FliPpr

A Prettier Invertible Printing System

Kazutaka Matsuda (Univ. of Tokyo)
Meng Wang (Chalmers Univ. of Tech.)

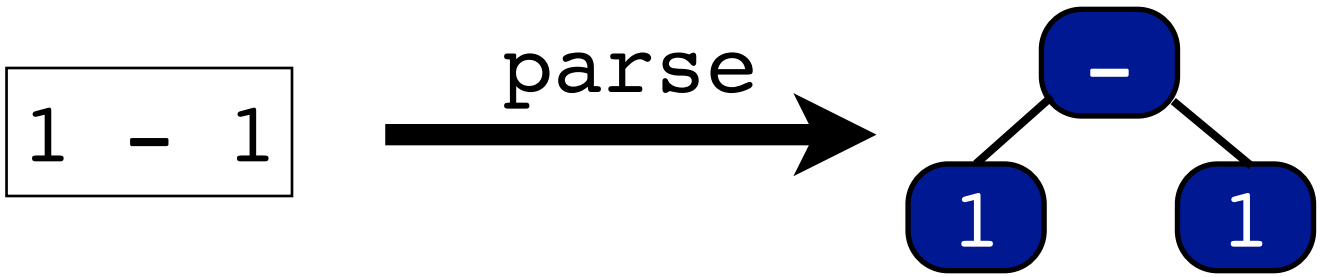
March 19th, 2013 @ESOP

FliPpr in 1 Slide

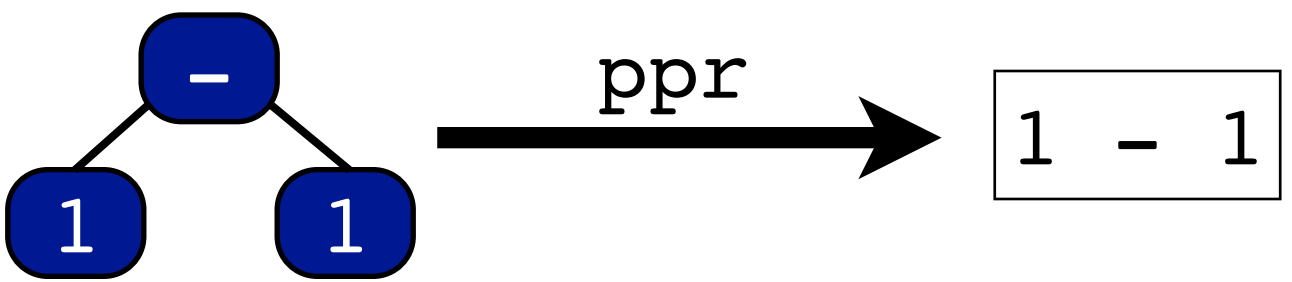
- A Prettier Invertible Printing System
  - takes a **pretty-printer**
    - written with Wadler's pretty-printing combinators [Wadler 03]
  - returns a **parser**
    - based on grammar-based inversion [M.+10]

Background

- To implement a programming lang, we often write ...
  - a parser

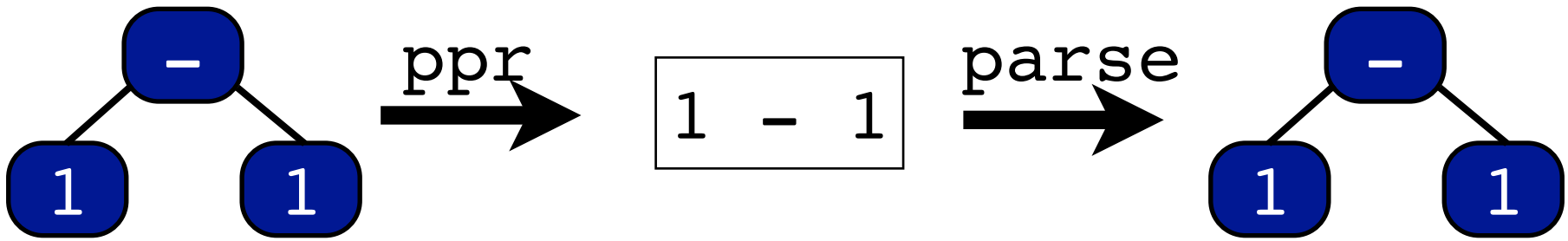


  - a pretty-printer



Desired Property

- A pretty-printed string must be correctly parsed




```
parse (ppr ast) = ast
```

Problem

- Separately-writing parser/ppr is ...
  - **tedious**
    - We have to write and maintain two programs
  - **error-prone**
    - A pretty-printed string may not be correctly parsed

```
parse (ppr ast) ≠ ast
```

Problem

- Separately-writing parser/ppr is ...
  - **tedious**
    - We have to write and maintain two programs
  - **error-prone**

```
*Main> "\n" :: Int

<interactive>:93:1:
    Couldn't match expected type `Int' with actual type `[Char]'
    In the expression: "" :: Int
    In an equation for `it': it = "" :: Int
```

In GHC 7.4.1

parse (ppr ast) ≠ ast

Our Goal

- Derive a **parser** from a **pretty-printer** by program inversion [Gries 81,...]

```
parse (ppr ast) = ast
```

