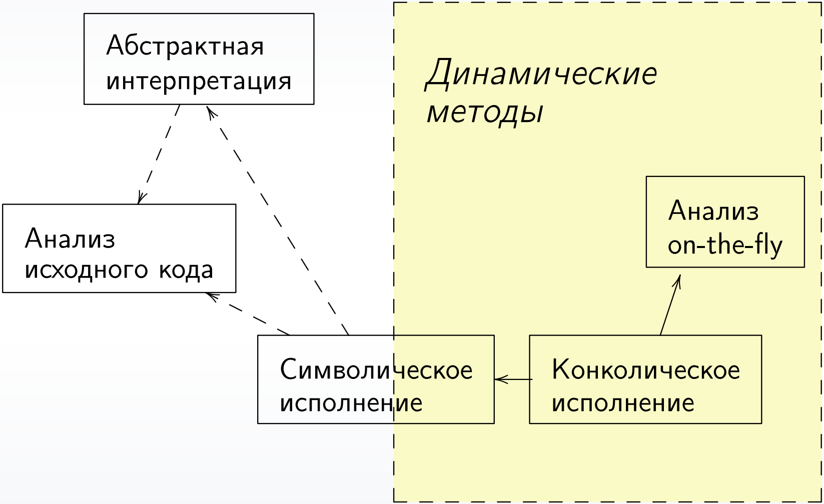
Индуктивный динамический анализ программ

Летняя практика, Переславль-Залесский

4–8 июля 2022 г.

Статический vs. динамический



Абстрактная интерпретация

- Основа метода — факторизация области определения программы с помощью введения гомоморфизма на данных.

Операция ++ — ассоциативная конкатенация, множество данных — строки.

```
f(x, x ++ y) = x ++ 'A' ++ f(x,y)
f(x, /* пустое слово */) = 'A'
```

- Если факторизовать данные так: $\sigma(x) = t_1 \Leftrightarrow x \in A^*$, $\sigma(x) = t_2 \Leftrightarrow \neg(x \in A^*)$, то очевидно, программа тривиализуется до $f(x, x) = x$.
- Факторизация $\sigma(x) = t_1 \Leftrightarrow x \in A^*$, $\sigma(x) = t_2 \Leftrightarrow x \in B^*$, $\sigma(x) = t_3 \Leftrightarrow (x \notin A^* \,\&\, x \notin B^*)$ некорректна из-за перекрытия прообразов абстрактных значений t_1 и t_2.

Символическая интерпретация

- Основа метода — замена конкретных данных на параметры и анализ возможных путей выполнения при конкретизации этих параметров.

```
init(x) = g(f(x ++ 'A'))
f('A' ++ x) = 'B' ++ f(x)       g(B ++ x) = g(x)
f(/* пустое слово */)           g(A ++ x) = False
     = /* пустое слово*/        g(/* пустое слово */)
                                                  = True
```

- Если вычисляется самый внешний вызов (нормальный порядок вычислений, call-by-name), символическая интерпретация доказывает, что `False`-ветка недостижима из `init(x)` для всех значений `x`.
- Плюс: не нужно под каждую задачу строить свой гомоморфизм; минус — слишком общий.

Конколическая интерпретация

- Исполнение на конкретных данных как на символических, с обратным просчётом по путям для порождения наиболее полного накрытия тестами.

```
f('A' ++ x) = f(x)
f(/* пустое слово */) = 'True'
f('Такого теста не будет') = 'AXAXA'
f( _ ++ x) = 'False'
```

- При конколическом запуске `f('AAA')` на первом же ветвлении будут построены тесты `f(/* пустое слово */)`, `f('BAA')` (или другая первая буква, но не `'A'`) и `f('Такого теста не будет')`.
- Требует уже реализованного механизма символической интерпретации и механизма восстановления исходных данных (часто — с привлечением SMT-солверов).

Исполнение программы в модели

- Строится накрытие (чаще) либо подмножество реальных путей исполнения программы.
- По «индукции» строится структура, отражающая некоторые свойства программы.

Индукция здесь инверсна порядку выполнения. Такая обращённая схема индукции иногда называется «коиндукция».

Исполнение программы в модели

- Строится накрытие (чаще) либо подмножество реальных путей исполнения программы.
- По «индукции» строится структура, отражающая некоторые свойства программы.

Индукция здесь инверсна порядку выполнения. Такая обращённая схема индукции иногда называется «коиндукция».

- База — терминальные (конечные) состояния.

