Разработка алгоритмов статического поиска выходов за пределы динамического массива в C/C++ программах

И. Е. Громаковский

Научный руководитель: М. А. Лукин

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Описание задачи

- Статический поиск выходов за пределы динамического массива в C и C++.
- Работа с большими программами за разумное время.
- В общем случае задача поиска всех ошибок неразрешима.
- Находить как можно больше ошибок, минимизируя число ложных срабатываний.

Актуальность проблемы

- Программное обеспечение всегда было и остаётся подвержено ошибкам в коде.
- Выход за пределы массива — одна из главных уязвимостей с точки зрения безопасности.
- Особенно опасны ошибки в операционных системах и сетевых программах, для которых популярны языки C и C++.

Анализ LLVM-IR

- Меньше языковых конструкций, проще анализ.
- Анализируемый код ближе к реально выполняемому.
- Встроенные оптимизации и средства для анализа кода.
- Автоматически поддерживается любой язык, для которого есть компилятор в LLVM-IR.
- Static Single Assignment.

Li, Cifuentes, Keynes: ranges

- Li L., Cifuentes C., Keynes N. Practical and Effective Symbolic Analysis for Buffer Overflow Detection.
- Symbolic ranges.
  - Symbolic expressions: 15, $x + 1$, $\bot$, $\top$.
  - Частичный порядок.
  - Symbolic range: $[s1, s2]$.
  - Операции: $\cup, \cap, +, -, \times, \div$.
- Define range, S_V:
  - в месте определения V.
- Use range, $S_{V,P}$:
  - в произвольном месте P.
- Выход за пределы массива размера n в инструкции P:
  - $S^{max}_{n,P} \prec S^{max}_{index,P} \vee S^{min}_{index,P} \prec -1$.

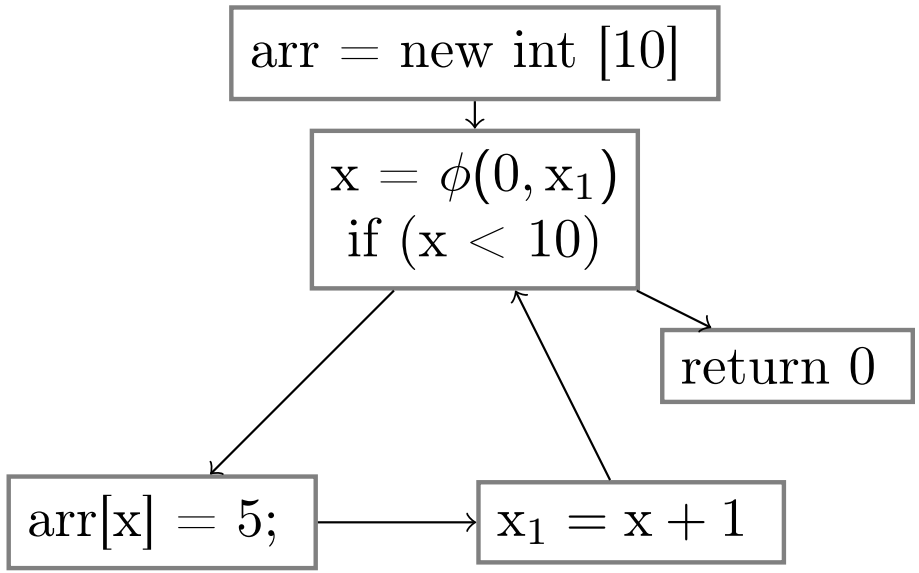
Li, Cifuentes, Keynes: зависимости

- Зависимости по данным.
  - Define range вычисляется через диапазоны аргументов.
  - $\mathrm{S}_{(\mathrm{a+b})} = \mathrm{S}_{\mathrm{a,P}} + \mathrm{S}_{\mathrm{b,P}}, \mathrm{P} = \mathrm{a} + \mathrm{b}$
  - $\mathrm{S}_{(\phi(\mathrm{a,b}))} = \mathrm{S}_{\mathrm{a,P}} \cup \mathrm{S}_{\mathrm{b,P}}, \mathrm{P} = \phi(\mathrm{a}, \mathrm{b})$
  - ...
- Зависимости потока управления.
  - Уточнение диапазона $\mathrm{S}_{\mathrm{V,P}}$ на основании условных переходов на пути к P.
  - Условные переходы, связанные с V, такие что:
    - P строго доминируется условным переходом;
    - P достижима только из одного потомка условного перехода.

