IST FP6-004779

PYPY

Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Lanugage and Community

STREP

IST Priority 2

D04.4: Release PyPy as a Research tool for Experimental Language Enhancements

Due date of deliverable: August 2005

Actual Submission date: 23th December 2005

Start date of Project: 1st December 2005   **Duration: 2 years**

Lead Contractor of this WP: Strakt

Authors: Samuele Pedroni (AB Strakt), Ludovic Aubry (Logilab)

<div align="right">

Revision: final

</div>

Abstract

This document describes the 0.8 release of the PyPy project, made on the 3rd of November 2005. The release is specifically intended to be used for experimental language enhancements.

Contents

1 Executive Summary

Traditional compilers turn source code into a series of tokens, which are then handled by a parser which generates the output codes. In modern compilers, there is still the same tokenizing step, but the parser generates an Abstract Syntax Tree as an intermediate representation. The output codes are then generated by a later pass which traverses the AST. There are several advantages to this approach, including easier and better identification of errors in the input, the possibility to manipulate the AST before the code generation pass and a simpler and more systematic code generator.

While CPython uses the traditional approach, PyPy implements an AST compiler written in RPython - a proper subset of the Python language. This allows us to compile the AST compiler into a static executable which can either be used as a stand alone bytecode compiler or be integrated as a part of the PyPy interpreter. It also provides the flexibility needed to easily add new syntax. This allows extending the language by hooking at the syntactic level.

The PyPy architecture also has a clear division between the bytecode interpreter and the object space, the latter implementing a well-defined interface (D04.2). This division easily enables hooks at the operational level and allows extending the language at the semantic level.

Both of these kinds of flexibility are needed for work packages 9 and 10, which focus on aspect-oriented programming, design-by-contract, advanced static checking capabilities, constraint satisfaction algorithms and inference engine constructs.

The 0.8 release contains both a complete and translatable AST-based compiler. It also contains an experimental object space, the *thunk object space*, which show-case semantic level extensibility.

The thunk object space adds lazy evaluated objects which are not present in conventional Python, and works by wrapping the Standard object space. The 0.8 release enables the experimental translation of the *thunk object space* through our toolchain.

2 Introduction

2.1 Purpose of this Document

This document describes the release 0.8 version of the PyPy interpreter. It documents the new availability of a flexible parser and bytecode compiler, making it reasonably easy to add new constructs to the Python language. It also illustrates briefly the *thunk object space* as a showcase for semantic flexibility.

2.2 Scope of this Document

This document gives an overview of the 0.8 release as a base platform for experimental language enhancements both at the syntactic and semantic level. It does not deal with any aspects of the translation of the PyPy interpreter into lower level code as such nor any aspects of optimization, even though such elements were part of the 0.8 release of the project. A more detailed description of the parsing of Python source code, the building of an Abstract Syntax Tree and the generation of bytecode from the AST is presented in D04.3 Report about the parser and bytecode compiler implementation (D04.3) .

2.3   Related Documents

This document has been prepared in conjunction with other tasks in work packages 3, 4 and 5. Before reading this document, a close look at the following deliverables is recommended.

- D04.2 Report about "D04.2 Complete Python implementation running on top of CPython" (D04.2), which contains a more general overview about our architecture and more detailed description of parts of the implementation.
- D04.3 Report about the parser and bytecode compiler implementation (D04.3)

3   Goals of the 0.8.0 Release

The third public release within the PyPy project, version numbered 0.8.0 was released on November, 3rd 2005.

The release is available as

- http://codespeak.net/download/pypy/pypy-0.8.0.tar.bz2,
- http://codespeak.net/download/pypy/pypy-0.8.0.tar.gz and
- http://codespeak.net/download/pypy/pypy-0.8.0.zip.

This deliverable distributes the completed results of WP4, Task 3. It also include some preparatory and incremental improvements along with some initial work for Phase 2:

Complete the interpreter with a Python source parser and bytecode compiler. Collaborate with various core developers involved with new parsing and compiling techniques.

- Implement a Python parser and bytecode compiler in Python by leveraging existing research and implementation work. Reuse code and ideas from ongoing open source projects.
- Design a flexible way to insert the compiler into the code base, easing future experimentation with the syntax.
- More generally, research and provide bridges between the interpreter source and the code it interprets, allowing the interpreter to delegate some of its "core" work to regular, interpreted, pure Python code. Decide when it is appropriate to make this regular code visible to language users (e.g. as modules, modifiable at run-time).

