DCE-Artifact源码实验

> **采用的代码：**
>
> Artifact-Version2【link】
>
> **采用的环境：**
>
> - Ubuntu20.04
> - Docker latest
>
> **硬件容量需求：**
>
> 15G+

1、配置环境

这边主要是为了还原原始实验场景，所以没有另外自己导入包去创建环境，优点是可以复现原始实验数据，缺点是后续自己做衍生实验受到限制。

To import the docker image:

第一次需要配置，以后这一步都不需要。

Plain Text | 复制代码

```
cat dce-artifact-image.tar | docker import - dce_artifact
```

Start an interactive session:

以后每次进入环境都需要在命令行键入以下命令：

```
docker run -it dce_artifact bash
```

2、实验流程以及数据

2.1 测试数据

使用的是CSmith生成的10000条C程序，其中每一个程序对应4个文件：

文件名	内容简介	内容展示	数据来源
test_casexxxx.c	对CSmith生成的程序进行marksers插入后的结果	//……前面276个markers的定义 void DCEFunc277(void); /* * This is a RANDOMLY GENERATED PROGRAM. * * Generator: csmith 2.3.0 * Git version: 30dccd7 * Options: ––argc ––arrays ––no–bitfields ––no–checksum ––comma–operators * ––compound–assignment ––consts ––no–divs ––no–embedded–assigns ––no–jumps * ––longlong ––no–force–non–uniform–	CSmith

		force-non-uniform-arrays --no-math64 --muls
		* --no-packed-struct --paranoid --no-pointers --structs --no-volatiles
		* --no-volatile-pointers --inline-function --return-structs --no-arg-structs
		* --no-dangling-global-pointers --no-unions --safe-math
		Seed: 769789783
		*/
		#include "csmith.h"
		volatile uint32_t csmith_sink_ = 0;
		static long __undefined;
		//程序具体内容	

test_caseXXXX.pred	记录每一个marker的前一个结点marker（计算使用）	DCEFunc6: DCEFunc1 DCEFunc7: DCEFunc1 DCEFunc271: DCEFunc268 DCEFunc268: DCEFunc42: DCEFunc38 DCEFunc269: DCEFunc268 DCEFunc277: DCEFunc269 DCEFunc270: DCEFunc268	/dce/generate_ground_truth.sh脚本生成； 具体的，.pred是由/dce/dce_instrumenter/src/FIPCFGExtractor.cpp生成的

		DCEFunc243: DCEFunc241 DCEFunc248 DCEFunc249 ……	
test_case.ground_truth_alive	作为ground_truth的alive markers	DCEFunc0 DCEFunc1 DCEFunc105 DCEFunc106 DCEFunc107 DCEFunc11 DCEFunc110 DCEFunc111 ……	
test_case.ground_truth_dead	作为ground_truth的dead markers	DCEFunc10 DCEFunc100 DCEFunc101 DCEFunc102 DCEFunc103 DCEFunc104 DCEFunc108 ……	
test_caseXXXX.COMP_alive_OPT(具体有gcc_o0~gcc_os/clag_o0~clang_os)	对应某个level编译器的alive markers	DCEFunc0 DCEFunc1 DCEFunc10 DCEFunc105 DCEFunc106 DCEFunc107 DCEFunc11	

		DCEFunc110 DCEFunc111 ……	

ground_truth生成原理：最新版本的llvm编译器。

但是最后论文给的都是差分的方法，应该是一开始想做ground truth但是后来发现了一些回归问题，改成差分的方法更客观一点。

2、实验流程以及测试结果

2.1 实验流程

1、设置环境

命令行启动

```bash
cd /dce
source setup.sh
```

setup.sh

```bash
export CLANG=/opt/compilers/bin/clangdce
export GCC=/opt/compilers/bin/gccdce
export FIND_DCE_PREDECESSORS=/dce/helper_scripts/find_dce_predecessors.py
export LIB_DCE_FIPCFGExtractor=/dce/dce_instrumenter/build/lib/libFIPCFGExtractor.so
export PATH="$PATH":/opt/llvm12/bin
export CSMITH_INC=/usr/include/csmith-2.3.0
```

