Unipycation

A Case Study in Cross-Language Tracing

Edd Barrett
Carl Friedrich Bolz
Laurence Tratt

KING’S College LONDON
Software Development Team
2013-10-28
Language Composition

PL X
- syntax
- runtime

PL Y
- syntax
- runtime

PL Z
- syntax
- runtime
Language Composition

![Diagram showing language composition]

- PL X
  - Syntax
  - Runtime
- PL Y
  - Syntax
  - Runtime
- PL Z
  - Syntax
  - Runtime

[2 / 21 http://soft-dev.org/]
Language Composition

Requirements

Fast

PL

<<<

Less effort

Current runtime composition techniques?
Traditional Approaches

- **PL X**
- **PL Y**
- **C/C++/JVM**

- Can build fast VMs.
- Development effort.
- JVM

- "Semantic mismatch."
Traditional Approaches

**C/C++**
- Can build fast VMs.
- Development effort.

**PL X**

**PL Y**

**C/C++/JVM**
Traditional Approaches

C/C++
- Can build fast VMs.
- > Development effort.

JVM
- < Development effort.
- “Semantic mismatch”.

PL X

PL Y

C/C++/JVM
Hypothesis:
Meta-tracing may overcome these issues.
Hypothesis:
Meta-tracing may overcome these issues.

- Tracer for free.
  - Cross-language tracing.

- Little engineering effort.
Meta-tracing

PL

Interpreter

Meta-tracing

PL

Interpreter

Tracing JIT
Meta-tracing

PL X

Interpreters

Glue

PL Y

Meta-tracing

PL Z

Interpreter

Tracing JIT
Unipycation

- Python
- Prolog

<table>
<thead>
<tr>
<th>Interpreters</th>
</tr>
</thead>
</table>

| Glue |

| RPython |

<table>
<thead>
<tr>
<th>Interpreter</th>
</tr>
</thead>
</table>

| Tracing JIT |

7 / 21 http://soft-dev.org/
Unipycation

existing interpreters

Python

Interpreters

Pypy

Glue

Prolog

Pyrolog

RPython

Unipycation

Interpreter

Tracing JIT

http://soft-dev.org/
Unipycation

existing interpreters

Python

Prolog

Interpreters

PyPy

Pyrolog

Glue

paradigms

RPython

Unipycation

Interpreter

Tracing JIT
Design Considerations

- type conversion
- database/theory
- partial lists
- cross language calls
- user expectations
- unbound variables
- non-determinism
Where can I get to from ’b’ via at most 4 nodes and how?
Example: Underground/Metro System

- **www server**
- **XML parser**
- **path finder**
- **visualisation**
Example: Underground/Metro System

www server

XML parser

path finder

visualisation

<edge src="a" dest="b" />
<edge src="a" dest="c" />
<edge src="b" dest="e" />
Example: Underground/Metro System

www server

XML parser

directed graph

path finder

paths

visualisation

<edge src="a" dest="b" />
<edge src="a" dest="c" />
<edge src="b" dest="e" />

route finder v1.0
Example: Underground/Metro System

www server
XML parser
path finder
visualisation

Python
import bs4
...

Python
import tkinter
...

10 / 21 http://soft-dev.org/
Example: Underground/Metro System

www server

XML

XML parser

directed graph

path finder

paths

visualisation

Prolog

path() :-


import bs4, uni, tkinter

# ...code for parsing would be here

edges = { node : sucessors(node) for node in ... }  
# e.g. edges = {
#     "a" : ["b", "c"],
#     "b" : ["e", "h"],
#     ...
# }

# prolog helper
def get_edges(src_node):
    return iter(edges[src_node])
import bs4, uni, tkinter

# ...code for parsing would be here

dges = { node : successors(node) for node in ... }
# e.g. edges = {
#    "a" : ["b", "c"],
#    "b" : ["e", "h"],
#    ...
# }

# prolog helper
def get_edges(src_node):
    return iter(edges[src_node])
import bs4, uni, tkinter

# ...code for parsing would be here

edges = { node : successors(node) for node in ... }  
# e.g. edges = { 
#     "a" : ["b", "c"], 
#     "b" : ["e", "h"], 
#     ...
# }

# prolog helper
def get_edges(src_node):
    return iter(edges[src_node])
import bs4, uni, tkinter

# ...code for parsing would be here

edges = { node : successors(node) for node in ... }  
# e.g. edges = {
#     "a" : ["b", "c"],
#     "b" : ["e", "h"],
#     ...
# }

# prolog helper
def get_edges(src_node):
    return iter(edges[src_node])
Example: Underground/Metro System

e = uni.Engine(""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead |], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
""
)