inverse

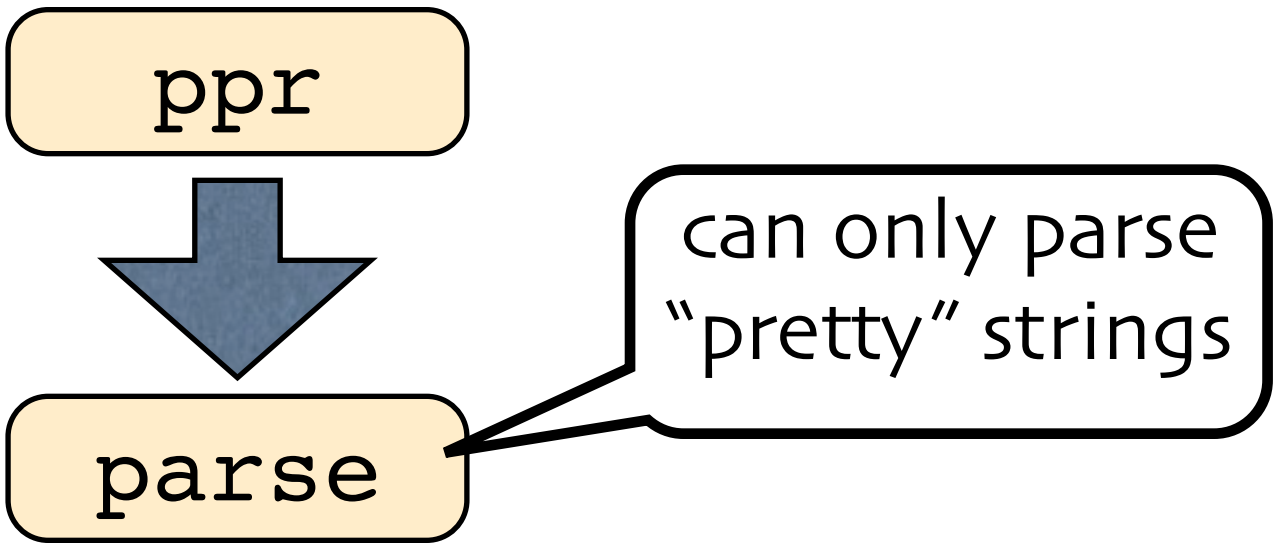
Why Pretty-Printing?

- Pretty-printing is important
  - It is the only way for a compiler to communicate to its users
    - Prettier means more productive
- Pretty-printing is more creative
  - More control on layouting is needed
    - indentation, spacing, putting parens, ...

We **do** want to write pretty-printers!

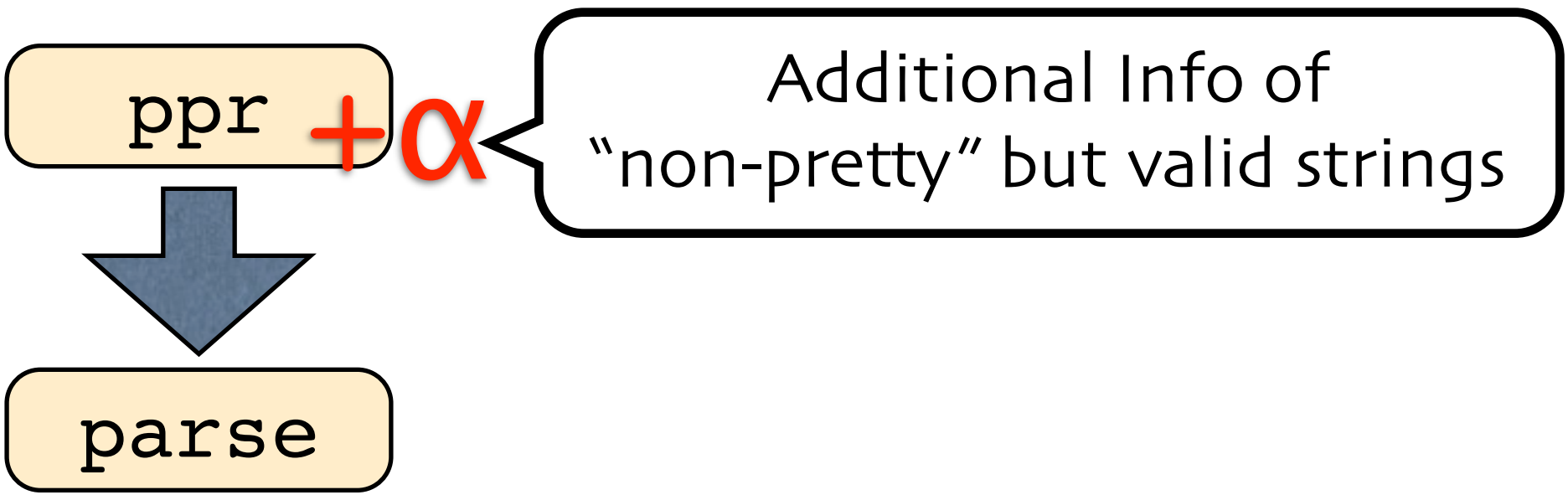
Issue

- Naively-derived parsers are useless



Additional Info

- Required to derive useful parsers



Our Proposal: FliPpr

- A Prettier Invertible Printing System
  - takes a pretty-printing program
    - written with Wadler's pretty-printing combinators [Wadler 03]
    - together with additional info for parsing
  - returns a parser as a CFG with actions
    - based on grammar-based inversion [M.+10]

Advantages

- Users define pretty-printers (fine-grained control)
- FliPpr can reuse existing efficient algorithms and implementations
  - For pretty-printers
    - [Wadler03, Swisstra&Chitil09, Kiselyov13,...]
  - For parsers
    - GLR, Early, [Frost+08], [Might+11], ...

Agenda

- Input of FliPpr
  - Wadler's Pretty-Printing Combinators
  - Additional Information for Parsing
- Quick Overview of FliPpr
- Related Work
- Conclusion

Wadler's Combinators

- `text :: String → Doc`
- `(<>) :: Doc → Doc → Doc`
- `line :: Doc`
- `nest :: Int → Doc → Doc`
- `group :: Doc → Doc`

`Doc`: A smart datatype for pretty-printing

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

```
ppr One = text "1"
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

```
ppr One = text "1"
ppr (Sub x y) =
  ppr x <> nest 2 (
    line <> text "-" <> text " " <> ppr y)
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

```
ppr One = text "1"
ppr (Sub x y) =
  ppr x <> nest 2 (
    line <> text "-" <> text " " <> ppr y)
ppr (Div x y) =
  ppr x <> nest 2 (
    line <> text "/" <> text " " <> ppr y) )
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

prec level 6

prec level 7

