Fine-grained language composition

Edd Barrett, Carl Friedrich Bolz, Lukas Diekmann, Geoff French, Sarah Mount, Laurence Tratt, Naveneetha Krishnan Vasudevan

Software Development Team
2016-01-28
A perfect programming language
Background

Solution
Background

Solution

A new programming language
Reality

Another imperfect programming language
What to expect from this talk

A

B
What to expect from this talk

$A \cup B$
What to expect from this talk

Python ∪ Prolog
What to expect from this talk

Python \cup PHP
Two levels of challenge

Tooling
Two levels of challenge

Tooling

Language friction
Tooling challenges

- Python
- PHP

PyHyp
Tooling challenges

- Python
  - Syntax
  - Runtime

- PHP
  - Syntax
  - Runtime

- PyHyp
  - Syntax
  - Runtime
Tooling challenges

Language boxes

Python
- syntax
- runtime

PHP
- syntax
- runtime

PyHyp
- syntax
- runtime

6/28 HTTP://SOFT-DEV.ORG/
Tooling challenges

Language boxes

- Python
  - syntax
  - runtime

- PHP
  - syntax
  - runtime

- PyHyp
  - syntax
  - runtime

Composed meta-tracing VMs
Parsing: recognising

Input:
Bill hits Ben

Grammar:
noun verb noun

7/28 HTTP://SOFT-DEV.ORG/
Input: Bill hits Ben
Input: Bill hits Ben
Grammar: noun verb noun
Parsing: Context-Free Grammars

Expr ::= Expr '+' Term | Term
Term ::= Term '*' Factor | Factor
Factor ::= '(' Expr ')' | 'INT'
Parsing: Context-Free Grammars

Expr ::= Expr ′+′ Term | Term
Term ::= Term ′*′ Factor | Factor
Factor ::= ′(′ Expr ′)′ | ′INT′
Expr ::= Expr ‘+’ Term | Term
Term ::= Term ‘*’ Factor | Factor
Factor ::= ‘(’ Expr ‘)’ | ‘INT’
Parsing: Context-Free Grammars

Expr ::= Expr ‘+’ Term | Term
Term ::= Term ‘*’ Factor | Factor
Factor ::= ‘(’ Expr ‘)’ | ‘INT’
Parsing: Context-Free Grammars

Expr ::= Expr '+' Term | Term
Term ::= Term '*' Factor | Factor
Factor ::= '()' Expr ')' | 'INT'
Parsing: Trees

\[ 2 + 3 \]

Diagram:
```
Expr
  /\   /
Expr Term
  /\   /
Term Factor
  /\   /
Factor 3
  /\   /
2
```
Syntax composition

```
<grammar>
expr ::= ... 
term ::= ... 
    | ... 
    | ... 
func ::= ...

<program>
for (j : js) {
    doStuff();
}
```

http://soft-dev.org/
Syntax composition

```
<grammar>
expr ::= ...
    | ...
    | ...
func ::= ...
<program>
for (j : js) {
    doStuff();
}
.
.
.
Parser
```

PL X

<grammar>
expr ::= ...
    | ...
    | ...
func ::= ...

PL Y

<program>
for (j : js) {
    doStuff();
}
.
.
.

10 / 28 HTTP://SOFT-DEV.ORG/
Syntax composition

<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...
</grammar>

<program>
for (j : js) {
    doStuff();
}
.
.
.
Parser
Parse Tree
Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

Generalised
Parse Tree
Syntax composition

\[ \text{PL X} \]
\[ \text{<grammar>} \]
\[ \text{expr::=} \ldots \]
\[ \text{term::=} \ldots \]
\[ \text{|} \ldots \]
\[ \text{|} \ldots \]
\[ \text{func::=} \ldots \]

\[ \text{PL Y} \]
\[ \text{<program>} \]
\[ \text{for (j : js) \{ doStuff(); \}} \]
\[ \ldots \]

\[ \text{PEG} \]

\[ \text{Parse Tree} \]
The only choice?

