Homework 5

Kevin Yang - 50244152

November 8, 2021

Link to repo:
https://github.com/keviny2/CPSC532W-Assignments/tree/main/HOPPL

Program 0

Code Snippets:

```
class Env(PClass):
    local_map = field()
    outer = field()

    def find(self, var):
        "Find the innermost Env where var appears."
        return self.local_map if (var in self.local_map) else self.outer.find(var)


class Procedure(PClass):
    "A user-defined Scheme procedure."
    parms = field()
    body = field()
    env = field()
    sig = field()

    def __call__(self, *args):
        return evaluate(self.body, Env(local_map=pmap(dict(zip(self.parms, args))), outer=self.env), self.sig)


def standard_env():
    "An environment with some Scheme standard procedures."
    env = Env(local_map=pmap(penv), outer=None)
    return env
```

```python
def evaluate(exp, env=None, sig=None):

    # evaluate the wrapper fn
    if env is None:
        env = standard_env()
        proc, sig = evaluate(exp, env, sig)
        return proc('')  # list with an empty string for pmap constructor for __call__ function in Procedure class

    # simply return constants and variables
    if isinstance(exp, str):         # variable reference
        try:
            return env.find(exp)[exp], sig
        except AttributeError:  # not found in environment, so just be a string primitive
            return exp, sig
    elif not isinstance(exp, list):  # constant
        return torch.tensor(float(exp)), sig

    op, *args = exp
    if op == 'sample':
        d, sig = evaluate(args[1], env, sig)
        return d.sample(), sig
    if op == 'observe':
        # sample from prior for now
        d, sig = evaluate(args[1], env, sig)
        return d.sample(), sig
    if op == 'if':             # conditional
        (test, conseq, alt) = args
        exp = (conseq if evaluate(test, env, sig)[0] else alt)
        return evaluate(exp, env, sig)
    elif op == 'define':         # definition
        (symbol, exp) = args
        env[symbol] = evaluate(exp, env, sig)[0]
    elif op == 'fn':             # procedure
        (parms, body) = args
        return Procedure(parms=parms[1:], body=body, env=env, sig=sig), sig
    else:                        # procedure call
        proc, sig = evaluate(op, env, sig)
        vals = [evaluate(arg, env, sig)[0] for arg in args[1:]]

        # if proc is a Procedure, we do not need to return the sig b/c it implicitly calls evaluate
        if isinstance(type(proc), type(Procedure)):
            return proc(*vals)
        else:
            return proc(*vals), sig
```

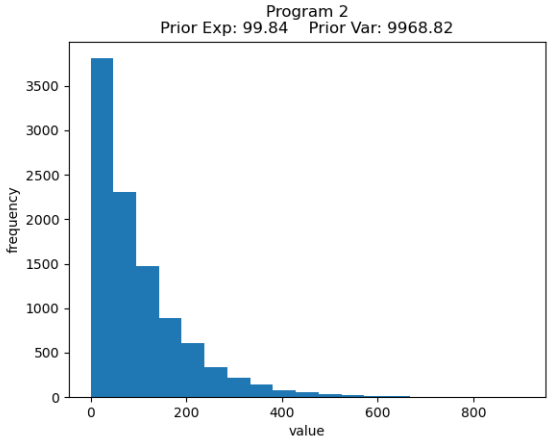
```python
env = {
    'alpha': '',
    'normal': Normal,
    'gamma': Gamma,
    'dirichlet': Dirichlet,
    'discrete': Categorical,
    'beta': Beta,
    'exponential': Exponential,
    'flip': Bernoulli,
    'uniform-continuous': UniformContinuous,
    'push-address': push_addr,
    '+': torch.add, '-': torch.sub, '*': torch.mul, '/': torch.div,
    '>': torch.gt, '<': torch.lt, '>=': torch.ge, '<=': torch.le, '=': torch.eq,
    'log': torch.log,
    'sqrt': torch.sqrt,
    'abs': torch.abs,
    'and': torch.logical_and,
    'or': torch.logical_or,
    'vector': vector,
    'hash-map': hash_map,
    'empty?': empty,
    'get': get,
    'put': put,
    'remove': remove,
    'first': first,
    'second': second,
    'last': last,
    'rest': rest,
    'append': append,
    'conj': append,
    'peek': peek,
    'mat-mul': torch.matmul,
    'mat-add': torch.add,
    'mat-repmat': mat_repmat,
    'mat-tanh': torch.tanh,
    'mat-transpose': torch.t
}
```

Program 1

```
FOPPL Test 1 passed
FOPPL Test 2 passed
FOPPL Test 3 passed
FOPPL Test 4 passed
FOPPL Test 5 passed
FOPPL Test 6 passed
FOPPL Test 7 passed
FOPPL Test 8 passed
FOPPL Test 9 passed
FOPPL Test 10 passed
FOPPL Test 11 passed
FOPPL Test 12 passed
FOPPL Test 13 passed
HOPPL Test 1 passed
HOPPL Test 2 passed
HOPPL Test 3 passed
HOPPL Test 4 passed
HOPPL Test 5 passed
HOPPL Test 6 passed
HOPPL Test 7 passed
HOPPL Test 8 passed
HOPPL Test 9 passed
HOPPL Test 10 passed
HOPPL Test 11 passed
HOPPL Test 12 passed
All deterministic tests passed
```

