Assignments

Assignment 1

- Download
  - Python 3.2.3 or later (http://www.python.org/)
  - Z3 'unstable' for your platform (http://z3.codeplex.com/)
  - Git client (http://www.github.com/)
  - Clone https://github.com/thomasjball/PyExZ3.git
- Or get the code I have on USB key for Windows (and Z3 for all platforms)
- Write a Python function to encode n-bit multiplication using Z3 Bools (you can use PyExZ3\examples\adder.py)
- Use Z3 to prove that your multiplier is equivalent to Z3's BitVector multiplier (you can use PyExZ3\examples\check_adder.py)

Assignment 2

1. Get new PyExZ3 from me or from www.github.com/thomasjball

2. PyExZ3 regression tests should pass (python run_tests.py test)

3. Write and submit new test cases
   - At least one test case that passes
   - At least one that shows off a deficiency in the implementation
   - Send email with new tests to tball@microsoft.com - I will add to github

Assignment 3: create a SymbolicDictionary

- Use Z3's array theory to support Python's dictionary (dict), modelling the following operations
  - Length: __length__(self)
  - Get: __getitem__(self,key)
  - Set: __setitem__(self,key,value)
  - Lookup: __contains__(self,key)

- Should support SymbolicInteger as a key and a value

- What about?
  - Delete: __delitem__(self,key)

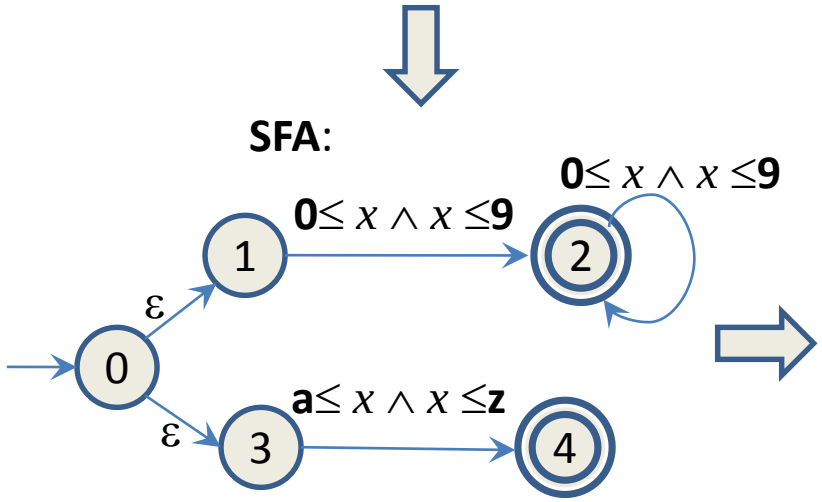
Assignment 4

- Add basic support for strings into PyExZ3

- Using the theory of lists, how much can we do?
  - Constants
  - Concatenation (+)
  - Substring matching (and extraction?)

SFA axioms ($Th(A)$)

Regex: **\d+|[a-z]**



Theory

(y:List<Σ>)
$Acc_0(y) \Leftrightarrow Acc_1(y) \vee Acc_3(y)$
$Acc_1(y) \Leftrightarrow y\neq[] \wedge$ '0'≤first(y)≤'9' $\wedge Acc_2(rest(y))$
$Acc_2(y) \Leftrightarrow y=[] \vee$
$(y\neq[] \wedge$ '0'≤first(y)≤'9' $\wedge Acc_2(rest(y)))$
$Acc_3(y) \Leftrightarrow y\neq[] \wedge$ 'a'≤first(y)≤'z' $\wedge Acc_4(rest(y))$
$Acc_4(y) \Leftrightarrow y=[]$