SDE
Challenge: SDE’s power + a text editor feel?
Runtime composition
Runtime composition

PL X
Interpreter

PL Y
Interpreter

C/C++
Runtime composition

Interpreter
Interpreter
Too slow

PL X
PL Y
C/C++
Runtime composition

PL X
Interpreter
JIT Compiler

PL Y
Interpreter
JIT Compiler

C/C++
Runtime composition

Too much engineering

PL X

Interpreter

PL Y

Interpreter

C/C++

JIT Compiler

JIT Compiler

14/28 HTTP://SOFT-DEV.ORG/
Runtime composition

PL X
Interpreter
JVM/CLR

PL Y
Interpreter
JIT Compiler

14/28 HTTP://SOFT-DEV.ORG/
Runtime composition

Semantic mismatch
Runtime composition

PL X
Interpreters
Glue
PL Y
Meta-tracing
PL Z
Interpreter
Tracing JIT
Meta-tracing translation with RPython

Interpreter
Meta-tracing translation with RPython

Interpreter

RPython translator
Meta-tracing translation with RPython
Meta-tracing translation with RPython
Meta-tracing translation with RPython

you write this

RPython translator

Optimised Interpreter

JIT

you get this for free

you get this for free

http://soft-dev.org/
Runtime composition recap

PL X

PL Y

Interpreters

Glue

Meta-tracing

PL Z

Interpreter

Tracing JIT

16/28 HTTP://SOFT-DEV.ORG/
Runtime composition recap

PyPy  Hippy  Meta-tracing  PyHyp
Interpreters  Glue  Tracing JIT
Interpreter

16/28  HTTP://SOFT-DEV.ORG/
## Composed Richards vs. other VMs

<table>
<thead>
<tr>
<th>Type</th>
<th>VM</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mono</td>
<td>CPython 2.7.7</td>
<td>9.475 ± 0.0127</td>
</tr>
<tr>
<td></td>
<td>HHVM 3.4.0</td>
<td>4.264 ± 0.0386</td>
</tr>
<tr>
<td></td>
<td>HippyVM</td>
<td>0.250 ± 0.0008</td>
</tr>
<tr>
<td></td>
<td>PyPy 2.4.0</td>
<td>0.178 ± 0.0006</td>
</tr>
<tr>
<td></td>
<td>Zend 5.5.13</td>
<td>9.070 ± 0.0361</td>
</tr>
</tbody>
</table>
## Composed Richards vs. other VMs

<table>
<thead>
<tr>
<th>Type</th>
<th>VM</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mono</td>
<td>CPython 2.7.7</td>
<td>9.475 ± 0.0127</td>
</tr>
<tr>
<td></td>
<td>HHVM 3.4.0</td>
<td>4.264 ± 0.0386</td>
</tr>
<tr>
<td>Mono</td>
<td>HippyVM</td>
<td>0.250 ± 0.0008</td>
</tr>
<tr>
<td></td>
<td>PyPy 2.4.0</td>
<td>0.178 ± 0.0006</td>
</tr>
<tr>
<td></td>
<td>Zend 5.5.13</td>
<td>9.070 ± 0.0361</td>
</tr>
<tr>
<td>Composed</td>
<td>PyHyp</td>
<td>0.335 ± 0.0012</td>
</tr>
</tbody>
</table>
Datatype conversion

PHPRoot

PHPObject  PHPInt  PHPFunc
Datatype conversion

PHPRoot

PHPObject  PHPInt  PHPFunc

PyRoot

PyObject  PyInt  PyFunc
Datatype conversion: primitive types

PHP  

Python
Datatype conversion: primitive types

PHP
2 : PHPInt

Python
Datatype conversion: primitive types

PHP

2 : PHPInt

Python

2 : PyInt

http://soft-dev.org/
Datatype conversion: user types

PHP

Python

o : PHPObject
Datatype conversion: user types

Diagram:
- PyRoot
  - PyObject
  - PyInt
  - PyFunc
Datatype conversion: user types

PyRoot

PyObject  PyInt  PyFunc

PyPHPAdapter
Datatype conversion: user types

PyRoot

PyObject

PyInt

PyFunc

PyPHPAdapter

php_obj : PHPObjec
Datatype conversion: user types

<table>
<thead>
<tr>
<th>PHP</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o$ : PHPObject</td>
<td></td>
</tr>
</tbody>
</table>
Datatype conversion: user types

