Санкт-Петербургский государственный университет

Кафедра системного программирования

Двунаправленное направляемое конфликтами резюмирование кода в символьной виртуальной машине V#

Седлярский Михаил Андреевич, 21.M07-мм группа

Научный руководитель: к.ф.-м.н. Д.А. Мордвинов, доцент кафедры системного программирования

Санкт-Петербург
2022

Введение

- Современная разработка нуждается в автоматическом создании тестов
- Символьное исполнение решает эту проблему, но плохо масштабируется
- Резюмирование (summurization) кода как способ борьбы с комбинаторным взрывом путей

Листинг 1: Программа со взрывом путей

```
int max = 0;
for (int i = 0; i < a.Length; ++i) {
  max = Math.max(max, Math.abs(a[i]));
}
assert(max >= 0);
```

Существующие подходы

- Резюмирование бывает слабо- и сверх- аппроксимирующим
- Резюме (сводки) можно представлять представлять в формулах КНФ и ДНФ
- Сводки можно обобщать интерполяцией Крейга или через unsat ядра

Листинг 2: Функция Abs

```
int abs(int x) {
    return x < 0 ? -x : x;
}
```

$$x < 0 \wedge abs(x) = -5 \tag{1}$$

$$x \neq INT\_MIN \Longrightarrow abs(x) > 0 \tag{2}$$

Существующие подходы

В данной работе будут использоваться сверх-аппроксимирующие резюме, прдеставленные в виде КНФ формул логики первого порядка. Резюме будут обобщаться с помощью unsat ядер

- Выводы
  - Подвести итог
  - Указать недостатки существующих подходов, на борьбу с которыми направленна данная работа
  - Чётко сформулировать существующую проблему, которая будет решаться в данной работе

Постановка задачи

Целью работы является разработка и реализация алгоритма двунаправленного направляемого конфликтами резюмирования кода в символьной виртуальной машине для платформы .net core

Задачи:

- реализовать извлечение unsat ядер из ограничений пути
- реализовать двунаправленное символьное исполнение в виртуальной машине V#
- реализовать двунаправленные эвристики на основе unsat-ядер

Задачи

Unsat ядра: Запишем конъюнкцию формул $c_1, ..., c_n$ как множество $PC = \{c_1, ..., c_n\}$. Пусть PC невыполнимо в некой теории Г. Тогда минимальное невыполнимое (unsat) ядро формулы PC это – минимальное невыполнимое в теории Г подмножество PC. Необходимо расширить существующую струткуру ограничений пути и интегрировать поиск unsat ядер SMT решателя Z3.

Двунаправленный поиск и эвристики: Текущая реализация двунаправленного исполнения в V# требует реинжениринга для увеличения гибкости. Необходио добавить стратегии поиска точек обратного распространения на основе unsat ядер.

Алгоритм

Двунаправленное исполнение состоит из следующих шагов:

- На вход символьной машины подаётся программа и набор целевых инструкций
- Стратегия прямого исполнения пробует найти пути, ведущие к назначенным целям
- Если кол-во путей с невыполнимыми ограничениями превышает заданный порог, то создаётся точка обратного распространения (inti point)
- От заданной точки начинается изолированная попытка дойти до целей. Все ранее известные значения переменных в изоляции будут заменены на символьные.
- Дойдя до цели формируется наислабейшее предусловие (WP), которое затем распространяется назад по контекстам.
- Теперь достигнутые цели можно сместить на точку обратного распространянения
- Полученное WP будет использоваться при создании лемм и дальнейшем резюмировании кода

