Алгоритам претраге Best-First DFS у оквиру алата за симболичко извршавање KLEE

Практични семинарски рад у оквиру курса
Верификација софтвера
Математички факултет

Петар Куколь, 1097/2020
mr13095@alas.matf.bg.ac.rs

6. август 2021.

Сажетак

У овом раду биће описана идеја и имплементација Best-First DFS, једног алгоритма за претрагу у оквиру алата за симболичко извршавање KLEE. Овај рад подељен је на три целине: у уводном делу биће речи о самој идеји Best-First DFS алгоритма, у другом делу рада биће размотрени детаљи имплементације ове стратегије претраге у оквиру самог алата KLEE, а у последњем делу биће представљене перформансе алгоритма Best-First DFS над подскупом пакета програма *GNU coreutils*. [1]

Садржај

1 Увод

Код симболичког извршавања обилазак свих могућих путања кроз програм често може бити скуп или готово немогућ са меморијског или временског аспекта, због чега се прибегава коришћењу разних стратегија навођења претраге заснованим на различитим хеуристикама и алгоритмима претраге. Једна од основних стратегија, подржана у готово свим алатима за симболичко извршавање, јесте претрага у дубину (енг. *depth-first search*, надаље скраћено *DFS*).

Главна предност класичне претраге у дубину приликом претраге стања у оквиру алата за симболичко извршавање је добра меморијска ефиксаност. Разлог томе је чињеница да DFS сваку започету путању прати до краја, због чега нема потребе да се истовремено чува велики број стања у меморији. Међутим, исти овај разлог због кога је DFS меморијски ефикасан доводи уједно и до његове највеће мане: претрага може да се "заглави" у петљама или рекурзијама које зависе од симболичких променљивих, што доводи до лоших перформанси овог алгоритма претраге у смислу покривености грана и инструкција над програмима који их садрже.

Best-First DFS алгоритам, заснован на *best-first search* алгоритму објашњеном у [2], настоји да исправи мане класичног DFS алгоритма, притом у што већој мери чувајући меморијску ефикасност. Наиме, Best-First DFS претрага представља DFS претрагу навођену тзв. *best-first* хеуристиком. Ова хеуристика од тренутно доступних стања бира оно које је заустављено на линији кода која је најмањи број пута извршена. По избору најбољег стања хеуристиком, Best-First DFS претрага наставља са DFS претрагом од тог стања надаље. Када DFS претрага обради одређени број инструкција од последњег избора најбољег стања, тада се DFS претрага зауставља и бира се ново најбоље стање. Идеја иза бирања стања заустављеног на најмање пута извршеној инструкцији је да се приоритизују мање истражена доступна стања, у нади да ће то довести до повећања покривености инструкција и грана. Чињеница да Best-First DFS алгоритам користи DFS претрагу ограничену буџетом инструкција, као и да се претрага даље наставља после избора стања које је заустављено на инструкцији која је најмањи број пута извршена, обезбеђује да се Best-First DFS претрага не "заглави" у петљама или рекурзијама. Стога, за овако осмишљену претрагу очекује се да од класичне DFS претраге буде ефикаснија како у погледу покривености инструкција, тако и у погледу покривености грана, што ће у поглављу 3 бити и експериментално проверено.

2 Имплементација

Имплементација новог алгоритма претраге у алат KLEE састоји се из следећа два корака:

1. У датотеке `Searcher.h` и `Searcher.cpp` додаје се класа за нови алгоритам претраге која наслеђује базну класу `Searcher`. Базна класа `Searcher` обезбеђује основни интерфејс претраживача (енг. *searcher*) који имплементира стратегију претраге. Главни методи који имплементирају стратегију претраге су `selectState` за одабир следећег стања у претрази и `update` за ажурирање скупа стања самог претраживача.

2. У датотеку `UserSearcher.cpp` додаје се нови претраживач на листу доступних претраживача, као и одговарајуће документационе поруке и евентуалне нове опционе аргументе командне линије потребне за нови претраживач.

