Program Analysis and Testing
Exam 1 Review

Khalid Hourani

March 1, 2022

1 Python for PAT

1.1 Inspecting objects in python

Python objects can be inspected with a handful of built-in functions.

function	description
`help`	Invoke the built-in help system
`type`	With one argument, return the type of an object
`dir`	Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object
`id`	Return the identity of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime.
`getattr`	Return the value of the named attribute of object
`callable`	Return `True` if the object argument appears callable, `False` if not.

1.2 Magic functions

A *Magic Method* is a function (always beginning and ending with `__`, called a *dunderstore*). Examples are given in **??**.

function	description
`__new__`	Called to create a new instance of class cls.
`__init__`	Called after the instance has been created (by `__new__()`), but before it is returned to the caller.
`__del__`	Called when the instance is about to be destroyed.
`__repr__`	Called by the `repr()` built-in function to compute the "official" string representation of an object
`__str__`	Called by `str(object)` and the built-in functions `format()` and `print()` to compute the "informal" or nicely printable string representation of an object.

1.3 Syntactic sugar

Syntactic Sugar is syntax within a programming language that is designed to make things easier to read or to express. For example, a function decorator can be used as shorthand for function composition:

```
@decorator
def func():
    # do whatever
```

is equivalent to

```
def func(args):
  # do whatever
func = decorator(func)
```

Other examples of syntactic sugar:

Compound inequalities:

```
1 < x < 10
```

```
1 < x and x < 10
```

List comprehension:

```
arr = [x for x in range(10)]
```

```
arr = []
for x in range(10):
    arr.append(x)
```

1.4 Regular expression

A *regular expression* is a sequence of characters that specify a search pattern in text.

```
re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
```

?? gives an outline of regular expression syntax.

expression	explanation
`.`	(Dot.) In the default mode, this matches any character except a newline. If the `DOTALL` flag has been specified, this matches any character including a newline.
`^`	(Caret.) Matches the start of the string, and in `MULTILINE` mode also matches immediately after each newline.
`$`	Matches the end of the string or just before the newline at the end of the string.
`*`	Causes the resulting RE to match 0 or more repetitions of the preceding RE
`+`	Causes the resulting RE to match 1 or more repetitions of the preceding RE
`?`	Causes the resulting RE to match 0 or 1 repetitions of the preceding RE.
`[]`	Used to indicate a set of characters

2 Concepts and application of concepts in PAT

2.1 Program Concrete/Abstract/Symbolic State

2.2 State space

2.3 Overapproximation

2.4 Reachability

2.5 Safety and Liveness properties

2.6 Meta-morphic relations

2.7 Undecidablity

2.8 Satisifiability

3 Control flow graph

A *control flow graph* is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

3.1 Basic blocks

The *nodes* in the graph correspond to regions of source code. A *basic block* is maximal program region with a single entry and single exit point.

3.2 Transitions

The (directed) *edges* of the graph correspond to the possibility that program execution proceeds from the end of one region directly to the beginning of another.

4 Data flow

4.1 Def/Use

Definition: where a variable gets a value

- Variable declaration (often the special value uninitialized)
- Variable initialization
- Assignment
- Values received by a parameter

Use: extraction of a value from a variable

- Expressions
- Conditional statements
- Parameter passing
- Returns

4.2 Def-use pairs

A *def-use* (du) pair associates a point in a program where a value is produced with a point where it is used.

4.3 Def/Use in presence of references

4.4 Data flow algorithms

Suppose we are calculating the reaching definitions of node v, and there is an edge (p, v) from an immediate predecessor node p.

- If the predecessor node p can assign a value to variable x, then the definition x_p reaches v. We say the definition x_p is generated at p.
- If a definition x_p of variable x reaches a predecessor node p, and if x is not redefined at that node (in which case we say the x_p is killed at that point), then the definition is propagated on from p to v.

Worklist algorithm iterate to a fixed point solution.
General idea:

- Initially all nodes are on the work list, and have default values
- Default for "any-path" problem is the empty set, default for "all-path" problem is the set of all possibilities (union of all gen sets)
- While the work list is not empty
  - Pick any node n on work list; remove it from the list
  - Apply the data flow equations for that node to get new values
  - If the new value is changed (from the old value at that node), then
    - Add successors (for forward analysis) or predecessors (for backward analysis) on the work list
- Eventually the work list will be empty (because new computed values = old values for each node) and the algorithm stops.