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 Inferencerm-> 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 infered? 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/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.