RPython
A Step Towards Reconciling Dynamically and Statically Typed OO Languages

Antonio Cuni – DISI, Università degli Studi di Genova

DLS'07 OOPSLA Montreal CA

October 22, 2007

Dynamic languages for .NET and JVM

- .NET and JVM: widespread platforms
- Designed for static languages
- Great Python implementations: **IronPython**, **Jython**
- Much slower than e.g. C# and Java

Dynamic vs. static

Dynamic languages

- Flexibility
- Rapid development cycle
- **Metaprogramming**

Static languages

- Speed
- Nothing more :-)

Dynamic vs. static

Dynamic languages

- Flexibility
- Rapid development cycle
- **Metaprogramming**

Static languages

- Speed
- Nothing more :-)

RPython Quick Facts

- Restricted subset of Python
- Statically typed (type inference)
- Still allows metaprogramming
- RPython programs still run under {C,J,Iron}Python
- Three backends: C, .NET, JVM
- Almost as fast as C, C#, Java

Type inference

- Top-down, starting from an entry point; whole program analysis
- Assign the most precise type to each variable
- Fail if you try to mix incompatible types

RPython

```
def main():
    print add(40, 2)

def add(a, b):
    return a+b
```

Not RPython

```
def fn(flag):
    if flag:
        return 42
    else:
        return 'hello'
```

Type inference

- Top-down, starting from an entry point; whole program analysis
- Assign the most precise type to each variable
- Fail if you try to mix incompatible types

RPython

```
def main():
    print add(40, 2)

def add(a, b):
    return a+b
```

Not RPython

```
def fn(flag):
    if flag:
        return 42
    else:
        return 'hello'
```

Type inference

- Top-down, starting from an entry point; whole program analysis
- Assign the most precise type to each variable
- Fail if you try to mix incompatible types

RPython

```
def main():
    print add(40, 2)

def add(a, b):
    return a+b
```

Not RPython

```
def fn(flag):
    if flag:
        return 42
    else:
        return 'hello'
```

Other restrictions

- Globals are assumed to be constant
- `yield` and generators not supported
- No special __methods__ (except __init__ and __del__)
- No run-time definition of new functions and classes
- Cannot modify classes at run-time
- Cannot change the __class__ of an object
- Single inheritance, with limited support for mixins

Still pythonic, though

- No syntactic restriction
- Functions and classes are first-order values
- Exceptions work

Lists and dictionaries

- Work, but they must be homogeneous
- list of int, dict from string to floats, etc. are OK
- list of *int and strings* is not
- Most of methods of `list`, `dict` and `str` are supported