PHP

\( o \): \texttt{PHPObject}  

Python

\( :\texttt{PyPHPAdapter} \)
Datatype conversion: user types

**PHP**

```
o : PHPObject
```

**Python**

```
:PyPHPAdapter
```

```
php_obj
```
Datatype conversion: user types

PHP

\( o: \text{PHPObject} \)

Python

\( :\text{PyPHPAdapter} \)

\( \text{php}_\text{obj} \)

\textit{Immutable field}
A good composition needs to reduce *friction*.
A good composition needs to reduce friction. Some examples:

- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)
A good composition needs to reduce friction. Some examples:

- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)
- PHP datatypes are immutable except for references and objects; Python’s are largely mutable (semantic and performance friction)
A good composition needs to reduce friction. Some examples:

- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)
- PHP datatypes are immutable except for references and objects; Python’s are largely mutable (semantic and performance friction)
- PHP has only dictionaries; Python has lists and dictionaries (semantic friction)
Unipyication

PyPy

Pyrolog

Interpreters

Glue

Meta-tracing

Interpreter

Tracing JIT

Software Development Team

http://soft-dev.org/
## Absolute timing comparison

<table>
<thead>
<tr>
<th>VM</th>
<th>Benchmark</th>
<th>Python</th>
<th>Prolog</th>
<th>Python → Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPython-SWI</td>
<td>SmallFunc</td>
<td>0.125s ±0.007</td>
<td>0.257s ±0.002</td>
<td>28.893s ±0.227</td>
</tr>
<tr>
<td></td>
<td>L1A0R</td>
<td>2.924s ±0.284</td>
<td>7.352s ±0.048</td>
<td>9.310s ±0.084</td>
</tr>
<tr>
<td></td>
<td>L1A1R</td>
<td>4.184s ±0.038</td>
<td>18.890s ±0.111</td>
<td>20.865s ±0.067</td>
</tr>
<tr>
<td></td>
<td>NdL1A1R</td>
<td>7.531s ±0.080</td>
<td>18.643s ±0.197</td>
<td>667.682s ±6.895</td>
</tr>
<tr>
<td></td>
<td>TCons</td>
<td>264.415s ±2.250</td>
<td>48.819s ±0.252</td>
<td>2185.150s ±18.225</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>9.374s ±0.059</td>
<td>25.148s ±0.221</td>
<td>2207.304s ±16.073</td>
</tr>
<tr>
<td>Unipyrcation</td>
<td>SmallFunc</td>
<td>0.001s ±0.000</td>
<td>0.006s ±0.001</td>
<td>0.001s ±0.000</td>
</tr>
<tr>
<td></td>
<td>L1A0R</td>
<td>0.085s ±0.000</td>
<td>0.086s ±0.000</td>
<td>0.087s ±0.000</td>
</tr>
<tr>
<td></td>
<td>L1A1R</td>
<td>0.112s ±0.000</td>
<td>0.114s ±0.000</td>
<td>0.115s ±0.000</td>
</tr>
<tr>
<td></td>
<td>NdL1A1R</td>
<td>0.500s ±0.003</td>
<td>0.548s ±0.085</td>
<td>2.674s ±0.012</td>
</tr>
<tr>
<td></td>
<td>TCons</td>
<td>6.053s ±0.288</td>
<td>2.444s ±0.003</td>
<td>36.069s ±0.225</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>0.845s ±0.002</td>
<td>1.416s ±0.003</td>
<td>5.056s ±0.035</td>
</tr>
<tr>
<td>Jython-tuProlog</td>
<td>SmallFunc</td>
<td>0.088s ±0.003</td>
<td>3.050s ±0.053</td>
<td>52.294s ±0.475</td>
</tr>
<tr>
<td></td>
<td>L1A0R</td>
<td>1.078s ±0.009</td>
<td>206.590s ±3.846</td>
<td>199.963s ±2.476</td>
</tr>
<tr>
<td></td>
<td>L1A1R</td>
<td>2.145s ±0.232</td>
<td>293.311s ±5.691</td>
<td>294.781s ±6.193</td>
</tr>
<tr>
<td></td>
<td>NdL1A1R</td>
<td>7.939s ±0.457</td>
<td>1857.687s ±5.169</td>
<td>1990.985s ±15.071</td>
</tr>
<tr>
<td></td>
<td>TCons</td>
<td>543.347s ±3.289</td>
<td>8014.477s ±17.710</td>
<td>8202.362s ±24.904</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>5.661s ±0.046</td>
<td>6981.873s ±18.795</td>
<td>5577.322s ±15.754</td>
</tr>
</tbody>
</table>
## Relative timing comparison