Our translatable compiler is based on the 'compiler' package shipped with CPython as part of the standard Python library. This is a pure Python implementation of a bytecode compiler for the language. CPython uses its internal compiler written in C for normal operations. The 'compiler' package is meant to be an extensible compiler for users that have such a need.

The compiler is based on traversal of the abstract syntax trees produced by our own parser. Trees are traversed twice, first to gather scoping information for names and second to emit code in the form of a flow graph containing bytecode instructions, which in a final phase is flattened and serialised as a bytecode string proper.

In our efforts to make the CPython 'compiler' package translatable we had to simplify it and make it conform to RPython restrictions. Also, CPython uses its own C-implemented version of the compiler and does particularly not use the Python-implemented compiler package. In the process of porting this original package to RPython, we discovered a number of unexpected problems and bugs. This made the porting more involved and time-consuming than expected.

4   The "Thunk" Object Space

The "Thunk" object space implements lazy evaluated objects:

```
PyPy 0.8.0 in thunk(StdObjSpace) on top of Python 2.4.1
(startuptime: 25.33 secs)
>>>> def f():
....    print "computing"
....    return 6*7
....
>>>> x = thunk(f)
>>>> x
computing
42
>>>> y = thunk(f)
>>>> y.__class__
computing
<type 'int'>
```

The 'thunk' builtin creates such objects taking a function corresponding to the computation we are deferring. *Forcing* will happen when the object is operated on. By itself, lazy computations are nothing new in language implementations but the ease with which we could implement it with the PyPy architecture (152 Lines of Code) proves the flexibility and extensibility of the 0.8 PyPy release.

The Thunk Object Space is implemented by wrapping our standard object space in such a way that arguments to space operations are *forced*, this exploits the object space interface as a surface to hook into language semantics.

In more detail, all objects grow a field possibly pointing to an object that should be used instead of them, or to a special marker in the case of a thunk. *Forcing* will return the content of the field or invoke the function in the thunk and return its result while at the same time caching it in the field so that it will be used directly the next time the thunk is operated on which will result in the stored object being used instead.

These changes require very little code, mostly the definition of the new "thunk" builtin; the wrapping of the standard space is achieved by using the reflective manipulation allowed by Python on classes at bootstrap time, where RPython limitations do not apply.

Summarising, the *thunk* object space illustrates how one can experiment with the Python language using the flexibility provided by PyPy the language implementation and PyPy the translation toolchain.

5 Mixing Application-level and Interpreter-level

PyPy builtin modules, which in CPython are written entirely in C, are implemented as mixed-modules (D04.2).

Module implementations are free to use and mix code written at interpreter-level (in RPython) or at application-level (pure Python). We have support through gateway classes (D04.2) to generate stubs at bootstrap time that allows application-level functions to call interpreter-level ones as if they were application-level. This is obviously how all builtins written at interpreter-level are exposed. Conversely, gateways can be generated to call application-level functions as if they lived at interpreter-level.

In a later phase of the project, we expect to experiment more with the possibilities opened by mixed-modules, mainly in terms of user application level code hooking into and overriding builtin module functionality.

6 The Public Announcement of the Release

The following release announcement was sent out to various mailing lists:

```
pypy-0.8.0: Translatable compiler/parser and some more speed
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

The PyPy development team has been busy working and we've now packaged
our latest improvements, completed work and new experiments as
version 0.8.0, our third public release.

The highlights of this third release of PyPy are:

- Translatable parser and AST compiler. PyPy now integrates its own
  compiler based on Python's own 'compiler' package but with a number
  of fixes and code simplifications in order to get it translated
  with the rest of PyPy.  This makes using the translated pypy
  interactively much more pleasant, as compilation is considerably
  faster than in 0.7.0.

- Some Speed enhancements. Translated PyPy is now about 10 times
  faster than 0.7 but still 10-20 times slower than
  CPython on pystones and other benchmarks.  At the same time,
  language compliance has been slightly increased compared to 0.7
  which had already reached major CPython compliance goals.

- Some experimental features are now translatable.  Since 0.6.0, PyPy
  shipped with an experimental Object Space (the part of PyPy
  implementing Python object operations and manipulation) implementing
  lazily computed objects, the "Thunk" object space. With 0.8.0 this
  object space can also be translated preserving its feature
  additions.

What is PyPy (about)?
++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 PyPy is a MIT-licensed research-oriented reimplementation of
 Python written in Python itself, flexible and easy to
```

experiment with. It translates itself to lower level
languages. Our goals are to target a large variety of
platforms, small and large, by providing a compilation toolsuite
that can produce custom Python versions. Platform, Memory and
Threading models are to become aspects of the translation
process - as opposed to encoding low level details into a
language implementation itself. Eventually, dynamic
optimization techniques - implemented as another translation
aspect - should become robust against language changes.