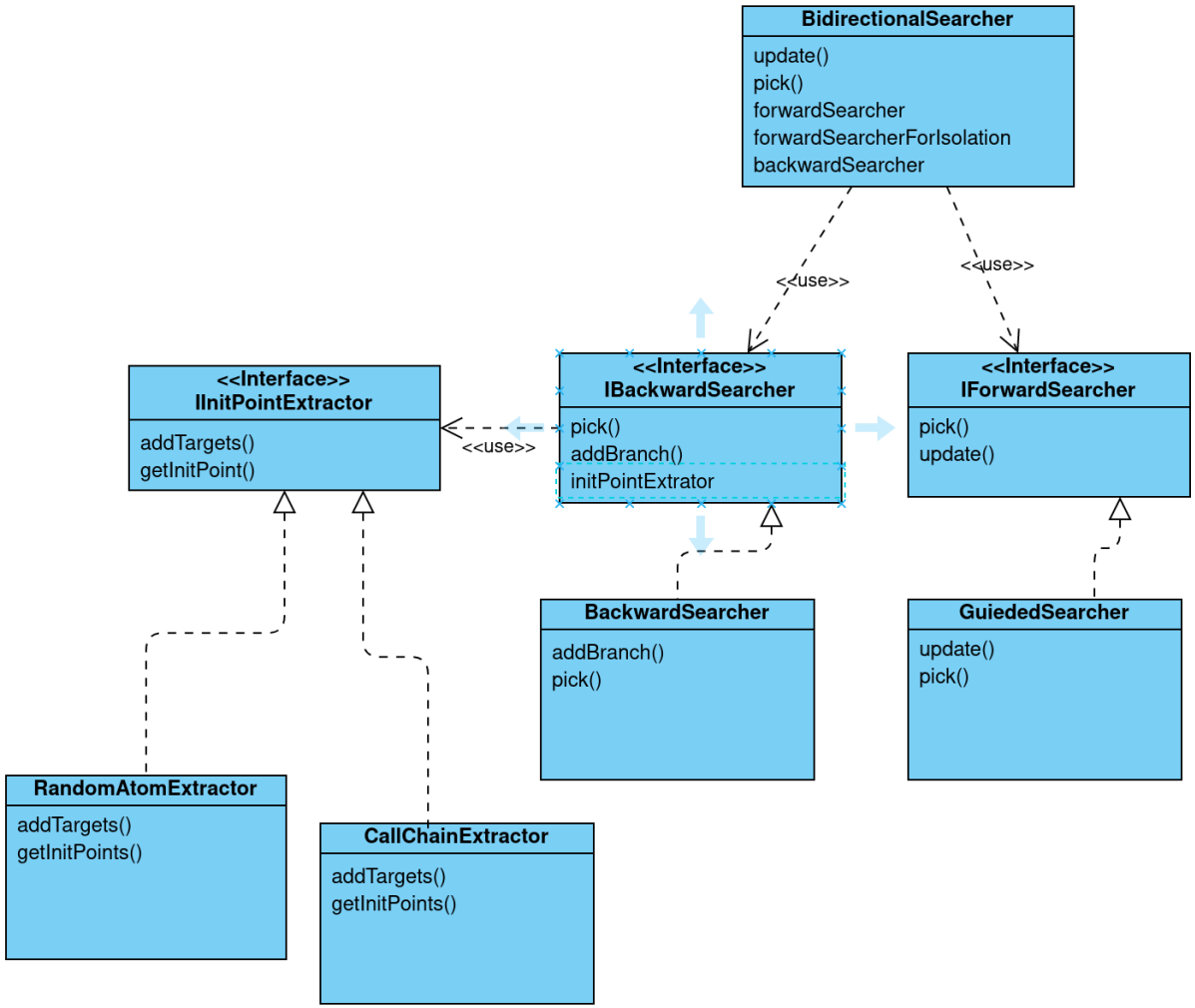
Эвристики извлечения init points

Под точкой обратного распространянения понимается пара ($from$, T). Где $from$ это локация в коде, а T – множество целевых локаций.

- Target-based эвристика. Если путь невыполним и имеет цели, то создаём точку от случайного атома unsat ядра до целей пути.
- Call-chain эвристика. Если путь невыполним, то создаётся ряд точек, ведущих по цепочке вызовов от первого атома до последнего. Атомы ядра упорядочиваются по мере добавления в ограничения пути. Если у пути есть цели, то подобным образом создаётся цепочка точек от последнего атома до целей. Таким образом генерируются больше WP, которые могут пригодиться в последующем резюмировании кода. По определнию, подход использует все unsat ядра

Архитектура решения

- Система символьных ограничений пути была расширена привязками к коду, через монотонно увелчивающийся счётчик была введена хронология ограничений
- Для экономиии времени, затрачиваемой на сборку мусора, была введена мемоизация объектов codeLocation, отвечающих за хранение точек в коде.
- Архитектура двунаправленного исполнения была расширена интерфейсом InitPointExtractor


BidirectionalSearcher
update()
pick()
forwardSearcher
forwardSearcherForIsolation
backwardSearcher
<<use>>
<<use>>
<<Interface>>
IBackwardSearcher
pick()
addBranch()
initPointExtrator
<<Interface>>
IForwardSearcher
pick()
update()
<<Interface>>
IInitPointExtractor
addTargets()
getInitPoint()
<<use>>
BackwardSearcher
addBranch()
pick()
GuiededSearcher
update()
pick()
RandomAtomExtractor
addTargets()
getInitPoints()
CallChainExtractor
addTargets()
getInitPoints()

