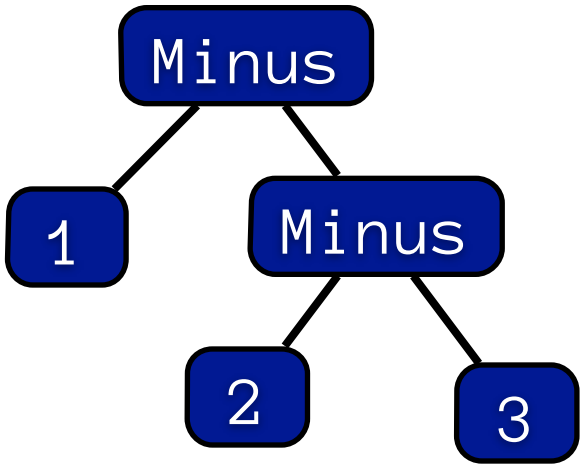
Relating a data structure and its text representation:

A case study of pretty-printing and parsing

Kazutaka Matsuda
(Univ. of Tokyo)

Background

- A data structure may have text-representations
  - for ***data exchange***
  - for ***human readability***



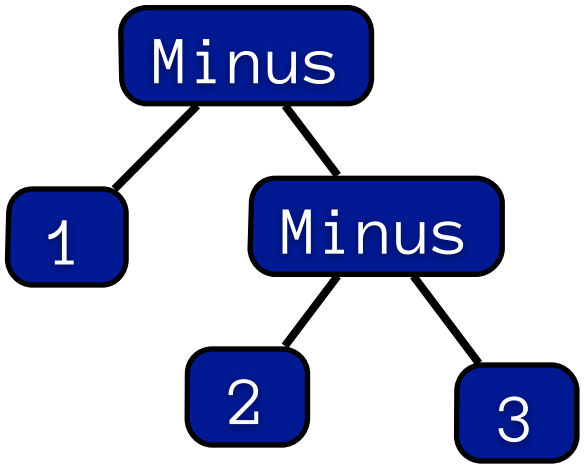
```
1 - (2 - 3)
```