Исполнение программы в модели

- Строится накрытие (чаще) либо подмножество реальных путей исполнения программы.
- По «индукции» строится структура, отражающая некоторые свойства программы.

- База — терминальные (конечные) состояния.
- Шаг (от n к $n - k$) — ??? Должен отражать закономерность, а не просто быть переходом к следующему вычислительному состоянию.

Исполнение программы в модели

- Строится накрытие (чаще) либо подмножество реальных путей исполнения программы.
- По «индукции» строится структура, отражающая некоторые свойства программы.

- База — терминальные (конечные) состояния.
- Шаг (от n к $n-k$) — ??? Должен отражать закономерность, а не просто быть переходом к следующему вычислительному состоянию.
- Индуктивный переход (метод сведения к предыдущему случаю) — ??? Как понять, что пути вычислений повторяют уже исследованные?

Порождение путей вычислений

- Дано: абстрактное (символическое, псевдосимволическое) состояние.
- Построить: множество достижимых из него состояний (при той же разметке или факторизации).

Императивный случай

- Рассматривается control-flow-graph.
- По графу строится множество состояний, за шаг достижимых из данного.

Функциональный случай

- Состояние унифицируется с определениями.
- Строится множество результатов применения правил переписывания.

Пример

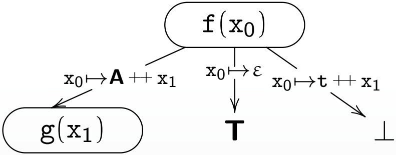
$$\begin{array}{ll} \mathtt{f}(\mathbf{A} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{g}(\mathtt{x}) & \mathtt{g}(\mathbf{B} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{f}(\mathtt{x}) \\ \mathtt{f}(\varepsilon) = \mathbf{T} & \mathtt{g}(\_) = \bot \\ \mathtt{f}(\mathtt{t} \mathbin{+\!\!+} \mathtt{x}) = \bot & \end{array}$$

Здесь $\bot$ — это аналог состояния «ошибка», приводящий к исключению. Семантика сопоставления — стандартная, сверху вниз. Операция $+\!\!+$ — операция конкатенации, которая в этом примере может быть заменена и на обычный cons. Знак _, как обычно при сопоставлении с образцом (например, в Python и в Haskell), означает произвольное значение. Также это правило можно записать в форме $\mathtt{g}(\mathtt{x}) = \bot$, однако использование _ подчёркивает, что этот элемент образца не используется в правой части.

Пример

$$\mathtt{f}(\mathbf{A} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{g}(\mathtt{x}) \qquad \mathtt{g}(\mathbf{B} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{f}(\mathtt{x})$$
$$\mathtt{f}(\varepsilon) = \mathbf{T} \qquad \mathtt{g}(\_) = \bot$$
$$\mathtt{f}(\mathtt{t} \mathbin{+\!\!+} \mathtt{x}) = \bot$$

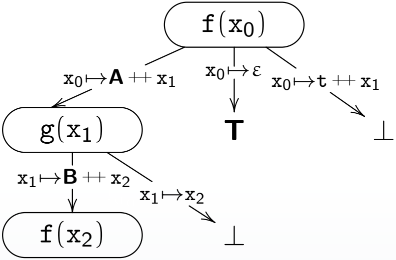
Обратим внимание на очерёдность сиблингов в графе: подстановка $\mathtt{x}_0 \mapsto \mathbf{A} \mathbin{+\!\!+} \mathtt{x}_1$ приоритетнее, чем $\mathtt{x}_0 \mapsto \mathtt{t} \mathbin{+\!\!+} \mathtt{x}_1$, поскольку находится левее.



По подстановкам $\mathtt{x}_0 \mapsto \varepsilon$ и $\mathtt{x}_0 \mapsto \mathtt{t} \mathbin{+\!\!+} \mathtt{x}_1$ мы дошли до базы индукции (терминальных состояний). Но в узле $\mathtt{g}(\mathtt{x}_1)$ шаг индукции делать рано: вызов на вершине стека другой.

