



ANASTASIA LABS

Audit Tracker
OpShin Audit

Contents

Milestone1 - Project Kickoff and Planning	2
Milestone2 - Language Analysis Report	2
Topics to cover	2
Areas Covered	6

Project Name: OpShin Audit

URL: Catalyst Proposal

Project Name: Opshin Language Audit

Audit Period: Oct 27th,2024

Team Member(s) Assigned: Suganya Raju, Eric Lee

Audit Start Date: Oct 2024

Audit End Date:

Milestone1 - Project Kickoff and Planning

The milestone1 was approved in November 2024.

Milestone2 - Language Analysis Report

As part of Milestone2 we are required to submit a detailed analysis report on the OpShin Language.

Topics to cover

1. UPLC debugging Architectural Decisions
 - How types are mapped to UPLC types
 - Static type Inferencer -> How ATI is implemented within OpShin
 - Mapping of Python -> Pluto -> UPLC
 - Storage of variables in statemonads
2. Opshin compiler pipeline
 - rewrites - how effective it is written and area of improvement
 - [CS] `rewrite/rewrite_cast_condition.py`: is the injected cast removed if it is redundant (i.e. `bool(b: bool)`)?
 - [CS] `rewrite/rewrite_empty_dicts.py`: this is a tiny AST transformation. Should this be merged with `rewrite_empty_lists.py`?
 - [CS] `rewrite/rewrite_import.py`: what does `spec.loader.exec_module(module)` do?
 - [CS] `rewrite/rewrite_import.py`: is the package path resolution correctly implemented, and does it always lead to the expected package being imported? Are there security risks?
 - type inference
 - [CS] `type_inference.py/AggressiveTypeInferencer`: are the more complex visitors (eg. `visit_Call`) implemented correctly
 - [CS] `type_inference.py/AggressiveTypeInferencer.visit_Expr`: why does not setting `.typ` still return a valid `TypedExpr` AST node?

-
- ▶ [CS] `type_inference.py/`
`AggressiveTypeInferencer.visit_NoneType`: the `TypedConstant` return type is confusing, is the `NoneType` AST node a type or a `None` value?
 - ▶ [CS] `type_inference.py/`
`AggressiveTypeInferencer.visit_ImportFrom`: is this the correct place for the remaining `ImportFrom` `node.module == "opshin.bridge"` assertion?
 - ▶ [CS] `type_inference.py/AggressiveTypeInferencer`: how is the type of empty lists and dicts inferred? What is the same empty list is subsequently used in two different lists with two different item types?
 - ▶ [CS] `type_inference.py`: can a top-down visitor pattern be considered aggressive? (function arg types must always be defined, so this is simple upstream to downstream type propagation)
 - optimization - how effective it is written and area of improvement
 - ▶ [CS] `optimize/optimize_const_folding.py`: some expressions might be valid python at this point, without actually passing the `OpShin` type checker (eg. `int(0.5)`). This leads to inconsistent behavior.
 - ▶ [CS] `optimize/optimize_const_folding.py`: is the evaluated math consistent with the on-chain math?
 - ▶ [CS] `optimize/optimize_const_folding.py/`
`DefinedTimesVisitor.visit_ImportFrom()`: is incrementing the number of writes of `"*"` spurious?
 - ▶ [CS] `optimize/optimize_const_folding.py`: can `print` be assigned to a variable (thus circumventing the `"print(" in unparse` output condition)? Are there other statements that shouldn't be optimized out?
 - ▶ [CS] `optimize/optimize_remove_comments.py`: the name is confusing. Are only comments being removed, or are all constants that are statements being removed? Are comments treated as constants internally?
 - ▶ [CS] `optimize/optimize_remove_comments.py`: why does this optimization remove AST nodes by returning `None`, and in
-

OptimizeRemoveDeadvars AST nodes are removed by returning `Pass()`?

