Experiences of implementing a VM with RPython

Laurence Tratt
http://tratt.net/laurie/

King's College London

2012/01/05

Language designers dilemma

Language designers dilemma

- How much implementation do I need to do to show my language design is good?

Language designers dilemma

- How much implementation do I need to do to show my language design is good?
- If too little: it will be dismissed out of hand as unusably slow.

Language designers dilemma

- How much implementation do I need to do to show my language design is good?
- If too little: it will be dismissed out of hand as unusably slow. *I may not even be sure myself that it can be made adequately efficient.*

Language designers dilemma

- How much implementation do I need to do to show my language design is good?
- If too little: it will be dismissed out of hand as unusably slow. *I may not even be sure myself that it can be made adequately efficient.*
- If too much: low-level fiddling will have consumed energy that should have gone into design.

Language implementation

The traditional routes:

Language implementation

The traditional routes:

- *Interpretation*: [a wonderfully vague phrase] e.g. running directly on ASTs. Pros: very easy. Cons: too slow.

Language implementation

The traditional routes:

- *Interpretation*: [a wonderfully vague phrase] e.g. running directly on ASTs. Pros: very easy. Cons: too slow.
- *Compilation*: compile to C. Pros: can be very efficient. Cons: hard to realise the full promise; often unbearably slow tool chain.

Language implementation

The traditional routes:

- *Interpretation*: [a wonderfully vague phrase] e.g. running directly on ASTs. Pros: very easy. Cons: too slow.
- *Compilation*: compile to C. Pros: can be very efficient. Cons: hard to realise the full promise; often unbearably slow tool chain.
- *VMification*: compile to bytecode. Pros: neatly separates out execution from compilation. Cons: use someone else's or create your own?

VMs: using someone else's

- Take an Off The Shelf (OTS) VM: e.g. JVM / HotSpot or CLR.
- Huge amounts of work spent optimising OTS VMs (rough estimate for HotSpot ~1000-4000 man years). Let's use that hard work!

VMs: using someone else's

- Take an Off The Shelf (OTS) VM: e.g. JVM / HotSpot or CLR.
- Huge amounts of work spent optimising OTS VMs (rough estimate for HotSpot ~1000-4000 man years). Let's use that hard work!
- But a VM necessarily reflects a specific language (or group of language's) semantics.
- Semantic mismatches destroy both language implementer efficiency and execution times.

VMs: using someone else's

- Take an Off The Shelf (OTS) VM: e.g. JVM / HotSpot or CLR.
- Huge amounts of work spent optimising OTS VMs (rough estimate for HotSpot ~1000-4000 man years). Let's use that hard work!
- But a VM necessarily reflects a specific language (or group of language's) semantics.
- Semantic mismatches destroy both language implementer efficiency and execution times.
- e.g. if you want tail calls on the JVM you'll have to painfully encode them; similarly for continuations on the CLR.
- e.g. Jython at best tends to reach CPython performance levels.

VMs: write your own

- You have a language which you can't really use an existing VM for.

VMs: write your own

- You have a language which you can't really use an existing VM for.
- This is where I found myself in 2004 (or so) with the Converge language: its Icon-based expression evaluation system allows backtracking. Try encoding that in an OTS VM!
- I took the traditional route: a C based VM.

VMs: write your own

- You have a language which you can't really use an existing VM for.
- This is where I found myself in 2004 (or so) with the Converge language: its Icon-based expression evaluation system allows backtracking. Try encoding that in an OTS VM!
- I took the traditional route: a C based VM.
- The first version was horrendous.

VMs: write your own

- You have a language which you can't really use an existing VM for.
- This is where I found myself in 2004 (or so) with the Converge language: its Icon-based expression evaluation system allows backtracking. Try encoding that in an OTS VM!
- I took the traditional route: a C based VM.
- The first version was horrendous.
- The second was merely awful.

Converge VM # 2

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.

Converge VM # 2

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.
- Highlights: implements C-level continuations using `setjmp` / `longjmp` for ease of library writing; uses conservative GC when necessary, so most C code needn't care about freeing memory.

Converge VM # 2

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.
- Highlights: implements C-level continuations using `setjmp` / `longjmp` for ease of library writing; uses conservative GC when necessary, so most C code needn't care about freeing memory.
- Lowlights: slow and increasingly hard to maintain / optimise.

Converge VM # 2

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.
- Highlights: implements C-level continuations using `setjmp` / `longjmp` for ease of library writing; uses conservative GC when necessary, so most C code needn't care about freeing memory.
- Lowlights: slow and increasingly hard to maintain / optimise.
- *Sophisticated uses of Converge's most interesting features are so slow that it brings their utility into question.*

PyPy and RPython

- The PyPy project is really two things:
  - PyPy: a new Python VM.