Best-First DFS стратегија претраге имплементирана је класом `BestFirstDFSSearcher` чија је декларација из заглавља `Searcher.h` дата у листингу 1. Атрибут `states` типа `vector<ExecutionState*>` садржи сва стања која су позната претраживачу, док атрибут `instsBeforeNewBest` типа `unsigned` чува инструкцијски буџет за DFS претрагу пре поновног коришћења хеуристике за одабир новог најбољег стања; логички атрибут `selectBest` служи да се означи да је инструкциони буџет за DFS претрагу испуњен, и да приликом следећег позива методу `selectState` треба извршити селекцију новог најбољег стања best-first хеуристиком. Атрибут `instructions` типа `uint64_t` садржи број инструкција које су биле извршене у тренутку када је последњи пут примењена best-first хеуристика и на основу њега се закључује када је прекорачен буџет инструкција.

```
0  class BestFirstDFSSearcher final : public Searcher {
1    std::vector<ExecutionState*> states;
2    unsigned instsBeforeNewBest;
3    bool selectBest;
4    uint64_t instructions;
5
6    public:
7      ExecutionState &selectState() override;
8      void update(ExecutionState *current,
9                  const std::vector<ExecutionState *> &addedStates,
10                 const std::vector<ExecutionState *> &removedStates
       ) override;
11     bool empty() override;
12     void printName(llvm::raw_ostream &os) override;
13 };
```

Листинг 1: Класа која имплементира Best-First DFS претрагу

Метод `update` није нарочито занимљив. Његов једини посао је да претраживачу доступна стања садржана у атрибуту `states` ажурира додавањем нових стања из колекције `addedStates` и брисањем стања означених за брисање из колекције `removedStates`. Сва логика Best-First DFS претраге одвија се у методу `selectState`, чији се комплетан код може видети у листингу 2. Уколико је `selectBest` постављен на `false`, то јест уколико није потребно хеуристиком бирати ново стање, тада се враћа последње додато стање из колекције познатих стања, што одговара DFS претраги. Успут се испитује да ли је прекорачен инструкциони буџет и евентуално ажурира `selectBest` ако јесте. Када је `selectBest` постављен на `true`, значи да је у претходном кораку прећен задати буџет инструкција и да је потребно применити хеуристику. Тада се петљом пролази кроз целокупну листу познатих стања и проналази се оно које је заустављено на најмање пута извршеној инструкцији. Након што је такво стање пронађено, ажурирају се атрибути `selectBest` и `instructions`, и враћа се то пронађено стање.

```
ExecutionState &BestFirstDFSSearcher::selectState() {
  if (selectBest)
  {
    ExecutionState* bestState = states.back();
    uint64_t instCount = theStatisticManager->getIndexedValue(
     stats::instructions, states.back()->pc->info->id);
    for (const auto state : states)
    {
      uint64_t currCount = theStatisticManager->getIndexedValue(
     stats::instructions, state->pc->info->id);
      if (currCount <= instCount)
      {
        bestState = state;
        instCount = currCount;
      }
    }
    selectBest = false;
    instructions = stats::instructions;
    return *bestState;
  }
  else
  {
    if (stats::instructions >= instructions + instsBeforeNewBest)
      selectBest = true;
    return *states.back();
  }
}
```

Листинг 2: Метод `selectState` класе `BestFirstDFSSearcher`

У датотеку `UserSearcher.cpp` додат је претраживач дефинисан класом `BestFirstDFSSearcher` и допуњен је одговарајући део `--help` блока информацијама о овом претраживачу. Такође, додат је аргумент командне линије `--best-first-dfs-instructions` којим се може контролисати буџет инструкција доступан пре поновне примене best-first хеуристике.

