Pretty Printing with Delimited Continuations

Olaf Chitil

University of Kent, UK

IFL 2005

What is Pretty Printing?

```
if True
   then if True then True else True
   else
      if False
         then False
         else False
```

Pretty printing library interface

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

What is Pretty Printing?

```
if True
   then if True then True else True
   else
      if False
         then False
         else False
```

User code

```
toDoc :: Exp -> Doc
toDoc (If e1 e2 e3) =
  group (nest 3 (
    group (nest 3 (text "if" <> line <> toDoc e1)) <> line <>
    group (nest 3 (text "then" <> line <> toDoc e2)) <> line <>
    group (nest 3 (text "else" <> line <> toDoc e3))))
```

What is Pretty Printing?

```
if True
   then if True then True else True
   else
      if False
         then False
         else False
```

Pretty printing library interface

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

Specification: Functionality

A document may be formatted *horizontally* or *vertically*.

```
type Horizontal = Bool
```

A document has many different layouts.

```
type Doc = Horizontal -> [String]
```

Layouts for each document:

```
(text t) _ = [t]
line True = [" "]
line False = ["\n"]
(d1 <> d2) h = [l1 ++ l2 | l1 <- d1 h, l2 <- d2 h]
(group d) True = d True
(group d) False = d False ++ d True
```

Prettiest: compare line by line; within width-limit longer line better.

```
pretty w d = minimumBy (compareLayout w) (d False)
```

Specification: Further Properties

- time:
  - linear in document size
  - independent of document width
- (optimally) bounded

  ```
  pretty 4 (group (text "Hi" <> line <> text "you" <> undefined))
  ```

  yields

  ```
  Program error: {undefined}
  ```

  Instead want

  ```
  Hi
  you
  Program error: {undefined}
  ```

- space (lazy input/output): linear in width-limit

Outline of a Linear Algorithm

```
text "a" <> group (text "b" <> line <> group (text "c" <> line))
```

Two passes:

1. Use position in document to determine width of each group.
2. Use remaining space on line to determine for each group if horizontal.

Linear but unbounded.

Correctness Problem

```
pretty 6 (group (text "Hi" <> line <> text "you") <> text "!")
```

algorithm yields

```
Hi you!
```

but specification says

```
Hi
you!
```

Correctness Problem

```
pretty 6 (group (text "Hi" <> line <> text "you") <> text "!")
```

algorithm yields

```
Hi you!
```

but specification says

```
Hi
you!
```

Only *group-closed* documents:

A `line` between end of each group and next `text`.

Represent Document as Token List

Represent

```
group (text "Hi" <> line <> text "you") <> text "!"
```

as

```
[Open, Text "Hi", Line, Text "you", Close, Text "!"]
```

Group-closed document via rewriting:

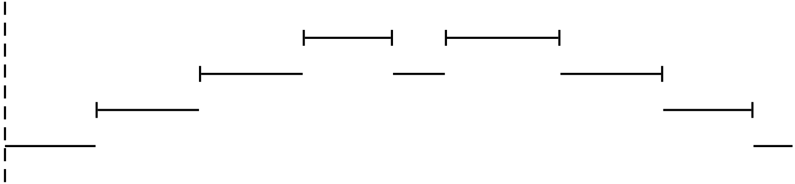
```
Close (Text s ts) ⇒  Text s (Close ts)
Open (Text s ts)  ⇒  Text s (Open ts)
Open (Close ts)   ⇒  ts
```

Effect of representation change:

- Rewritten document describes same set of layouts.
- Algorithm always selects correct layout.