Note that PyPy is mainly a research and development project
and does not by itself focus on getting a production-ready
Python implementation although we do hope and expect it to
become a viable contender in that area sometime next year.

PyPy is partially funded as a research project under the
European Union's IST programme.

Where to start?
++++++++++++++++++++++++

Getting started: http://codespeak.net/pypy/dist/pypy/doc/getting-started.html

PyPy Documentation: http://codespeak.net/pypy/dist/pypy/doc/

PyPy Homepage: http://codespeak.net/pypy/

The interpreter and object model implementations shipped with
the 0.8 version can run on their own and implement the core
language features of Python as of CPython 2.4. However, we still
do not recommend using PyPy for anything else than for education,
playing or research purposes.

Ongoing work and near term goals
+++++++++++++++++++++++++++++++++++++

At the last sprint in Paris we started exploring the new directions of
our work, in terms of extending and optimizing PyPy further. We
started to scratch the surface of Just-In-Time compiler related work,
which we still expect will be the major source of our future speed
improvements and some successful amount of work has been done on the
support needed for stackless-like features.

This release also includes the snapshots in preliminary or embryonic
form of the following interesting but yet not completed sub projects:

- The OOtyper, a RTyper variation for higher-level backends
  (Squeak, ...)
- A JavaScript backend
- A limited (PowerPC) assembler backend (this related to the JIT)
- some bits for a socket module

PyPy has been developed during approximately 16 coding sprints across
Europe and the US. It continues to be a very dynamically and
incrementally evolving project with many of these one-week workshops
to follow.

```
PyPy has been a community effort from the start and it would
not have got that far without the coding and feedback support
from numerous people.   Please feel free to give feedback and
raise questions.

    contact points: http://codespeak.net/pypy/dist/pypy/doc/contact.html


have fun,

    the pypy team, (Armin Rigo, Samuele Pedroni,
    Holger Krekel, Christian Tismer,
    Carl Friedrich Bolz, Michael Hudson,
    and many others: http://codespeak.net/pypy/dist/pypy/doc/contributor.html)

PyPy development and activities happen as an open source project
and with the support of a consortium partially funded by a two
year European Union IST research grant. The full partners of that
consortium are:

    Heinrich-Heine University (Germany), AB Strakt (Sweden)
    merlinux GmbH (Germany), tismerysoft GmbH (Germany)
    Logilab Paris (France), DFKI GmbH (Germany)
    ChangeMaker (Sweden), Impara (Germany)
```

7   Glossary of Abbreviations

The following abbreviations may be used within this document:

7.1   Technical Abbreviations:

AST	Abstract Syntax Tree
CPython	The standard Python interpreter written in C. Generally known as "Python". Available from www.python.org.
codespeak	The name of the machine where the PyPy project is hosted.
docutils	The Python documentation utilities.
GenC backend	The backend for the PyPy translation toolsuite that generates C code.
GenLLVM backend	The backend for the PyPy translation toolsuite that generates LLVM code.
Graphviz	Graph visualisation software from AT&T.
Jython	A version of Python written in Java.
LLVM	Low Level Virtual Machine - a compiler infrastructure available from University of Illinois at Urbana-Champaign
LOC	Lines of code.
Object Space	A library providing objects and operations between them, available to the bytecode interpreter via a well-defined API.

Pygame	A Python extension library that wraps the Simple Directmedia Library - a cross-platform multimedia library designed to provide fast access to the graphics framebuffer and audio device.
ReST	reStructuredText, the plaintext markup system used by docutils.
RPython	Restricted Python; a less dynamic subset of Python in which PyPy is written.
Standard Interpreter	The subsystem of PyPy which implements the Python language. It is divided in two components: the bytecode interpreter, and the standard object space.
Standard Object Space	An object space which implements creation, access and modification of regular Python application level objects.

7.2   Partner Acronyms:

DFKI	Deutsches Forschungszentrum für künstliche Intelligenz
HHU	Heinrich Heine Universität Düsseldorf
Strakt	AB Strakt
Logilab	Logilab
CM	Change Maker
mer	merlinux GmbH
tis	Tismerysoft GmbH

References

(D04.2) Report about "D04.2 Complete Python implementation running on top of CPython"

(D04.3) D04.3 Report about the parser and bytecode compiler implementation