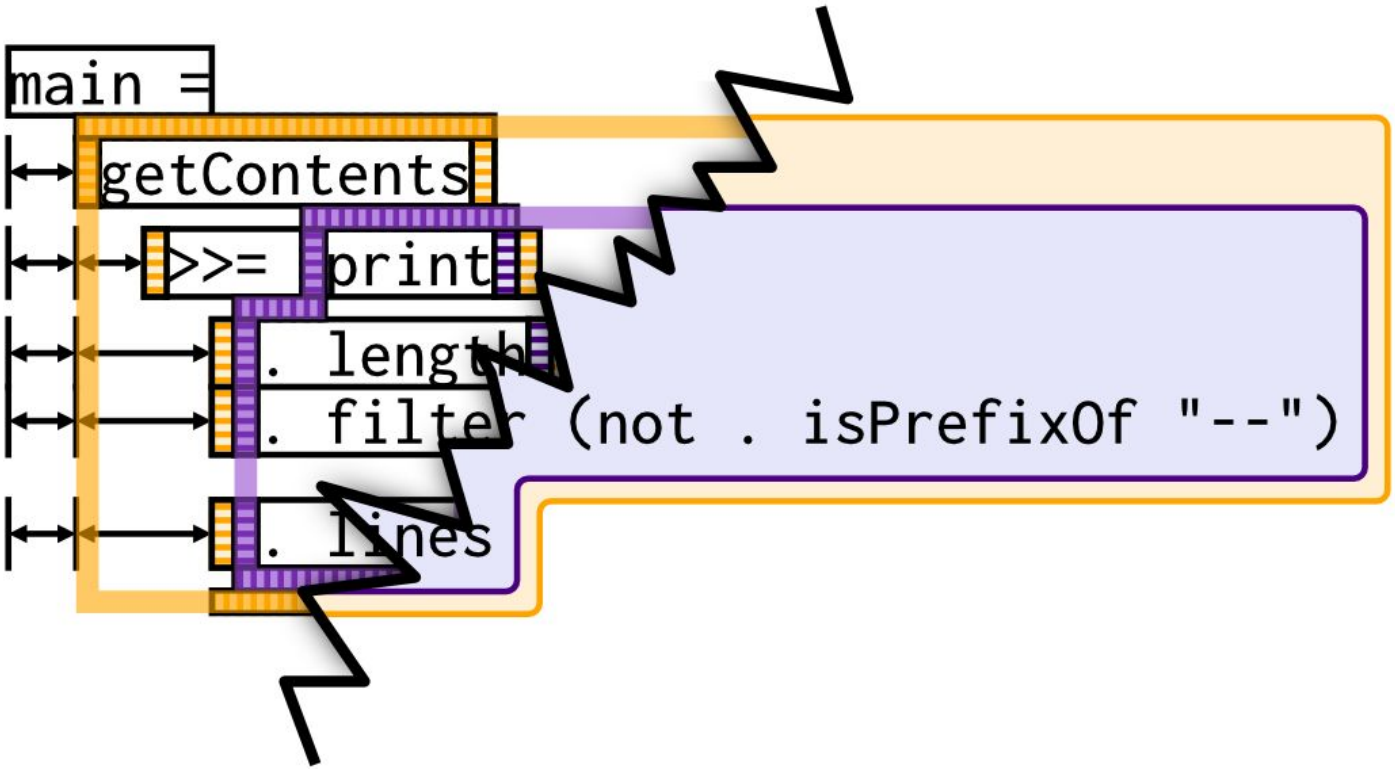
Insert vertical gadgets:

- Above the first line
- Above the second line
- Below the second-to-last line
- Below the last line

