Neon: a Library for Language Usage Analysis

Jurriaan Hage   Peter van Keeken
e-mail: jur@cs.uu.nl
homepage: http://www.cs.uu.nl/people/jur/

Department of Information and Computing Sciences, Universiteit Utrecht

January 9, 2009

1. Introduction

The starting point: Helium

- Helium for learning Haskell
  - Implemented in Haskell
- To get some idea where to improve further we need to know how students "use" the language?
  - do they avoid certain parts of the language?
  - which parts of the syntax are often involved in mistakes?
  - how long does it take to solve a type error?
  - when does it help to know Java, or works against them?
  - and so on...
- Each of these questions is a study by itself.
- Today I only talk about the tool, Neon, we developed to help answer these questions.

The first (easy) step: log the compilations

- Logging facility added was added from the outset.
- Compiler logs every compile via a socket connection to a Java server.

```
bigbrother | T |
  1.7.0 (Tue Dec 4 11:00:00 CET 2007) |
  -P/usr/local/helium/lib:. --overloading
  --enable-logging -v /tmp/Interpreter.hs |
  bigbrother/2007-12-14@13_58_20_250/Dummy.hs
```

- Helium has been in use since 2002.
  - Over 68,000 “full” compilation contexts (later is fuller)
- Collection is “in vivo”, so polluted to some extent,
- but loggings have been cleaned up (by Peter).
- Now to analyze the loggings...

The design criteria of Neon

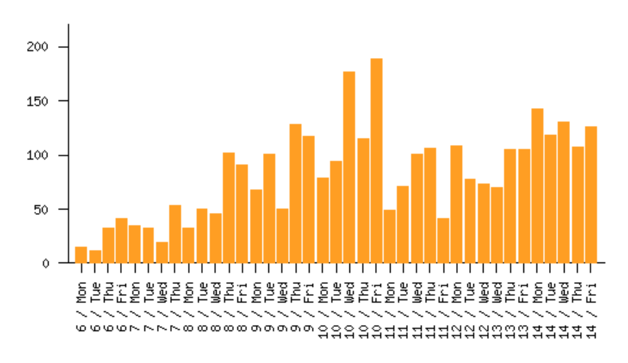
- To do this effectively, we need support. Hence, Neon.
  - Why is effectiveness so important?
- Queries should be concise and conceptually close to what they intend to express.
- Implementation based on a small set of well-understood primitives and combinators,
  - Eases argumentation that implementation of Neon is correct.
- It should be easy to reuse code from the compiler.
  - We can reuse the lexer, parser etc.
- Generate esthetically pleasing output.
  - Support multiple output formats, e.g., HTML tables and PNG files.

The elements of this talk

- Slick and slippery examples
  - For which we need more queries...
- Descriptive statistics
- The Neon library that
  - implements these ideas
  - ... and allows us to generate these slick examples,
  - ... by writing a bit of simple, reusable code.
- Concluding remarks, future work, points of discussion.

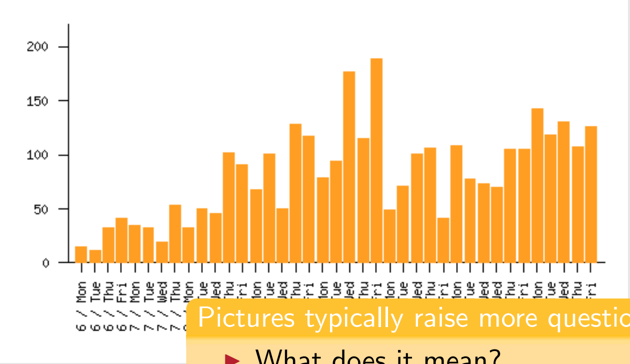
2. The slick examples

Example 1: Module Size



- Average number of lines for compiled modules, given per day (year 2003-2004)