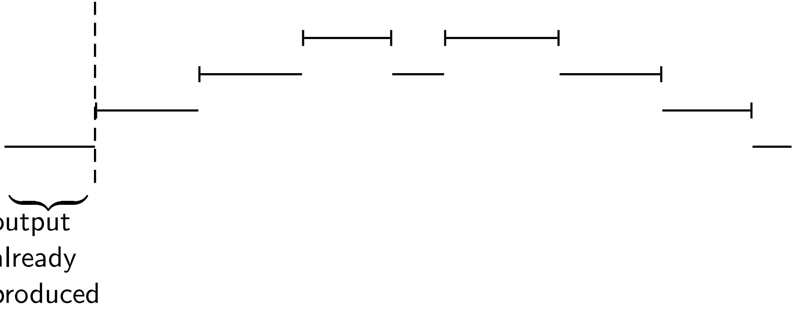
A Linear Unbounded Algorithm

- single pass
- no laziness



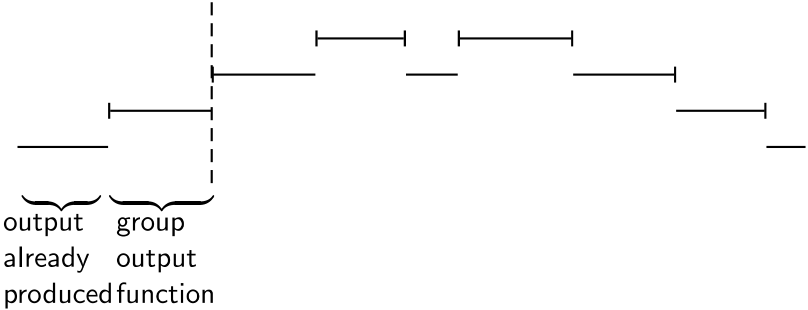
A Linear Unbounded Algorithm

- single pass
- no laziness



A Linear Unbounded Algorithm

- single pass
- no laziness

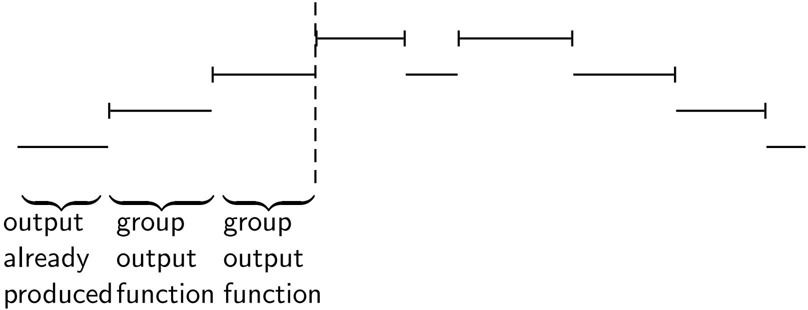



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out
```

```
inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

A Linear Unbounded Algorithm

- single pass
- no laziness

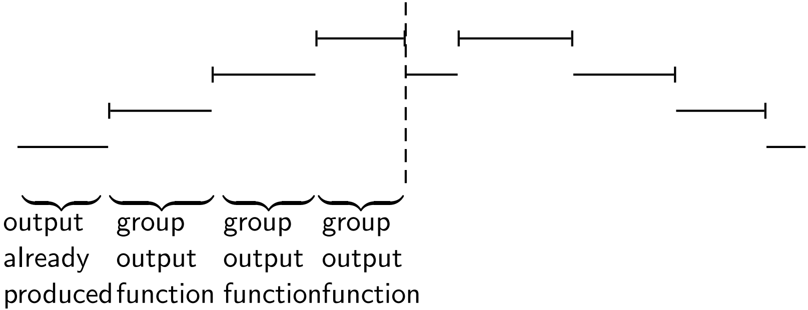



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out

inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

A Linear Unbounded Algorithm

- single pass
- no laziness

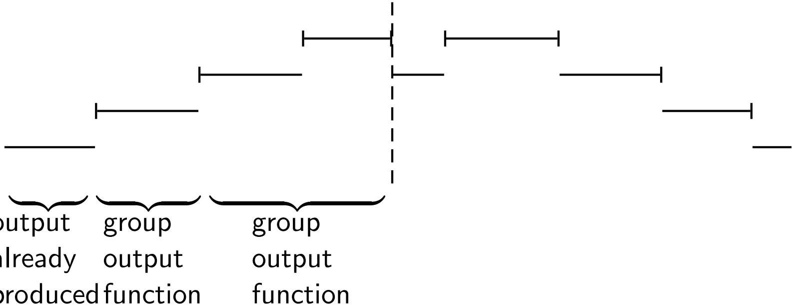



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out
```

```
inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

A Linear Unbounded Algorithm

- single pass
- no laziness

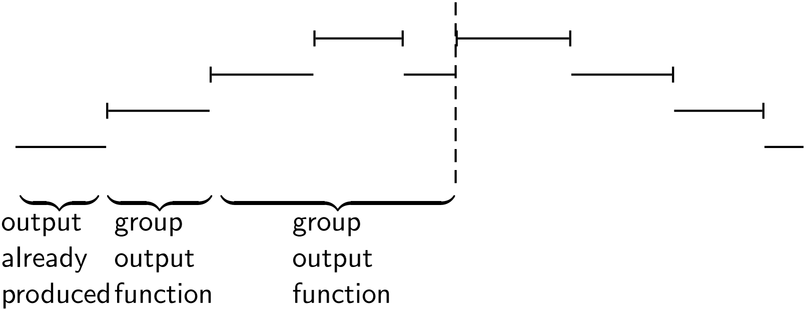



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out
```

```
inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

A Linear Unbounded Algorithm

- single pass
- no laziness

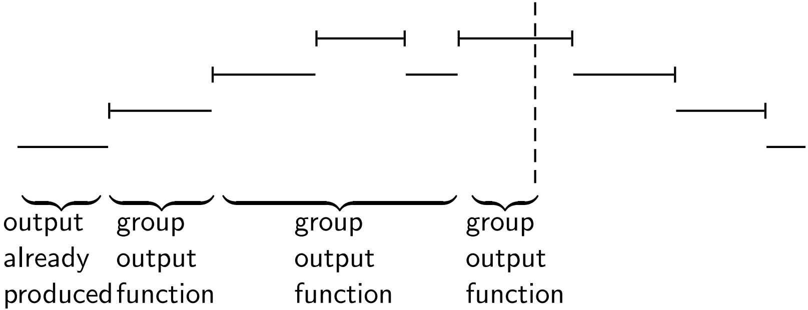



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out

inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

A Linear Unbounded Algorithm

- single pass
- no laziness

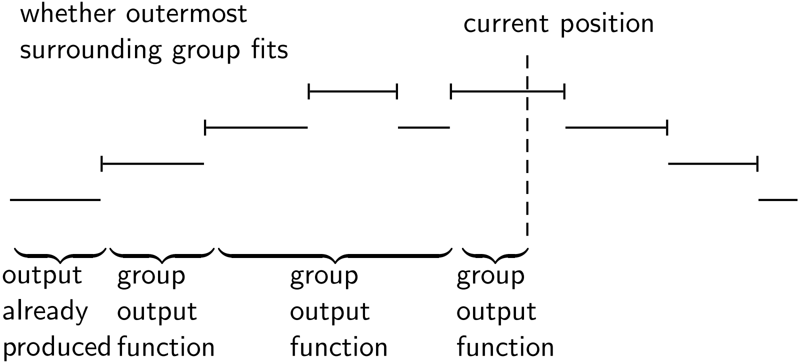



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out