其中setup.sh主要是为了设置环境变量

2、生成100个新的test cases

```bash
./make_corpus.sh test_corpus 100
```

其中make_corpus.sh内容如下：

make_corpus.sh

Bash 复制代码

```bash
#!/bin/env bash

corpusdir="$1"
n=$2

if [[ -d "$corpusdir" ]]; then
    echo "$corpusdir" already exists
    exit 0
fi

mkdir -p "$corpusdir"

check_size() {
    origsize=$(du -b "$1" | cut -f 1)
    if [ $origsize -lt 100000 ] || [ $origsize -ge 1000000 ] ; then
      echo "Abort: small/large file"
      return 1
    fi
    return 0
}

instrument() {
    if  /opt/llvm12/bin/dcei "$1" --extra-arg=-isystem/usr/include/csmith-2.3.0 -- > /dev/null 2>&1 ;
    then
       :
    else
        return 1
    fi
    grep -q DCEFunc "$1"
    return $?
}

gen_case(){
    timeout 10s ./helper_scripts/gen-csmith-seed-program.sh  "$corpusdir"/test_case${1}.c
    f=$(echo "$corpusdir"/test_case${1}.c)
    until check_size "$f" && ./helper_scripts/sanitize.sh "$CLANG" "$GCC" "$f" && instrument "$f"
    do
        timeout 10s ./helper_scripts/gen-csmith-seed-program.sh  "$corpusdir"/test_case${1}.c
    done
}

```

```bash
export corpusdir
export -f gen_case
export -f check_size
export -f instrument

seq -f '%04g' "$n" | parallel --progress gen_case
rm -f "$corpusdir"/platform.info
```

3、生成ground truth

```bash
./generate_ground_truth.sh test_corpus/
```

其中generate_ground_truth.sh的内容如下：

generate_ground_truth.sh Bash | 复制代码

```bash
#!/usr/bin/env bash

corpusdir="$1"

if [[ ! -d "$corpusdir" ]]; then
    echo "$corpusdir" does not exist
    exit 0
fi


gen_ground_truth(){
    TMPDIRECTORY=$(mktemp -d)
    trap '{ rm -rf -- "$TMPDIRECTORY"; }' EXIT


    filename="$1"
    dir=$(dirname "$filename")
    cd "$dir"
    filename=$(basename "$filename")


    tmpcopy=$(mktemp --suffix='.c' --tmpdir="$TMPDIRECTORY")

    echo "#include <stdio.h>" > "$tmpcopy"
    cat "$filename" >> "$tmpcopy"
    sed -i 's/^void DCEFunc\(.*\)(void);/void DCEFunc\1(void){printf("DCEFunc\1\\n");}/g' "$tmpcopy"

    tmpexe=$(mktemp --suffix='.exe' --tmpdir="$TMPDIRECTORY")
    alive_file=$(echo ${filename%.c}.ground_truth_alive)
    gcc "$tmpcopy" -o "$tmpexe"  -w -I /usr/include/csmith-2.3.0
    "$tmpexe"  | grep DCEFunc --color=never | sort -u > "$alive_file"

    dead_file=$(echo ${filename%.c}.ground_truth_dead)
    diff --unchanged-line-format="" --new-line-format="" \
        <(grep 'DCEFunc.*()' --color=never "$filename" | cut -d '(' -f 1 | awk '{$1=$1;print}' | sort -u) \
                "$alive_file" > "$dead_file"

    pred_file=$(echo ${filename%.c}.pred)
    $FIND_DCE_PREDECESSORS "$filename" /usr/include/csmith-2.3.0 $LIB_DCE_FIPCFGExtractor > "$pred_file"

    rm -rf "$TMPDIRECTORY"
}
```

```
export -f gen_ground_truth

find "$corpusdir" -name '*.c' | parallel --progress gen_ground_truth
```