Example 1: Module Size

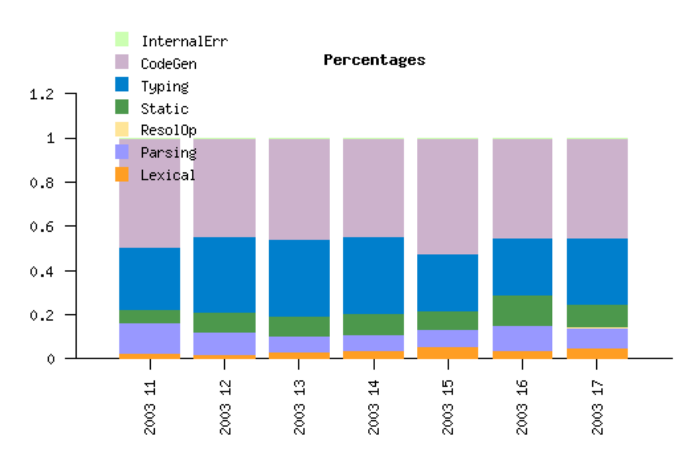


- Average nu
  day (year 2

Pictures typically raise more questions

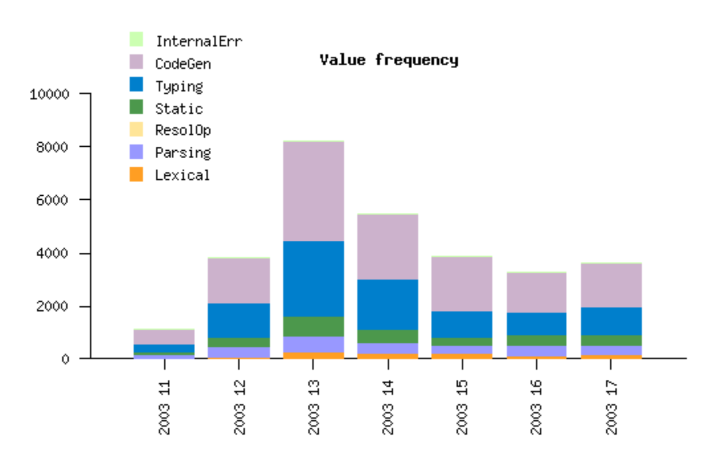
- What does it mean?
- Do all students show the same pattern?
- Correlation with grades, experience, ...?
- Counting code, comments and blank lines separately.

Example 2: Phase Analysis (relative)



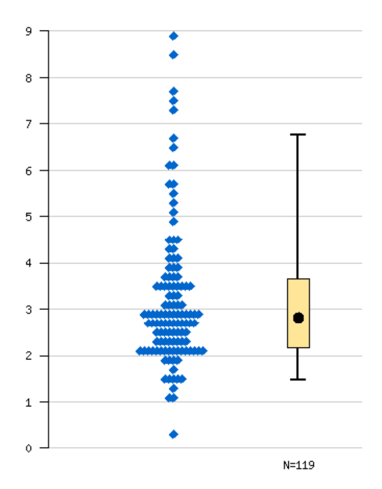
- Why does the ratio of parse errors increase again?
- Do recidivists muddy the picture?

Example 2: Phase Analysis (absolute)



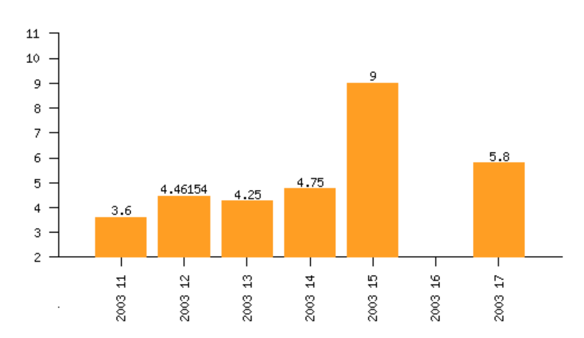
- Absolute gives an idea of weight: how significant are the ratios.
- We want these queries to be similar.

Example 3



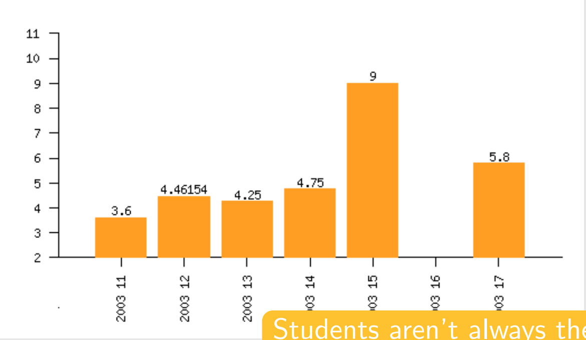
- Average in-between compile time in minutes per student

Example 4



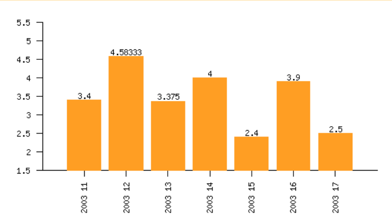
- Average number of compiles needed to “solve” a type error, for a particular student.
- How does one measure this at all?

