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):

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

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.

$$
\begin{gathered}
\{\} : [\![\{\}]\!] = \alpha[] \\
\{E_1, ..., E_n\} : [\![E_1]\!] = ... = [\![E_n]\!] \land [\![\{E_1, ..., E_n\}]\!] = [\![E_1]\!][] \\
E[E_1] : [\![E]\!] = \alpha[] \land [\![E_1]\!] = \text{int} \land [\![E[E_1]]\!] = \alpha \\
E[E_1] = E_2 : [\![E]\!] = \alpha[] \land [\![E_1]\!] = \text{int} \land [\![E_2]\!] = \alpha
\end{gathered}
$$

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:

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

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: $(\perp L_1, \perp L_2, ... \perp L_n)$
2. Какая высота произведения решеток? Она равна сумме высот производящих решеток. Так как самый длинный путь от $\top$ до $\perp$ 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 $\forall a : A.a \to \perp$.
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 $\perp$ 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 $\perp$ (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.