```
ppr One = text "1"
ppr (Sub x y) =
  ppr x <> nest 2 (
    line <> text "-" <> text " " <> ppr y)
ppr (Div x y) =
  ppr x <> nest 2 (
    line <> text "/" <> text " " <> ppr y) )
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

prec level 6

prec level 7

```
ppr One = text "1"
ppr (Sub x y) =
  ppr x <> nest 2 (
    line <> text "-" <> text " " <> ppr y)
ppr (Div x y) =
  ppr x <> nest 2 (
    line <> text "/" <> text " " <> ppr y) )
```

```
ifParens b x = if b then parens x else x
parens x = text "(" <> x <> text ")"
```

A Pretty Printer

```
data AST = One
         | Sub AST AST
         | Div AST AST
```

prec level 6

prec level 7

```
pprMain x = ppr 5 x
ppr i One = text "1"
ppr i (Sub x y) = ifParens (i>=6) (
  ppr 5 x <> nest 2 (
    line <> text "-" <> text " " <> ppr 6 y)
ppr i (Div x y) = ifParens (i>=7) (
  ppr 6 x <> nest 2 (
    line <> text "/" <> text " " <> ppr 7 y) )
```

```
ifParens b x = if b then parens x else x
parens x = text "(" <> x <> text ")"
```

Smart Layouting

```
pprMain x = ppr 5 x
ppr i One = text "1"
ppr i (Sub x y) = ifParens (i>=6) (
  ppr 5 x <> nest 2 (
    line <> text "-" <> text " " <> ppr 6 y) )
ppr i (Div x y) = ifParens (i>=7) (
  ppr 6 x <> nest 2 (
    line <> text "/" <> text " " <> ppr 7 y) )
```

```
pprMain (Sub One One)
```

```
1
  - 1
```

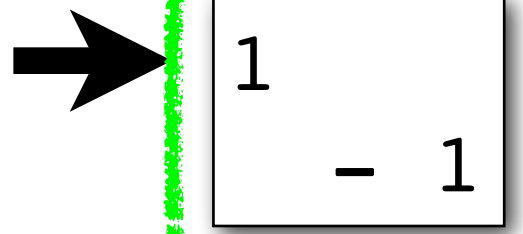
Smart Layouting

```
pprMain x = ppr 5 x
ppr i One = text "1"
ppr i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
    line <> text "-" <> text " " <> ppr 6 y)))
ppr i (Div x y) = ifParens (i>=7) (group (
  ppr 6 x <> nest 2 (
    line <> text "/" <> text " " <> ppr 7 y)))
```

`group` `x` chooses between

- `lines` as (indented) newlines
- `lines` as (single) spaces

```
pprMain (Sub One One)
```

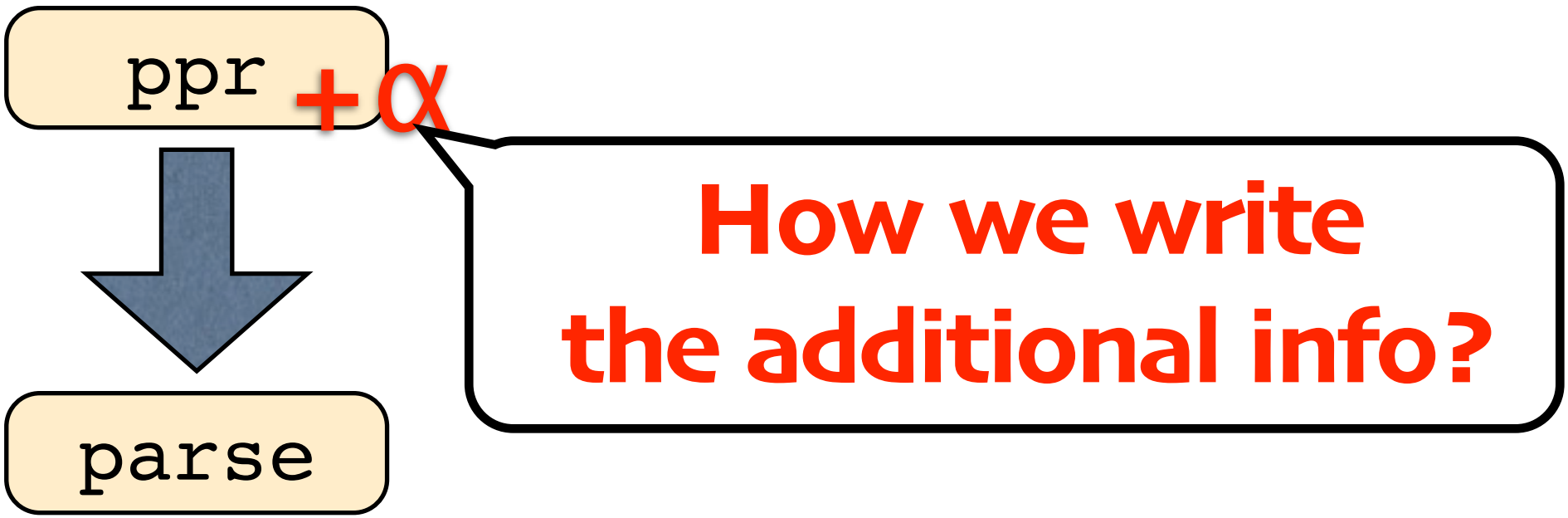


```
1
  - 1
```

```
1 - 1
```

Additional Info

- Additional info is required to parse "non-pretty" strings



Ideas

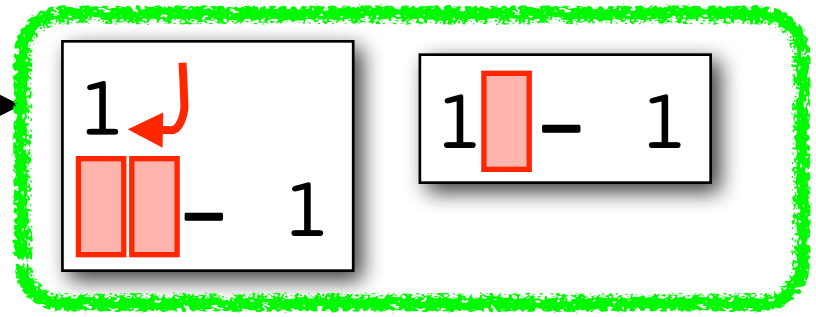
- Reinterpretation of `line`
- Biased-choice operator <+ for additional-information