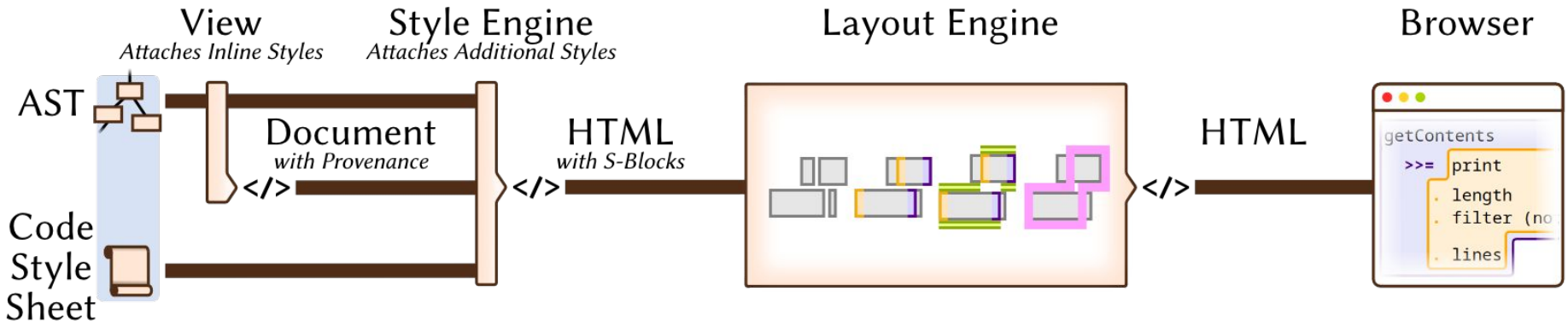
Phase 3: Height Resolution



Phase 3: Height Resolution

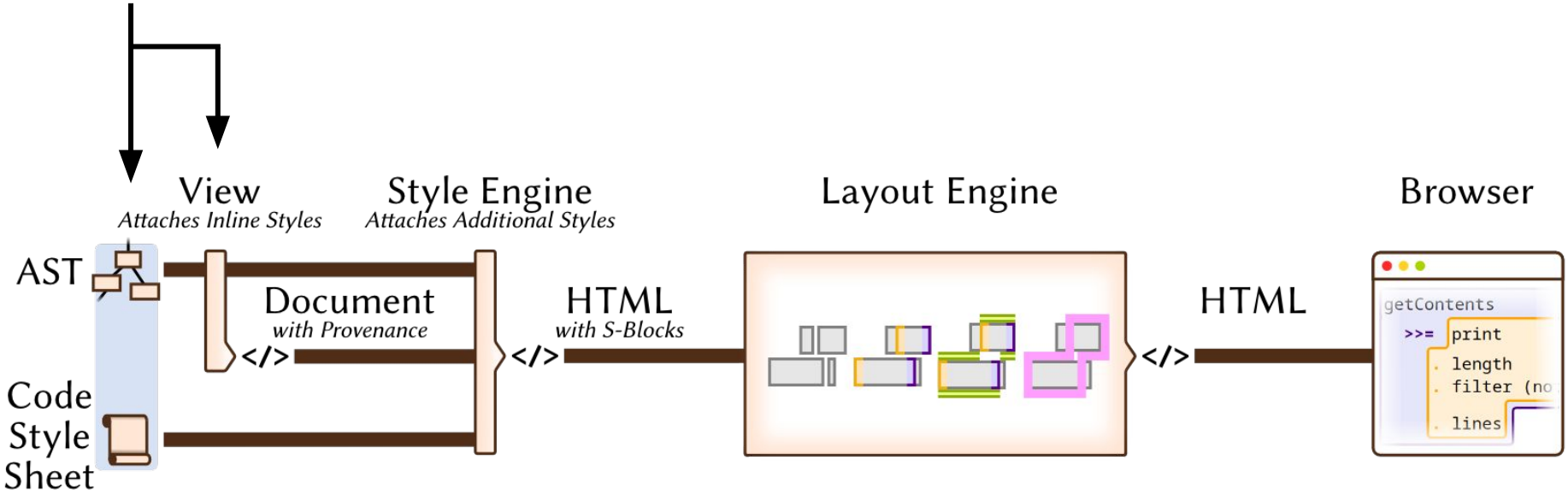


Implementation (5.7k LOC of Haskell)