In Data Structure

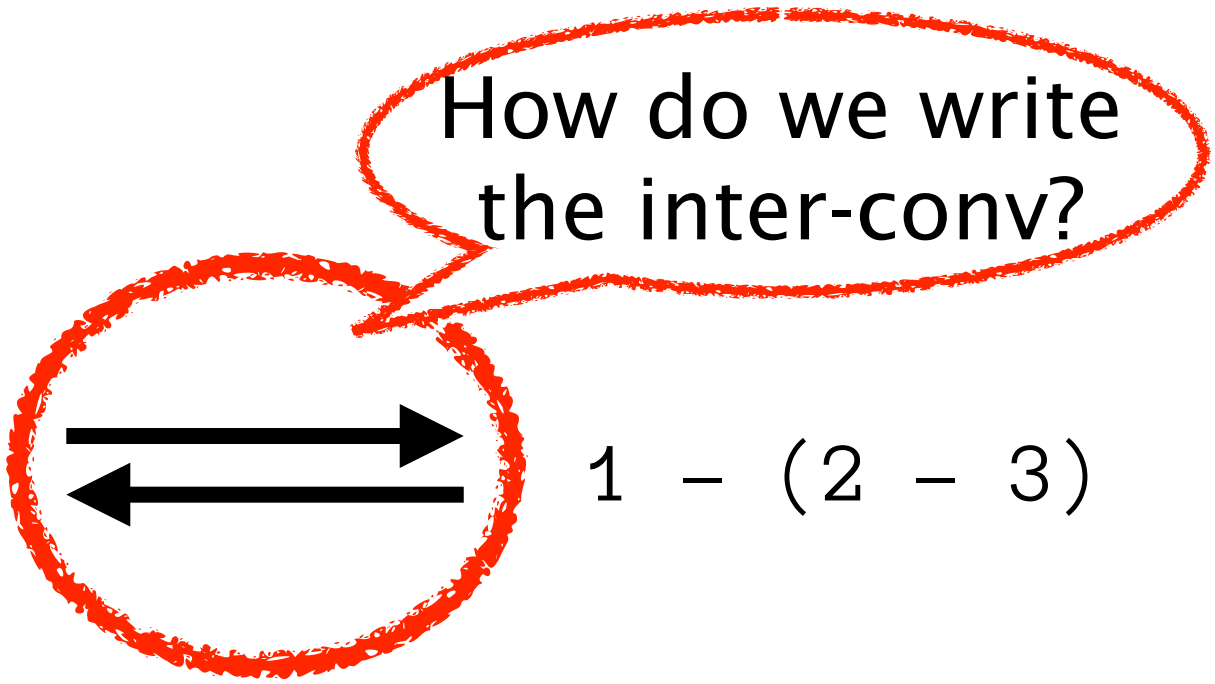
In Text

Motivation

- Writing inter-conversion between
  - *data structure*
  - its *text-representation*



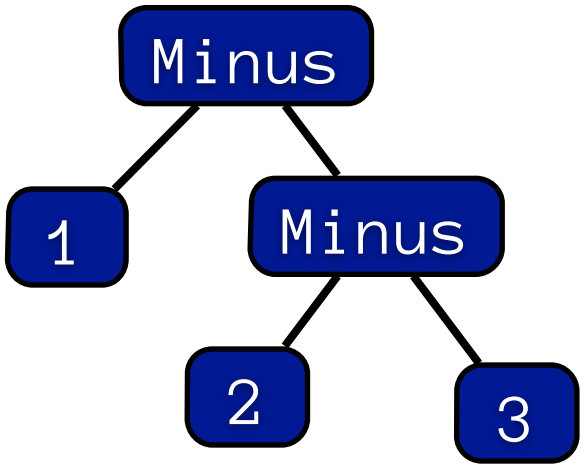
In Data Structure



In Text

A Case Study

- Inter-conversion between ***abstract syntax trees*** and ***codes***
  - pretty-printing and parsers