4 、生成编译器的DCE数据

Bash | 复制代码

```
    ./generate_compiler_dce_results.sh test_corpus/
```

其中generate_compiler_dce_results.sh的内容如下：

generate_compiler_dce_results.sh

Bash

复制代码

```bash
#!/usr/bin/env bash

corpusdir="$1"

if [[ ! -d "$corpusdir" ]]; then
    echo "$corpusdir" does not exist
    exit 0
fi


gen_ground_truth(){
    TMPDIRECTORY=$(mktemp -d)


    filename="$1"
    dir=$(dirname "$filename")
    cd "$dir"

    filename=$(basename "$filename")

    tmpclangs=$(mktemp --suffix='.c' --tmpdir="$TMPDIRECTORY")
    tmpgccs=$(mktemp --suffix='.c' --tmpdir="$TMPDIRECTORY")

    optlevels=(O0 O1 Os O2 O3)

    for optlevel in "${optlevels[@]}"; do
        "$CLANG" -S -"$optlevel" -w -I "$CSMITH_INC" -o "$tmpclangs" "$filename" >& /dev/null
        "$GCC"   -S -"$optlevel" -w -I "$CSMITH_INC" -o "$tmpgccs"   "$filename" >& /dev/null

        clang_alive_file=$(echo ${filename%.c}.clang_alive_"$optlevel")
        gcc_alive_file=$(echo ${filename%.c}.gcc_alive_"$optlevel")

        grep DCEFunc "$tmpclangs" | grep -E "call|jmp" | sort -u | awk '{print $2}' > "$clang_alive_file"
        grep DCEFunc "$tmpgccs" | grep -E "call|jmp" | sort -u | awk '{print $2}' > "$gcc_alive_file"
    done

    rm -rf "$TMPDIRECTORY"
}

export -f gen_ground_truth

```

```
find "$corpusdir" -name '*.c' | parallel --progress gen_ground_truth
```

5、验证

This will randomly select 100 c files in corpus10000, regenerate the ground truth and per compiler DCE data, and compare these with the original in corpus10000, if there are any differences they will be printed.

Bash | 复制代码

```bash
    ./validate.sh corpus10000/ 100
```

其中validate.sh的内容如下：

validate.sh

Bash | 复制代码

```bash
#!/usr/bin/env bash

corpusdir="${1%/}"
n="$2"

if [[ ! -d "$corpusdir" ]]; then
    echo "$corpusdir" does not exist
    exit 0
fi


readonly TMPDIRECTORY=$(mktemp -d)
trap '{ rm -rf -- "$TMPDIRECTORY"; }' EXIT

ls "$corpusdir"/*.c | sort -R | head -n "$n" | while read file;
do
    cp "$file" "$TMPDIRECTORY"
done

./generate_ground_truth.sh "$TMPDIRECTORY"
./generate_compiler_dce_results.sh "$TMPDIRECTORY"

ls "$TMPDIRECTORY" | while read file;
do
   diff <(sort "$TMPDIRECTORY"/"$file") <(sort "$corpusdir"/$(basename "$file"))
done
```

2.2 实验结果数据处理

生成DCE统计信息

```
./print_dce_stats.py corpus10000
```

将会生成论文中的section4中的表格1和表格2（GCC vs LLVM以及不同的优化级别）

运行结果如下：

```
|    % dead blocks that are missed

––––––––––––––––––––––––––––––––––––––––

O0 | 85.21% (2373352) | 83.82% (2334830)

O1 |   8.18% (227807) |   5.20% (144757)

Os |   5.94% (165332) |   4.75% (132375)

O2 |   5.66% (157720) |   4.35% (121111)

O3 |   5.60% (155945) |   4.31% (120003)


   | % dead blocks that are primary missed

–––––––––––––––––––––––––––––––––––––

O0 | 15.30% (132313) | 4.75% (426261)

O1 |   1.76% (40908) |  1.47% (49086)

Os |   1.56% (39705) |  1.43% (43427)

O2 |   1.53% (38385) |  1.38% (42655)

O3 |   1.53% (38194) |  1.37% (42478)


GCC at –O3 eliminates 3781 dead blocks that LLVM misses, 396 are primary.

LLVM at –O3 eliminates 39723 dead blocks that GCC misses, 4749 are primary.


LLVM at –O3 misses 456 dead blocks but it eliminates them at –O2 and/or –O1, 54 are primary.

GCC at –O3 misses 308 dead blocks but it eliminates them at –O2 and/or –O1, 24 are primary.
```