Observation

- Many ways to interpret `lines`
  - `nest` inserts indentation after line
  - `group` can replace a `line` with a space

```
ppr i (Sub x y) = ifParens (i>=6) (group (
   ppr 5 x <> nest 2 (
    line <> text "-" <> text " " <> ppr 6 y)))
```

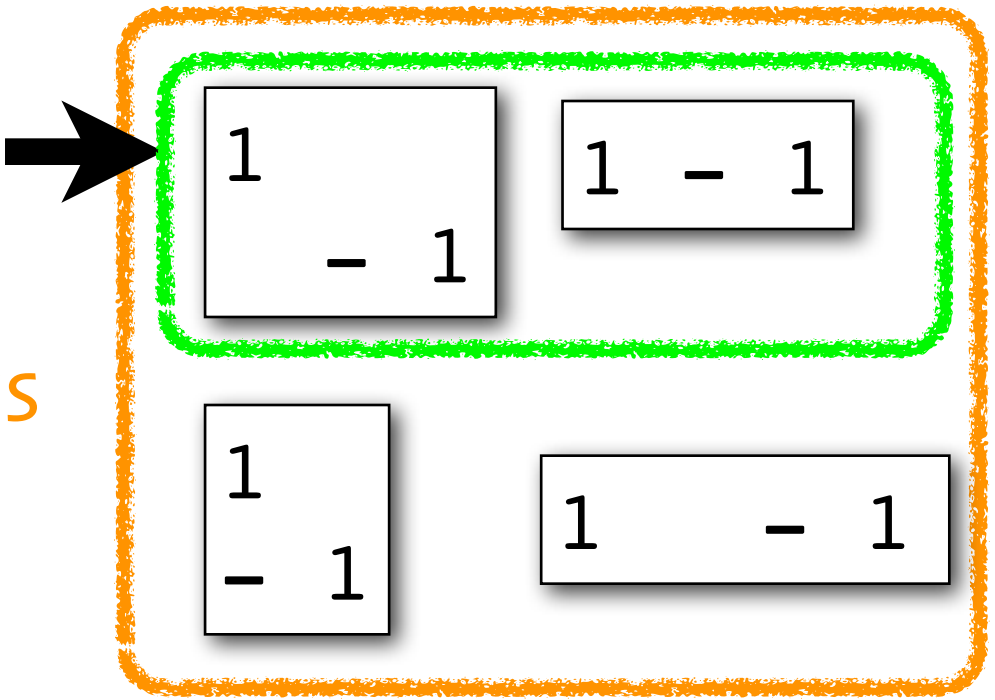
```
pprMain (Sub One One)
```



Reinterpret `lines`

- `lines` are interpreted as whitespaces in parsing

```
pprMain (Sub One One)
```



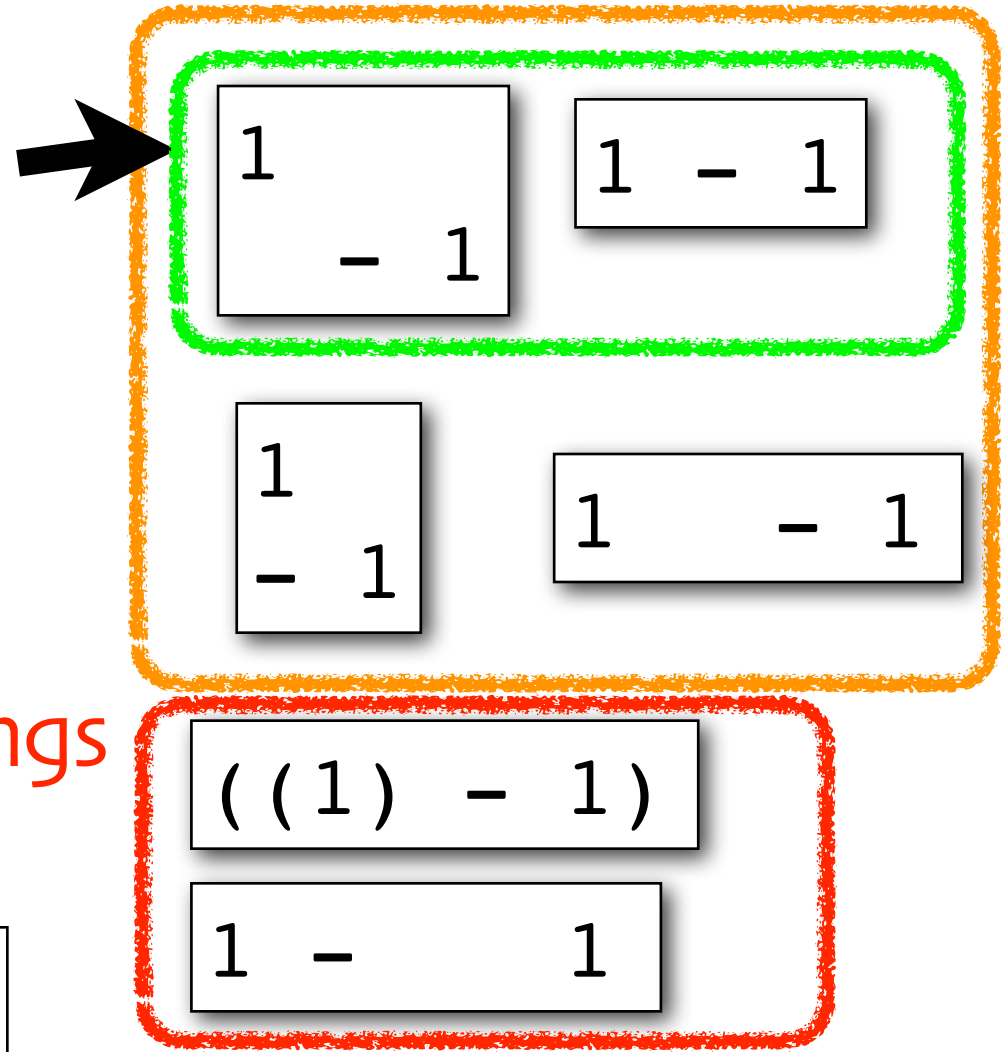
The pretty-printer knows these non-pretty strings