Пример

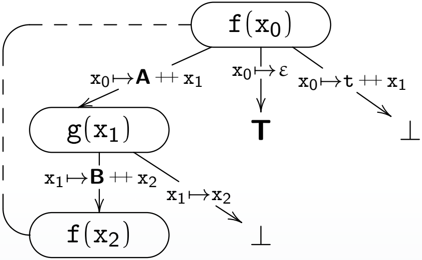
$$
\begin{array}{ll}
\mathtt{f}(\mathbf{A} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{g}(\mathtt{x}) & \mathtt{g}(\mathbf{B} \mathbin{+\!\!+} \mathtt{x}) = \mathtt{f}(\mathtt{x}) \\
\mathtt{f}(\varepsilon) = \mathbf{T} & \mathtt{g}(\_) = \bot \\
\mathtt{f}(\mathtt{t} \mathbin{+\!\!+} \mathtt{x}) = \bot &
\end{array}
$$



А вот теперь пришло время сделать переход от $\mathtt{f}(\mathtt{x}_2)$ к $\mathtt{f}(\mathtt{x}_0)$. Заметим, что состояние $\mathtt{f}(\mathtt{x}_0)$ формально «старше», т.к. получается из $\mathtt{f}(\mathtt{x}_2)$ за два шага вычислений.

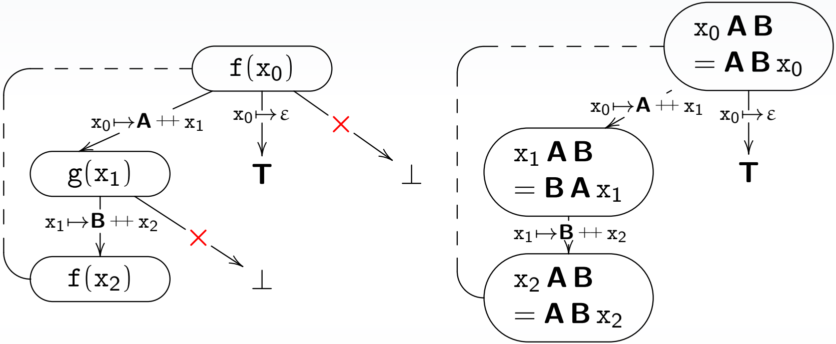
Пример

$$f(\mathbf{A} +\!\!+\, x) = g(x) \qquad g(\mathbf{B} +\!\!+\, x) = f(x)$$
$$f(\varepsilon) = \mathbf{T} \qquad g(\_) = \bot$$
$$f(t +\!\!+\, x) = \bot$$



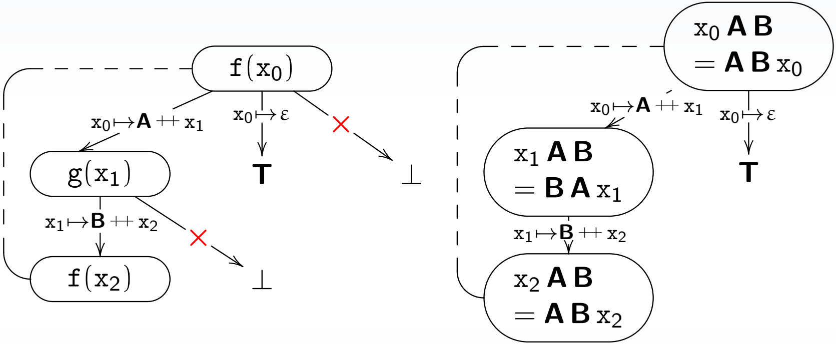
Развёртка путей вычисления замкнулась: полученный граф в точности описывает все такие пути (поскольку переход был сделан простой переименовкой x_2 в x_0).

(Ко)индукция повсюду



Справа — схема решения уравнения в словах. Если стереть метки узлов и не обращать внимание на пути вычислений, приводящие к исключениям, обе схемы идентичны.

(Ко)индукция повсюду



Справа — схема решения уравнения в словах. Можно сказать, что метод Матиясевича решения уравнений выполняет их символическую интерпретацию в DSL уравнений в словах.

Когда пора остановиться?

- Слишком рано — не удастся сделать индуктивный переход.

Пример: попытка возврата по переходу от $\mathtt{f}(\mathtt{x}_0)$ до $\mathtt{g}(\mathtt{x}_1)$.

- Слишком поздно — экспоненциальный взрыв путей вычисления.

Пример: добавочный переход от $\mathtt{f}(\mathtt{x}_2)$ к $\mathtt{g}(\mathtt{x}_3)$ по ветке $\mathtt{x}_2 \mapsto \mathbf{A} +\!\!+\, \mathtt{x}_3$ (с порождением ещё двух переходов).

- Главное — когда-нибудь остановиться вообще.

Когда пора остановиться?

- Слишком рано — не удастся сделать индуктивный переход.

Пример: попытка возврата по переходу от $\mathtt{f}(\mathtt{x}_0)$ до $\mathtt{g}(\mathtt{x}_1)$.

- Слишком поздно — экспоненциальный взрыв путей вычисления.

