Hyperledger Mentorship Project Presentation

November 2021
Solang Compiler Passes

Introduction

- **Name:** Lucas Steuernagel
- **Location:** Joinville, Brazil
- **University:** Aeronautics Institute of Technology - São José dos Campos, Brazil
- **Mentor(s):** Sean Young
- **Hyperledger Project:** Solang Solidity Compiler
Solang Compiler Passes

› Project Description:
○ Solang is a Solidity compiler that uses LLVM as the backend.
○ It is written in Rust.
○ It allows for many optimizations, considering the context of the Solidity Language.
○ Apply compiler’s theory to optimize the intermediate representation of Solidity contracts.
Solang Compiler Passes

Project Objectives:

- Obj 1: Detect and remove unused variables.
- Obj 2: Raise warnings for undefined variables.
- Obj 3: Remove common subexpressions.

```java
function passes(int a, int b) public returns (int) {
    int x = a*b-5;
    int p;
    if (x > 0) {
        x = a*b-19;
    } else {
        p = a*b*a;
    }
    int32 v = int32(x-3)-p;
    return x+a*b;
}
```
Project Deliverables:

Code deliverables:
- Deliverable 1: Unused variable detection and warnings.
- Deliverable 2: Unused variable elimination.
- Deliverable 3: Undefined variable warning.
- Deliverable 4: Available expressions analysis.
- Deliverable 5: Common subexpression elimination.

Documentation deliverables:
- Deliverable 1: Write documentation about the optimization passes at Solang docs.

```java
function passes(int a, int b) public
returns (int) {
    int x = a*b-5;
    int p;
    if (x > 0) {
        x = a*b-19;
    } else {
        p = a*b*a;
    }
    int32 v = int32(x-3)-p;
    return x+a*b;
}
```
Solang Compiler Passes

Project Execution & Accomplishments:

Deliverables:
- All deliverables have been completed 😊.

Challenges:
- There were so many edge cases for each implementation.
- All cases should be extensively considered in a compiler.

Proud of:
- Common subexpression elimination is a really complex compiler optimization to design and implement.
Solang Compiler Passes

Recommendations for future work:

- Solang’s existing optimization analysis implementation should be improved:
  - We couldn’t detect the usage of storage variables.
  - Common subexpression elimination does not work for storage variables.
- Solang has complex instructions that should be broken down.

Continued involvement:

- Make contributions during free time.
- Create an article containing a detailed design of implementation.
Solang Compiler Passes

› Project Output or Results:
  ○ Links:
    ■ Pull requests
    ■ Solang docs
    ■ Hyperledger wiki
  ○ Medium article: coming soon
Solang Compiler Passes

Basic demo:

Function passes(int a, int b) public returns (int) {
    int x = a*b - 5;
    if (x > 0) {
        x = a*b - 19;
    }
    uint32 v = uint32(x-3);
    return x + a*b;
}

Generated warning

Warning: local variable 'v' has been assigned, but never read
Line 8:
    int32 v = int32(x-3);

Solang's intermediate representation

Block0:
    ty:int256 %a = (arg #0)
    ty:int256 %b = (arg #1)
    ty:int256 %1.cse_temp = ((arg #0) * (arg #1))
    ty:int256 %x = (%1.cse_temp - int256 5)
    branchcond (%x > int256 0), block1, block2
Block1:
    ty:int256 %x = (%1.cse_temp - int256 19)
    branch block2
Block2:
    return (%x + %1.cse_temp)
Solang Compiler passes

› Insights Gained:
  ○ Better knowledge of compiler optimizations.
  ○ Open source code can impact many other projects: your contribution is truly impactful.

› Advices for new mentees:
  ○ Make a project plan.
  ○ Talk to your mentors early.
  ○ Get to know code styles and available tools beforehand to avoid back-and-forths during PR review.
THANK YOU!