```

```
inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

The Linear Bounded Algorithm

- continuously check whether outermost surrounding group fits

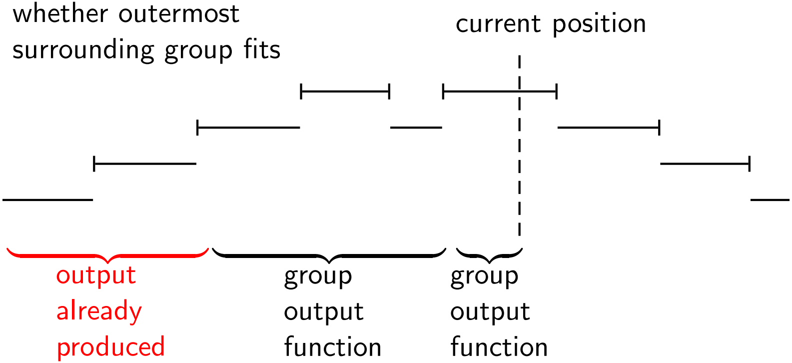



```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out

inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

The Linear Bounded Algorithm

- continuously check whether outermost surrounding group fits




```
type Out = Remaining -> String
type OutGroup = Horizontal -> Out -> Out

inter :: Tokens -> Width -> Position -> ⟨(Position,OutGroup)⟩ -> Out
```

Optimisation by Specialisation

Replace

```
inter :: Tokens -> Width -> Position -> ⟨OutGroup⟩ -> Out
```

by

```
noGroup    :: Tokens -> Width -> Position -> Out
oneGroup   :: Tokens -> Width -> Position ->
              Position -> OutGroup -> Out
multiGroup :: Tokens -> Width -> Position ->
              Position -> OutGroup ->
              ⟨(Position,OutGroup)⟩ ->
              Position -> OutGroup -> Out
```

Summary

- Delimited continuations express explicitly switching between consuming input and producing output.
- Dequeue is buffer between input consumed and output produced.
- Specialisation improves performance but duplicates code.
- Laziness gives space linear in width, but irrelevant for correctness and linearity.
- Higher-order functions essential; defunctionalised algorithm incomprehensible.
- How to prove equivalence of specification and implementation?