Note: a move $(p, \varphi[x], q)$ encodes the *set* of transitions

$$\{(p, x^M, q) \mid M \vDash \varphi[x]\}$$

Code Review

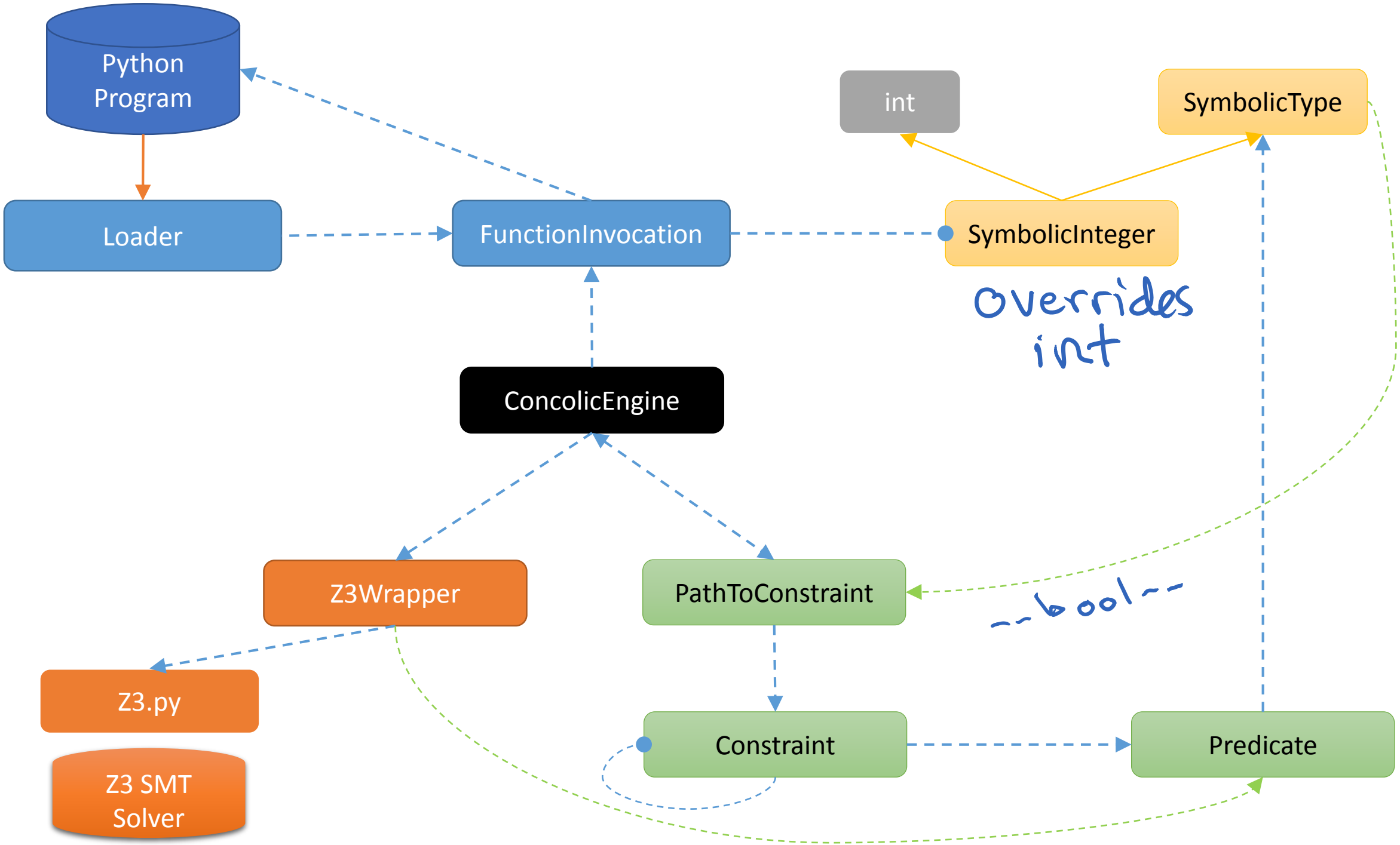
Design and Implementation of Dynamic Symbolic Execution (for Python, in Python)

https://github.com/thomasjball/PyExZ3

Classes

- Loader
- FunctionInvocation
- SymbolicType
  - SymbolicInteger
- PathToConstraint
- Constraint
- Predicate
- Z3Wrapper
- ConcolicEngine

- Identify the code under test (CUT)
- Identify symbolic inputs
- Reinterpret instructions
- Trace the control flow
- Collect path constraint
- Generate new input from path constraint
- Restart execution of CUT (from initial state)
- Search strategy to expose new paths


Python Program
Loader
FunctionInvocation
int
SymbolicType
SymbolicInteger
overrides int
ConcolicEngine
Z3Wrapper
PathToConstraint
__bool__
Z3.py
Z3 SMT Solver
Constraint
Predicate

Loader

Loader

FunctionInvocation

Uses reflection to

- load the code under test and identify function entry point F
- determine the number of arguments to F

- Creates a FunctionInvocation object to encapsulate
  - entry point F and
  - symbolic argument values

- Creates a SymbolicInteger for each argument

symbolic\loader.py

SymbolicType

SymbolicType

- An abstract base class representing a pair of
  - a concrete value of type T
  - a symbolic value of type T (represented by an abstract syntax tree)