pretty-print

parse

```
1 - (2 - 3)
```

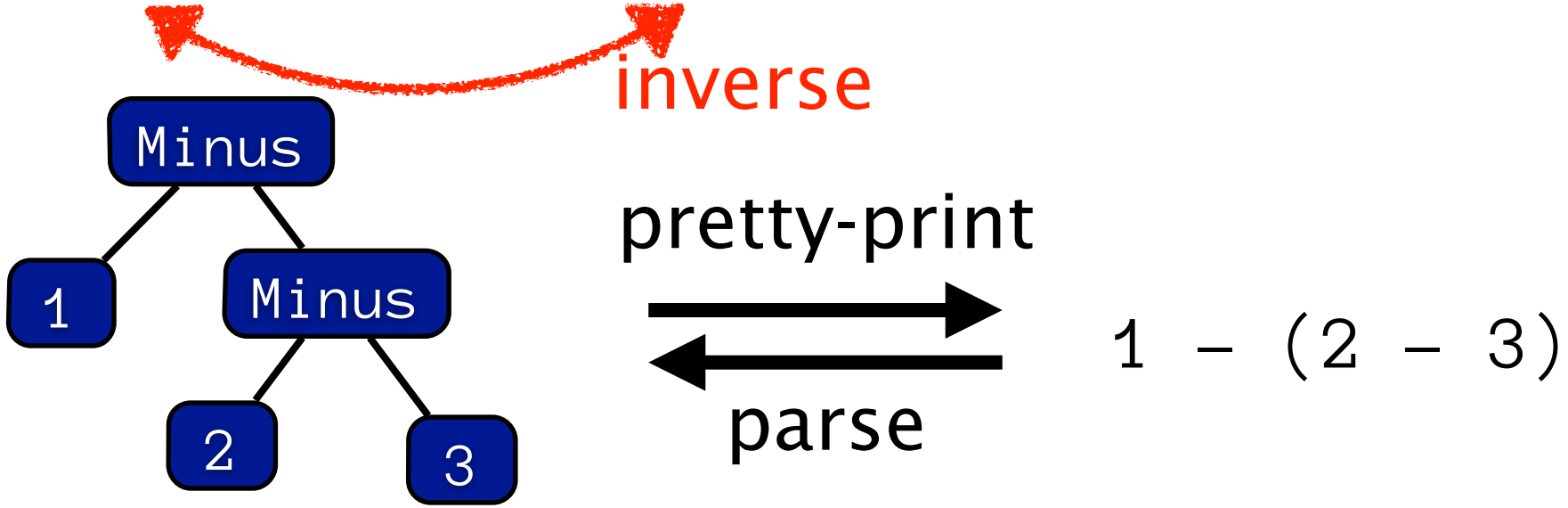
In Data Structure

In Text

Bidirectional Property

- A pretty-printing result is correctly parsed

```
parse (pretty_print ast) = ast
```



Bidirectional Property

- A pretty-printing result is correctly parsed

```
parse (pretty_print ast) = ast
```

Minus

```
*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
Still in GHC 7.6.3

Our Approach

- To derive a parser from a pretty-printer by program inversion

```
parse (pretty_print ast) = ast
```

  - Why this direction?
    - We have a data structure at first
    - Pretty-printing is more creative
      - more control on layouting is needed

Our Approach

- To derive a parser from a pretty-printer by program inversion

```
parse (pretty_print ast) = ast
```

```
int f(x) {
  return x
}
```

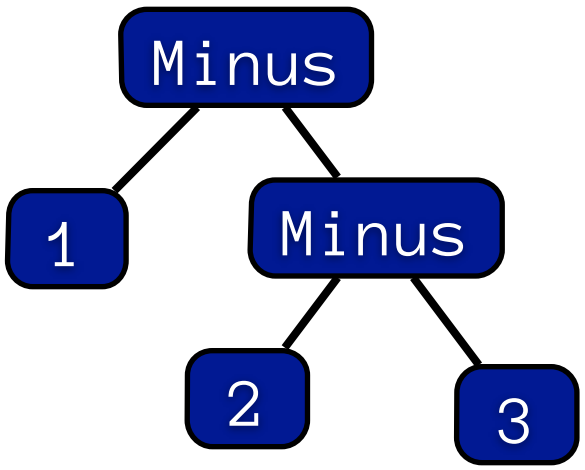
vs.

```
int f(x)
{
    return x
}
```

  - Why this direc
    - We have a dat
    - Pretty-printing is more creative
      - more control on layouting is needed

Technical Challenge

- "Information Mismatch"
  - A pretty-printer knows a "pretty" code but no other valid codes
  - A parser knows all the valid codes but no "prettiness"



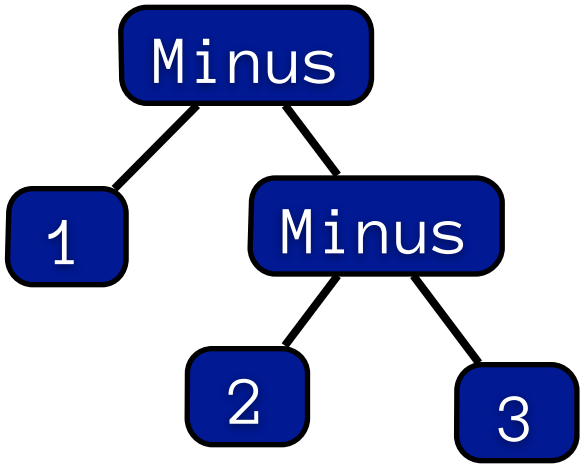
In Data Structure

```
(1) - (2 - 3)

1 -
  (2 - 3)

