Dynamic Languages in Practice

Python Dynamicity & Other Ideas

Beatrice Åkerblom

Department of Computer and Systems Sciences
Stockholm University
beatrice@dsv.su.se

“Historically” in Language Research

- Type inference (Smalltalk, Various Python projects, Diamondback Ruby)
- Gradual typing
- Soft typing
- Pluggable types

- Generally tries to make dynamic languages more “controllable” and predictable, that is static
- Assumptions are made about programs

Approaches used before

Selected examples:

- “Usually, no further properties are defined after the initialization and the type of the properties rarely changes.”
  -- Peter Thiemann

- “Giving people a dynamically-typed language does not mean that they write dynamically-typed programs”
  -- John Aycock

- “Yet while the presence of such abundant dynamism makes traditional static optimization impossible, in most programs, there is surprisingly little dynamism present.”
  -- Michael Salib

Approaches used before

Selected examples:

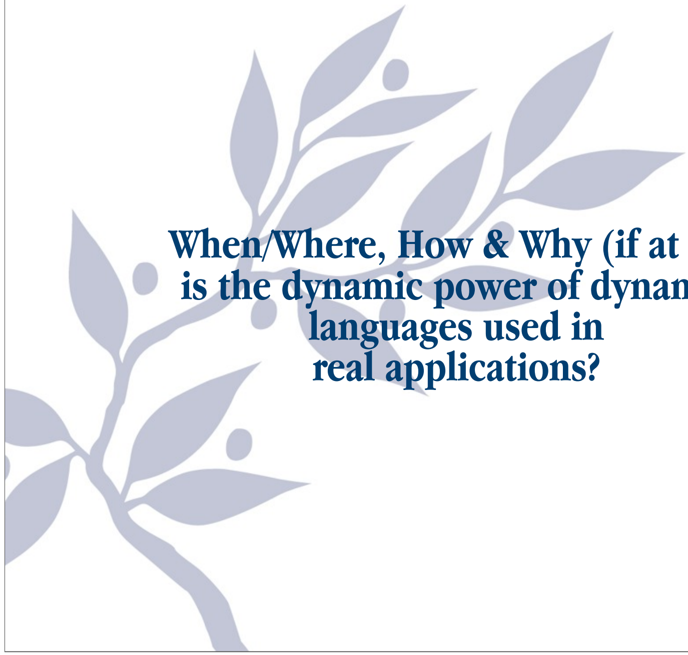
- “Usually, no further properties are defined after the initialization and the type of the properties rarely changes.”
  -- Peter Thiemann

- “Giving people a dynamically-typed language does not mean that they write dynamically-typed programs”
  -- John Aycock

- “Yet while the presence of such abundant dynamism makes traditional static optimization impossible, in most programs, there is surprisingly little dynamism present.”
  -- Michael Salib

True?

We don’t know



When/Where, How & Why (if at all) is the dynamic power of dynamic languages used in real applications?

What's Dynamic?

do variables change type?
will different paths lead to different types?
how polymorphic are method calls?
can common supertypes be found?

- Duck typing - polymorphism without need for inheritance or declared interfaces
- Dynamic features - introspection, reflection, reification

What's Dynamic?

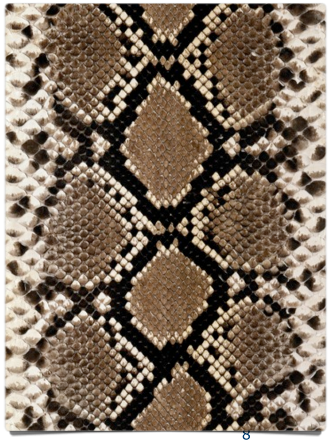
- Duck typing - polymorphism without need for inheritance or declared interfaces
- Dynamic features - introspection, reflection, reification

what is the program?
do our objects reflect the classes?
how dynamic are variable accesses, etc?

Why is this important?

We'll be able to:

- know how much of a "typical" Python program could be annotated with types
- know how well Python source code does represent the running program
- know to what extent we need to support dynamic behaviour e.g. when building tools or new language constructs for Python
- emphasize the focus on how Python is used when designing new constructs



Why is this important?