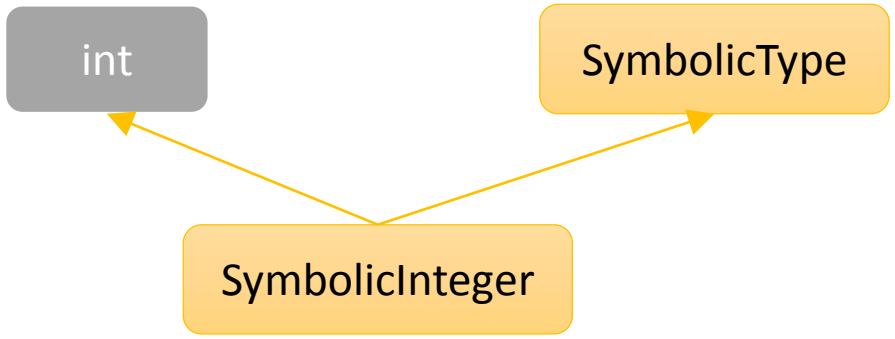
- Overrides basic object operations:

  - Comparisons: __eq__, __ne__, __lt__, __le__, __gt__, __ge__

  - Coercion to Boolean: __bool__

symbolic\symbolic_types

SymbolicInteger



- A SymbolicInteger is
  - a Python int, and
  - a SymbolicType

- SymbolicInteger overloads arithmetic operations for which we know how to translate to logic and solver with Z3

- For other operations, we default to concrete execution

symbolic\symbolic_types

Python Execution

x : SymbolicInt ⟨ 0, "x" ⟩

y = x + 1

y : SymbolicInt ⟨ 1, "x" + 1 ⟩

Intercepting Control-flow in Python

- Conditionals
  - if e1, while e1, e1 and e2 , e1 or e2, not e

- *Any* object can be used in a conditional test
  - Python calls __bool__ method to get a Boolean from object
  - Used whenever a conditional test (predicate) encountered

- We override __bool__ in order to intercept control-flow and determine which way predicate will evaluate (true, false)

The Power of the Object Protocol

- As long as code is using the Python object protocol, comparisons via SymbolicType will be visible

- Example: print(self.toString())
  - D = [j for j in range(len(A)) if A[j]]
  - if x in D # implies equality comparison of x against each element of D

- Sometimes the runtime uses concrete value rather than object protocol (for efficiency)
  - if A[j] # j is cast out of object domain to a system32

Predicate

Predicate

- Tracks a Boolean expression in the program and which direction it took (T,F)