1 - (2 - 3)
```

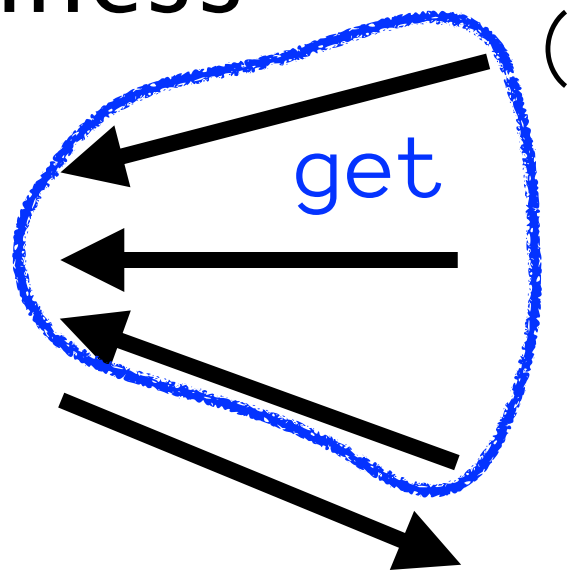
In Text

Technical Challenge

- "Information Mismatch"
  - A pretty-printer knows a "pretty" code but no other valid codes
  - A parser knows all the valid codes but no "prettiness"



In Data Structure

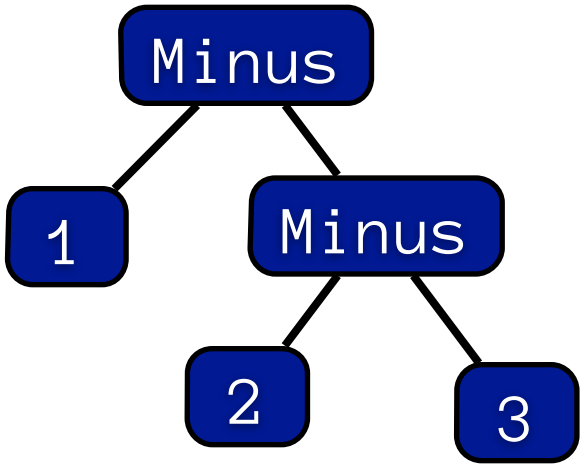



```
(1) - (2 - 3)

1 -
  (2 - 3)

1 - (2 - 3)
```

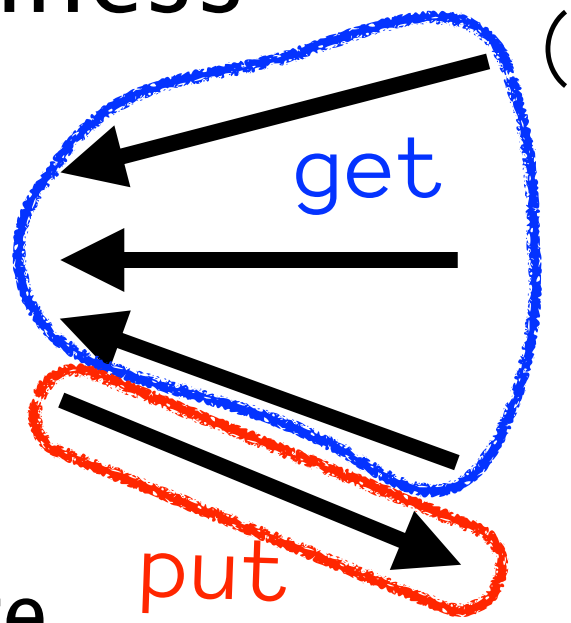
In Text

Technical Challenge

- "Information Mismatch"
  - A pretty-printer knows a "pretty" code but no other valid codes
  - A parser knows all the valid codes but no "prettiness"



In Data Structure




```
(1) - (2 - 3)

1 -
  (2 - 3)