3 Резултати упоредног тестирања

У овом поглављу биће извршено упоредно тестирање стратегија претраге у алату KLEE над подскупом пакета програма *GNU coreutils*. Биће тестиране следеће четири стратегије:

1. Best-First DFS са подразумеваним буџетом инструкција од 10 000 (у резултатима означена као *bfdfs10k*)
2. Best-First DFS са буџетом инструкција од 20 000 (у резултатима означена као *bfdfs20k*)
3. Класичан DFS (у резултатима означена као *dfs*)
4. Подразумевани алгоритам претраге алата KLEE – насумични одабир путања (енг. *Random Path Selection*) преплет са неуниформном насумичном претрагом (енг. *Non Uniform Random Search*) која више вреднује стања која су скорије покрила нову инструкцију (у резултатима означена као *default*)

Стратегије претраге биће тестиране на подскупу алата из *GNU coreutils* пакета којег чини следећих десет алата:

- *chown*
- *cp*
- *dd*
- *echo*
- *expr*
- *kill*
- *mkdir*
- *printf*
- *sha512sum*
- *touch*

За тестирање користе се најновије верзије доступне на званичним git репозиторијумима *GNU coreutils* пакета [3] и алата KLEE [4]. Од опција алата KLEE приликом тестирања користе се опције [5] које су аутори алата KLEE користили за тестирање *GNU coreutils* пакета програма у раду [6]. У листингу 3 може се видети пример позивања алата KLEE са поменутим опцијама за алат *mkdir* и претраживачем Best-First DFS са подразумеваним опцијама.

```
$ klee --simplify-sym-indices --write-cvcs --write-cov --output-
    module --max-memory=1000 --disable-inlining --optimize --use-
    forked-solver --use-cex-cache --libc=uclibc --posix-runtime
    --external-calls=all --only-output-states-covering-new --max-
    sym-array-size=4096 --max-solver-time=30s --max-time=60min --
    watchdog --max-memory-inhibit=false --max-static-fork-pct=1
    --max-static-solve-pct=1 --max-static-cpfork-pct=1 --switch-
    type=internal --search=best-first-dfs coreutils/obj-llvm/src/
    mkdir.bc --sym-args 0 3 10 --sym-files 1 8 --sym-stdin 8 --
    sym-stdout
```