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
Example: Underground/Metro System

e = uni.Engine("""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead ], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
"""
)

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
Example: Underground/Metro System

```python
e = uni.Engine(""

    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead |], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
""

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```
Example: Underground/Metro System

```prolog
e = uni.Engine('""'
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
""

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```
Example: Underground/Metro System

e = uni.Engine(""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead ], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
""
)

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
Example: Underground/Metro System

```
e = uni.Engine(""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead ], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
"")

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```
Example: Underground/Metro System

```
e = uni.Engine(""")
  path(From, To, MaxLen, Nodes) :-
    path(From, To, MaxLen, Nodes, 1).
  path(Node, Node, _, [Node], _).
  path(From, To, MaxLen, [From | Ahead], Len) :-
    Len < MaxLen, edge(From, Next),
    Len1 is Len + 1,
    path(Next, To, MaxLen, Ahead, Len1).

  edge(From, To) :- python:get_edges(From, To).
  """)

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
  # show path in GUI...
```
Example: Underground/Metro System

e = uni.Engine(""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).

    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead ], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).

    edge(From, To) :- python:get_edges(From, To).
""
)

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
Example: Underground/Metro System

```python
path(From, To, MaxLen, Nodes) :-
    path(From, To, MaxLen, Nodes, 1).

path(Node, Node, _, [Node], _).
path(From, To, MaxLen, [From | Ahead |], Len) :-
    Len < MaxLen, edge(From, Next),
    Len1 is Len + 1,
    path(Next, To, MaxLen, Ahead, Len1).

edge(From, To) :- python:get_edges(From, To).

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```
Example: Underground/Metro System

```python
e = uni.Engine(""
  path(From, To, MaxLen, Nodes) :-
    path(From, To, MaxLen, Nodes, 1).

  path(Node, Node, __, [Node], __).
  path(From, To, MaxLen, [From | Ahead |], Len) :-
    Len < MaxLen, edge(From, Next),
    Len1 is Len + 1,
    path(Next, To, MaxLen, Ahead, Len1).

  edge(From, To) :- python:get_edges(From, To).
"")
paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```

```python
def get_edges(src_node):
    return iter(edges[src_node])
```
Example: Underground/Metro System

```python
from uni import Engine

e = Engine(""
    path(From, To, MaxLen, Nodes) :-
        path(From, To, MaxLen, Nodes, 1).
    path(Node, Node, _, [Node], _).
    path(From, To, MaxLen, [From | Ahead], Len) :-
        Len < MaxLen, edge(From, Next),
        Len1 is Len + 1,
        path(Next, To, MaxLen, Ahead, Len1).
    edge(From, To) :-
        python:get_edges(From, To).
"

paths = e.db.path.iter
for (to, nodes) in paths("b", None, 4, None):
    # show path in GUI...
```

How much effort was our composition?
Development Effort

- # LoC: 600
- 4 months

http://soft-dev.org/
Experimental Methodology

func1

func2

micro-benchmark
design

15 / 21 http://soft-dev.org/
Experimental Methodology

1. .py
   .py

2. .pl
   .pl

func1
---
func2

micro-benchmark
design
Experimental Methodology

Baseline measurements

1. py
   ---------
   py

2. pl
   ---------
   pl
Experimental Methodology
Experimental Methodology

1. python
2. python
3. python
4. python

Diagram:

1. python
   - python

2. python
   - python

3. python
   - python

4. python
   - python
Experimental Methodology

Performance comparison
7 micro-benchmarks, e.g.: 
Experimental Methodology

7 micro-benchmarks, e.g.:

Benchmark: SmallFunc

```
func1 func2
```
call

Benchmark: Lists

```
[1,2,3,...]
```

produce consume
Experimental Methodology

7 micro-benchmarks, e.g.:

Benchmark: SmallFunc

```
func1 -> call -> func2 (inlineable)
```

Benchmark: Lists

```
[1,2,3,...] -> produce (func1) -> call -> consume (func2)
```
Experimental Evaluation

Performance of Synthetic Benchmarks

Time (normalised as a ratio of Python)

- **SmallFunc**
- **LoopArgResult**
- **LoopArg1Result**
- **NonDetLoop**
- **Lists**
- **PythonInstances**
- **TermConstruction**

- **Python**
- **Prolog**
- **Python->Prolog**
- **Prolog->Python**

- 98.6895
- 149.7116
In Summary

[Image of a stick figure with a less than sign]
Confirmed
In Summary

Promising
Future Work

What is next?
Future Work

Faster
for i in path(X, Y, 3, [a, b])

Syntax
Future Work

PL vs. PL' vs. jvm rpython

PL
jvm

vs.

PL'
rpython