- conversion -> pluto
 - [CS] **pluthon** seems quite low-level. Does this complicate the code generation unnecessarily? Investigate precisely what constructs **pluthon** allows.
 - [CS] verify that the `PlutoCompiler` visitor pattern generates correct **pluthon**
 - [CS] verify python-like builtins defined in `fun_impls.py` and `type_impls.py` are correctly implemented (manual review and unit tests)
 - [CS] verify `check_integrity` generates correct **pluthon** code.
 - [CS] `util.py/opshin_name_scheme_compatible_varname`: is there a performance penalty to using `f"1{n}"`. This seems to be a very frequently used function. Perhaps simple string concatenation makes more sense. At least add a comment about what this name mapping is necessary.
 - [CS] If `str()` is used mainly for debugging we must ensure that it is implemented in a way that can be optimize out. However if `str()` is also used for validation logic a fast, error-throwing, alternative should be made available.

3. Performance and script size testing

- [CS] Investigate root cause of maximum recursion limit error
- [CS] Running `opshin compile examples/smart_contracts/assert_sum.py -O3` creates a completely unacceptable output, the contract is extremely simple, yet the purpose debugging print statement can't be optimized out. With the print statement removed the output is still unacceptable: 470 bytes. The Muesli swap governance code-base seems to use some external compression mechanism, how exactly is that done?

4. Overall language constructs

- [CS] Does the AST only contain a single Module?

-
- [CS] Can polymorphic builtins defined in `fun_impls.py` and `type_impls.py` be assigned to variables?
 - [CS] Can helper functions like `bytes.fromhex` and `.decode` be accessed as first class citizens, or only be called directly? If only called: ensure the error message is sensible when trying to access without calling.
 - [CS] Attempting to import `Self` or `Dict`, `List`, `Union` from typing directly failed
 - [CS] Does the ability to use a getter defined on any member of a `Union` type add a lot of overhead because the field can be in a different position?
 - [CS] Why does down-casting not perform the `isinstance` check?
 - [CS] Verify that bundled std library, written in OpShin itself, is correctly implemented.

Areas Covered

1. UPLC debugger through Gastronomy
2. Tested Language constructs – output of opshin eval, eval_uplc,
3. Code Coverage - using Pytest -cov tool
4. Checked build artifacts
5. Tried to replicate aiken acceptance tests in opshin(only first 20 test cases)
 - Challenge - opshin supports only a limited language constructs, its a challenge to replicate most of the aiken acceptance tests.
6. [SR] rewrite/rewrite_import.py -
 - Observations:
 1. Relative imports are not supported as the package is always set to none (test cases for relative imports, package is a single or multiple, relative import from local) created two files , tried with imports starting with “.” not working()
 2. assumption of __spec__ for the parent module, what if __spec__ is not available for the parent module– may be try catch block?
 3. what if spec.loaded.exec_module fails to load -
 4. Line no 76 to 84, checks are for length, empty asnames, and * as name but no checks on duplicate imports, I am able to import as many times and no warnings/errors thrown
 5. sys.modules- As per Python docs If you want to iterate over this global dictionary always use sys.modules.copy() or tuple(sys.modules) to avoid exceptions as its size may change during iteration as a side effect of code or activity in other threads.
7. [SR] rewrite/rewrite_empty_dicts.py
 - Observations -None
8. [SR] rewrite/rewrite_empty_lists.py
 - Observations

-
1. List of Lists is not considered in the empty_list function the below compiles,

```
def validator(a:List[List[int]])-> List[List[int]]: empty_List :  
List[List[int]]= [[1,2],[2,3]] return empty_List
```

but this fails

```
def validator(a:List[List[int]])-> List[List[int]]: empty_List :  
List[List[int]]= [[]] return empty_List (have to look at the type inference)
```

9. [SR] rewrite/rewrite_import_dataclasses.py

- Observations

1. error messages can be more descriptive, for all the assertions , the error message is almost same

Example :

```
from dataclasses import dataclass as dc def dataclass(cls): return cls  
  
@dc class MyClass(PlutusData): pass
```

I get the error as “The program must contain one ‘from dataclasses import dataclass”

2.Validate Decorator Source: If I define a custom dataclass(rewrite over)

```
from dataclasses import dataclass
```

```
def dataclass(cls): return cls
```

```
@dataclass class MyClass(PlutusData): pass
```

```
def validator(a:int)-> None : return None
```

now the dataclass refers to the custom dataclass instead of import dataclass, the rewrite code checks the format “from import dataclass” and @dataclass decorator is present but it does not validates the source of it.