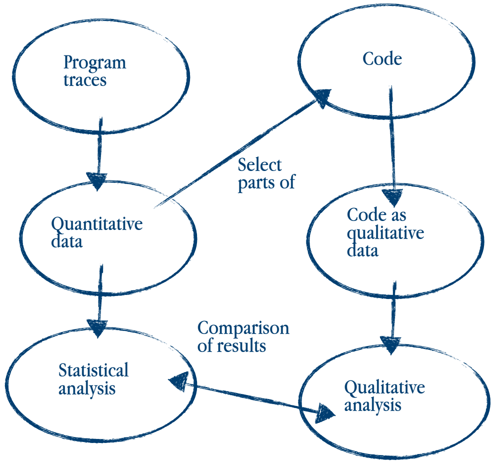
We'll be able to:

- know how much of a "typical" [redacted] program could be annotated with types
- know how well [redacted] source code does represent the running program
- know to what extent we need to support dynamic behaviour e.g. when building tools or new language constructs [redacted]
- emphasize the focus on how [redacted] is used when designing new constructs

Different Sources, different methods

- Programs (Quantitative)
  - Static analysis (what is the program?)
  - Dynamic analysis: Measure behaviour at runtime, e.g. use of language constructs, inheritance hierarchies, polymorphic call sites, etc.
- Code snippets (Qualitative)
  - Search for language constructs usage patterns
  - Read to understand how/why
- Programmers (Sociological)
  - Interview
  - Observe

Different Sources, different methods



Program traces

Code

Select parts of

Quantitative data

Code as qualitative data

Comparison of results

Statistical analysis

Qualitative analysis

What Have We Done?

- Modified the Python 2.6 interpreter to log information about running programs
  - method and function calls
  - instance member access
  - use of dynamic features
- Python programs selected from Source Forge
- Programs run on a Debian machine
  - interactive
  - tests
  - examples
- Program runs documented
  - tests
  - recordings
  - use cases

Dynamic Features in Python Programs

- Anomos, Bleachbit, Comix, ConvertAll, Exaile, Kodos, Mcomix, Pysolfc, Rednotebook, Retext, Sbackup, Solfege, Task coach, Torrent Search, Wikidpad, Zmail

- hasattr, eval, reload, getattr, __delattr__, __getattr__, execfile, __getattribute__, del attribute, __import__, exec, setattr, vars, __setattr__, delattr

0: Id-nummer
1: the path, filename and row number from which the call was made,
2: Caller id.
3: Caller type.
4: Target Id
5: Target type
6: Feature name
7: Argument types
8: Results

Number of Features Used by Programs

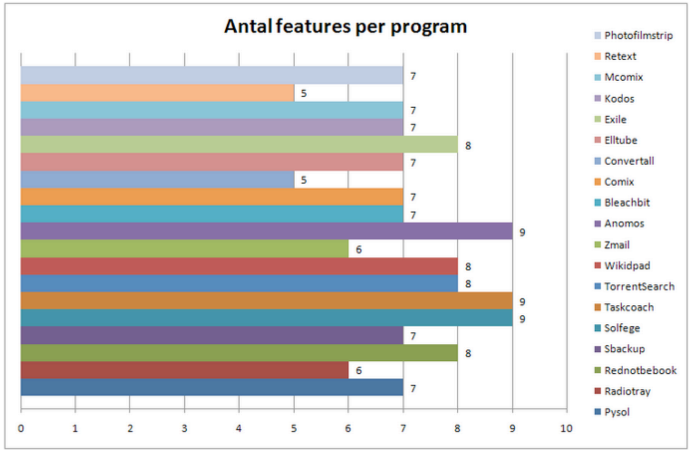
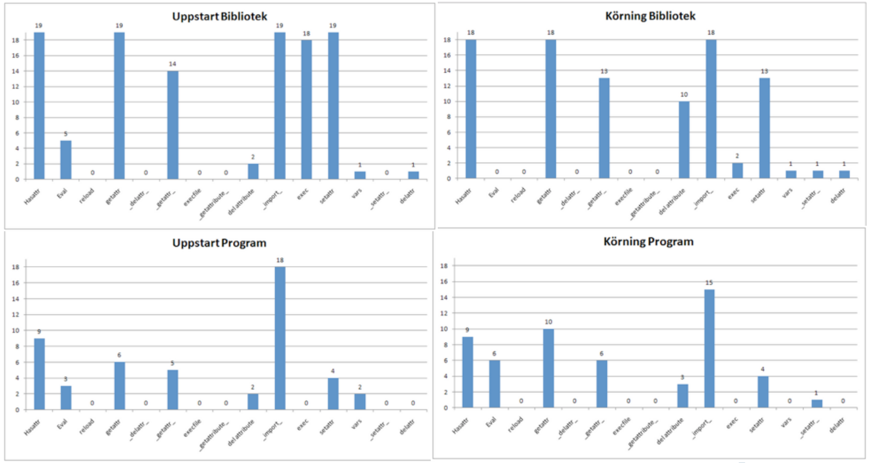


Chart by Jonatan Stendahl och Mattias Tumlin

When and Where?