其中print_dce_stats.py内容如下：

print_dce_stats.py Python 复制代码

```python
#!/usr/bin/env python3

import argparse
from pathlib import Path
from operator import itemgetter
from dataclasses import dataclass
from collections import defaultdict

extensions = [
    ".clang_alive_Os",
    ".gcc_alive_Os",
    ".clang_alive_O0",
    ".gcc_alive_O0",
    ".clang_alive_O1",
    ".gcc_alive_O1",
    ".clang_alive_O2",
    ".gcc_alive_O2",
    ".clang_alive_O3",
    ".gcc_alive_O3",
    ".ground_truth_alive",
    ".ground_truth_dead",
]


def all_dce_files_exist(cfile):
    return all(cfile.with_suffix(ext).exists() for ext in extensions)


def read_function_set(file):
    with open(str(file), "r") as f:
        return set(l.rstrip() for l in f.readlines())


def read_dce_predecessors(file):
    pred = defaultdict(set)
    with open(str(file), "r") as f:
        for l in f.readlines():
            try:
                s, preds = l.strip().split(":")
                preds = preds.strip().split()
            except:
                s = l.strip().split(":")
                preds = []
            for pr in preds:
                pred[s].add(pr)
    return pred


@dataclass
class DCESets:
    dead: set[str]
    alive: set[str]


@dataclass
class CompilerDCESets:
    Os: DCESets
    O0: DCESets
    O1: DCESets
    O2: DCESets
    O3: DCESets


@dataclass
class DCEData:
    cfile: str
    predecessors: dict[str, set[str]]
    ground_truth: DCESets
    gcc: CompilerDCESets
    clang: CompilerDCESets


def read_data_from_files(cfile):
    raw_data = {
        ext[1:]: read_function_set(cfile.with_suffix(ext)) for ext in extensions
    }
    all_funcs = raw_data["ground_truth_dead"] | raw_data["ground_truth_alive"]
    ground_truth = DCESets(
        raw_data["ground_truth_dead"], raw_data["ground_truth_alive"]
    )
    predecessors = read_dce_predecessors(cfile.with_suffix(".pred"))

    def read_compiler_data(cc, opt):
        cc_alive = raw_data[f"{cc}_alive_{opt}"]
        cc_dead = all_funcs - cc_alive
        return DCESets(cc_dead, cc_alive)

    return DCEData(
        cfile,
        predecessors,
        ground_truth,
        CompilerDCESets(
            *(read_compiler_data("gcc", opt) for opt in ("Os", "O0", "O1", "O2", "O3"))
        ),
        CompilerDCESets(
            *(
                read_compiler_data("clang", opt)
                for opt in ("Os", "O0", "O1", "O2", "O3")
            )
        ),
    )


def read_data(directory):
    for cfile in Path(directory).glob("*.c"):
        assert all_dce_files_exist(cfile)
        yield read_data_from_files(cfile)


def find_diff_cases(data, cc):
    pred = data.predecessors
    O1 = getattr(data, cc).O1
    O2 = getattr(data, cc).O2
    O3 = getattr(data, cc).O3

    missed_O1 = O1.dead - O3.dead
    missed_O2 = O2.dead - O3.dead
    missed_O2 = missed_O2 - missed_O1

    def filter_and_print(opt, missed):
        for m in missed:
            if all(p in data.ground_truth.alive or p in O3.dead for p in pred[m]):
                print(f"{data.cfile} {opt} {m}")

    filter_and_print("O1", missed_O1)
    filter_and_print("O2", missed_O2)


def number_critical_missed_dead_wrt_gt(data, cc, opt):
    pred = data.predecessors
    opt_data = getattr(getattr(data, cc), opt)
    missed = data.ground_truth.dead - opt_data.dead
    return len(
        [
            m
            for m in missed
```

```python
            if all(p in data.ground_truth.alive or p in opt_data.dead for p in pred[m])
        ]
    )


def number_critical_dead_wrt_gt(data, cc, opt):
    pred = data.predecessors
    opt_data = getattr(getattr(data, cc), opt)
    missed = data.ground_truth.dead - opt_data.dead
    return len(
        [
            m
            for m in missed
            if all(p in data.ground_truth.alive or p in opt_data.dead for p in pred[m])
        ]
    )


def number_differential(data, cc1, opt1, cc2, opt2):
    pred = data.predecessors
    opt_data1 = getattr(getattr(data, cc1), opt1)
    opt_data2 = getattr(getattr(data, cc2), opt2)
    missed = opt_data2.dead - opt_data1.dead
    return len(missed)


def number_differential_lower(data, cc):
    pred = data.predecessors
    cc_data = getattr(data, cc)
    O3d = cc_data.O3.dead
    O2d = cc_data.O2.dead
    O1d = cc_data.O1.dead
    missed = (O1d | O2d) - O3d
    return len(missed)


def number_critical_differential(data, cc1, opt1, cc2, opt2):
    pred = data.predecessors
    opt_data1 = getattr(getattr(data, cc1), opt1)
    opt_data2 = getattr(getattr(data, cc2), opt2)
    missed = opt_data2.dead - opt_data1.dead
    return len(
        [
            m
            for m in missed
            if all(p in data.ground_truth.alive or p in opt_data1.dead for p in pred[m])
        ]
    )


def number_critical_differential_lower(data, cc):
```

