Advanced Compiler 2024 - HW2 Pointer Analysis

Experiment report

Definition: There are some variables are used to record the information, like the following:

1. std::vector<std::string> VarName: recording the variable name in the instruction.
2. std::unordered_map<std::string, int> TDEF: used to record OUT of last statement
3. std::unordered_map<std::string, std::string> TEQUIV_IN: used to record the TEQUIV_OUT of last statement (TEQUIV_IN for current statement)
4. static int isSubtree(std::string s1, std::string s2): check if s1 is the subtree of s2, if yes, return the index of s1 in s2, if no, return -1. If return 0 means s1 == s2, > 0, s1 is proper subtree of s2.

Step1: Seperate statement

1. Find the variable and its subtree in a statement (ignore constant), if a variable in instruction has no name, it is a dereference of the last one.
2. Stop until encounting store instruction.

Ex: We can separate statements by identifying the store instruction, for the following example, in the **varName** vector of each statement
statment1: [p]
statement2: [pp]
statment3: [pp, *pp]
statment4: [p, *p]

```
%x = alloca i32, align 4
%y = alloca i32, align 4
%p = alloca ptr, align 8
%pp = alloca ptr, align 8
store ptr %x, ptr %p, align 8
store ptr %p, ptr %pp, align 8
%0 = load ptr, ptr %pp, align 8
store ptr %y, ptr %0, align 8
%1 = load ptr, ptr %p, align 8
store i32 3, ptr %1, align 4
ret void
```

Step2: Computing **TREF**

1. In the **varName** vector of each statement, add all elements into the TREF set except for the last element; that is, add all RHS variables and their expression trees and the proper subtree of LHS.
2. TREF run through the TEQUIV_IN set to check alias

Step3: Computing **TGEN**

1. Add the last element in the varName vector into TGEN set
2. Searching TEQUIV_IN to find the alias of the LHS variable and add it to TGEN

Step4: Find Dependence

1. Flow dependence: TREF(Si) ∩ TDEF(Si)
2. Output dependence: TGEN(Si) ∩ TDEF(Si)

Step5: Update the **TDEF** for next statement

1. First, there are two cases, the element in TDEF should be removed
    1. TGEN(Si) ∩ TDEF(Si)
    2. If any element of TGEN(Si) is a proper subtree of any of the element in TDEF(Si), the element is removed
2. Add the variables in TGEN into TDEF like (varname, current_stmtid)

Step6: Compute the **TEQUIV_OUT**: TEQUIV_OUT = (TEQUIV_IN - TEQUIV_KILL) U TEQUIV_GEN

1. Remove the element in **EQUIV_KILL** from the original EOUIV_IN
    - EQUIV_KILL = If any element of TGEN(Si) is a proper subtree of any of the element in EQuiv_IN(Si), the pair is removed
2. **TEQUIV_GEN**: check the type of src of the store instruction, if the type is pointer, it is a pointer assignment, and we should add new element into TEQUIV_IN, there are two cases, like the following:

```
Value *V2 = SI->getValueOperand();
if (V2->getType()->isPointerTy())
{
    if (V2->hasName())
    { // for pointer assignment: a = &b;
      TEQUIV_IN[("*" + varNames.back())] = V2->getName().str();
    }
    else
    { // for pointer assignment: a = b; (assume only one variable in RHS)
      TEQUIV_IN[("*" + varNames.back())] = "*" + varNames.front();
    }
}
```

3. compute the **transitive closure** of **TEQUIV_IN**

```
// transitve closure of TEQUIV_OUT (the TEQUIV_IN of the next statement)
for (auto &name : TEQUIV_IN)
{
  for (auto &name2 : TEQUIV_IN)
  {
    if (name == name2)
      continue;
    int pos = isSubtree(name.first, name2.first);
    if (pos > 0)
    {
      TEQUIV_IN[name2.first.substr(0, pos) + name.second] = name2.second;
    }
    pos = isSubtree(name.second, name2.first);
    if (pos != -1)
    {
      TEQUIV_IN[name2.first.substr(0, pos) + name.first] = name2.second;
    }
    pos = isSubtree(name.second, name2.second);
    if (pos > 0)
    {
      TEQUIV_IN[name2.first] = name2.second.substr(0, pos) + name.first;
    }
  }
}
```

Step 7: After processing all instructions, output the json file by using the **llvm::json::Object**

example: the output of icpp.c: icpp.json

```
{
  "S1": {
    "DEP": [],
    "TDEF": {
      "p": 1
    },
    "TEQUIV": [
      [
        "*p",
        "x"
      ]
    ],
    "TGEN": [
      "p"
    ],
    "TREF": []
  },
  "S2": {
    "DEP": [],
    "TDEF": {
      "p": 1,
      "pp": 2
    },
    "TEQUIV": [
      [
        "**pp",
        "x"
      ],
      [
        "*p",
        "x"
      ],
      [
        "*pp",
        "p"
      ]
    ],
    "TGEN": [
      "pp"
    ],
    "TREF": []
  },
  "S3": {
    "DEP": [
      {
        "dst_stmt": 3,
        "src_stmt": 2,
        "type": "flow",
```

```
        "var": "pp"
      },
      {
        "dst_stmt": 3,
        "src_stmt": 1,
        "type": "output",
        "var": "p"
      }
    ],
    "TDEF": {
      "*pp": 3,
      "p": 3,
      "pp": 2
    },
    "TEQUIV": [
      [
        "**pp",
        "y"
      ],
      [
        "*p",
        "y"
      ],
      [
        "*pp",
        "p"
      ]
    ],
    "TGEN": [
      "*pp",
      "p"
    ],
    "TREF": [
      "pp"
    ]
  },
  "S4": {
    ...
  }
}
```