⇓

A derived parser can parse these strings

Still Not Enough

```
pprMain (Sub One One)
```



Uncovered non-pretty strings we want to parse

```
ppr i (Sub x y) =
    … text "-" <> text " "
      <> ppr 6 y …
```

Biased Choice: <+

- <+ for additional info
  - `x <+ y` equals to `x` in pretty-printing
    - No need to change pretty-printing system
  - `x <+ y` also conveys the info of `y`

```
ppr p = pretty <+ nonpretty
```

`ppr` knows both `pretty` and `nonpretty` are related to `p`

Original Pretty-Printer

```
pprMain x = ppr 5 x

ppr i One = text "1"
ppr i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
   line <> text "-" <> text " " <> ppr 6 y)))
…
```

Original Pretty-Printer

```
pprMain x = ppr 5 x

ppr i One = text "1"
ppr i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
   line <> text "-" <> text " " <> ppr 6 y)))
…
```

```
manyParens x = x <+ parens (manyParens x)
space = (text " " <+ text "\n") <> nil
nil   = text "" <+ space
```

Modified Pretty-Printer

Extra Parens

```
pprMain x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i One = text "1"
aux i (Sub x y) = ifParens (i>=6) (group (
   ppr 5 x <> nest 2 (
    line <> text "-" <> space     <> ppr 6 y)))
…
```

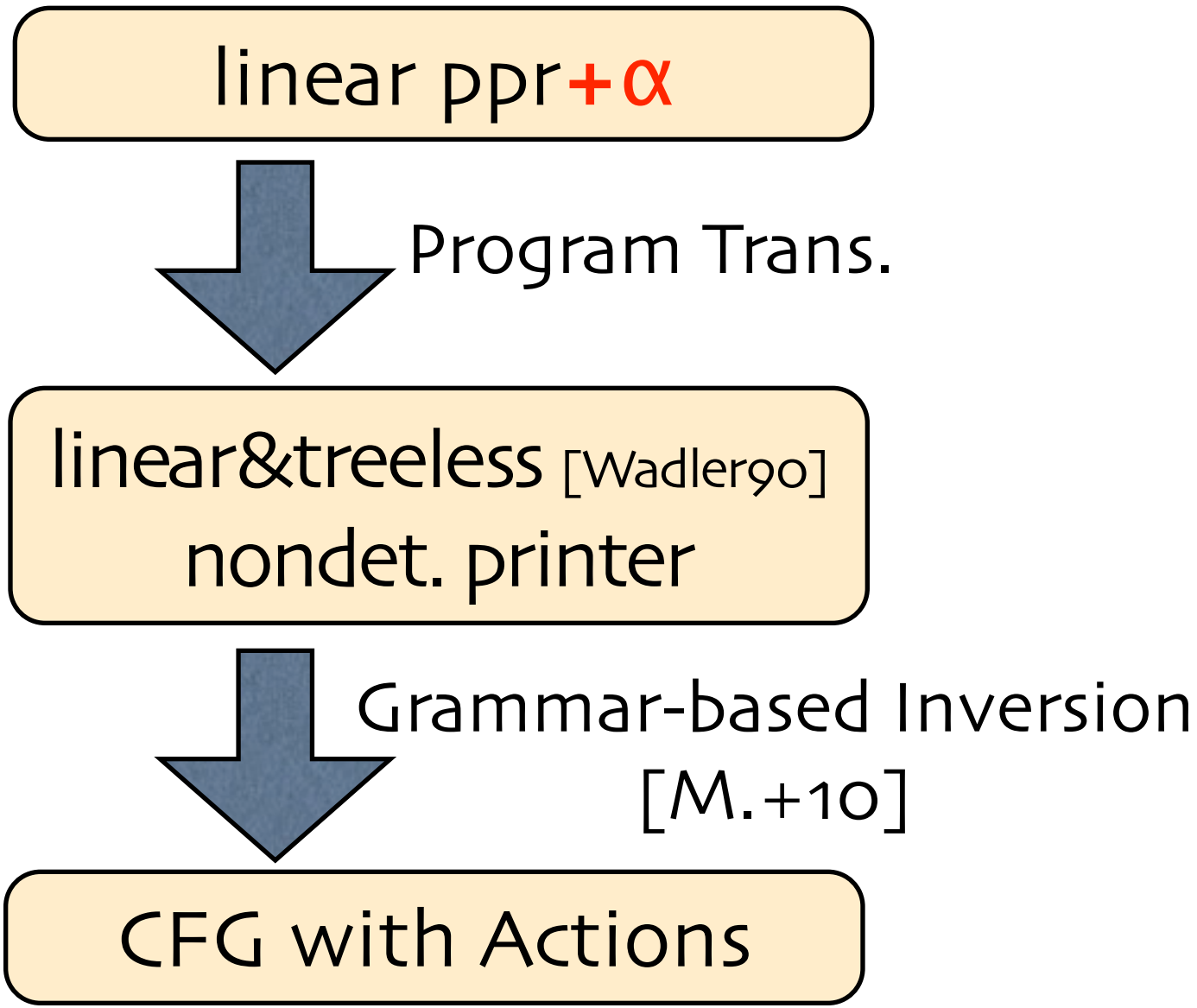
Extra Spaces

```
manyParens x = x <+ parens (manyParens x)
space = (text " " <+ text "\n") <> nil
nil   = text "" <+ space
```