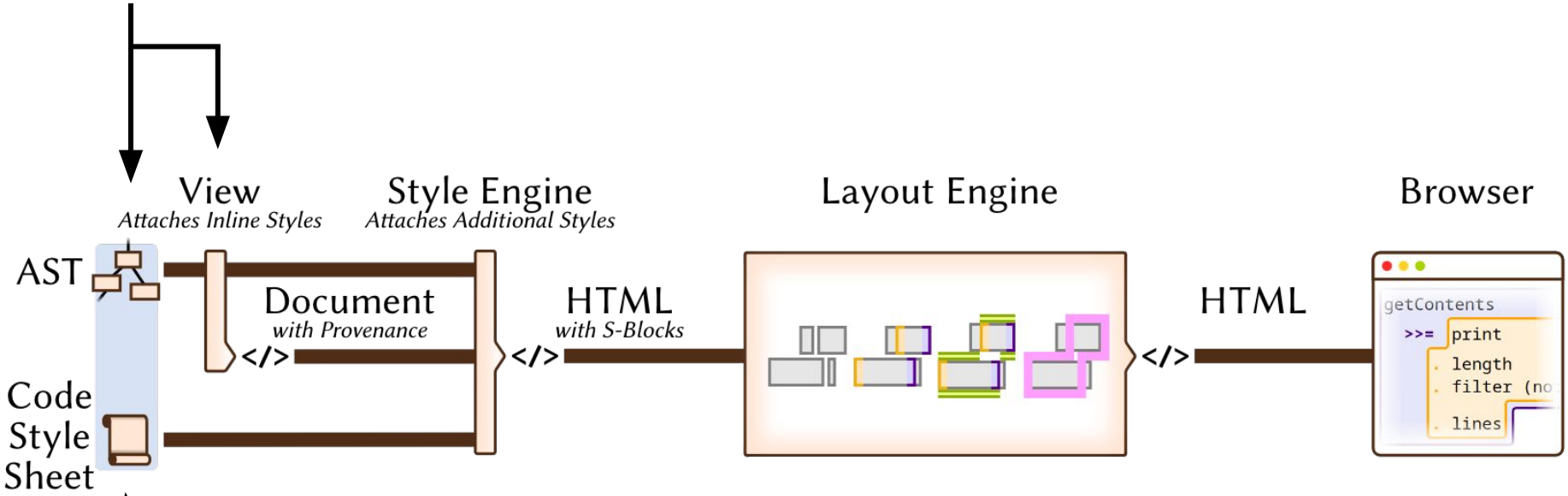
Implementation (5.7k LOC of Haskell)

A small Haskell subset interpreter, typechecker



Implementation (5.7k LOC of Haskell)

A small Haskell subset interpreter, typechecker



Includes 1.2k LOC of Style Sheets

Skeleton Code

```
x@.comment ->
x {
  font-family: "Noto Serif", serif;
}
x@Equation (Left (Located _ (PVar (Located _
             (y@Ident "main"))), _), _, _) ->
x y {
  color: gray;
}
    :
    :
    :
```

```
-- myZip takes two lists and returns
-- a single list where corresponding elements
-- have been grouped into a tuple.
--
-- If one list is longer than the other, remaining
-- elements are discarded.
myZip :: [a] -> [b] -> [(a, b)]
myZip = {- your code here -} undefined

main = do
  let input1 = [1, 2, 3]
      input2 = [True, False, True]
  print $ myZip input1 input2
```

Result

Skeleton Code

```
x@.comment ->
x {
  font-family: "Noto Serif", serif;
}
x@Equation (Left (Located _ (PVar (Located _
             (y@Ident "main")))), _), _, _) ->
x y {
  color: gray;
}
```