PyPy and RPython

- The PyPy project is really two things:
  - PyPy: a new Python VM.
  - RPython: a language suited to writing VMs.

PyPy and RPython

- The PyPy project is really two things:
  - PyPy: a new Python VM.
  - RPython: a language suited to writing VMs.
- My belief: **RPython will revolutionise language design and implementation.**

PyPy and RPython

- The PyPy project is really two things:
  - PyPy: a new Python VM.
  - RPython: a language suited to writing VMs.
- My belief: **RPython will revolutionise language design and implementation.**
- Why?

RPython: overview

- RPython is a *strict* subset of Python. Every RPython program can be run with Python (but not the other way around).

RPython: overview

- RPython is a *strict* subset of Python. Every RPython program can be run with Python (but not the other way around).
- The subset allows meaningful whole-program static analysis.
- e.g. every RPython program must be statically typeable (with a roughly Java-esque type system).

RPython: overview

- RPython is a *strict* subset of Python. Every RPython program can be run with Python (but not the other way around).
- The subset allows meaningful whole-program static analysis.
- e.g. every RPython program must be statically typeable (with a roughly Java-esque type system).
- RPython translates the RPython program into efficient C.

RPython: overview

- RPython is a *strict* subset of Python. Every RPython program can be run with Python (but not the other way around).
- The subset allows meaningful whole-program static analysis.
- e.g. every RPython program must be statically typeable (with a roughly Java-esque type system).
- RPython translates the RPython program into efficient C.
- So what?

RPython for VMs

- RPython is a ‘full’ programming language, albeit rather strict: it won’t be e.g. replacing PHP.
- It is however rather well suited to VMs (and VM authors).

RPython for VMs

- RPython is a ‘full’ programming language, albeit rather strict: it won’t be e.g. replacing PHP.
- It is however rather well suited to VMs (and VM authors).
- First pro: you have a high-level language which gives you good GC, useful datatypes (e.g. dictionaries) and so on.

RPython for VMs

- RPython is a ‘full’ programming language, albeit rather strict: it won’t be e.g. replacing PHP.
- It is however rather well suited to VMs (and VM authors).
- First pro: you have a high-level language which gives you good GC, useful datatypes (e.g. dictionaries) and so on.
- Second pro: it provides a neat FFI to C. Integrating libraries and system calls is easy and efficient.

RPython for VMs

- RPython is a ‘full’ programming language, albeit rather strict: it won’t be e.g. replacing PHP.
- It is however rather well suited to VMs (and VM authors).
- First pro: you have a high-level language which gives you good GC, useful datatypes (e.g. dictionaries) and so on.
- Second pro: it provides a neat FFI to C. Integrating libraries and system calls is easy and efficient.
- Third pro: it gives you a *custom JIT for your language for free*.

RPython JITs (1)

- With < 10 lines of user code, RPython gifts a tracing JIT: hot loops in the language the VM implements translate into machine code.
- [RPython automatically generates guards too, so stale, or incorrect, machine code isn't executed.]

RPython JITs (1)

- With < 10 lines of user code, RPython gifts a tracing JIT: hot loops in the language the VM implements translate into machine code.
- [RPython automatically generates guards too, so stale, or incorrect, machine code isn't executed.]
- *The JIT isn't for RPython, it's for the language the VM is written for*.

RPython JITs (1)

- With < 10 lines of user code, RPython gifts a tracing JIT: hot loops in the language the VM implements translate into machine code.
- [RPython automatically generates guards too, so stale, or incorrect, machine code isn't executed.]
- *The JIT isn't for RPython, it's for the language the VM is written for*.
- No magic needed.
- RPython assumes (reasonably) VMs have a main loop. A couple of annotations do the rest.

Converge VM # 3 (1)

- I decided to experiment with creating an RPython VM for Converge. How far could I get?

Converge VM # 3 (1)

- I decided to experiment with creating an RPython VM for Converge. How far could I get?
- Size: ˜5.5KLoC.

Converge VM # 3 (1)

- I decided to experiment with creating an RPython VM for Converge. How far could I get?
- Size: ˜5.5KLoC. *vs. 13KLoC*.
- Effort: under 3 man months (started Sep 1st, feature complete by Dec 19th; fun activities such as new job and new course to teach soaked up much time). *vs. 18 man months*.

Converge VM # 3 (2)

- The Dec 19th version of the VM is extremely readable: a textbook (i.e. simple) VM.