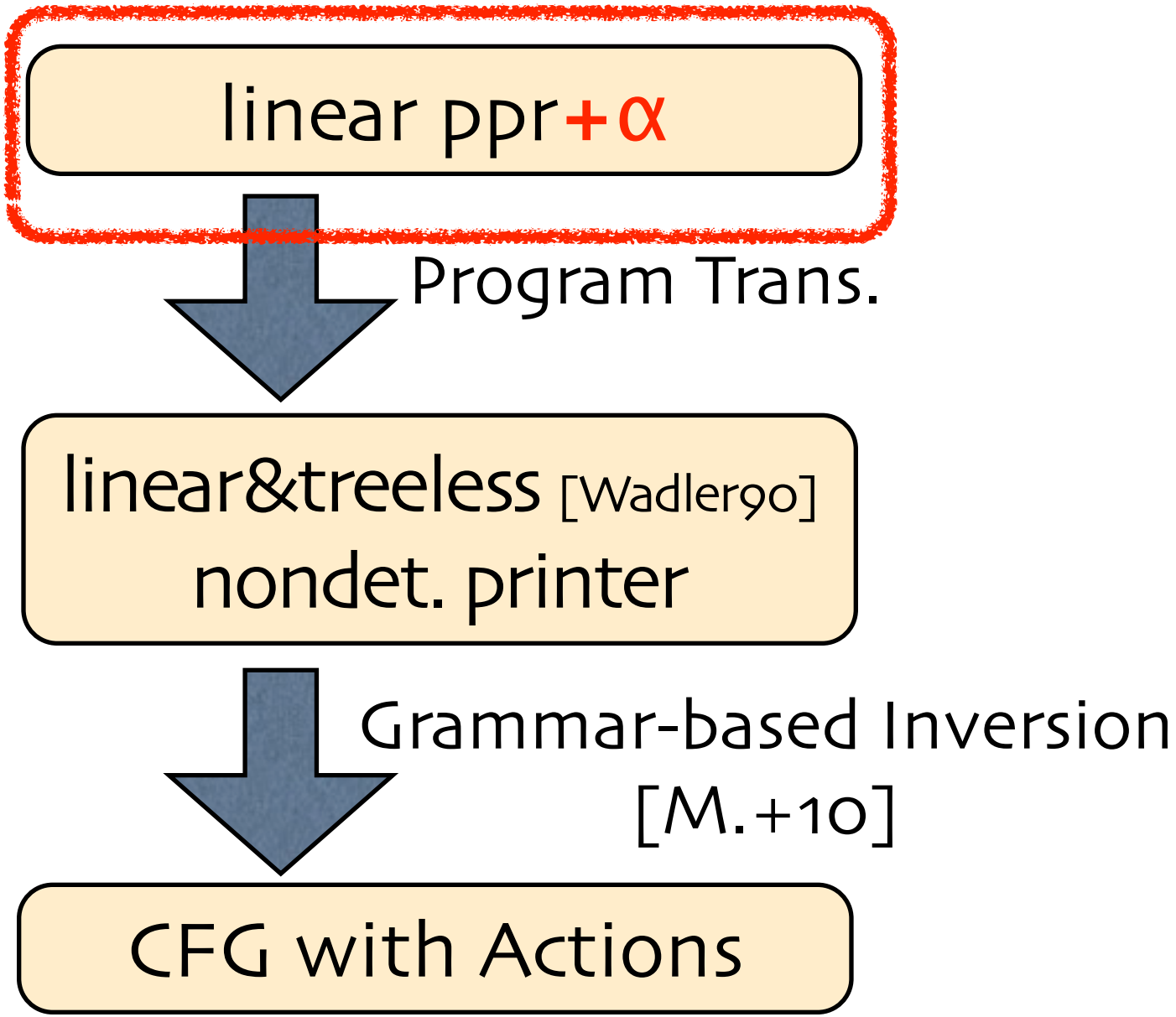
Agenda

- Input of FliPpr
- Quick Overview of FliPpr
- Related Work
- Conclusion

Architecture of FliPpr



Architecture of FliPpr

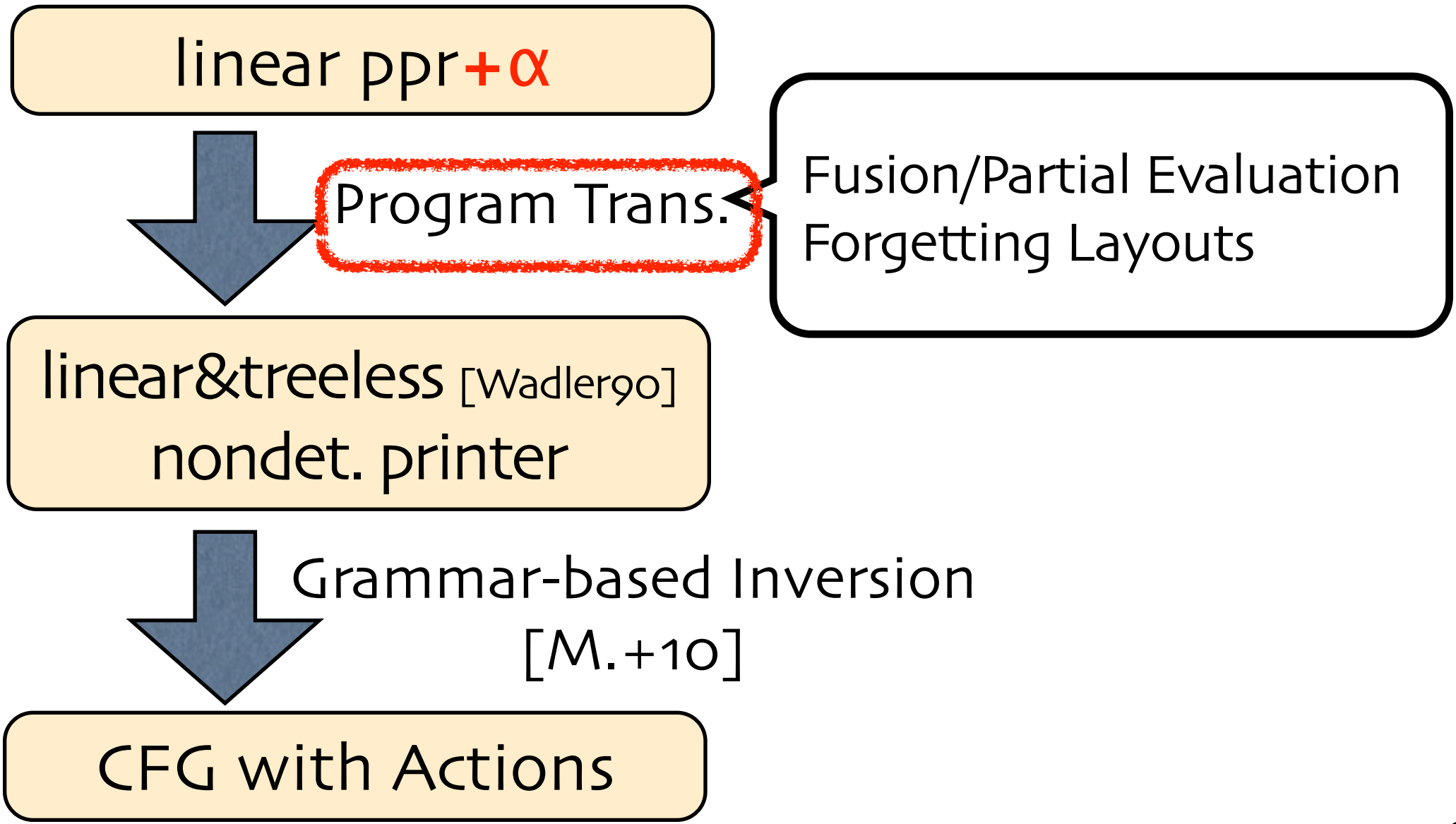


Input Program

- 1st-order linear functional programs with Wadler's combinators
  - restrictions:
    - limited nested calls (see paper)
    - distinguished statically-computed data

```
pprMain x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i One = text "1"
aux i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
   line <> text "-" <> space <> ppr 6 y)))
…
```

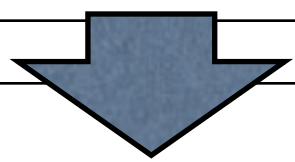
Architecture of FliPpr



Fusion/Partial Eval

```
pprMain x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i One = text "1"
aux i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
   line <> text "-" <> space <> ppr 6 y)))
…
```

```
