P4 report

Ruoyu Wu

Helper Functions

reset

For `mVarDefs` and `mIncompletePhis`, we delete all their nested maps and then delete them.

addBlock

We create new entry and add new `SubMap` and `SubPHI`. Invoke `sealBlock` if `isSealed` is True.

Local Value Numbering

writeVariable

We just set `mVarDefs[block][var]` as the `value`.

readVariable

If `mVarDefs[block][var]` exists, we jsut return it. Otherwise, we invoke `readVariableRecursive`.

Global Value Numbering

We follow the Algorithm 2 and Algorithm 3 in the paper to implement the following functions.

readVariableRecursive

There are three cases: (1) If the current block is not sealed, we create a phi instruction at the beginning and add it into `mIncompletePhis`. (2) If the current block is sealed, and it only has one predecessor, we use `readVariable` to find the def in its predecessor. (3) If the current block is sealed and has multiple predecessor, we create a phi instruction, and construct its operands by invoking `addPhiOperands`.

addPhiOperands

For each predecessor, we use `addIncoming` to add phi node's operands. When return, we use `tryRemoveTrivialPhi` to remove the trivial the phi node.

tryRemoveTrivialPhi

We first check if there are different operands in the phi node. If there are, then it is not trivial and we just return. If the phi node is trivial, we do the following three things: (1) we replace all uses of phi node to `same`, and iterate through `mVarDefs` (every basic block and every identifier) to replace all phi node. (2) we remove the phi node from the its basic block (3) for every other phi node that uses this trivial phinode, we try `tryRemoveTrivialPhi`.

Sealing Blocks

For every phi node in `mIncompletePhis`, we invoke `addPhiOperands`. At the end, we add this block to `mSealedBlocks`.

Integrating SSABuilder

We integrate the `SSABuilder` following the instructions.