Example 4



- Average number error, for a partic
- How does one m

Students aren't always the same



3. The basic concepts

Descriptive statistics

- Easy presentation of results in multiple formats
  - `ploticus` pictures, HTML tables, $\LaTeX$
- Grouping loggings, repeatedly
  - For each student, for each week, compute the list of loggings.
- Filtering on (groups of) loggings.
  - Only lists with at least 10 compiles for a given student
  - Only loggings from the 19th of September
- Computing statistics for groups and other metrics.
  - Only the lengths of the logged programs.
  - The average length of the lists of loggings.

Why use Haskell?

- Advantages
  - General purpose language with strong typing.
  - Analyses are functions.
  - Built-in support for easy composition and abstraction.
    - Particularly higher-orderness, type classes and polymorphism.
  - Reuse Helium code base
  - Library uses combinators.
    - Facilitates building analyses from others.

Why use Haskell?

- Advantages
  - General purpose language with strong typing.
  - Analyses are functions.
  - Built-in support for easy composition and abstraction.
    - Particularly higher-orderness, type classes and polymorphism.
  - Reuse Helium code base
  - Library uses combinators.
    - Facilitates building analyses from others.
- Drawbacks
  - Generating pictures can only be done via existing tools.
    - New libraries are mushrooming.
  - Speed could become an issue.
    - Haskell is actually doing well these days.
  - Limited audience.
    - But getting less so.

4. The combinator library

The basics

- An analysis result is represented by $[(key, value)]$.
- The key datatype allows us to describe what the value represents,
- by remembering how $value$s have been computed.
- key inhabits the $DescrKey$ class so that the description can be updated automatically.
- From it, we can generate legend information, filenames and so on.

Some primitives

A few of the (not so primitive) primitives

$$\mathbf{type}\ An\ k\ a\ b = [(k,a)] \to [(k,b)]$$

$$\begin{aligned} &basicAnalysis :: (DescrKey\ k) \Rightarrow \\ &\quad String \to (a \to b) \to An\ k\ a\ b \end{aligned}$$

$$\begin{aligned} &groupAnalysis :: (DescrKey\ k, Enum\ a, DataInfo\ b) \Rightarrow \\ &\quad (a \to b) \to ([a] \to [[a]]) \to An\ k\ [a]\ [a] \end{aligned}$$

$$mapAnalysis :: (DescrKey\ k) \Rightarrow An\ k\ a\ b \to An\ k\ [a]\ [b]$$

$$\diamond :: An\ k\ b\ c \to An\ k\ a\ b \to An\ k\ a\ c$$

$$runAnalysis :: a \to k \to An\ k\ a\ b \to [(k,b)]$$

Group analysis in more detail

$$groupAnalysis :: (DescrKey\ k, Enum\ a, DataInfo\ b) \Rightarrow$$
$$(a \to b) \to ([a] \to [[a]]) \to An\ k\ [a]\ [a]$$

- First function argument describes which values belong to the same group.
- Second function computes the actual grouping.
- But why is the result type not $[[a]]$?

Group analysis in more detail

$$groupAnalysis :: (DescrKey\ k, Enum\ a, DataInfo\ b) \Rightarrow$$
$$(a \to b) \to ([a] \to [[a]]) \to An\ k\ [a]\ [a]$$

- First function argument describes which values belong to the same group.
- Second function computes the actual grouping.
- But why is the result type not $[[a]]$?
- Flatten: $[[1,2,3],[2,4]]$ grouped on parity is not $[[[1,3],[2]],[[2,4]]]$, but $[[1,3],[2],[2,4]]$.
- But how do you know that $[1,3]$ and $[2]$ belonged to the same list?

Group analysis in more detail

$$groupAnalysis :: (DescrKey\ k, Enum\ a, DataInfo\ b) \Rightarrow (a \rightarrow b) \rightarrow ([a] \rightarrow [[a]]) \rightarrow An\ k\ [a]\ [a]$$

- First function argument describes which values belong to the same group.
- Second function computes the actual grouping.
- But why is the result type not $[[a]]$?
- Flatten: $[[1,2,3],[2,4]]$ grouped on parity is not $[[[1,3],[2]],[[2,4]]]$, but $[[1,3],[2],[2,4]]$.
- But how do you know that $[1,3]$ and $[2]$ belonged to the same list?
- We store that in the *key*: $[(k1,[1,2,3]),(k2,[2,4])]$ maps to $[(k1',[1,3]),(k1',[2]),(k2',[2,4])]$
- Avoids arbitrarily nested values.