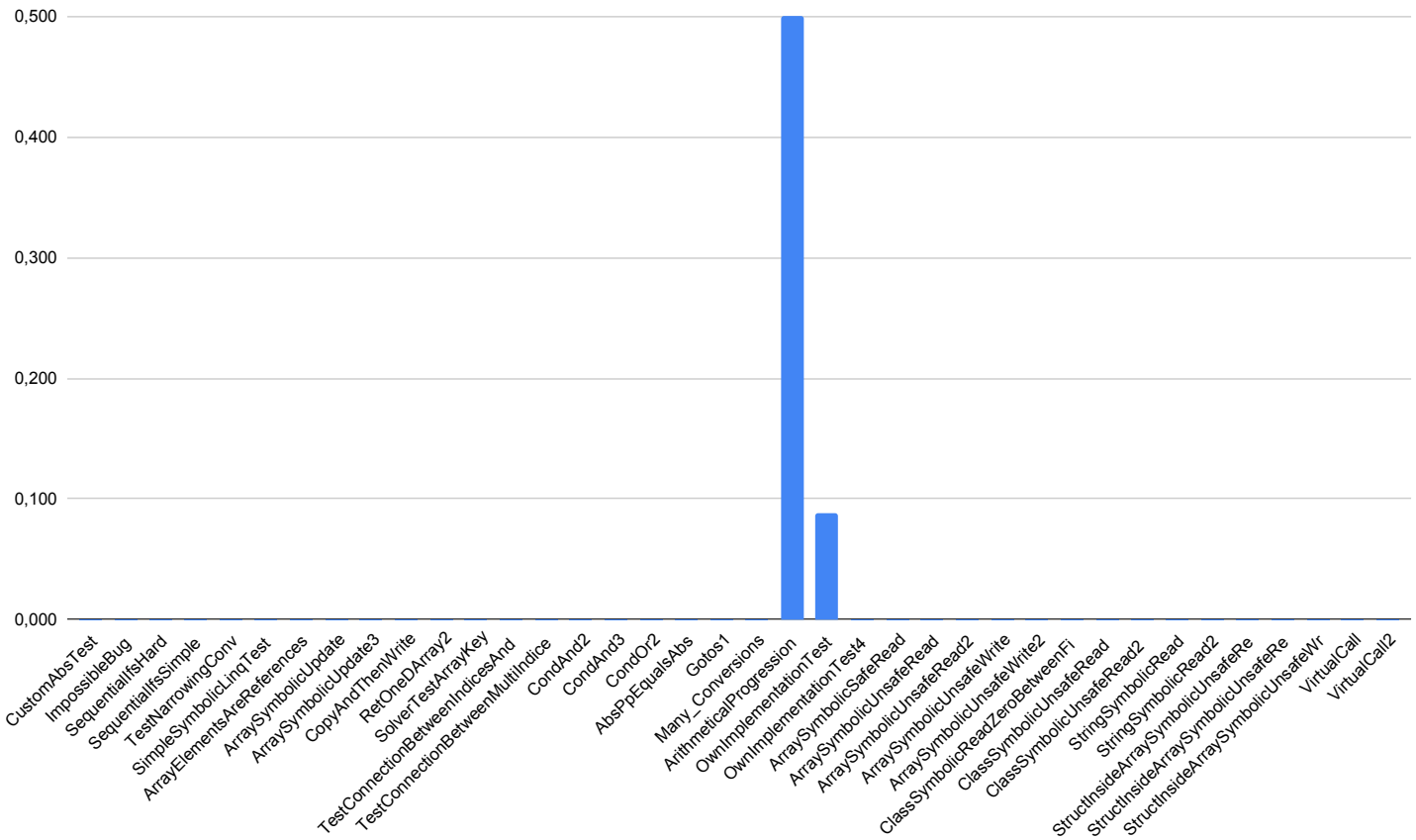
Экспериментальное исследование

Постановка эксперимента

- Проверим степень утилизации unsat ядер в target-based подходе
- Символьная машина будет запускаться на 511 неболших программах
- Из 511 тестов только в 38 появляются невыполнимые ограничения пути. На графике избражена доля использованных unsat ядер. Только 2 из 38 тестов смотрят на информацию, извлекаюемую из конфликтов. В среднем, в target-based подходе отбрасывается $98,5\%$ ядер

Экспериментальное исследование

Рис.: График утилизации unsat ядер в target-based эвристике. По оси X тесты, по оси Y доля использованных unsat ядер от общего числа



Результаты

- Реализовано извлечение unsat ядер из ограничений пути
- Реализован каркас двунаправленного исполнения
- Предложено две эвристики извлечения точек обратного распространения
- Одна из эвристик опробована на практических тестах и не показала стоящих результатов. Вторая эвристика находится в стадии доработки

Планы на следующий семестр:

- Реализовать эффективную эвристику извлечения точек обратного распространения
- Реализован базу данных лемм
- Реализовать механизм распространения относительно индуктивных лемм