Листинг 3: Пример позивања алата KLEE са поменутим опцијама

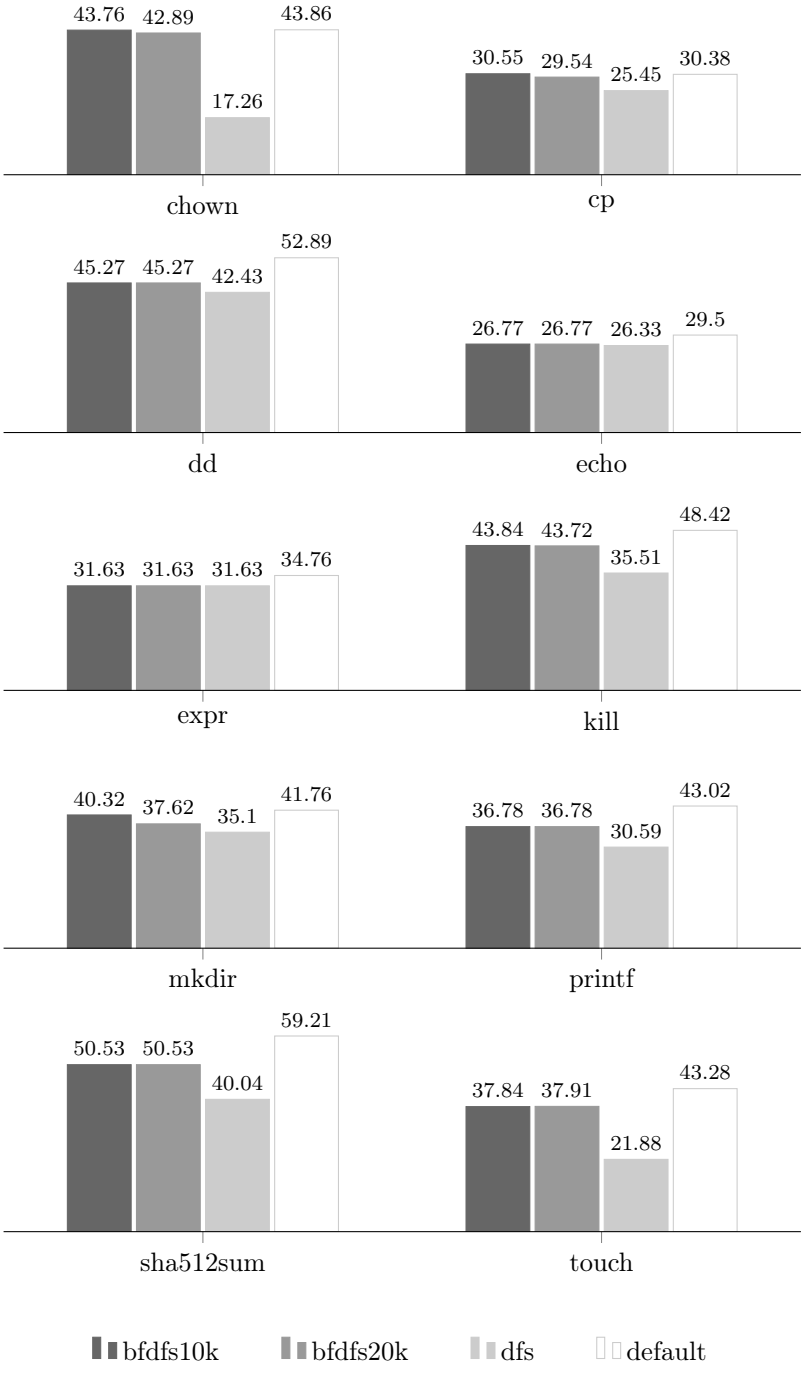
У табели 1 може се видети остварена покривеност инструкција, а у табели 2 остварена покривеност грана за све четири стратегије претраге над поменутим подскупом *GNU coreutils* алата. Из датих остварених резултата може се приметити да је Best-First DFS алгоритам претраге на готово свим алатима постигао бољу покривеност инструкција и грана од класичног DFS алгоритма. Подразумевана стратегија претраге алата KLEE је остварила боље резултате од Best-First DFS стратегије, што се и дало очекивати због тога како је она дизајнирана. У просеку, Best-First DFS претрага остварује 34% бољу покривеност инструкција и 45% бољу покривеност грана од класичне DFS претраге, док подразумевана претрага алата KLEE остварује 10% бољу покривеност инструкција и 16% бољу покривеност грана од Best-First DFS претраге. Такође, дуплирање инструкционог буџета (са десет на двадесет хиљада) Best-First DFS претрази није донело значајније промене оствареног резултата. Визуелизација резултата упоредног теста из табела 1 и 2 може се видети на графиконима 1 и 2.

	bfdfs10k	**bfdfs20k**	**dfs**	**default**
chown	43.76	42.89	17.26	43.86
cp	30.55	29.54	25.45	30.38
dd	45.27	45.27	42.43	52.89
echo	26.77	26.77	26.33	29.50
expr	31.63	31.63	31.63	34.76
kill	43.84	43.72	35.51	48.42
mkdir	40.32	37.62	35.10	41.76
printf	36.78	36.78	30.59	43.02
sha512sum	50.53	50.53	40.04	59.21
touch	37.84	37.91	21.88	43.28