最后是bisect输出统计：

Bash | 复制代码

```bash
./print_commit_info.py
```

运行结果为：

LLVM

Component	# Commits	# Files
Instruction Operand Folding	2	1
Jump Threading	1	1
Loop Transformations	1	1
Pass Management	2	2
Peephole Optimizations	7	10
SSA Memory Analysis	2	1
Target Info	1	2
Value Constraint Analysis	1	1
Value Propagation	4	2
Value Tracking	1	1

GCC

Component	# Commits	# Files
Alias Analysis	3	1
C−family Frontend	1	4
Common Subexpression Elimination	3	2
Constant Propagation	4	2
Control Flow Graph Analysis	1	2
Copy Propagation	1	1
Inlining	3	2
Interprocedural Analyses	1	1
Interprocedural SRoA	1	1
Jump Threading	1	3
Loop Transformations	3	2
Pass Management	2	2
Peephole Optimizations	1	1

```
Target Info                |1        |1

Value Numbering               |3        |2

Value Propagation             |6        |7
```

其中，print_commit_info.py的内容如下：

print_commit_info.py

Python | 复制代码

```python
            for line in f.readlines():
                line = line.strip()
                if Path(line).parts[1] == "testsuite":
                    continue
                if Path(line).parts[1] == "doc":
                    continue
                if line.strip().endswith("match.pd"):
                    continue
                if line.strip().endswith("params.opt"):
                    continue
                if line.strip().endswith("timevar.def"):
                    continue
                if line.strip().endswith("ChangeLog"):
                    continue
                if line.strip().endswith("params.def"):
                    continue
                if line.strip().endswith("Makefile.in"):
                    continue
                if Path(line).parts[0] == "gcc":
                    yield (
                        commit_files.stem,
                        str(Path().joinpath(*Path(line).parts[1:])),
                    )
                else:
                    print(line)
                    assert False


def has_many_parts(file):
    return len(Path(file).parts) > 1


def generate_grouped_by_prefix(files, counts, ntabs=0):
    grouped_by_prefix = defaultdict(list)
    tabs = "\\quad" * ntabs + " "
    for file, count in zip(files, counts):
        if has_many_parts(file):
            parts = Path(file).parts
            grouped_by_prefix[parts[0]].append(
                (Path(parts[1]).joinpath(*parts[2:]), count)
            )
        else:
            grouped_by_prefix[Path(file)].append(("", count))

    for head, subpaths in sorted(
        grouped_by_prefix.items(), key=lambda x: str(x[0]).lower()
    )
    ):
        if len(subpaths) == 1:
            subpath = subpaths[0]
            subpath_str = str(Path(head) / subpath[0]).strip().replace("_", "\\_")
            yield tabs, subpath_str, subpath[1]
            continue
        head_str = str(head).strip().replace("_", "\\_")
        yield tabs, head_str
        files, counts = zip(*subpaths)
        yield from generate_grouped_by_prefix(files, counts, ntabs + 1)


def line_to_str(line):
    if len(line) == 2:
        tabs, head_str = line
        return f"\\ {tabs}{head_str}/ &"
    elif len(line) == 3:
        tabs, subpath_str, subpath_1 = line
        return f"\\ {tabs}{subpath_str} &{subpath_1}"
    else:
        return line + "&"


def print_cc(cc):
    category_map = {
        "Analysis/ValueLattice.h": "Value Constraint Analysis",
        "Transforms/Scalar/SCCP.cpp": "Value Propagation",
        "Analysis/InstructionSimplify.cpp": "Instruction Operand Folding",
        "Transforms/IPO/PassManagerBuilder.cpp": "Pass Management",
        "Passes/PassBuilder.cpp": "Pass Management",
        "Transforms/InstCombine/InstCombineShifts.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineSimplifyDemanded.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombinePHI.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineAndOrXor.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineSelect.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineAddSub.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineInternal.h": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineNegator.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstructionCombining.cpp": "Peephole Opti