Монотонно изменяющиеся переменные

```
int main()
{
    int * arr = new int[10];
    for (int x = 0; x < 10; ++x)
        arr[x] = 5;

    return 0;
}
```

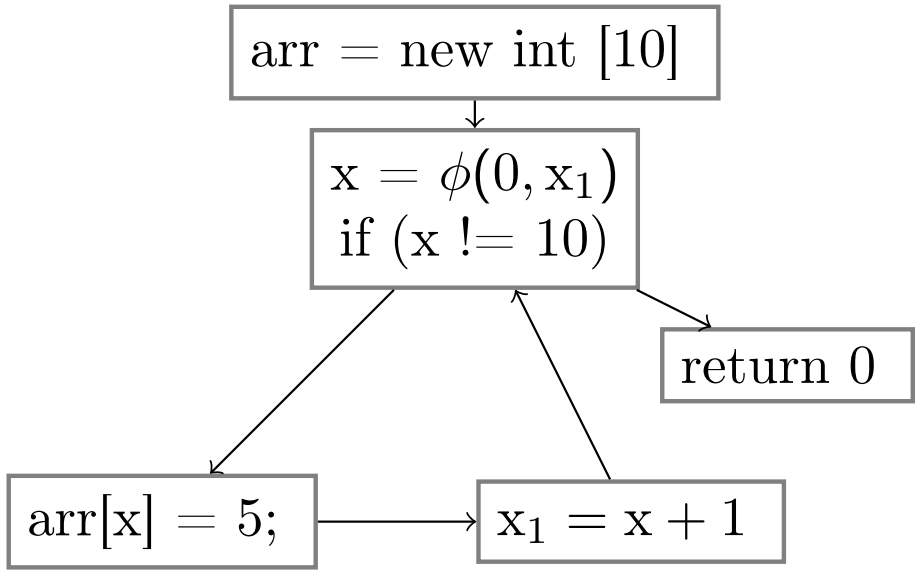


- Нет предикатов, ограничивающих значение снизу.
- Если $x = \phi(a, b), b = f(x)$ и последовательность $x, f(x), f(f(x)) \ldots$ монотонна (например, возрастает), то применяется предикат $x \geq a (x \leq a)$.

Обработка предиката «не равно»

```
int main()
{
    int * arr = new int[10];
    for (size_t x = 0; x != 10; ++x)
        arr[x] = 5;

    return 0;
}
```



- Алгоритм из статьи учитывает $x! = y$, только если значение на границе отрезка.
- Если $x = \phi(a, f(x)), \exists c : y = f^c(a)$ и последовательность $x, f(x), f(f(x)), \ldots$ монотонна (например, возрастает), то применяется предикат $x < y$.

Межпроцедурный анализ

```
#define ISDN_MAX_DRIVERS 32
#define ISDN_MAX_CHANNELS 64

struct isdn_driver * drivers[ISDN_MAX_DRIVERS];

struct isdn_driver * get_drv_by_nr(int di)
{
    struct isdn_driver * drv;
    drv = drivers[di];
    // ...
}

struct isdn_slot * get_slot_by_minor(int minor)
{
    int di, ch;
    struct isdn_driver * drv;
    for (di = 0; di < ISDN_MAX_CHANNELS; di++)
    {
        drv = get_drv_by_nr(di);
        // ..
    }
    // ..
}
```

- $32 \prec \mathrm{S}^{\max}_{\mathrm{di,P}}$.
- $\mathrm{S}^{\min}_{\mathrm{di,P}} \prec -1$.
- P — инструкция вызова.

- Триггер — условие ошибки: $\mathrm{e}_1 \prec \mathrm{e}_2$.
- Функции анализируются от вызываемой к вызывающей.
- Построение триггеров при анализе обращения к массиву.
- Проверка триггеров при анализе вызова функции.

Сравнение: Multi Theft Auto

Multi Theft Auto 1.3.1: 760 тыс. строк кода, 182 мегабайта биткода.

–	TP	FP	Время работы (м:с)
Cppcheck	1	8	11:53
PVS-Studio	4	0	10:36
Splint	—	—	—
Li et at.	7	202	37:55
Улучшенная версия	7	4	42:49

TP (true positive) — число корректно найденных ошибок.

FP (false positive) — число ложных срабатываний.

Сравнение: CMake

CMake 1.3.1: 350 тыс. строк кода, 31 мегабайт биткода.

–	TP	FP	Время работы (м:с)
Cppcheck	1	0	5:59
PVS-Studio	1	0	5:18
Splint	—	—	—
Li et at.	5	79	9:12
Улучшенная версия	5	3	11:03

TP (true positive) — число корректно найденных ошибок.

FP (false positive) — число ложных срабатываний.

Сравнение: синтетические тесты

Синтетические тесты: различные использования массива, 12 ошибочных.

–	TP	FP	FN
Clang Analyzer	0	0	0
CppCheck	3	0	9
PVS-Studio	4	0	8
Splint	9	3	3
Li et at.	12	8	0
Улучшенная версия	12	0	0

TP (true positive) — число корректно найденных ошибок.
FP (false positive) — число ложных срабатываний.
FN (false negative) — число реальных ошибок, не найденных анализатором.

Заключение

- За основу взят известный подход из статьи Li et al.
- Выявлены недостатки подхода.
- Сделаны улучшения, направленные на устранение выявленных недостатков.
- Получился анализатор, способный находить ошибки в больших программах, работающий лучше доступных анализаторов в некоторых случаях.

Спасибо за внимание!

Вопросы?