⋮

Result

```
-- myZip takes two lists and returns
-- a single list where corresponding elements
-- have been grouped into a tuple.
--
-- If one list is longer than the other, remaining
-- elements are discarded.
myZip :: [a] -> [b] -> [(a, b)]
myZip = {- your code here -} undefined

main = do
  let input1 = [1, 2, 3]
      input2 = [True, False, True]
  print $ myZip input1 input2
```

“Semantic” Highlighting

```
x@(Ident @RnPhase (IdentUsage _ binder) _)
     if pred binder ->
x {
  color: purple;
}
  where pred binder = binder `mod` 5 == 3
```

⋮

Result

```
gcd a b =
  case (a, b) of
    (0, b) -> b
    (a, 0) -> a
    (a, b) -> gcd b (mod a b)

main = print (gcd 270 192)
```

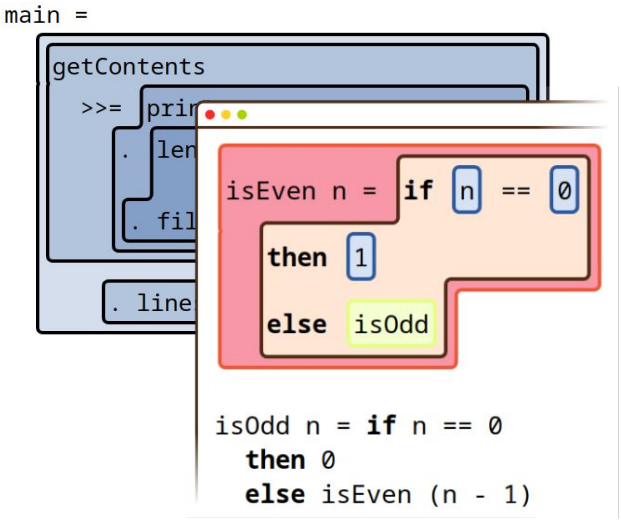
Heat Map

```
e@(_::Exp_ EvalPhase)
(EvalExpInfo { temp })
if temp > 0.6 && temp ⩽ 0.8 ->
e {
color: "red";
}
```

Result

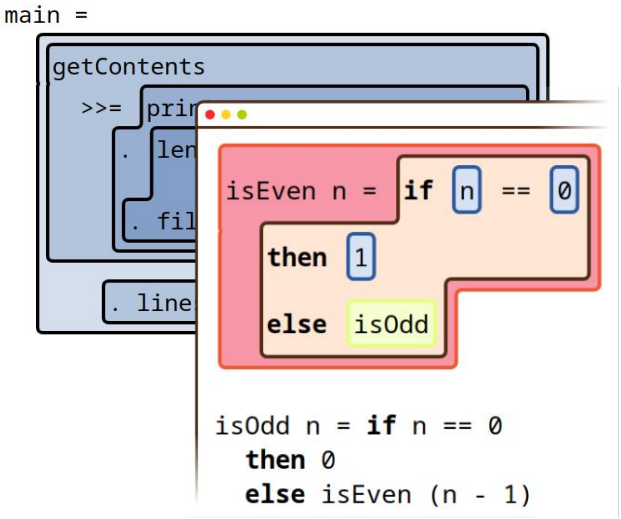
```
r = 100

inCircle x y r =
  x*x + y*y < r*r

calcPi iters = do
  (circ, sqr) <- go 0 0 iters
  pure (div (4 * circ) sqr)

go circ sqr i =
  if i == 0 then pure (circ, sqr)
  else do
    x <- randomRIO (-r, r)
    y <- randomRIO (-r, r)
    if inCircle x y r
      then go (circ+1) (sqr+1) (i-1)
      else go circ (sqr+1) (i-1)

main = do
  pi <- calcPi 1000
  print pi
```

