Advanced Compilers 2023f hw2

Author: 311554032 廖子濬 CS@NYCU

How to run the program?

Put the hw2 directory alongside the `llvm-build` directory as hw1. Go into hw2 and do `make run[1-4]` for testcase `icpp.c, icpp2.c, icpp3.c, foo.c` . For example, do `make run1` for testcase `icpp.c`

Display the output of the released testcases that you can handle

- icpp.c

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP: {}
TDEF: {(p, S1)}
TEQUIV: {(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP: {}
TDEF: {(p, S1), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, x), (*p, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP: {
    p: S1-0->S3
    pp: S2--->S3
}
TDEF: {(*pp, S3), (p, S3), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

```
S4:--------------------
TREF: {*pp, p}
TGEN: {*p, y}
DEP: {
    *pp: S3--->S4
    p: S3--->S4
}
TDEF: {(*p, S4), (*pp, S3), (p, S3), (pp, S2), (y, S4)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

- icpp2.c

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP: {}
TDEF: {(p, S1)}
TEQUIV: {(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP: {}
TDEF: {(p, S1), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, x), (*p, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP: {
    p: S1-0->S3
    pp: S2--->S3
}
TDEF: {(*pp, S3), (p, S3), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}

S4:--------------------
TREF: {*pp, p}
TGEN: {*p, y}
DEP: {
    *pp: S3--->S4
    p: S3--->S4
}
TDEF: {(*p, S4), (*pp, S3), (p, S3), (pp, S2), (y, S4)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

```
S5:--------------------
TREF: {**pp, *p, *pp, p, pp, y}
TGEN: {**pp, y}
DEP: {
    *p: S4--->S5
    *pp: S3--->S5
    p: S3--->S5
    pp: S2--->S5
    y: S4--->S5
    y: S4-O->S5
}
TDEF: {(**pp, S5), (*p, S4), (*pp, S3), (p, S3), (pp, S2), (y, S5)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

- icpp3.c

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP: {}
TDEF: {(p, S1)}
TEQUIV: {(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP: {}
TDEF: {(p, S1), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, x), (*p, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP: {
    p: S1-O->S3
    pp: S2--->S3
}
TDEF: {(*pp, S3), (p, S3), (pp, S2)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

```
S4:--------------------
TREF: {*pp, p}
TGEN: {*p, y}
DEP: {
    *pp: S3--->S4
    p: S3--->S4
}
TDEF: {(*p, S4), (*pp, S3), (p, S3), (pp, S2), (y, S4)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}

S5:--------------------
TREF: {*pp, p, pp}
TGEN: {**pp, y}
DEP: {
    *pp: S3--->S5
    p: S3--->S5
    pp: S2--->S5
    y: S4-0->S5
}
TDEF: {(**pp, S5), (*p, S4), (*pp, S3), (p, S3), (pp, S2), (y, S5)}
TEQUIV: {(*pp, p), (**pp, y), (*p, y)}
```

- foo.c

```
S1:--------------------
TREF: {b, c}
TGEN: {a}
DEP: {}
TDEF: {(a, S1)}
TEQUIV: {}

S2:--------------------
TREF: {}
TGEN: {p}
DEP: {}
TDEF: {(a, S1), (p, S2)}
TEQUIV: {(*p, y)}

S3:--------------------
TREF: {b, c}
TGEN: {d}
DEP: {}
TDEF: {(a, S1), (d, S3), (p, S2)}
TEQUIV: {(*p, y)}
```

```
S4:--------------------
TREF: {*p, a, d, x, y}
TGEN: {f}
DEP: {
    a: S1--->S4
    d: S3--->S4
}
TDEF: {(a, S1), (d, S3), (f, S4), (p, S2)}
TEQUIV: {(*p, y)}

S5:--------------------
TREF: {*p, a, d, x, y}
TGEN: {g}
DEP: {
    a: S1--->S5
    d: S3--->S5
}
TDEF: {(a, S1), (d, S3), (f, S4), (g, S5), (p, S2)}
TEQUIV: {(*p, y)}

S6:--------------------
TREF: {i, p}
TGEN: {*p, y}
DEP: {
    p: S2--->S6
}
TDEF: {(*p, S6), (a, S1), (d, S3), (f, S4), (g, S5), (p, S2), (y, S6)}
TEQUIV: {(*p, y)}

S7:--------------------
TREF: {*p, a, d, x, y}
TGEN: {h}
DEP: {
    *p: S6--->S7
    a: S1--->S7
    d: S3--->S7
    y: S6--->S7
}
TDEF: {(*p, S6), (a, S1), (d, S3), (f, S4), (g, S5), (h, S7), (p, S2), (y
TEQUIV: {(*p, y)}

S8:--------------------
TREF: {*p, y}
TGEN: {f}
DEP: {
    *p: S6--->S8
    f: S4-O->S8
    y: S6--->S8
}
TDEF: {(*p, S6), (a, S1), (d, S3), (f, S8), (g, S5), (h, S7), (p, S2), (y
TEQUIV: {(*p, y)}
```

Experiment report – how you implement the pass

1. **Finish the generate of `TEQUIV`**
   I've implemeted the TEQUIV with `vector<list<string>>` , list is to simulate the forest of trees and vector is the collection of trees.
   For example of the forest we may see the end result of `icpp.c` .

```
Tree: pp -> p -> y -> 3 -> NULL
Tree: p -> y -> 3 -> NULL
Tree: y -> 3 -> NULL
Tree: x -> NULL
Tree: 3 -> NULL
```

The forest stores trees and some proper subtrees of trees.
Everytime the proper subtree is updated, the tree where the proper subtree comes from also update its child.

2. **Update `TREF , TGEN`**

Since these two sets only update in current statement, so I implement these two sets at step 2.

3. **Add `TDEF`**

For each statement add/update the defined elements in current statement. Also, we add all the aliasing members into it, or we update it.

4. **Calculate `DEP`**

```
The intersection of TREF(Si) and TDEFi is taken. Any elements common
to both sets represent flow dependence arcs into this statement based
on those elements. Similarly, any elements in the intersection of
TDEFi and TGEN(Si) represent output dependence arcs into this
statement based on those elements.
```

For **Flow Dependence**, intersect the `TREFi` and `TDEFi−1` , and for **Output Dependnece**, intersect the `GENi` and `TDEFi−1` .