<table>
<thead>
<tr>
<th>VM</th>
<th>Benchmark</th>
<th>Python → Prolog</th>
<th>Prolog → Python</th>
<th>Unipyication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPython-SWI</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>231.770 ×</td>
<td>±13.136</td>
<td>112.567 × ±1.242</td>
<td>27821.079 × ±2331.665</td>
</tr>
<tr>
<td>L1A0R</td>
<td>3.184 ×</td>
<td>±0.300</td>
<td>1.266 × ±0.014</td>
<td>107.591 × ±0.995</td>
</tr>
<tr>
<td>L1A1R</td>
<td>4.987 ×</td>
<td>±0.049</td>
<td>1.105 × ±0.007</td>
<td>181.899 × ±0.590</td>
</tr>
<tr>
<td>NdL1A1R</td>
<td>88.654 ×</td>
<td>±1.368</td>
<td>35.814 × ±0.554</td>
<td>249.737 × ±2.922</td>
</tr>
<tr>
<td>TCons</td>
<td>8.264 ×</td>
<td>±0.101</td>
<td>44.760 × ±0.453</td>
<td>60.583 × ±0.637</td>
</tr>
<tr>
<td>Lists</td>
<td>235.459 ×</td>
<td>±2.314</td>
<td>87.772 × ±1.017</td>
<td>436.609 × ±4.415</td>
</tr>
<tr>
<td><strong>Unipyication</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>1.295 ×</td>
<td>±0.105</td>
<td>0.182 × ±0.054</td>
<td>1.000 ×</td>
</tr>
<tr>
<td>L1A0R</td>
<td>1.020 ×</td>
<td>±0.002</td>
<td>1.012 × ±0.002</td>
<td>1.000 ×</td>
</tr>
<tr>
<td>L1A1R</td>
<td>1.025 ×</td>
<td>±0.002</td>
<td>1.002 × ±0.003</td>
<td>1.000 ×</td>
</tr>
<tr>
<td>NdL1A1R</td>
<td>5.349 ×</td>
<td>±0.045</td>
<td>4.879 × ±0.924</td>
<td>1.000 ×</td>
</tr>
<tr>
<td>TCons</td>
<td>5.959 ×</td>
<td>±0.282</td>
<td>14.756 × ±0.092</td>
<td>1.000 ×</td>
</tr>
<tr>
<td>Lists</td>
<td>5.982 ×</td>
<td>±0.045</td>
<td>3.569 × ±0.026</td>
<td>1.000 ×</td>
</tr>
<tr>
<td><strong>Jython-tuProlog</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>592.904 ×</td>
<td>±19.517</td>
<td>17.143 × ±0.338</td>
<td>50354.204 × ±4341.413</td>
</tr>
<tr>
<td>L1A0R</td>
<td>185.460 ×</td>
<td>±2.818</td>
<td>0.968 × ±0.021</td>
<td>2310.844 × ±28.093</td>
</tr>
<tr>
<td>L1A1R</td>
<td>137.427 ×</td>
<td>±14.537</td>
<td>1.005 × ±0.028</td>
<td>2569.873 × ±52.847</td>
</tr>
<tr>
<td>NdL1A1R</td>
<td>250.776 ×</td>
<td>±14.666</td>
<td>1.072 × ±0.009</td>
<td>744.699 × ±6.726</td>
</tr>
<tr>
<td>TCons</td>
<td>15.096 ×</td>
<td>±0.106</td>
<td>1.023 × ±0.004</td>
<td>227.409 × ±1.592</td>
</tr>
<tr>
<td>Lists</td>
<td>985.149 ×</td>
<td>±8.674</td>
<td>0.799 × ±0.003</td>
<td>1103.206 × ±8.338</td>
</tr>
</tbody>
</table>

Software Development Team

25 / 28

http://soft-dev.org/
What can we use this for?
First-class languages
What can we use this for?

First-class languages

Language migration
Thanks to our funders

- EPSRC: COOLER and Lecture.
- Oracle: various.
Thanks for listening

HTTP://SOFT-DEV.ORG/