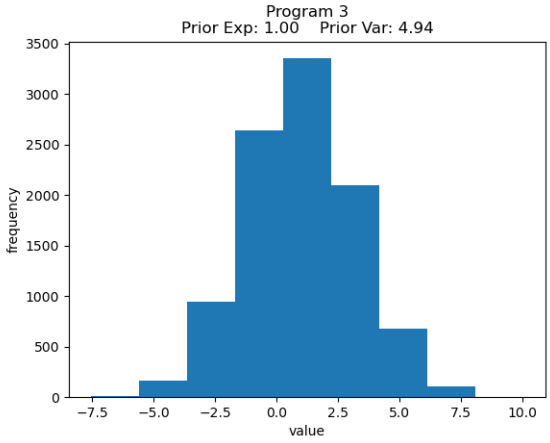
```
('normal', 5, 1.4142136)
p value 0.28711711109407734
Probabilistic Test 1 passed
('beta', 2.0, 5.0)
p value 0.4398311608511676
Probabilistic Test 2 passed
('exponential', 0.0, 5.0)
p value 0.37476190681428545
Probabilistic Test 3 passed
('normal', 5.3, 3.2)
p value 0.7546709877560348
Probabilistic Test 4 passed
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.4743866870827953
Probabilistic Test 5 passed
('normal', 0, 1.44)
p value 0.5093347451510053
Probabilistic Test 6 passed
All probabilistic tests passed
```

Program 2



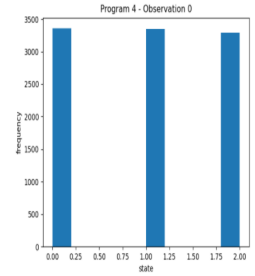

```
Sample of prior of program 2:
Took 549.94 seconds to finish Program 2
```

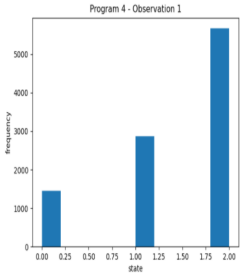
Program 3

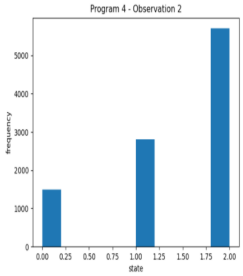


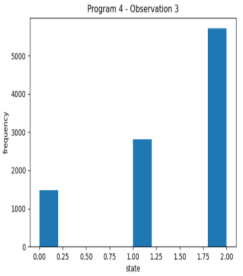

```
Sample of prior of program 3:
Took 37.93 seconds to finish Program 3
```

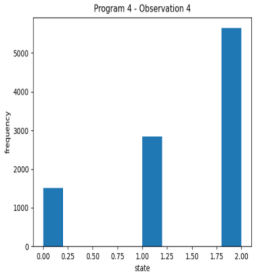
Program 4

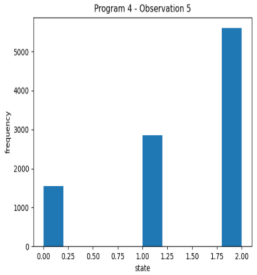


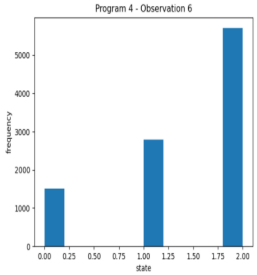


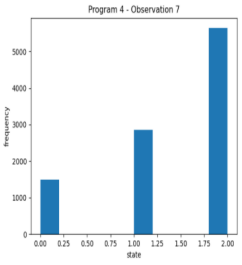


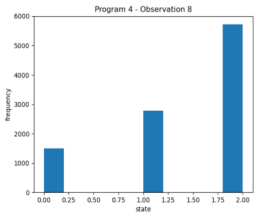


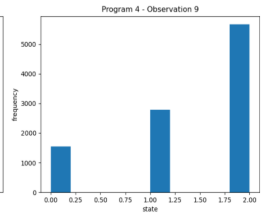


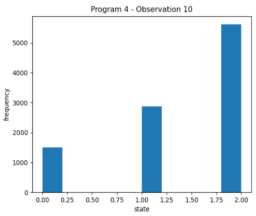


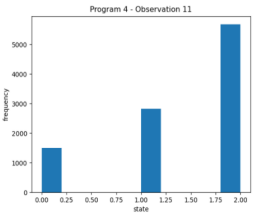


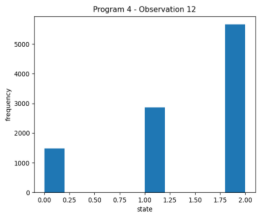


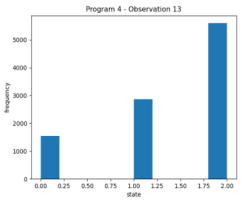


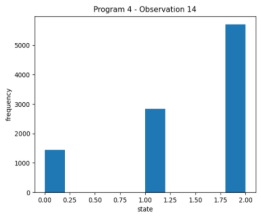


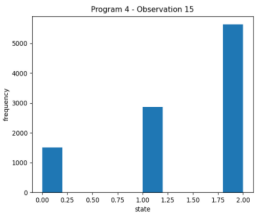


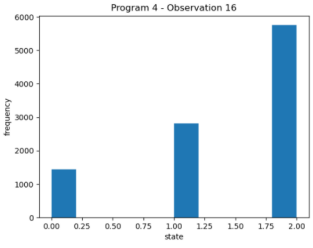













```
Sample of prior of program 4:
Took 222.18 seconds to finish Program 4
```