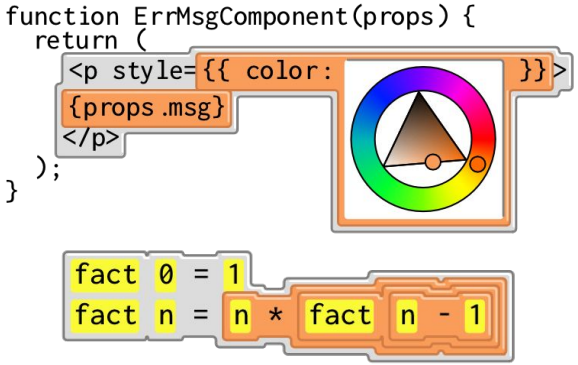
Applications & Future Work



Read-Only Displays

Applications & Future Work

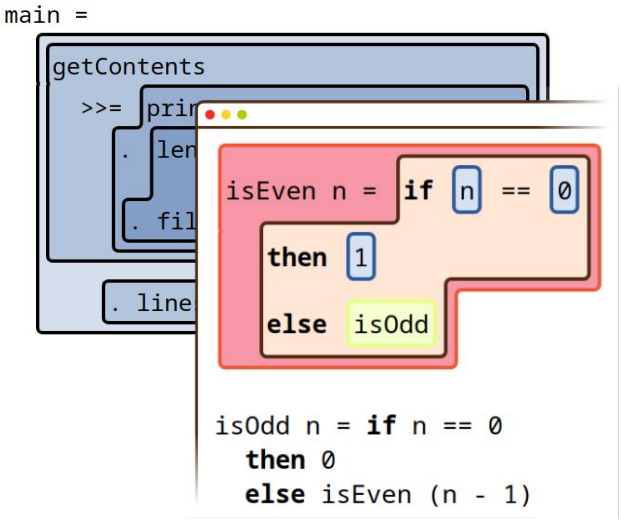




Read-Only Displays → Improved Algorithms*

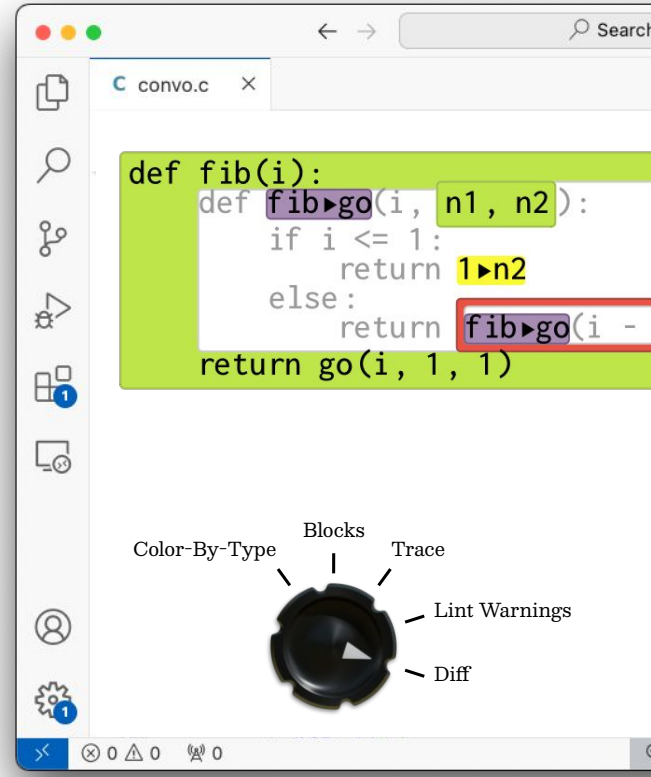
*See our more recent work, Ragged Blocks: Rendering Structured Text with Style, *UIST 2025*

Applications & Future Work




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

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



Read-Only Displays → Improved Algorithms* → Code Style Sheets in the Editor

*See our more recent work, Ragged Blocks: Rendering Structured Text with Style, *UIST 2025*