A principled approach to REPLs*

*Read-eval-print-loops, consoles, interactive shells, notebooks, command-lines

Thomas van Binsbergen
Universiteit van Amsterdam

Mauricio Verano
CWI TU/e

Tijstr van der Storm
University of Groningen

Joint work with Pierre Jeanjean, Benoit Combemale, Olivier Barais (INRIA, University of Rennes)

Problem: REPLs are popular tools, but their semantics is not well-defined

Solution: see REPLs as a (modular) language extension

How: define a sequential operator \( \circ \) and compose semantics.

**REPLs: old but popular**

<table>
<thead>
<tr>
<th>Type</th>
<th>Mar 2023</th>
<th>Mar 2021 Change</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>&lt; Python</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>&lt; C</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>&lt; Java</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>= C++</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>= C</td>
<td></td>
</tr>
</tbody>
</table>

JOSS (1964)

**Very diverse design space:**

Sequential languages:

- **Python**
- **Java**

“A language is sequential if the concatenation of two programs is again a program”

\[
[p_1 \circ p_2] = [p_2] \circ [p_1]
\]

Non-seq stack language:

```plaintext
syntax Prog = Op* Print;
syntax Op = Num | "+" | "dup";
syntax Print = "print";
syntax Stack = list[int];
int eval((Prog)\:<Op* ops\: print>);
```

Sequentialize:

```plaintext
syntax Cmd = Op | Print | assoc Cmd Cmd;
Conf eval((Cmd)`<Cmd c1>` `<Cmd c2>``, Conf c) = eval(c2, eval(c1, c));
```

rascal-mp.org

**MiniJava Notebook**

**QL: questionnaire DSL**

**eFLINT: norms DSL**