A Prettier Invertible Printing System

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

March 19th, 2013 @ESOP
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]
To implement a programming language, we often write ...

- a parser

\[
\begin{array}{c}
1 - 1
\end{array}
\]

- a pretty-printer

\[
\begin{array}{c}
- \\
1 & 1
\end{array}
\]

\[
\begin{array}{c}
- \\
1 & 1
\end{array}
\]

\[
1 - 1
\]
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

\[
\text{parse (ppr ast) } \neq \text{ ast}
\]
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

parse (ppr ast) /= ast
```
Our Goal

- Derive a parser from a pretty-printer by program inversion \([\text{Gries 81,}...]\)

\[
\text{parse (ppr ast) = ast}
\]
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

Diagram:

```
ppr

parse

can only parse "pretty" strings
```
Additional Info

- Required to derive useful parsers

```
ppr +\alpha \downarrow \text{parse}
```

Additional Info of “non-pretty” but valid strings
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&Chitilo9, 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

\[
\text{data } \text{AST} = \text{One} \mid \text{Sub AST AST} \mid \text{Div AST AST}
\]

\[
\begin{align*}
\text{ppr } \text{One} &= \text{\texttt{text } } "1" \\
\text{ppr } \text{(Sub } x \text{ y}) &= \text{ ifParens (i}\geq5) \left( \text{group (ppr } 5 \text{ x } <> \text{ nest } 2 (\text{line } <> \text{ text } "-" <> \text{ text } " " <> \text{ ppr } y)) } \right) \\
\end{align*}
\]
A Pretty Printer

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

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)))
```
A Pretty Printer

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

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)))
```
A Pretty Printer

\[ \text{data } \text{AST} = \begin{align*} \text{One} \\ \text{Sub AST AST} \\ \text{Div AST AST} \end{align*} \]

\[ \text{pprMain } x = \text{ppr } 5 x \]

\[ \text{ppr } i \text{ (Sub } x \text{ y) = } \begin{cases} \text{ppr } x <> \text{nest 2 (line <> text "-" <> text " " <> ppr } y) & \text{ifParens } b x = \text{if } b \text{ then parens } x \text{ else } x \\
\text{ppr } i \text{ (Div } x \text{ y) = } \begin{cases} \text{ppr } x <> \text{nest 2 (line <> text "/" <> text " " <> ppr } y) & \text{parens } x = \text{text "(" <> x <> \text{text ")"} \end{cases} \end{cases} \]
A Pretty Printer

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

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 ")"
```

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) → 
\[
\begin{array}{c}
1 \\
-1 \\
\end{array}
\]
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 " " <> pp 6 y)))
ppr i (Div x y) = ifParens (i>=7) (group (ppr 6 x <> nest 2 (line <> text "/" <> text " " <> pp 7 y)))

pprMain (Sub One One) →

\[
\begin{array}{c}
\text{1 - 1}
\end{array}
\]
Additional Info

- Additional info is required to parse “non-pretty” strings

How we write the additional info?
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

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

```
pprMain (Sub One One) →
```

```
[Diagram showing a simple calculation]
```

```
1
1
```

```
1 - 1
```

```
1
```

```
1 - 1
```
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
pprMain (Sub One One)

Uncovered non-pretty strings we want to parse

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

1 - 1
1 - 1

(((1) - 1)
1 - 1
1 - 1

1 - 1
1 - 1
1 - 1
Biased Choice: $\leftrightarrow$

- $\leftrightarrow$ for additional info
  - $x \leftrightarrow y$ equals to $x$ in pretty-printing
    - No need to change pretty-printing system
  - $x \leftrightarrow y$ also conveys the info of $y$

\[
\text{ppr } p = \text{pretty } \leftrightarrow \text{nonpretty}
\]

ppr knows both pretty and nonpretty are related to $p$
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))))

...
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
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)))

...
Agenda

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

linear ppr + $\alpha$

Program Trans.

linear & treeless [Wadler90]
nondet. printer

Grammar-based Inversion [M.+10]

CFG with Actions
Architecture of FliPpr

- linear ppr $+\alpha$
  - Program Trans.
    - linear&treeless \cite{Wadler90}
      - nondet. printer
        - Grammar-based Inversion \cite{M.+10}
          - CFG with Actions
Input Program

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

```haskell
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

**linear ppr + α**

**Program Trans.**

**linear & treeless** [Wadler90]

nondet. printer

**Grammar-based Inversion**

[Mc. +10]

**CFG with Actions**

Fusion/Partial Evaluation

Forgetting Layouts
Fusion/Partial Eval

```plaintext
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{text} \text{"s"} \\
x <> y \\
line
nest k x \\
group x \\
x <+ y
```

```
\text{"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

- linear ppr + $\alpha$
- Program Trans.
- linear&treeless [Wadler90]
- nondet. printer
- Grammar-based Inversion [M.+10]
- CFG with Actions

An inverse of a function in a certain class can be given by a parser.
Example

```
pprMain x = ppr5 x
ppr5 x = aux5 x ? "(" ++ nil ++ ppr5 x ++ nil ++ ")"
aux5 One = "1"
avx5 (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 }
...
```
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 ")"
In the Paper ...

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

- Extensions

- An Involved Example
  - models first-order functional programs

```ml
ppr (Var x) = text (x as [a-z]+)
ppr (Int x) = text (itoa x as [0-9]+)
```
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 combiners [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&Chitilo9, Kiselyov13,...]
    - Parsing
      - LR-k, GLR, Early, ...
      - [Frost et al. 08]