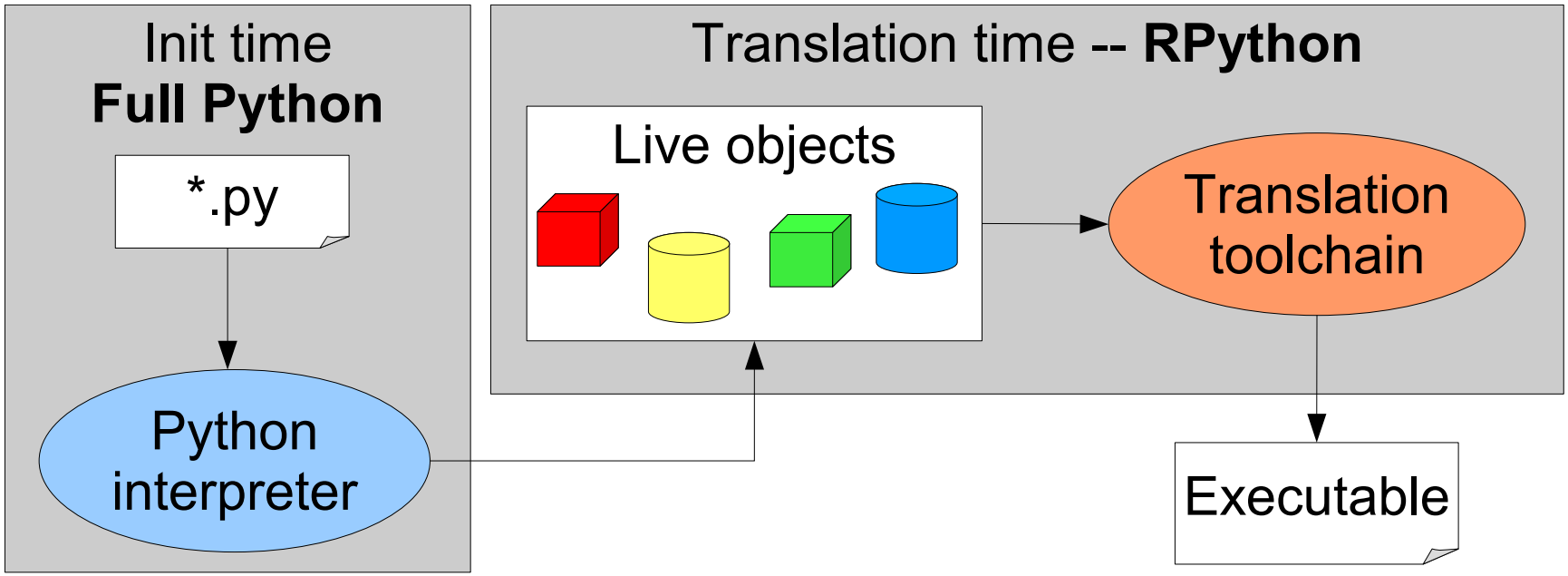
Still pythonic, though

- No syntactic restriction
- Functions and classes are first-order values
- Exceptions work

Lists and dictionaries

- Work, but they must be homogeneous
- list of int, dict from string to floats, etc. are OK
- list of *int and strings* is not
- Most of methods of `list`, `dict` and `str` are supported

Init-time, translation-time, run-time



Metaprogramming

- RPython restrictions only apply to live objects
- No restrictions about how they are created
  - Full Python is allowed at init-time
- Python as a metaprogramming language for RPython
- Code generation considered harmful

Compute complex constants

Fibonacci's numbers

```
def fibo(N):
    sequence = []
    a, b = 1, 1
    for i in xrange(N):
        sequence.append(a)
        a, b = b, a+b
    return sequence

# computed at init-time
fibo_numbers = fibo(100)
```

Metaclasses run at init-time

__extend__ metaclass

```
class MyClass(object):
    def foo(self):  ...

class __extend__(MyClass):
    def bar(self):  ...

def main():
    obj = MyClass()
    obj.bar()
```

Dynamic classes/functions at init-time

"Static" nested scopes work

```
def make_adder(N):
    def add(x):
        return x+N
    return add

add10 = make_adder(10)
add20 = make_adder(20)

def main():
    print add10(32)
    print add20(22)
```

The Translation Toolchain

- **CPython**: *.py --> Python bytecode
- **FlowObjSpace**: bytecode --> flow graphs
- **Annotator**: type inference on flow graphs
  - High level Python types (`List(Integer)`)
- **RTyper**: high level types -> low level types
  - lltype for C, ootype for CLI and JVM
- **Backends**: code generation
  - C, CLI (.NET), JVM

Benchmarks

- Classic Martin Richard's test
- Available in Java, C#, RPython

Language	**Result**	**Factor**
Results on Microsoft CLR		
C#	6.94 ms	1.00x
RPython	7.25 ms	1.04x
IronPython	1675.00 ms	241.35x
Results on JVM		
Java	1.77 ms	1.00x
RPython	2.10 ms	1.18x
Jython	2918.90 ms	1641.80x

What's good about RPython

- Pythonic enough to be usable
- Very fast
- Testable under CPython

Things to improve

- Originally an implementation detail
- Not designed to be user-friendly; terse error messages
- Lack of documentation/reference manual
- Lack of separate compilation
- Integration with the hosting platform
  - Good for C/Posix
  - Proof of concept for .NET
  - Doesn't exist for JVM

About PyPy (1)

Python in (R)Python

- High level interpreter written in RPython
- Easy to understand
- Easy to extend

Translation Toolchain

- Written in full Python
- Works as a general compiler
- Especially for interpreters (e.g. Javascript, Prolog)

About PyPy (1)

Python in (R)Python

- High level interpreter written in RPython
- Easy to understand
- Easy to extend

Translation Toolchain

- Written in full Python
- Works as a general compiler
- Especially for interpreters (e.g. Javascript, Prolog)

About PyPy (2)

Low-level aspects inserted by the TT

- Garbage collector
- Threading model/Stackless
- Additional language features
- JIT compiler (only for the C backend so far)

PyPy you can get

- pypy-c (about 2x slower than CPython)
- pypy-c-jit (up to 60x **faster** than CPython)
- pypy.NET (about 6x slower than IronPython)
- pypy-jvm (about 30% **faster** than Jython)

About PyPy (2)

Low-level aspects inserted by the TT

- Garbage collector
- Threading model/Stackless
- Additional language features
- JIT compiler (only for the C backend so far)

PyPy you can get

- pypy-c (about 2x slower than CPython)
- pypy-c-jit (up to 60x **faster** than CPython)
- pypy.NET (about 6x slower than IronPython)
- pypy-jvm (about 30% **faster** than Jython)

Acknowledgments

- The whole PyPy Team
  - RPython is **not** mine :-)
- Davide Ancona
- Massimo Ancona
- Nicholas D. Matsakis