1 - (2 - 3)
```

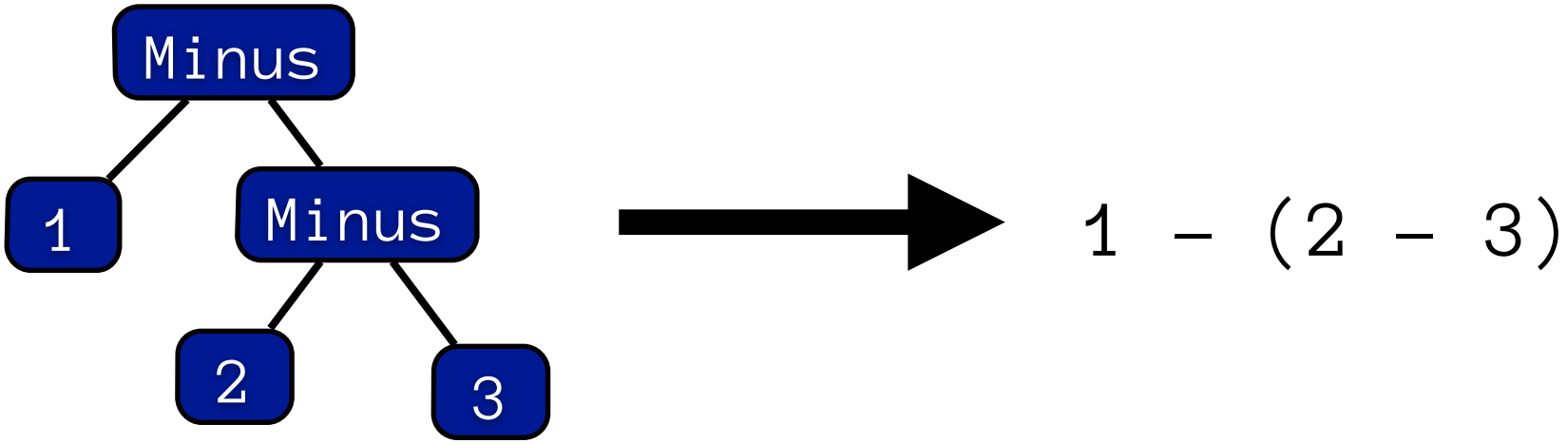
In Text

Our Proposal: FliPpr

- Invertible pretty-printing system [M.&Wang, 2013]
  - Takes a pretty-printer
    - based on an existing pretty-printing DSL [Wadler 2003]
    - ***with annotation of parsing information***
  - Returns a parser corresponding to the pretty-printer
    - Based on (full) CFG

Example of Inputs

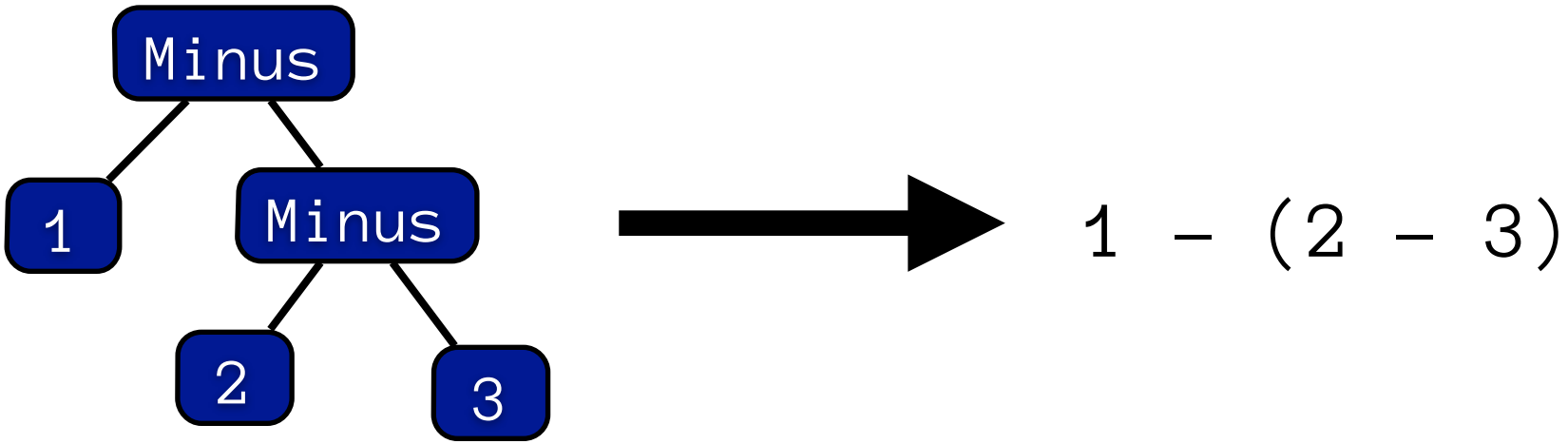
```
pretty_print x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i (Num i) = text (itoa x as [0-9]+)
aux i (Minus x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
    line <> text "-" <> space <> ppr 6 y)))
…
```



Example of Inputs

