Principled Parsing for Indentation-Sensitive Languages

Revisiting Landin's Offside Rule

Michael D. Adams

Portland State University

Haskell Implementors Workshop, September 14, 2012

Indentation in Haskell

- 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)

Indentation in Haskell 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 ≠ 0 and parse-error(t)
L ( t :ts)     ms =  t  : (L ts ms)
L       []     [] =  []
L       [] (m:ms) = '}' : L [] ms          if m ≠ 0
```

Indentation in GHC

Magic!

Erdweg et al.

- GLR Parsing
  - Filter out invalid indentations

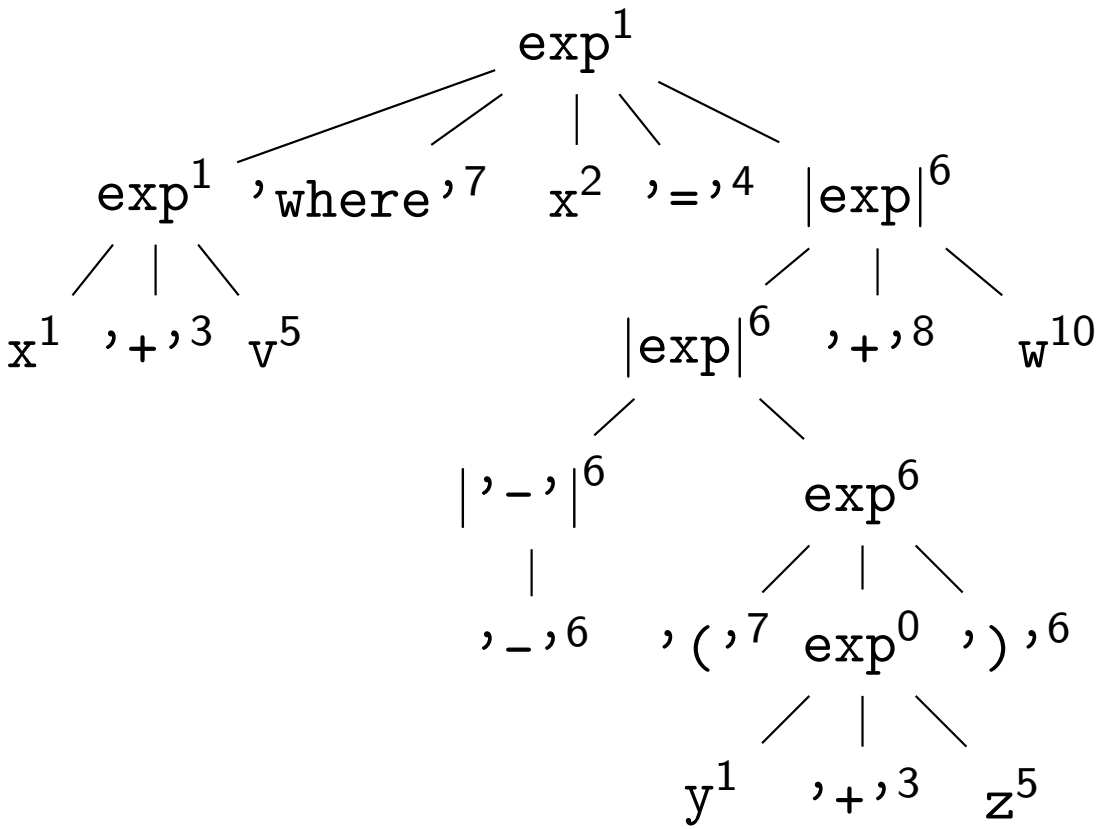
Sebastian Erdweg, Tillmann Rendel, Christian Kästner and Klaus Ostermann. Layout-sensitive Generalized Parsing. In Conference on Software Language Engineering (SLE), 2012.

My work

- Different formalism
- LR Parsing
- LL Parsing
- No filtering
- Formal derivation

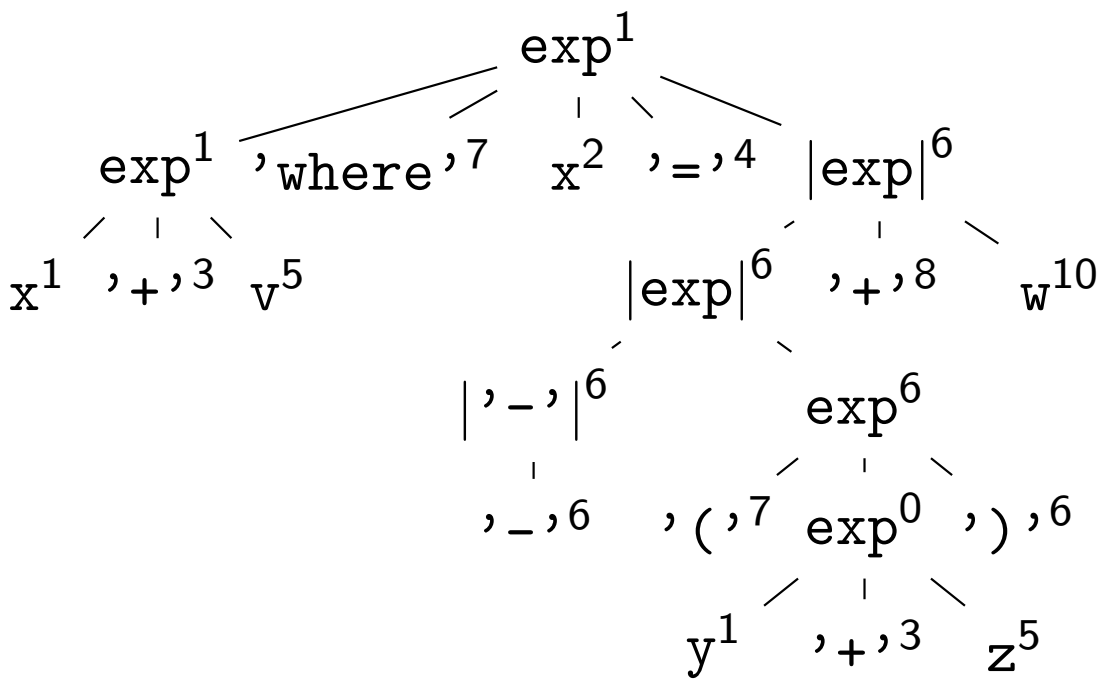
Code