manyParens x = x <+ parens (manyParens x)
parens x = text "(" <> nil <> x <> nil <> text ")"
```



```
pprMain x = ppr5 x
ppr5 x = aux5 x
     <+ text "(" <> nil <> ppr5 x <> nil <> text ")"
aux5 One = text "1"
aux5 (Sub x y) =
  group (ppr5 x <> nest 2 (
   line <> text "-" <> space <> ppr6 y))
…
```

Forgetting Layouts

- Clarify reinterpretation of `lines` by program transformations

```
text "s"
  x <> y
    line
nest k x
 group x
  x <+ y
```

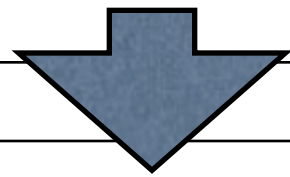
```
  "s"
x ++ y
 space
   x
   x
 x ? y
```

Nondeterministic choice

```
space = (" " ? "\n") ++ nil
nil   = "" ? space
```

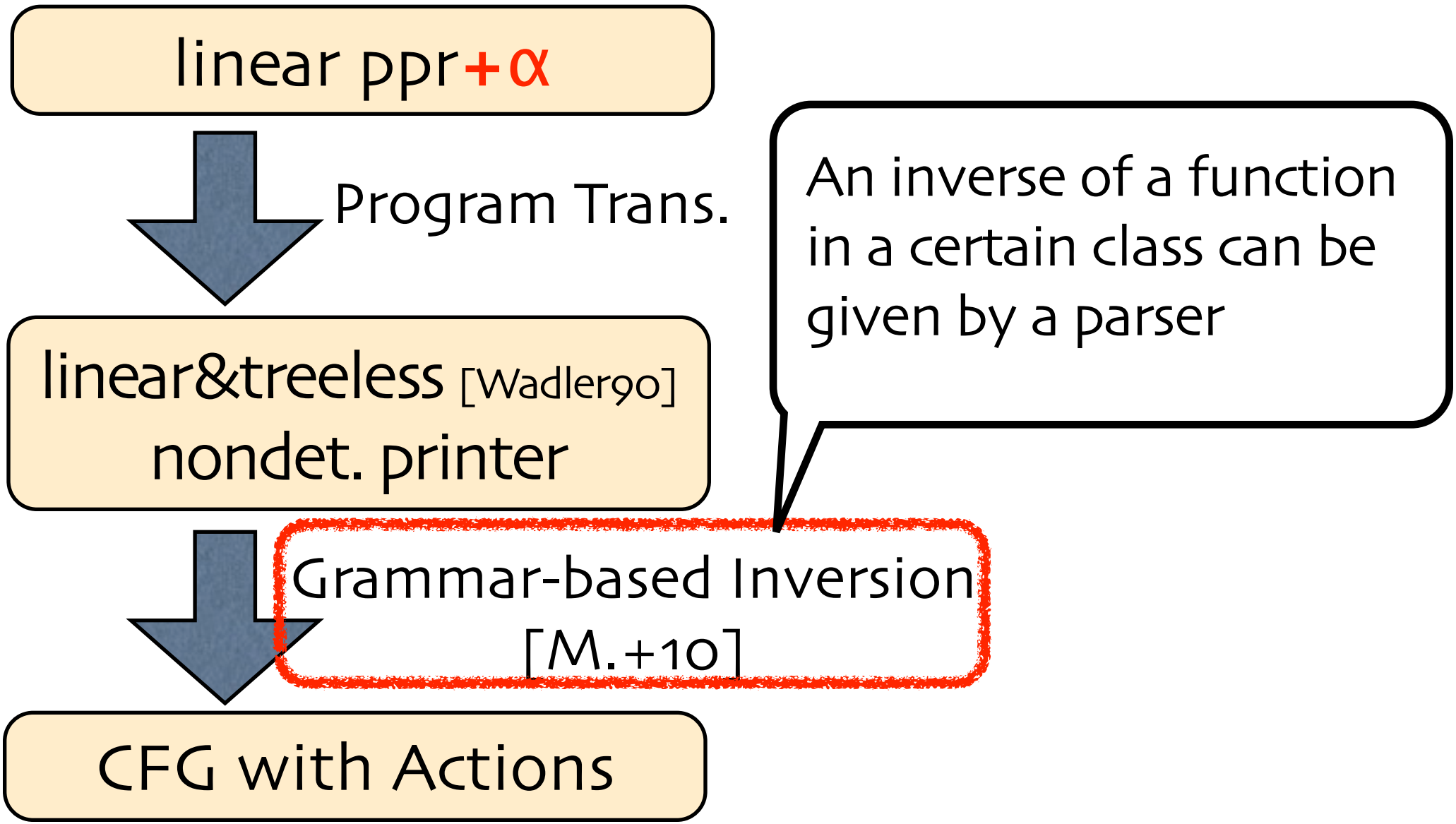
Example

```
pprMain x = ppr5 x
ppr5 x = aux5 x
     <+ text "(" <> nil <> ppr5 x <> nil <> text ")"
aux5 One = text "1"
aux5 (Sub x y) =
  group (ppr5 x <> nest 2 (
   line <> text "-" <> space <> ppr6 y))
…
```



