Principled Parsing for Indentation-Sensitive Languages
Revisiting Landin’s Offside Rule

Michael D. Adams
Portland State University

Haskell Implementors Workshop, September 14, 2012
• If a `let`, `where`, `do`, or `of` keyword is not followed by the lexeme `{`, the token `{n}` is inserted after the keyword, where `n` is the indentation of the next lexeme if there is one, or 0 if the end of file has been reached.

• If the first lexeme of a module is not `{` or `module`, then it is preceded by `{n}` where `n` is the indentation of the lexeme.

• Where the start of a lexeme is preceded only by white space on the same line, this lexeme is preceded by `<n>`, where `n` is the indentation of the lexeme, provided that it is not, as a consequence of the first two rules, preceded by `{n}`.

(Haskell Report 2010)
L (<n>:ts) (m:ms) = ';': (L ts (m:ms)) if m = n
    = '}'': (L (<n>:ts) ms) if n < m
L (<n>:ts) ms = L ts ms
L ({n}:ts) (m:ms) = '{' : (L ts (n:m:ms)) if n > m
L ({n}:ts) [] = '{' : (L ts [n]) if n > 0
L ({n}:ts) ms = '{' : '}' : (L (<n>:ts) ms)
L ('}':ts) (0:ms) = '}' : (L ts ms)
L ('}':ts) ms = parse-error
L ('{':ts) ms = '}' : (L ts (0:ms))
L ( t :ts) (m:ms) = '}' : (L (t:ts) ms)
    if m \neq 0 and parse-error(t)
L ( t :ts) ms = t : (L ts ms)
L [] [] = []
L [] (m:ms) = '}' : L [] ms if m \neq 0
Magic!
GLR Parsing

- Filter out invalid indentations

My work

- Different formalism
- LR Parsing
- LL Parsing
- No filtering
- Formal derivation
\[ x + v \text{ where} \\
\quad x = -(y + z) + w \]
Grammar

exp → exp 'where' ID '=' | exp |
exp → exp '++' exp
exp → '−' exp
exp → '()' exp'
exp → ID
|exp| → |exp| '++' exp
|exp| → |'−'| exp
|'−'| → '−'

exp1

exp1 'where' exp1 x2 '=' exp6
x1 '+' exp1 v5
x1 '+' exp6
'−' exp6
'−' exp6
'−' exp6
'()' exp0 ')' exp6
y1 '+' exp6
z5
Grammar

\[
\begin{align*}
\text{exp} & \rightarrow \text{exp}\text{=} \ 'where' \geq \ \text{ID} \geq \ '==' \geq |\text{exp}| \\
\text{exp} & \rightarrow \text{exp}\text{=} \ '+' \geq \ \text{exp}\text{=} \\
\text{exp} & \rightarrow \ '-' \geq \ \text{exp}\text{=} \\
\text{exp} & \rightarrow \ '(' \geq \ \text{exp}^{\circ} \ ')' \geq \\
\text{exp} & \rightarrow \ \text{ID} \\
|\text{exp}| & \rightarrow |\text{exp}|\text{=} \ '+' \geq \ \text{exp}\text{=} \\
|\text{exp}| & \rightarrow | '-' |\text{=} \ \text{exp}\text{=} \\
| '-' | & \rightarrow | '-' | \\
\end{align*}
\]
Does it work?

Languages

SRFI-49
ISWIM
Mirranda
Gofer
Orwell
Curry
Habit
Occam
Python
Haskell
F# (?)
YAML (?)
Does it work?

Languages

- SRFI-49
Does it work?

Languages

- SRFI-49
- ISWIM
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Occam
- Python
- Haskell
- F# ( {?})
- YAML ( {?})
Does it work?

Languages
- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Habit
- YAML (?)
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Occam
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry

- Habit
- Occam
- Python

- F# (??)
- YAML (??)
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Occam
- Python
- Haskell
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Occam
- Python
- Haskell
- F# (?)
Does it work?

Languages

- SRFI-49
- ISWIM
- Mirranda
- Gofer
- Orwell
- Curry
- Habit
- Occam
- Python
- Haskell
- F# (?)
- YAML (?)
Status

- Math (!!!)
  - Lots of subtleties
- Implementation in Happy
  - Based on haskell-src
  - Parses all of base (with qualifications)
- Under review at POPL 2013
a = do b
do { c
}
d
Scannerless

```
a = do b
   do { c
   }    d

a = do b
   do { c
   }   d
```
a = do b
    do { c } d

a = do b
    do { c } d d do e

a = do b
    do { c } d
Scannerless

\[
a = \text{do } b \text{ do } \{ c \} \text{ d}
\]

\[
a = \text{do } b \text{ do } \{ c \} \text{ d } \text{ do } e
\]

\[
a = \text{do } b \text{ do } \{ c \} \text{ d}
\]

\[
a = \text{do } b \text{ do } \{ c \} \text{ d } \text{ do } e
\]