```
x + v where
 x = -(
y + z) + w
```



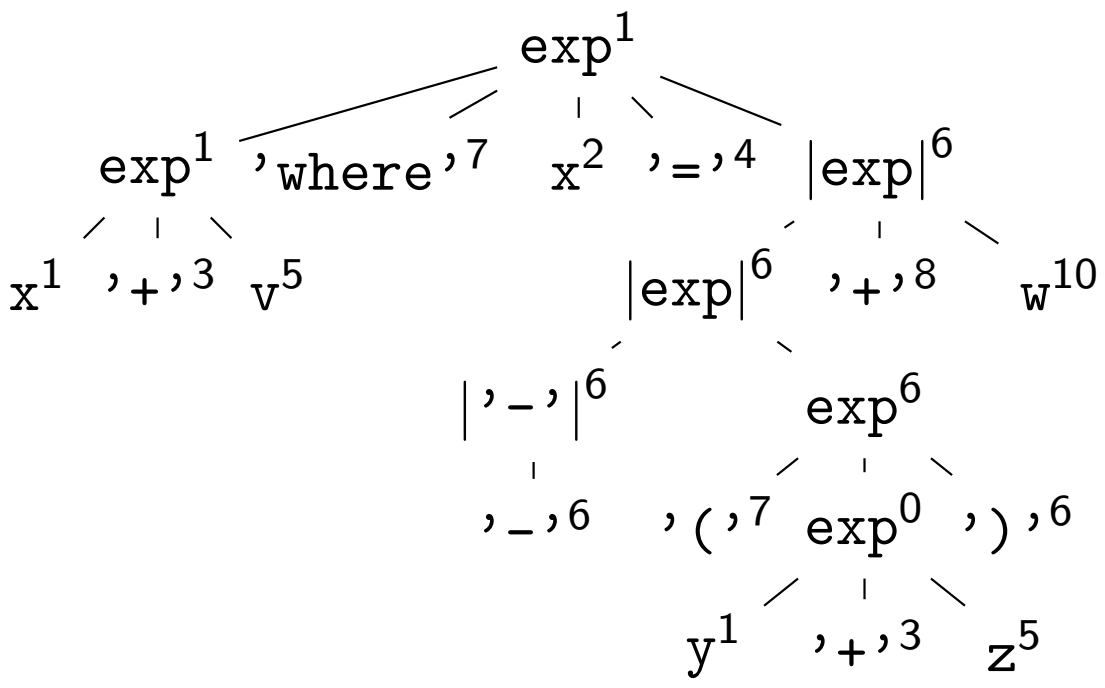
Grammar

$$
\begin{aligned}
\texttt{exp} &\rightarrow \texttt{exp 'where' ID '=' |exp|} \\
\texttt{exp} &\rightarrow \texttt{exp '+' exp} \\
\texttt{exp} &\rightarrow \texttt{'-' exp} \\
\texttt{exp} &\rightarrow \texttt{'(' exp ')'} \\
\texttt{exp} &\rightarrow \texttt{ID} \\
\texttt{|exp|} &\rightarrow \texttt{|exp| '+' exp} \\
\texttt{|exp|} &\rightarrow \texttt{|'-'| exp} \\
\texttt{|'-'|} &\rightarrow \texttt{'-'}
\end{aligned}
$$



Grammar

$$
\begin{array}{lcl}
\texttt{exp} & \rightarrow & \texttt{exp}^{=}\ \texttt{'where'}^{\geq}\ \texttt{ID}^{\geq}\ \texttt{'='}^{\geq}\ |\texttt{exp}|^{\geq} \\
\texttt{exp} & \rightarrow & \texttt{exp}^{=}\ \texttt{'+'}^{\geq}\ \texttt{exp}^{=} \\
\texttt{exp} & \rightarrow & \texttt{'-'}^{\geq}\ \texttt{exp}^{=} \\
\texttt{exp} & \rightarrow & \texttt{'('}^{\geq}\ \texttt{exp}^{\circledast}\ \texttt{')'}^{\geq} \\
\texttt{exp} & \rightarrow & \texttt{ID}^{\geq} \\
|\texttt{exp}| & \rightarrow & |\texttt{exp}|^{=}\ \texttt{'+'}^{\geq}\ \texttt{exp}^{=} \\
|\texttt{exp}| & \rightarrow & |\texttt{'-'}|^{=}\ \texttt{exp}^{=} \\
|\texttt{'-'}| & \rightarrow & \texttt{'-'}^{=}
\end{array}
$$



Does it work?

Languages

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

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

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

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
  `http://michaeldadams.org/papers/layout_parsing/`

Scannerless

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

Scannerless

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

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

Scannerless

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

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

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

Scannerless

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

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

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

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