Implement two compiler passes for the Solang Solidity Compiler

<table>
<thead>
<tr>
<th>Project Title</th>
<th>Implement two compiler passes for the Solang Solidity Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>Difficulty</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

**Description**

The Solang Solidity Compiler is a new compiler project, which compiles Solidity to wasm or bpf. The compiler does not yet detect if a variable is used before it is assigned, or detecting that a value set is never used. For example:

```solidity
contract test {
    function f(bool condition) public return (int) {
        int x;
        int y = 5;
        if (condition) {
            x = 102;
        }
        return x;
    }
}
```

This code has two problems: y is never used, and the value of x might be returned without it being given a value. In compiler theory this is done with reaching definitions. Solang already has an implementation of reaching definitions for the constant propagation pass, which will have to be generalized. Unused variables should be removed from the generated code.

Secondly, implement a common subexpression elimination pass, which also uses the same reaching definitions implementation, to ensure subexpressions can safely be substituted.

**Additional Information**

https://en.wikipedia.org/wiki/Reaching_definition

https://en.wikipedia.org/wiki/Common_subexpression_elimination

https://en.wikipedia.org/wiki/Solidity

https://github.com/hyperledger-labs/solang

**Learning Objectives**

- First and foremost the mentee will learn how to be a positive collaborator and contributor in an active open source project.
- Learn how to work within the Hyperledger open source ecosystem and culture.
- Understand smart contracts and the Solidity language
- Gain a greater understanding of compiler optimizations and compiler theory

**Expected Outcome**

- Solang gives errors when undefined variables are used
- Solang gives warnings when variables are not used, and removes them from the generated code
- Solang does a common subexpression elimination pass on the generated code
- Tests and documentation

**Relation to Hyperledger**

This is part of the Solang Hyperledger Labs project. Hyperledger Burrow and Hyperledger Sawtooth can run smart contracts compiled using Solang. So, this tool will help users write smart contracts in Solidity for those projects.
Education Level

The ideal mentee is a university student or a developer with one or two years of experience with a solid background in Computer Science, especially compiler theory.

Skills

The following skills are required:

- Working knowledge of rust
- Understanding of compiler theory; understanding of control flow graphs, code generation from the ast, and compiler optimization

Future plans

There are other compiler passes which can be done, and the existing passes can be extended.

Preferred Hours and Length of Internship

This project can be done by a full-time or part-time mentee.

Mentor(s) Names and Contact Info

Sean Young
sean@mess.org
hyperledger chat: seanyoung
twitter/telegram: iamseanyoung

Mentee

Lucas Steuernagel
Github: LucasSte
LinkedIn: Lucas Steuernagel

Project Results:

Github Pull Requests

1. Warnings for unused variables
2. Unused variable elimination
3. Raise warnings for undefined variables
4. Available expressions analysis
5. Common subexpression elimination

In addition, there were other pull requests to fix bugs, to raise warnings for events that have never been emitted and to prettify the warnings. The complete list of pull requests is available here.

Final Report