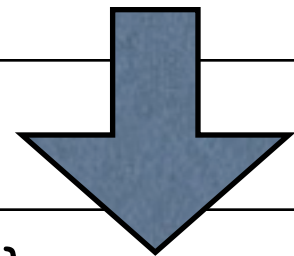
```
pprMain x = ppr5 x
ppr5 x = aux5 x ? "(" ++ nil ++ ppr5 x ++ nil ++ ")"
aux5 One = "1"
aux5 (Sub x y) =
   ppr5 x ++ space ++ "-" ++ space ++ ppr6 y
…
```

Architecture of FliPpr



Example

```
pprMain x = ppr5 x
ppr5 x = aux5 x ? "(" ++ nil ++ ppr5 x ++ nil ++ ")"
aux5 One = "1"
aux5 (Sub x y) =
   ppr5 x ++ space ++ "-" ++ space ++ ppr6 y
…
```

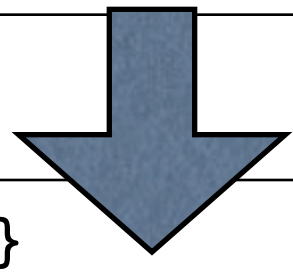


```
PprMain → Ppr5 { $1 }
Ppr5 → Aux5                          { $1 }
     | "(" Nil Ppr5 Nil ")"          { $3 }
Aux5 → "1"                             { One }
Aux5 → Ppr5 Space "-" Space Ppr6 { Sub $1 $5 }
…
```

Summary

```
pprMain x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i One = text "1"
aux i (Sub x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
   line <> text "-" <> space <> ppr 6 y)))
…
```

```
manyParens x = x <+ parens (manyParens x)
parens x = text "(" <> nil <> x <> nil <> text ")"
```



```
PprMain → Ppr5 { $1 }
Ppr5 → Aux5                          { $1 }
     | "(" Nil Ppr5 Nil ")"          { $3 }
Aux5 → "1"                              { One }
Aux5 → Ppr5 Space "-" Space Ppr6 { Sub $1 $5 }
…
```

In the Paper ...

- Formal definition of input programs
  - Types for binding-time analysis
  - Tiered-treelessness
- Extensions

```
ppr (Var x) = text (x as [a-z]+)
ppr (Int x) = text (itoa x as [0-9]+)
```

- An Involved Example
  - models first-order functional programs

Agenda

- Input of FliPpr
- Quick Overview of FliPpr
- Related Work
- Conclusion

Related Work

- Ppr/parser from one description
  - Invertible Syntax Description [Rendel&Ostermann10]
  - BNFC-meta [Duregård&Jansson11]
  - Syn [Boulton96]

No natural and fine control on pretty-printing

Conclusion

- FliPpr
  - takes a pretty-printing program
    - written with Wadler's pretty-printing combinators [Wadler 03]
    - together with additional info for parsing
  - returns a parser as a CFG with actions
    - based on grammar-based inversion [M.+10]

http://www-kb.is.s.u-tokyo.ac.jp/~kztk/FliPpr/

Future Work

- Solution to more general situation
  - A sender uses a certain representation
    - `a3b1` for `aaab` in runlength encoding
  - A receiver must accept more representations
    - `a3b1`, `a2a1b1`, `a1a2b1`,`a1a1a1b1` for `aaab` in runlength encoding

Future Work

- Enhance usability
  - More flexible pretty-printer descriptions
    - higher-order functions in surface lang
    - smart way to handle "lexing" issues
  - Injectivity analysis
  - Grammars beyond CFG
    - offside rules
      - Haskell, Python, YaML

Conceptual Change

```
ppr :: AST → Doc
```

Original

Doc is ...

Set of **Pretty Strings**

+

A Smart Chooser

Ours

Doc is ...

Set of **All Valid Strings**

+

A Smart Chooser

Related Work

- Quotient Lenses [Foster et al. 08]
  - Extra spaces and parens can be viewed as a quotient.
  - No direct connection to efficient implementations
    - Pretty-printing
      - [Wadler03, Swisstra&Chitil09, Kiselyov13,...]
    - Parsing
      - LR-k, GLR, Early, ...
      - [Frost et al. 08]