Ответы на вопросы с лекций

Евгений Букреев

02-simple

1. Что будет, если в нашу систему ввести тип Bool?

Перепишем правила (те, которые не выписаны, оставлены без изменений):

$$E_1 > E_2 : [\![E_1]\!] = [\![E_2]\!] = \text{int} \wedge [\![E_1 > E_2]\!] = \text{bool}$$
$$E_1 == E_2 : [\![E_1]\!] = [\![E_2]\!] \wedge [\![E_1 == E_2]\!] = \text{bool}$$
$$E_1 \texttt{ op } E_2 : [\![E_1]\!] = [\![E_2]\!] = [\![E_1 \texttt{ op } E_2]\!] = \text{int}$$
$$\text{output } E : [\![E]\!] = \alpha$$
$$\text{if } (E)S : [\![E]\!] = \text{bool}$$
$$\text{if } (E)S_1 \text{ else } S_2 : [\![E]\!] = \text{bool}$$
$$\text{while } (E)S : [\![E]\!] = \text{bool}$$

Полученный анализ не изменит точность, потому что он был и есть soundness. Но снизится полнота, потому что станут отвергаться некоторые выражения, которые имеют корректную семантику. Например, (x == y) + 1.

2. Что будет, если в нашу систему ввести тип Array?

Дополним правила типизации новыми конструкциями. Старые остались без изменений.

$$\{\} : [\![\{\}]\!] = \alpha[]$$
$$\{E_1, ..., E_n\} : [\![E_1]\!] = ... = [\![E_n]\!] \wedge [\![\{E_1, ..., E_n\}]\!] = [\![E_1]\!][]$$
$$E[E_1] : [\![E]\!] = \alpha[] \wedge [\![E_1]\!] = \text{int} \wedge [\![E[E_1]]\!] = \alpha$$
$$E[E_1] = E_2 : [\![E]\!] = \alpha[] \wedge [\![E_1]\!] = \text{int} \wedge [\![E_2]\!] = \alpha$$

Протипизируем программу со слайда:

```
main() {
  var x,y,z,t;
  x = {2,4,8,16,32,64}; // [|x|] = [|{2,4,8,16,32,64}|]
  y = x[x[3]];          // [|y|] = [|x[x[3]]|]
  z = {{},x};           // [|z|] = [|{{},x}|]
  t = z[1];             // [|t|] = [|z[1]|]
  t[2] = y;             // [|t|] = alpha[] and [|y|] = alpha
}
```

Решим уравнения:

$$[\![x]\!] = \text{int}[]$$
$$[\![y]\!] = \text{int}$$
$$[\![z]\!] = \text{int}[][]$$
$$[\![t]\!] = \text{int}[]$$

3. Подумайте, что происходит в получившейся реализации, если в программе есть рекурсивный тип?

Тогда программа все равно типизируется, т.к. используется регулярная унификация на основе Union-Find и регулярные рекурсивные термы разрешены.

03-lattices

1. Как выглядит $\sqcup L_1 \times L_2 \times ... \times L_n$? $(\top L_1, \top L_2, ... \top L_n)$. Нижняя аналогично: $(\bot L_1, \bot L_2, ... \bot L_n)$
2. Какая высота произведения решеток? Она равна сумме высот производящих решеток. Так как самый длинный путь от $\top$ до $\bot$ будет проходить через самые длинные пути исходных решеток.
3. Для решетки отображений $A \to L$ точная верхняя грань это отображение $\forall a : A . a \to \top$, а точная нижняя $\forall a : A . a \to \bot$.
4. Решетку отображений $A \to L$ можно выразить как L^n, где n = sizeof(A), поэтому $\text{height}(A \to L) = \text{height}(\text{L}) * \text{sizeof}(A)$
5. Можно ли выразить анализ типов с предыдущей лекции как анализ над решетками? Да, если взять решетку flat от множества возможных типов, где $\bot$ представляет полиморфную типовую переменнную, а $\top$ ошибку типизации.
6. Можно ли выразить анализ над решетками как анализ типов? Да, если мы сами выбираем систему типов. Тогда необходимо явно ввести $\top$ (в разных языках это Any, Object...) и $\bot$ (Nothing). Отношение выражается через subtyping.

04-flow

1. Какова сложность структурного алгоритма для live variables analysis? Сложность структурного алгоритма в общем случае это O(n · h · k), где n – количество узлов CFG, h – высота решетки, а k – время вычисления constraint функции. Тогда для live variables analysis с n узлами и b переменными k = O(b), а ответом будет O(n · b · b). Наличие циклов не меняет оценку.
2. Сложность по памяти O(b) т.к. храним состояние для текущего узла CFG и следующих const узлов.

05-intervals

Для вычисления возможных размеров переменных требуется знать интервал значений, которые эта переменная может достигать в процессе выполнения программы. Для представления этих значений необходимо взять интервальную решетку целых чисел.

Для практических же целей, чтобы получить конечную версию решетки, добавим операцию widening, которая будет аппроксимировать значения до краевых значений предложенных типов. $[0, 1]$ для bool, $[-2^8, 2^8 - 1]$ для short, ... $[-\infty, +\infty]$ для bigint или any.

Также для целей анализа нам потребуется решетка отображений $\text{Vars} \to \text{Intervals}$.

06-path

1. Напишите вариант программы, для которой анализ открытости-закрытости файлов не показывает корректный результат даже с учётом всех возможных условий в переходах.

```
main() {
  var flag;
  input(flag)

  if (flag) {
    open();
  }

  if (flag) {
    close();
  }
}
```

Проблема заключается в том, что значение флага приходит извне. А наш анализ не может доказать, что поведение программы всегда корректно.

2. Предложите, каким образом можно решить описанные в лекции проблемы в этой ситуации.

Для случаев, когда на интересующее нас поведение влияют неизвестные значения, необходимо разбить эти значения на классы эквивалентности и провести анализ для каждого из них. Удастся либо доказать невозможность какого то факта, либо найти контрпример.

08-interprocedural

1. Напишите вариант программы, для которой контекстно-чувствительный анализ знаков требует коэффициент k > 1.

```
bar(y) {
  if (x == 0) return 0;
  if (x < 0) return x + bar(x + 1)
  return x + bar(x - 1)
}

foo(x) {
  return bar(-x);
}
```

```
main() {
  foo(10) // -
  foo(-10) // +
}
```

2. Приведите пример решётки, для которой контекстно-чувствительный анализ в функциональном стиле является более ресурсозатратным, чем контекстно-чувствительный анализ по месту вызова с глубиной 2

Например, решетка $\mathcal{P}(\text{Expressions})$, тогда $\text{States} = \text{Vars} \to 2^{\text{Expression}}$.

1. Размер решетки для анализа по месту вызова $|\text{Nodes}| * 2 * |\text{Vars}| * 2^{|\text{Nodes}|}$
2. В функциональном стиле: $|\text{Nodes}| * |\text{Vars}| * 2^{|\text{Nodes}|} * |\text{Vars}| * 2^{|\text{Nodes}|}$