Assignment 2

Optimizations with SSA

Luca Della Toffola
luca.dellatoffola@inf.ethz.ch
SSA form and optimizations
Assignment 2

2 steps

1. Apply the following optimizations
   a) Constant Folding (CF)
   b) Copy Propagation (CP)
   c) Common Subexpression Elimination (CSE)

2. Find un-initialized variables
Legend

AST Node

+  

CFG Node

\[ x_0 = 5 \]

DT Node

A
Framework

All the optimizations needs to be in 
\texttt{cd.cfg.Optimizer}
Constant Folding

- Static evaluation of expressions
- Operands are constants

\[
\begin{align*}
i_1 &= 3 + 2 \\
j_2 &= 30 \times 3
\end{align*}
\]

\[
\begin{align*}
i_1 &= 5 \\
j_2 &= 90
\end{align*}
\]
Framework

- Work directly with AST
- `cd.ir.AstRewriteVisitor` serve the purpose

```
+ 2 2 ➔ 4
```
Copy Propagation

\[ x_0 = 5 \]

\[ \text{write}(x_0) \]

\[ x_1 = y_1 \]

\[ x_2 = \phi(x_0, x_1) \]
Copy Propagation

\[ x_0 = 5 \]

write(5)

\[ x_1 = y_1 \]

\[ x_2 = \text{phi}(5, y_1) \]
Copy Propagation

• Given a **definition** of a variable \( x = \text{RHS}(\ldots) \)

• **Replace** later **uses** of \( x \) with \( \text{RHS}(\ldots) \)
  – **No other definitions** of \( x \) in between uses
  – **SSA form has one definition** per version
Copy Propagation

\[ \text{RHS}(x) \text{ must be} \]

- Constant
- Local variable
- A phi-node with all the same operands
  - This can happen by applying optimizations
Common Subexpression Elimination

\[ i_1 = a_1 + b_1 \]
\[ i_2 = a_1 + b_1 \]
\[ \text{tmp}_1 = a_1 + b_1 \]
\[ i_1 = \text{tmp}_1 \]
\[ i_2 = \text{tmp}_1 \]
Common Subexpression Elimination

• Look for identical expressions
• Save time by evaluating the expression once
• Replace computation with caching
Common Subexpression Elimination

Problem?

(AST) Expressions equivalence
Common Subexpression Elimination

(a + b) Elimination

Difficult but not impossible

(a + b) Elimination

Difficult but not impossible
Common Subexpression Elimination

What about this?
Common Subexpression Elimination

• For each expression try construct a key that represents the expression
  – Build the **canonical form** of the string
  – Using a **string** is fine
Common Subexpression Elimination

• **Full credit** for detecting expressions
  – \( a + b = b + a \)

• **Not** necessary to detect expressions
  – \( (a + b) + c = a + (b + c) \)
Common Subexpression Elimination

• **Expressions** can occur at all levels
  – **Method** (globally), **Basic Block** (locally)

• We want to **exploit global** information
  – **How** to **discover** global expression?
  – **Where** to **insert** the temporary value?
Use the dominator tree

First use dominates the second use
The expression must have been evaluated
Common Subexpression Elimination

```python
for block in DT.blocks:
    canonicalize(block.expr)
```

Find

```python
for block in DT.blocks:
    if block has new CSE.expr
        substitute matching CSE.expr
```
Common Subexpression Elimination

\[ b_1 + c_2 \]
Common Subexpression Elimination

\[ b_1 + c_2 \]
Big Picture

- Individual optimizations are not powerful
- Combination may lead to trigger opportunities
Uninitialized variables

• Detect **potentially uninitialized** variables
  – Generate a semantic error

• Modify existing `cd.cfg.SSA`
SSA + Optimizations

\[ x_1 = \phi(x_0, y_1) \]
\[ y_1 = \phi(y_0, x_1) \]

\[ t = x_1 \]
\[ x_2 = y_1 \]
\[ y_2 = t \]
SSA + Optimizations

\[ x_1 = \text{phi}(x_0, y_1) \]
\[ y_1 = \text{phi}(y_0, x_1) \]

Changed behavior: no longer swaps
SSA + Optimizations

t = x_1
x_2 = y_1
y_2 = x_1

c_1 = y_1
y_1 = x_1
x_1 = c_1

x_1 = 1
y_1 = 5

t = x_1 = 1
x_2 = y_2 = 5
y_2 = x_1 = 1

x_1 = 5
y_1 = 1
x_1 = 5
Summary

• Assignment 2 overview
  – Optimizations to implement
  – Strategies

• Example of SSA + optimizations problems

• Handling of floating point operations
  – Not required!