Корректность статического анализа

Статические анализаторы пытаются автоматически установить некоторые свойства программ без их выполнения.

Статические анализаторы можно использовать по-разному, например:

– для поиска ошибок, совершаемых программистами;
– в качестве верификаторов, гарантирующих соответствие программы определенным свойствам.

При поиске ошибок анализ должен быть точным (слишком большое количество ложных срабатываний делает инструмент непригодным), однако не ожидается и не предоставляется гарантия обнаружения всех ошибок определенного класса. С другой стороны, при верификации программ *корректность анализа* имеет первостепенное значение: если анализатор не выдает предупреждений, программа должна быть свободна от класса отслеживаемых анализатором ошибок.

Наша задача – разработать корректный статический анализатор для простого императивного языка программирования. Для этого мы будем использовать абстрактную интерпретацию, а доказательство корректности осуществлять с помощью Rocq.

Абстрактная интерпретация на примере

Абстрактная интерпретация – это способ корректной аппроксимации семантики языка программирования. Необходимость аппроксимации объясняется неразрешимостью анализа семантических свойств программ[1]. При аппроксимации:

– конкретные значения переменных заменяются абстрактными;
– конкретная семантика заменяется на семантику, работающую с этими абстрактными значениями.

Например, рассмотрим следующую простую программу:

```
if x > 0 then
  y := x + 1
else
  y := 1
x := x / y
```

Добавим к программе аннотации, в которых показано ее состояние. На рисунке 1.a приведены конкретные значения переменных для случая, когда выполнение программы

[1] https://en.wikipedia.org/wiki/Rice's_theorem

начинается в состоянии, где $x = 10, y = 0$. На рисунке 1.б конкретные значения заменяются интервалами, что позволяет перейти от одного конкретного выполнения программы к абстрактному выполнению программы на всех возможных значениях переменных.

```
{ x = 10, y = 0 }
if x > 0 then
  { x = 10, y = 1 }
  y := x + 1
  { x = 10, y = 11 }
else
  { }
  y := 1
  { }
{ x = 10, y = 11 }
x := x / y
{ x = 0, y = 11 }
```

```
{ x ∈ [−∞, +∞], y ∈ [−∞, +∞] }
if x > 0 then
  { x ∈ [1, +∞], y ∈ [−∞, +∞] }
  y := x + 1
  { x ∈ [1, +∞], y ∈ [2, +∞] }
else
  { x ∈ [−∞, 0], y ∈ [−∞, +∞] }
  y := 1
  { x ∈ [−∞, 0], y ∈ [1, 1] }
{ x ∈ [−∞, +∞], y ∈ [1, +∞] }
x := x / y
{ x ∈ [−∞, +∞], y ∈ [1, +∞] }
```

Рис. 1: Выполнение программы а) на конкретных значениях, б) на абстрактных значениях.

Перед выполнением операции деления на абстрактных значениях мы видим, что переменная $y \in [1, +\infty]$ и не может принимать нулевое значение. Полученная аппроксимация является корректной, т.к. абстрактные значения содержат все возможные конкретные значения. Это позволяет утверждать, что выполнение операции не приводит к ошибке деления на 0 для любого конкретного выполнения программы.

Корректность абстрактной интерпретации

Перейдем от наглядных примеров к формальным определениям.

Абстрактные значения должны обладать:

- структурой решетки;
- отображением конкретизации, связывающим абстрактные значения с конкретными;
- абстрактными операциями, соответствующими конкретным операциям языка программирования.

Решеткой называется частично упорядоченное множество, в котором любое конечное подмножество обладает точной верхней и точной нижней гранями.

Отображением конкретизации называется монотонное отображение $\gamma : A \to \mathcal{P}C$ из абстрактного домена в подмножества значений конкретного.

Абстрактное состояние – это конечное отображение переменных в абстрактные значения.

Задание 1. Определите операции решетки на абстрактных состояниях с помощью операций решетки на абстрактных значениях. Затем, покажите, что на абстрактном состоянии определено отображение конкретизации, индуцированное отображением конкрети-

зации на абстрактных значениях (в файле `AbsInt.v` закончите определения `astateLatticeOp` и `astateConcretization`).

Структура решетки на абстрактных состояниях позволяет задать абстрактную семантику. В частности, существование точной верхней грани позволяет определить семантику условного оператора, а существование неподвижной точки у монотонного отображения на решетке – определить семантику циклического оператора:

$$\begin{aligned}
[\![\text{skip}]\!](S) &= S \\
[\![x := e]\!](S) &= S[x \mapsto [\![e]\!](S)] \\
[\![c_1; c_2]\!](S) &= [\![c_2]\!]([\![c_1]\!](S)) \\
[\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!](S) &= [\![c_1]\!](S) \vee [\![c_2]\!](S) \\
[\![\text{while } e \text{ do } c]\!](S) &= \text{postfixpoint of } F(X) = S \vee [\![c]\!](X)
\end{aligned}$$

Задание 2. Докажите корректность абстрактной интерпретации: абстрактная семантика команд аппроксимирует конкретную семантику (в файле `AbsInt.v` закончите доказательство теоремы `aceval_sound`, для этого сначала нужно доказать корректность определения неподвижной точки, лемма `postfixpoint_sound`, и корректность определения абстрактной семантики выражений, лемма `aeval_sound`).

Применение: распространение констант

Чтобы воспользоваться, полученным абстрактным интерпретатором нужно определить абстрактный домен, на котором он будет работать. Рассмотрим задачу распространения констант (constant propagation), возникающую при компиляции программ. Этот вариант статического анализа определяет имеет ли переменная постоянное значение, и передает его в места использования данной переменной, например, для выполнения дальнейшей свертки констант, уменьшающей избыточные вычисления. Например, статический анализ следующей программы показывает, что $y = 12$, а $z = 11$:

```
x := 1; y := 10; z := x + y;
if x > 0 then
  y := x + z; x := 0
else
  y := 12
```

Задание 3. Реализуйте абстрактный домен для задачи распространения констант (в файле `AbsInt.v` закончите определения `flatZLatticeOp`, `flatZConcretization` и `flatZAbsValue`).

Анализ условий

Текущий вариант абстрактной интерпретации не проводит анализ условий при ветвлениях – абстрактно интерпретируются обе ветви условного оператора, полученные ре-

зультаты объединяются:

$$[\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!](S) = [\![c_1]\!](S) \vee [\![c_2]\!](S)$$

Можно улучшить результаты анализа, если добавить информацию о результате абстрактной интерпретация условия в каждую из ветвей, например:

$$[\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!](S) = [\![c_1]\!](\text{assume\_true}(e, S)) \vee [\![c_2]\!](\text{assume\_false}(e, S)),$$

где assume_true(e, S) возвращает абстрактное состояние $S' \leqslant S$, в котором условие e истинно, а assume_false(e, S) возвращает $S' \leqslant S$, в котором e ложно.

Аналогичное улучшение можно применить к анализу условий циклов:

$$[\![\text{while } e \text{ do } c]\!](S) = \text{assume\_false}(e, \text{postfixpoint of } F(X) = S \vee [\![c]\!](\text{assume\_true}(e, X)))$$

Задание 4. Реализуйте анализ условий и докажите его корректность (создайте файл `AbsIntCond.v`, скопировав файл `AbsInt.v`, внесите необходимые изменения).

Применение: интервалы

…

Задание 5. (в файле `AbsIntCond.v` определите …)