Example: number of loggings per phase

$$
\begin{aligned}
&\mathit{groupPerPhase} :: \mathit{DescrKey}\ \mathit{key} \Rightarrow \mathit{An}\ \mathit{key}\ [\mathit{Logging}]\ [\mathit{Logging}] \\
&\mathit{groupPerPhase} = \\
&\quad \mathit{groupAnalysis}\ \mathit{phase}\ (\mathit{groupAllUnder}\ \mathit{phase})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{countNumberOfLoggings} :: \mathit{DescrKey}\ \mathit{key} \Rightarrow \mathit{An}\ \mathit{key}\ [a]\ \mathit{Int} \\
&\mathit{countNumberOfLoggings} = \\
&\quad \mathit{basicAnalysis}''\ \texttt{"number of loggings"}\ \mathit{length}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{loggingsPerPhase} :: \mathit{An}\ \mathit{KeyHistory}\ [\mathit{Logging}]\ \mathit{Int} \\
&\mathit{loggingsPerPhase} = \mathit{countNumberOfLoggings} \diamond \mathit{groupPerPhase}
\end{aligned}
$$

And now to present this

$$
\begin{array}{l}
\mathit{presentLoggingPerPhase} :: \mathit{FilePath} \rightarrow \mathit{FilePath} \rightarrow \mathit{IO}\ ()\\
\mathit{presentLoggingPerPhase}\ \mathit{logfile}\ \mathit{outputfp} = \mathbf{do}\\
\quad \mathit{loggings} \leftarrow \mathit{parseLogfile}\ \mathit{logfile}\\
\quad \mathbf{let}\ \mathit{analysisResult} = \mathit{runAnalysis}\ \mathit{loggings}\ \mathit{loggingsPerPhase}\\
\quad \mathit{barChart} \leftarrow \mathit{renderBarChart}\ \mathit{outputfp}\ \mathit{analysisResult}\\
\quad \mathit{writeFile}\ (\mathit{outputfp} +\!\!\!+ \texttt{"/analysis.tex"})\\
\qquad (\mathit{renderLateX}\ \$\ \mathit{showAsTable1D}\ \mathit{analysisResult}) +\!+\!+\\
\qquad \mathit{plotToFigure}\ \mathit{barChart}
\end{array}
$$

The same, but now per week

Given the definition of *groupPerWeek*

$$
\begin{aligned}
&\mathit{loggingsPerPhasePerWeek} :: \mathit{An}\ \mathit{KeyHistory}\ [\mathit{Logging}]\ \mathit{Int}\\
&\mathit{loggingsPerPhasePerWeek} =\\
&\quad \mathit{countNumberOfLoggings}\\
&\quad \diamond\ \mathit{groupPerPhase}\\
&\quad \diamond\ \mathit{groupPerWeek}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{phaseResearch} :: \mathit{FilePath} \rightarrow [(\mathit{KeyHistory}, [\mathit{Logging}])] \rightarrow \mathit{IO}\ ()\\
&\mathit{phaseResearch}\ \mathit{outputpath}\ \mathit{input} = \mathbf{do}\\
&\quad \mathit{barChartStacked} \leftarrow \mathit{renderBarChartDynamic}\ \mathit{outputpath}\\
&\qquad\qquad (\mathit{loggingsPerPhasePerWeek} < \$ > \mathit{input})\\
&\quad \mathit{writeFile}\ \ldots
\end{aligned}
$$

The same, but now per week

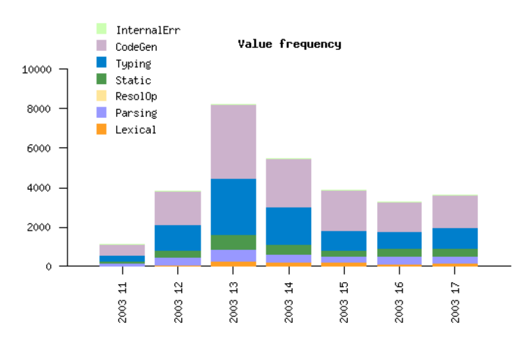
Given the definition

$loggingsPerPhaseP$
$loggingsPerPhaseP$
$countNumberOf$
$\diamond\ groupPerPhas$
$\diamond\ groupPerWeek$

$phaseResearch :: F$
$phaseResearch\ out$
$barChartStacked$
$(loggingsPerPhasePerWeek <\$> input)$

$writeFile \ldots$

Slight reprise



5. To conclude

What more do we want from Neon?

- In-depth studies
  - Students do not seem particularly interested in doing this kind of study
- The use of student properties
  - Who is (s)he? What grade was obtained? First language or not?
- Easy integration with different versions of Helium.
  - Not as easy as it may seem.

What more do we want, period?

- Money (to hire a PhD student).
- More loggings
  - In the process of extending Helium to include type classes in full.
- I am looking for expertise in empirical research.