Fine-grained language composition

Edd Barrett  Carl Friedrich Bolz  Lukas Diekmann  Laurence Tratt  Naveneetha Krishnan Vasudevan

KING’S College LONDON

Software Development Team
2015-11-07
What to expect from this talk

A

B
What to expect from this talk

A ∪ B
What to expect from this talk

Python ∪ Prolog
What to expect from this talk

Python ∪ PHP
Two levels of challenge

Tooling
Two levels of challenge

Tooling

Language friction
Tooling challenges
Tooling challenges

Python

PHP

PyHyp
Tooling challenges

Language boxes

Python
- Syntax
- Runtime

PHP
- Syntax
- Runtime

PyHyp
- Syntax
- Runtime
Tooling challenges

- Composed meta-tracing VMs
- Language boxes
  - Python
    - syntax
    - runtime
  - PHP
    - syntax
    - runtime
  - PyHyp
    - syntax
    - runtime

4/24 HTTP://SOFT-DEV.ORG/
Syntax composition

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

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

Parser

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

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

Parser

Software Development Team

5 / 24

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

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

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

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

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

LR
Parse Tree
Syntax composition

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

<program>
for (j : js) {
    doStuff();
}
.
.
.
LR
Parse Tree
Undefined

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

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

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

<program>
for (j : js) {
    doStuff();
}
.
.
.
Generalised Parse Tree
```

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

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

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

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

Generalised Parse Tree
Ambiguous

Parse Tree
Syntax composition

\[ <grammar> \\
expr ::= ... \\
term ::= ... \\
    | ... \\
    | ... \\
func ::= ... \\
\]

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

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...
PLY Y
<program>
for (j : js) {
    doStuff();
}
.
.
.
PEG
Parse Tree
Shadows
The only choice?
The only choice?

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

PL X
Interpreter

PL Y
Interpreter

C/C++
Runtime composition

C/C++

Interpreter

Too slow
Runtime composition

PL X
JIT Compiler
Interpreter

PL Y
JIT Compiler
Interpreter

C/C++
Runtime composition

PL X
JIT Compiler
Interpreter

PL Y
JIT Compiler
Interpreter

C/C++

Too much engineering
Runtime composition

PL X

Interpreter

PL Y

Interpreter

JVM/CLR

JIT Compiler

http://soft-dev.org/
Runtime composition
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
Meta-tracing translation with RPython

Diagram:
- Interpreter
- RPython translator
- Optimised Interpreter
- JIT
Meta-tracing translation with RPython

Interpreter

Optimised Interpreter

JIT

you write this
Meta-tracing translation with RPython

you write this

you get this for free

you get this for free

you get this for free
... 
pc := 0 
while 1: 

    instr := load_next_instruction(pc) 
    if instr == POP: 
        stack.pop() 
        pc += 1 
    elif instr == BRANCH: 
        off = load_branch_jump(pc) 
        pc += off 
    elif ...: 
        ... 

Observation: interpreters are big loops.
... pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...

Observation: interpreters are big loops.
Runtime composition recap
Runtime composition recap

- **PL X**: Interpreters
- **PL Y**: Interpreters
- **PL Z**: Interpreter, Tracing JIT

Glue connects PL X and PL Y, and Meta-tracing connects PL Y and PL Z.
Runtime composition recap

PyPy
Hippy
Interpreters

Glue

Meta-tracing

PyHyp
Interpreter
Tracing JIT
## 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></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>Time (μs) ± Error</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>
<tr>
<td>Composed</td>
<td>PyHyp</td>
<td>0.335 ± 0.0012</td>
</tr>
</tbody>
</table>
Datatype conversion

Diagram:
- 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
Datatype conversion: user types

<table>
<thead>
<tr>
<th>PHP</th>
<th>Python</th>
</tr>
</thead>
</table>

16 / 24
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 : PHPObj
Datatype conversion: user types

PHP

\( o : \text{PHPObject} \)

Python

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

PHP

```
o : PHPObject
```

Python

```
:PyPHPAdapter
```

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

PHP

```
o : PHPObj
```

Python

```
:PyPHPAdapter
```

`php_obj`

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)
Unipycation

- PyPy
- Hippy
- Interpreters
- Glue
- Meta-tracing
- PyHyp
  - Interpreter
  - Tracing JIT

18/24 HTTP://SOFT-DEV.ORg/
Unipycation

**PyPy**

**Pyrolog**

**Interpreters**

**Glue**

**Meta-tracing**

**Unipycation**

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