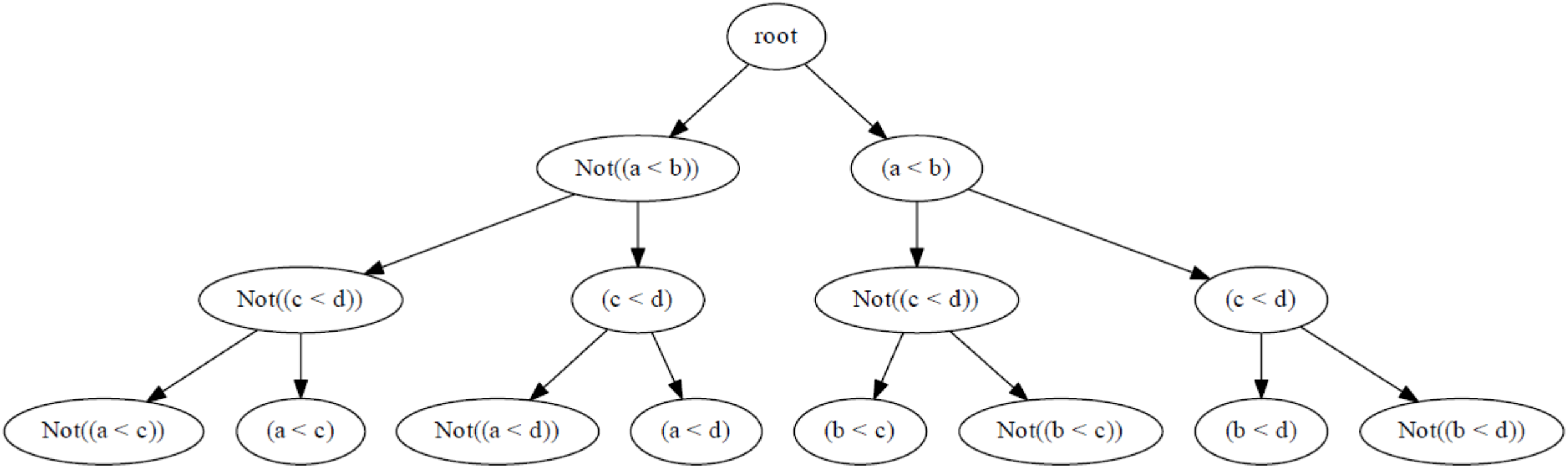
Constraint

Constraint

- A sequence of predicates corresponding to an execution path

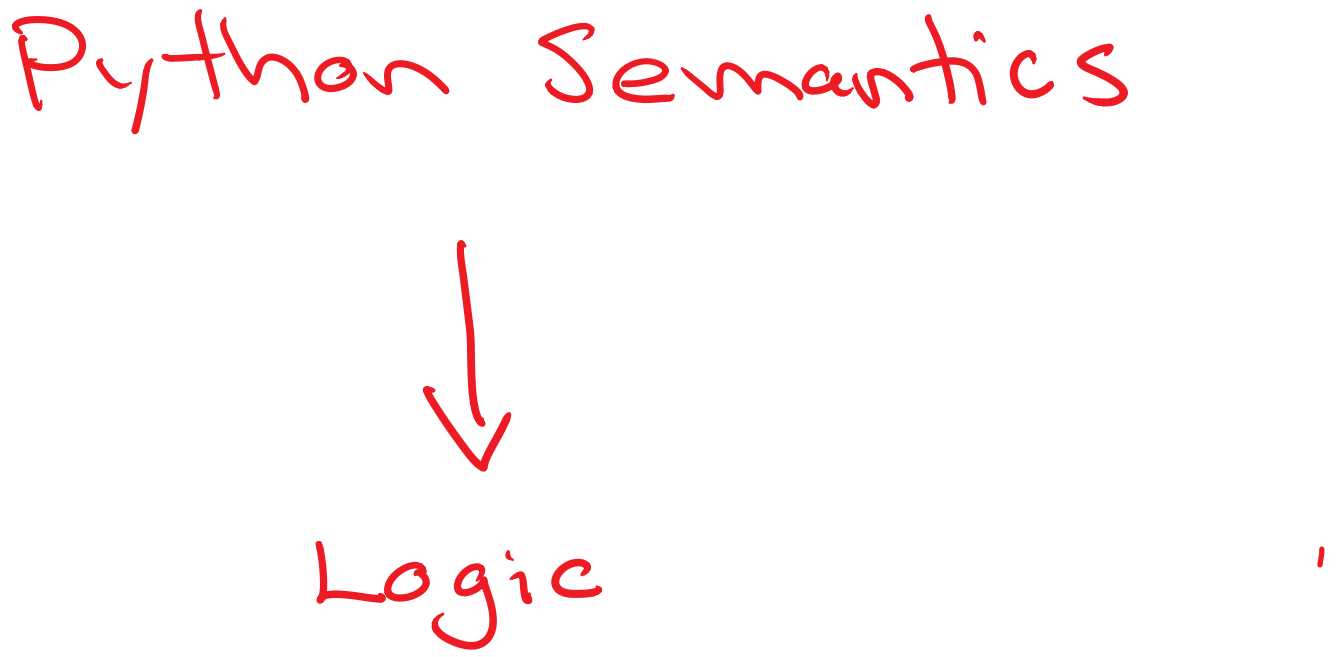
PathToConstraint

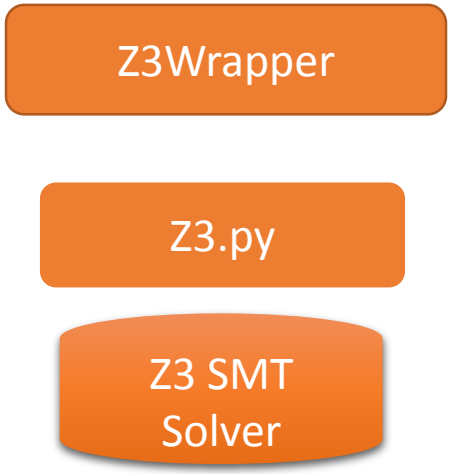
PathToConstraint



Z3Wrapper

- Translate from AST expression (in SymbolicType) into Z3 expression





ConcolicEngine

ConcolicEngine

- Generational search procedure
- Uses a queue to perform breadth-first exploration of paths