PyPy – where we are now

(So no talk about our plans to conquer the world)

PyCon 2006, Dallas, Texas

Michael Hudson mwh@python.net
Heinrich-Heine-Univeristät Düsseldorf

Christian Tismer tismer@stackless.com
tismerysoft GmbH

What is PyPy?

- PyPy is:
  - An implementation of Python, written in Python
  - An open source project
  - A STREP (“Specific Targeted REsearch Project”), partially funded by the EU
  - A lot of fun!

Demo

- We can currently produce a binary that looks very much like CPython to the user
- It's fairly slow (around the same speed as Jython)
- Can also produce binaries that are more capable than CPython -- stackless, thunk, ...
- Talk is mainly about how we got here

Overview

- PyPy has two major components:
  - The “standard interpreter” which implements the evaluator loop and object semantics of CPython
  - The analysis tool chain that can analyze the standard interpreter and, for example, compile it to C
- Also some generic tools

The Standard Interpreter

- Written in a mostly-static subset of Python ("RPython") which means it runs on CPython too
- Consists of a parser/compiler, a bytecode interpreter and the Standard Object Space
- Functionally equivalent to CPython

The “What is RPython?” question

- Restricted Python, or RPython for short, is a subset of Python that is static enough for our analysis toolchain to cope with
- First and foremost it *is* Python -- this lets us unit test our standard interpreter
- Definition is basically “what our tools accept” -- so changes as toolchain does

The Interpreter/ Object Space split

- The byte code interpreter treats objects as black boxes and consistently references a “space” object to manipulate them
- This allows us to use a funky object space to help analysis (later)
- The Standard Object Space implements objects that look very much like CPython’s

The Parser/ Compiler

- The standard interpreter includes a parser and bytecode compiler for Python
- Based on work by Jonathan David Riehl and CPython's compiler package (but with less bugs)
- Allows/will allow runtime modification of syntax and grammar

The Analysis Tool Chain

- Has four main parts:
  - The Flow Object Space
  - The Annotator
  - The RTyper
  - The Low Level backend

Flow Object Space

- Technically speaking, an “abstract domain” for the bytecode interpreter
- Treats objects as either “Constants” or “Variables”
- Works on a code object at a time
- Produces a control flow graph
- Basically stable since early 2005

The Annotator

- The RPythonAnnotator analyzes an entire RPython program to infer types and inter-function control flow
- Interesting stuff, but well documented -- see “Compiling dynamic language implementations” on the website for more.
- More-or-less stable since early summer 2005

The RTyper

- First of all: "the RTyper" is badly named
- Converts the still-fairly-high-level output of the annotator into lower level concepts that are easier to translate into languages like C
- Makes decisions about memory layout of objects of translated program
- Basically working since summer 2005

The Low-Level Backend(s)

- Take the low-level operations produced by the RTyper and converts to a low-level language
- At time of writing, C, LLVM and JavaScript(!) are the supported targets
- C backend present but not stable from late 2004

A little about the project

- Open Source, of course (MIT license)
- About 10 people work on PyPy full time
- Distributed -- full timers live in six(?) countries, contributors from several more
- Welcoming -- watch out, you might get hired! (easier for Europeans, admittedly)
- Sprinting after the conference

Current work

- Writing a Just In Time compiler
- Integration of logic programming/constraint solving
- Modularizing the garbage collection system
- Generally refactoring – a noticeable trend has been to do less and at source generation time