        "Transforms/InstCombine/InstructionCombining.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineCompares.cpp": "Peephole Optimizations",
        "Transforms/InstCombine/InstCombineCompares.cpp": "Peephole Optimizations",
        "Transforms/Scalar/CorrelatedValuePropagation.cpp": "Value Propagation",
        "Analysis/MemorySSA.cpp": "SSA Memory Analysis",
        "Transforms/Utils/LoopUtils.cpp": "Loop Transformations",
        "Support/KnownBits.cpp": "Value Tracking",
        "Analysis/BasicAliasAnalysis.cpp": "Alias Analysis",
        "Transforms/Scalar/JumpThreading.cpp": "Jump Threading",
        "Analysis/TargetTransformInfo.h": "Target Info",
        "Analysis/TargetTransformInfoImpl.h": "Target Info",
        "tree-ssa-loop-ivcanon.c": "Loop Transformations",
        "ipa-fnsummary.c": "Interprocedural Analyses",
        "tree-ssa-sccvn.c": "Value Numbering",
        "tree-ssa-sccvn.h": "Value Numbering",
        "tree-ssa-pre.c": "Common Subexpression Elimination",
        "gcse.c": "Common Subexpression Elimination",
        "tree-ssa-alias.c": "Alias Analysis",
        "cfganal.h": "Control Flow Graph Analysis",
        "cfganal.c": "Control Flow Graph Analysis",
        "tree-inline.c": "Inlining",
        "ipa-inline.c": "Inlining",
        "tree-ssa-ccp.c": "Constant Propagation",
        "tree-ssa-propagate.c": "Value Propagation",
        "tree-pass.h": "Pass Management",
        "tree-vrp.c": "Value Propagation",
        "tree-vrp.h": "Value Propagation",
        "vr-values.c": "Value Propagation",
        "vr-values.h": "Value Propagation",
        "tree-ssa-propagate.h": "Value Propagation",
        "fold-const.c": "Constant Propagation",
        "c-family/c-common.c": "C-family Frontend",
        "c-family/c-common.h": "C-family Frontend",
        "c/c-decl.c": "C-family Frontend",
        "cp/typeck2.c": "C-family Frontend",
        "combine.c": "Peephole Optimizations",
        "predict.def": "Target Info",
        "ipa-sra.c": "Interprocedural SRoA",
        "cgraph.c": "Call Graph Handling",
        "cgraphclones.c": "Call Graph Handling",
        "cgraph.h": "Call Graph Handling",
        "passes.def": "Pass Management",
        "tree-ssa-threadedge.c": "Jump Threading",
        "tree-ssa-threadbackward.c": "Jump Threading",
        "tree-ssa-threadbackward.h": "Jump Threading",
        "tree-ssa-copy.c": "Copy Propagation",
        "gimple-loop-versioning.cc": "Loop Transformations",
        "gimple-ssa-evrp.c": "Value Propagation",
    }
    file_counts = defaultdict(int)
    commits = set()
    per_commit_files = defaultdict(list)
    commits_per_category = defaultdict(set)
    files_per_category = defaultdict(set)
    categories = set()

    def read_data():
        if cc == "llvm":
            return read_llvm_commit_and_files()
        if cc == "gcc":
            return read_gcc_commit_and_files()

    for commit, file in read_data():
        commits.add(commit)
        file_counts[file] += 1
        try:
            category = category_map[file]
            categories.add(category)
            commits_per_category[category].add(commit)
            files_per_category[category].add(file)
        except KeyError as e:
            print(f"Uncategorized file: {e} (commit {commit})")

    rows = [("Component", "# Commits", "# Files")]
    for category in sorted(categories):
        rows.append(
            (
                category,
                str(len(commits_per_category[category])),
                str(len(files_per_category[category])),
            )
        )

    pad_lens = [max(map(len, column)) for column in zip(*rows)]
    for row in rows:
        print(
            " | ".join(
                col + ((pad - len(col)) * " ") for col, pad in zip(row, pad_lens)
            )
        )
```

```python
if __name__ == "__main__":
    print("LLVM")
    print_cc("llvm")
    print()
    print("GCC")
    print_cc("gcc")
```