Charts by Jonatan Stendahl och Mattias Tumlin

Polymorphism in Python Programs

- Task Coach, SciPy, Pootle, Virtaal & the Translate Toolkit, PhotoFilmStrip, Brain Workshop, Eric4, PyMol, Childsplay, GNU Solfege, WikiPad, BleachBit, Mnemosyne, RedNotebook, DispcalGUI, Scikit Learn, Python parsing module, PDF-Shuffler, Link checker, Mcomix, Python megawidgets, Autocomplete for Notepad++, PyTruss, Idle, Radiotray

0. Event ID
1. Source file path
2. Caller ID (current this at the call-site)
3. Caller type
4. Target ID (the receiver of the method call)
5. Class name of target + : + class id
6. Name of called function/method
7. Argument types
8. Call line
9. A list of all super classes of the target type

Questions Asked

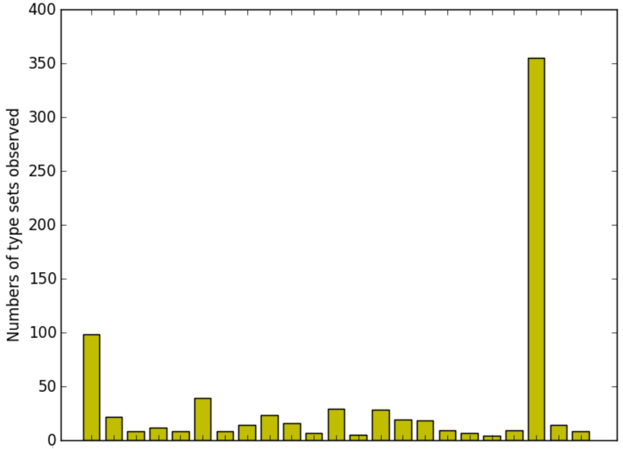
- How many unique call-sites?
- How many call-sites are monomorphic?
  - Trivially monomorphic vs. monomorphic
- How many polymorphic call-sites?
- Distribution of the degree of polymorphism seen

- For call-sites that saw several different types as receiver, what were the types and do they share a common supertype containing the method called?

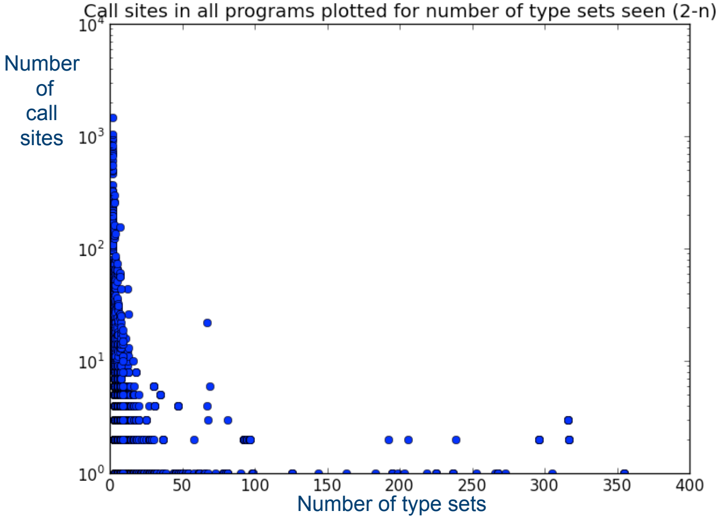
Monomorphic Call Sites

- Trivially monomorphic: We have only recorded one single execution of this call site
- Monomorphic: We have recorded more than one execution of this call site, and the types seen were almost the same (or the same set)

How Polymorphic are Python Call Sites?



How Polymorphic are Python Call Sites?



What do you think we would find if we did the same thing for Ruby?