```
pretty_print x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i (Num i) = text (itoa x as [0-9]+)
aux i (Minus x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
    line <> text "-" <> space <> ppr 6 y)))
…
```

Extra parens

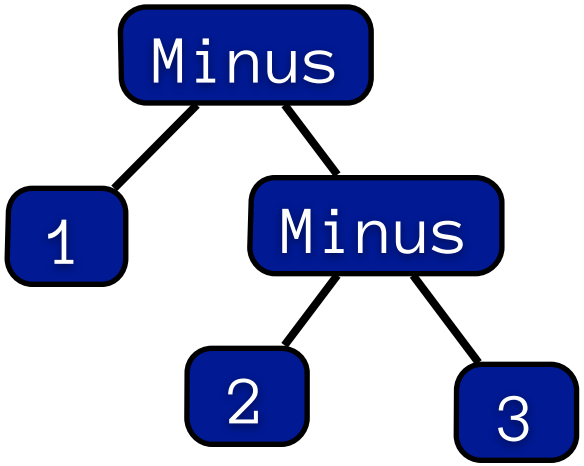


Example of Inputs

```
pretty_print x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i (Num i) = text (itoa x as [0-9]+)
aux i (Minus x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
    line <> text "-" <> space <> ppr 6 y)))
…
```

Extra parens

Extra spaces



```
1 - (2 - 3)
```

Example of Inputs

```
pretty_print x = ppr 5 x
ppr i x = manyParens (aux i x)
aux i (Num i) = text (itoa x as [0-9]+)
aux i (Minus x y) = ifParens (i>=6) (group (
  ppr 5 x <> nest 2 (
    line <> text "-" <> space <> ppr 6 y)))
…
```

Extra parens

Extra spaces

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

Advantages of FliPpr

- Fine-grained control on pretty-printing
- Efficiency
  - Reusability of existing efficient algorithms and implementations
    - for pretty-printers
      - [Wadler03, Swisstra&Chitil09, Kiselyov13,...]
    - for parsers
      - GLR, Early, [Frost+08], [Might+11], ...

Summary

- FliPpr:
  Invertible pretty-printing system
  - Takes a pretty-printer with parsing-annotations
    - Wadler (2003)'s pretty-printing combinators to write pretty-printers
  - Generates a parser based on CFG

Prototype Implementation:

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

Future Directions

- More expressive grammars
  - Indent-sensitive languages
    - Haskell, Python, YAML, …
- More expressive pretty-printer
  - User-defined prettiness
  - Type-system for safe programs
- More expressive input data-structure beyond trees
- More stable implementation ;)

Future Directions

- Truly bidirectional version
  - "prettiness" depends on the initial source

```
put s v =
    render c (pretty_print v)
    where
      c = … {- previous rendering
               info -} …
```