Converge VM # 3 (2)

- The Dec 19th version of the VM is extremely readable: a textbook (i.e. simple) VM.
- Virtually 100% bytecode compatible with the old VM. Only major change: a single stack to a per-function stack (to help the RPython optimiser).

Converge VM # 3 (2)

- The Dec 19th version of the VM is extremely readable: a textbook (i.e. simple) VM.
- Virtually 100% bytecode compatible with the old VM. Only major change: a single stack to a per-function stack (to help the RPython optimiser).
- Code is still malleable: e.g. a week ago I moved from full RPython continuations (rather slow) to RPython generators. Touched every builtin function, and core parts of the VM. Time taken < 1 day.

Converge VM # 3 (3)

- Performance: current benchmarks range between 3x (large programs) and 20x time faster (micro-benchmarks).

Converge VM # 3 (3)

- Performance: current benchmarks range between 3x (large programs) and 20x time faster (micro-benchmarks).
- Initial boost: good GC and string hashing (slot lookups).
- Both come for free from RPython.

Converge VM # 3 (3)

- Performance: current benchmarks range between 3x (large programs) and 20x time faster (micro-benchmarks).
- Initial boost: good GC and string hashing (slot lookups).
- Both come for free from RPython.
- Later boost: the JIT.

Converge VM # 3 (3)

- Performance: current benchmarks range between 3x (large programs) and 20x time faster (micro-benchmarks).
- Initial boost: good GC and string hashing (slot lookups).
- Both come for free from RPython.
- Later boost: the JIT.
- Note: the JIT works best if the VM is written in a specific style; changes often permeate the whole VM.

Converge VM # 3 (3)

- Performance: current benchmarks range between 3x (large programs) and 20x time faster (micro-benchmarks).
- Initial boost: good GC and string hashing (slot lookups).
- Both come for free from RPython.
- Later boost: the JIT.
- Note: the JIT works best if the VM is written in a specific style; changes often permeate the whole VM.
- Some work started on this; many optimisations still left. Some will require bytecode changes. Some will hurt VM readability...
- ...but at least a further 2x speed up seems plausible.

RPython: the bad points

- Nothing is perfect.

RPython: the bad points

- Nothing is perfect.
- Documentation is sparse (though that should change soon). Takes a fairly dedicated programmer to create a VM.

RPython: the bad points

- Nothing is perfect.
- Documentation is sparse (though that should change soon). Takes a fairly dedicated programmer to create a VM.
- Translation is ‘whole program’: the VM is translated in one go. Every time you change something. This is very slow. Quick (low optimised) Converge VM translation is about 3-4 minutes. PyPy (which is much bigger) is 30-60 minutes. Ouch.

RPython: the bad points

- Nothing is perfect.
- Documentation is sparse (though that should change soon). Takes a fairly dedicated programmer to create a VM.
- Translation is 'whole program': the VM is translated in one go. Every time you change something. This is very slow. Quick (low optimised) Converge VM translation is about 3-4 minutes. PyPy (which is much bigger) is 30-60 minutes. Ouch.
- Incorrect RPython programs often trigger assertions in the translator (not very helpful).
- Static analysis leads to error messages which exceed exotically typed languages like Haskell in their complexity and baroqueness.
- Identifying the cause can be tricky.

Summary

- The use of OTS VMs tends to constrain language implementations and, therefore, language designs.

Summary

- The use of OTS VMs tends to constrain language implementations and, therefore, language designs.
- RPython is the first in a new class of languages: those suited to creating custom JIT VMs.
- It allows ‘fast enough’ VMs to be created in ‘fast enough’ time.

Summary

- The use of OTS VMs tends to constrain language implementations and, therefore, language designs.
- RPython is the first in a new class of languages: those suited to creating custom JIT VMs.
- It allows ‘fast enough’ VMs to be created in ‘fast enough’ time.
- There’s no reason why similar languages can’t be created, but RPython is here now and demonstrably works.

Summary

- The use of OTS VMs tends to constrain language implementations and, therefore, language designs.
- RPython is the first in a new class of languages: those suited to creating custom JIT VMs.
- It allows ‘fast enough’ VMs to be created in ‘fast enough’ time.
- There’s no reason why similar languages can’t be created, but RPython is here now and demonstrably works.
- **This new class of language changes the rules of the language implementation game**.

Summary

- The use of OTS VMs tends to constrain language implementations and, therefore, language designs.
- RPython is the first in a new class of languages: those suited to creating custom JIT VMs.
- It allows ‘fast enough’ VMs to be created in ‘fast enough’ time.
- There’s no reason why similar languages can’t be created, but RPython is here now and demonstrably works.
- **This new class of language changes the rules of the language implementation game**.

Thank you for listening.