10. [SR] optimize/optimize_const_folding.py - executes the constant expressions in compile time instead of run time ex:

`x=10 y = x +5`

optimize transforms the code like this

`x =10 y =15`

1. Visit_ImportForm in class DefinedTimesVisitor - this class collects how often variables are written from ast import * - how can this node name be incremented, How to test this?
11. [SR] rewrite/rewrite_import_plutusdata.py
 1. all assertions have the same error message, can be informative
 2. suppose the imports are spilt like below, it doesn't work

Ex : from pycardano import Datum as Anything from pycardano import PlutusData
12. [SR] rewrite/rewrite_import_typing.py from typing import Dict, List, Union
 1. More descriptive error messages
 2. What if my program uses only List, still need to import Dict, List , Union – (combined with 4)
 3. Self has to be imported by typing(This case also fails , – from typing import Self – from typing import Dict, List, Union, Self)
 4. visit_classDef, checks replace self with classname(probably for type checking) only checks for Name and Union, why not List[Self] if the datum classes are used part of List , also for dictionaries - importerror, self is not from typing anymore? - TODO : create a test with List[Self] Python docs to be checked - to give the example self-> Self (it nor rewritten)
13. [SR] rewrite/rewrite_forbidden_overwrites.py
 1. Only names are checked against overriding.(covered under finding 27)
14. [SR] rewrite/rewrite_forbidden_return.py - No issues
15. [SR] rewrite/rewrite_import_hashlib.py

-
1. Aliased imports – this transformer handles alias name, What happens if the alias conflicts with an existing variable or function in the scope ?
 - It throws a type inference error, a check for name conflicts can be used instead.
 16. [SR] - rewrite/rewrite_import_integrity_check.py Similar to 15, if line 55 has a alias name it will be added to INITIAL_SCOPE as a new pair
 17. [SR] - rewrite/rewrite_subscript38.py
 - Tested for index (python old version of slice)
 - Tested for nested index - worked `def validator(x:List[List[int]]) -> int: x = [[1,2],[3,4]] b = x[1][0] return b`
 - Tried for a complex case - works Ex : `def validator(x:List[int]) -> int: x = [1,2,3,4] index = 1 return (x[index + 1])`
 - mixed slicing and index - works Ex : `def validator(x:List[List[int]]) -> List[int]: x = [[1, 2], [3, 4], [5, 6]] return (x[1:3][0])` Ex2:

```
@dataclass
class Buy(PlutusData):
    CONSTR_ID = 0
    index : int

    Const_list = [1,2,3,4]

    def validator(x:Buy) -> int:
        return (Const_list[x.index])
```

Edge case:

- Empty index doesn't work - its an invalid syntax in python itself(valid case)
18. [SR]- rewrite/rewrite_cast_condition.py -
 - if the condition is already a boolean and if its a constant node, explicit cast to bool is redundant
 - small performance overhead - using `timeit` performance with and without bool was analysed
 19. [SR] - rewrite/rewrite_augassign.py
-

-
- checked the order of precedence Ex!: `def validator(x:int,y:int, z:int) -> int: x += y*z return x -> it becomes x= x+(y*z)` Ex2: (Finding) `def validator(x:List[int]) -> int: x =[1,2,3,4] x[0] += 1 return x` Error - “Can only assign to variable names, no type deconstruction” - this is possible in python

20. [SR] - `rewrite/rewrite_remove_type_stuff.py` - No issues

21. [SR] - `rewrite/rewrite_tuple_assign.py` - need for it if the tuple is complex, reuse and efficiency What is the need of temporary variables like `2_uid_tup` , why can't it be assigned using the tuple itself Ex : `a,b =(1,2) a =(1,2)[0] b =(1,2)[1]`

22- [SR] - `rewrite/rewrite_inject_builtins.py` - no issues

Q - until the aggressive type inference occurs , there wouldn't be any typed modules ,so how can it take an `typedmodule` node? - maintainability issue, Q - polymorphic functions are skipped, two different ways to find if the node is polymorphic - maintainability - polymorphic func can be known only after type checking

23 - [SR] - `rewrite_inject_builtin_constr.py` Q - For all the builtins, `constr_type` is polymorphic only, that check is redundant or may be for a future use casez – already there (finding 11 - try to add the comment)