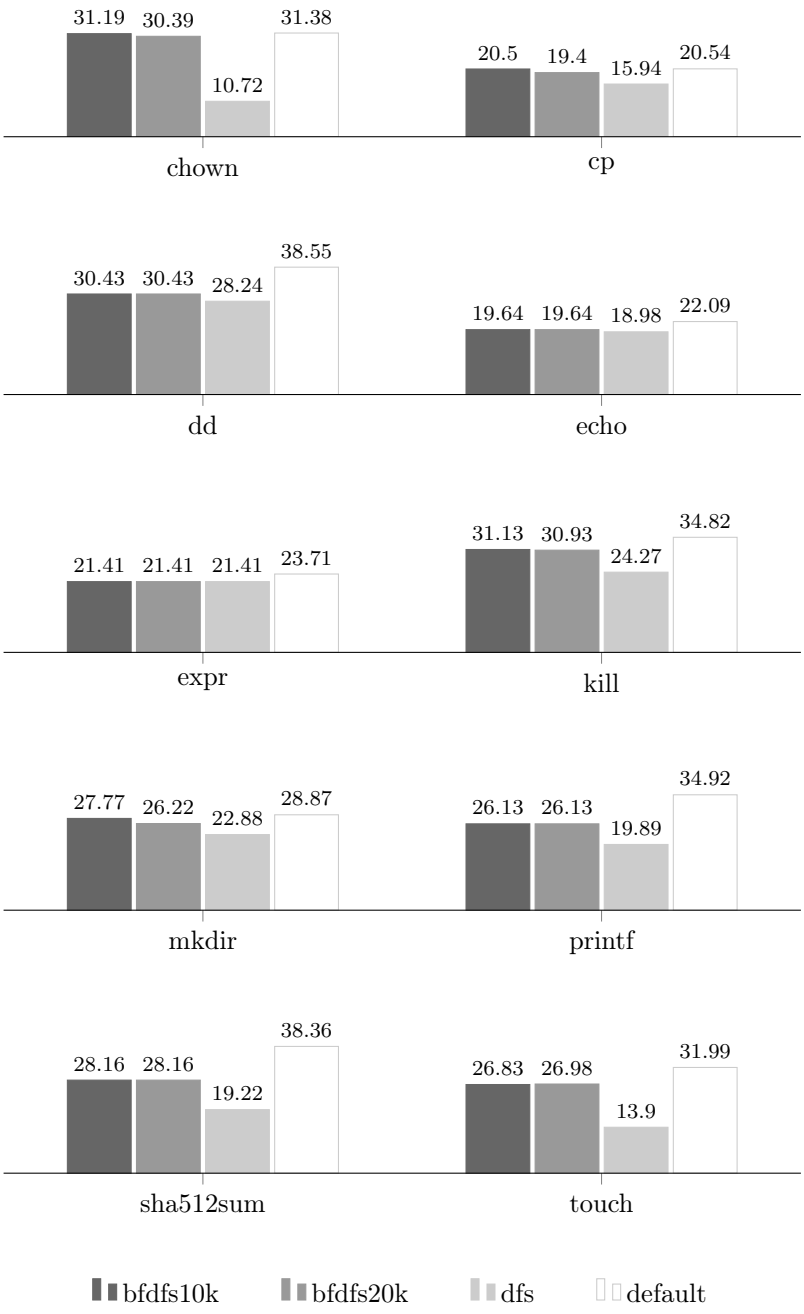
Табела 1: Проценат покривености инструкција

	bfdfs10k	**bfdfs20k**	**dfs**	**default**
chown	31.19	30.39	10.72	31.38
cp	20.50	19.40	15.94	20.54
dd	30.43	30.43	28.24	38.55
echo	19.64	19.64	18.98	22.09
expr	21.41	21.41	21.41	23.71
kill	31.13	30.93	24.27	34.82
mkdir	27.77	26.22	22.88	28.87
printf	26.13	26.13	19.89	34.92
sha512sum	28.16	28.16	19.22	38.36
touch	26.83	26.98	13.90	31.99