2.3 一些实验结果

官方给的实验的结果文件在：

There are four end-to-end regression examples in `/dce/end_to_end_examples` , two for GCC and two for LLVM.
Each example contains most of the following files:

- `code.c` : CSmith生成的带有优化标记（markers）的源代码
- `scenario.json` : 描述了编译器种类和优化级别
- `interesting_settings.json` : describes the `bad_setting` , that is the compiler which misses a marke, and `good_settings` the compiler(s) which can eliminate it（描述了哪个编译器错过了对marker的清除以及哪个编译器对marker的成功清除）

```json
{"bad_setting": {"compiler_config": "clang", "rev": "4c8b8e0154f075e463428acc0640388c40d60097", "opt_level": "3", "additional_flags": ["-I/usr/include/csmith-2.3.0"]}, "good_settings": [{"compiler_config": "clang", "rev": "ed403e4cb2e5c9c61d2fbb44bae03c5603290bf1", "opt_level": "3", "additional_flags": ["-I/usr/include/csmith-2.3.0"]}]}
```

- `marker.txt` : 错过的marker
- `reduced_code_0.c` : a reduced (via creduce) version of `code.c` 通过CReduce版本的Creduce)
    - Creduce后需要编译的代码大小大大缩小

C 复制代码

```c
static int b = -1;
void DCEMarker3_(void);
char(a)(char c, int d) { return d >= 2 ? c : c << d; }
static char e(unsigned char c) {
    if (0 == a(~0, c))
      DCEMarker3_();
    return c;
}
char f(int *c, unsigned d) { return d; }
int main() {
    int *g = &b, *h = &b;
    char i;
    i = f(h, *h);
    e(i);
    *g = 0;
}
```

- `bisection_0.txt` : the commit which introduced the regression(带来regression的提交)
- `massaged_code.c` : a cleaned up version of `reduced_code_0.c` which we used for reporting
- `bug_report` : the bug tracker url
- `fixed_by` : the commit that fixes the regression(修复了regression的提交)