Табела 2: Проценат покривености грана



Графикон 1: Проценат покривености инструкција

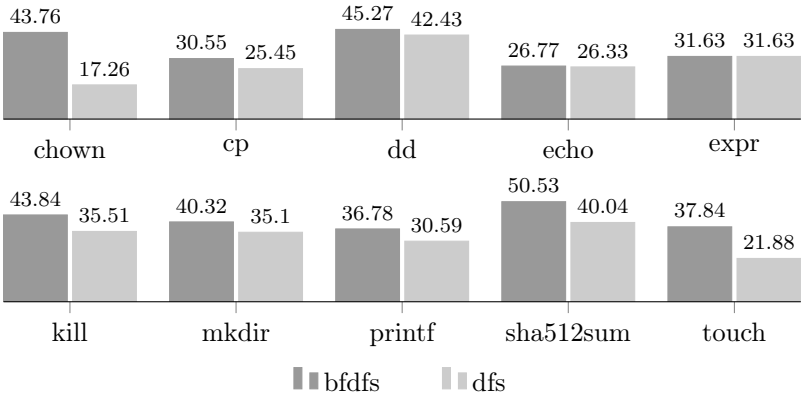


Графикон 2: Проценат покривености грана

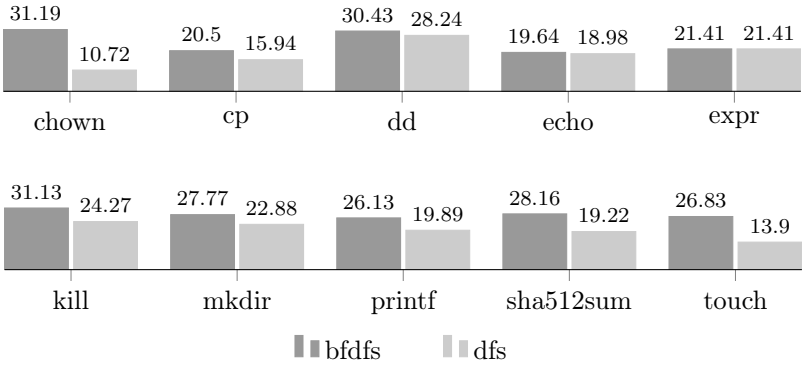
4 Закључак

Best-First DFS у просеку остварује значајно бољу покривеност и инструкција и грана, што се може лако видети на издвојеним графиконима 3 и 4. Наравно, ова стратегија због ограничене могућности DFS претраге коју користи да допре до непосећених стања, нешто је лошија од подразумеване стратегије алата KLEE, али за разлику од ње користи доста мање меморије. У току извођења тестирања над алатима из *GNU coreutils* пакета, због коришћења меморијског ограничења од 1000 мегабајта, подразумевана стратегија алата KLEE је веома често морала да одбацује започета стања, док код стратегије Best-First DFS то није био случај.

Даљи развој ове стратегије могао би да иде у правцу модификовања best-first хеуристике тако да бира најбоље стање по неком другом критеријуму, или у правцу иницијалног коришћења претраге у ширину као у раду [7], са циљем да се прошири спектар стања од којих best-first хеуристика може да бира.



Графикон 3: Покривеност инструкција стратегијама Best-First DFS и DFS



Графикон 4: Покривеност грана стратегијама Best-First DFS и DFS

Литература

[1] Free Software Foundation. GNU coreutils. on-line at: https://www.gnu.org/software/coreutils/.

[2] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically Generating Inputs of Death. *ACM Trans. Inf. Syst. Secur.*, 12(2), December 2008.

[3] Free Software Foundation. GNU coreutils git repository. on-line at: git://git.sv.gnu.org/coreutils – commit hash: `02fc0e3`.

[4] The KLEE Team. KLEE Symbolic Execution Engine. on-line at: https://github.com/klee/klee – commit hash: `df04aea`.

[5] The KLEE Team. OSDI'08 Coreutils Experiments. on-line at: https://klee.github.io/docs/coreutils-experiments/.

[6] Cristian Cadar, Daniel Dunbar, and Dawson Engler.
KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. OSDI'08, page 209–224, USA, 2008. USENIX Association.

[7] Strahinja Stanojević. Proširivanje alata KLEE naprednim algoritmom pretrage stabla izvršavanja programa. Master's thesis, Matematički fakultet, Univerzitet u Beogradu, 2020.