Пример: добавочный переход от $\mathtt{f}(\mathtt{x}_2)$ к $\mathtt{g}(\mathtt{x}_3)$ по ветке $\mathtt{x}_2 \mapsto \mathbf{A} +\!\!+\, \mathtt{x}_3$ (с порождением ещё двух переходов).

- Главное — когда-нибудь остановиться вообще.

Частичный порядок $\preceq$ есть Well-Quasi-Order (хороший предпорядок) на множестве $\mathcal{M}$, если для любой бесконечной последовательности $\{C_i\}$ элементов из $\mathcal{M}$ $\exists k_1, k_2 (k_1 < k_2 \;\&\; C_{k_1} \preceq C_{k_2})$.

Когда пора остановиться?

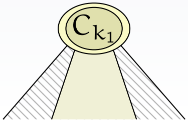
Пробуем сделать переход

$$C_1 \to C_2 \to \dots \to C_{k_1} \longrightarrow \dots \longrightarrow C_{k_2} \to \dots$$

Частичный порядок $\preceq$ есть Well-Quasi-Order (хороший предпорядок) на множестве $\mathcal{M}$, если для любой бесконечной последовательности $\{C_i\}$ элементов из $\mathcal{M}$ $\exists k_1, k_2 (k_1 < k_2 \,\&\, C_{k_1} \preceq C_{k_2})$.

- «Похожие» — не значит «одинаковые».
- WQO не обязан быть «QO» (может нарушаться транзитивность) и не обязан быть «W» на всём $\mathcal{M}$ (достаточно — на последовательностях элементов из $\mathcal{M}$, которые могут порождаться программой).

Индуктивный переход



Состояние C_{k_2} **вкладывается** в состояние C_{k_1} графа вычислений программы, если все пути вычислений, порождаемые C_{k_2}, могут порождаться также и C_{k_1}.

Если заштрихованная часть пуста, то вложение **точное**: то есть индуктивный переход порождает описание множества путей из C_{k_2} в терминах путей из C_{k_1} без потери информации. В частности, точными являются переходы, требующие только переименовки параметров.

Индуктивный переход

Состояние C_{k_2} **вкладывается** в состояние C_{k_1} графа вычислений программы, если все пути вычислений, порождаемые C_{k_2}, могут порождаться также и C_{k_1}.

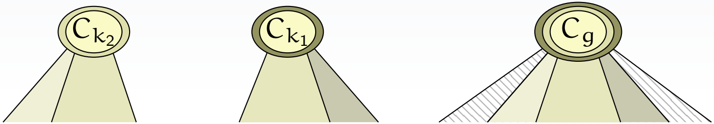
В общем случае нельзя ожидать не только точных индуктивных переходов, но и просто вложений.

При анализе путей из $\mathtt{f}(x_0, \varepsilon)$ при развёртке по правилам:

$$\mathtt{f}(\mathbf{A} +\!\!+\, x,\ y) = \mathtt{g}(x,\ y +\!\!+\, \mathbf{B}) \qquad \mathtt{f}(\varepsilon,\ y) = \mathtt{g}(\mathbf{A} +\!\!+\, y)$$

появится набор состояний вида $\mathtt{f}(x_i, \mathbf{B}^i)$. Каждое из этих состояний порождает дочернее состояние вида $\mathtt{g}(\mathbf{AB}^i)$, не являющееся дочерним ни для какого другого.

Индуктивный переход



Состояние C_g **является обобщением** состояний C_{k_1} и C_{k_2} графа вычислений программы, если все пути вычислений, порождаемые C_{k_1} и C_{k_2}, могут порождаться также и C_g.

Чем меньше заштрихованная часть, тем меньше потеря информации при обобщении. Точное обобщение (с нулевой такой частью) построить не удаётся почти ни в каком случае.

Вложение и обобщение состояний

На практике отношение вложения путей, порождаемых состояниями, не разрешимо (не существует алгоритма, который его определяет). Поэтому используются приближённые отношения.

Состояние C_{k_2} **вкладывается** в состояние C_{k_1} графа вычислений программы, если существует подстановка σ такая, что $C_{k_1}\sigma = C_{k_2}$.

Состояние C_g **является обобщением** состояний C_{k_1} и C_{k_2} графа вычислений программы, если существуют подстановки σ_1 и σ_2 такие, что $C_g\sigma_1 = C_{k_1}$, $C_g\sigma_2 = C_{k_2}$.

- Синтаксические
- Легко верифицируемые

Нётеровость

- Множество путей, порождаемых обобщённым состоянием, включает в себя множества путей обобщаемых состояний. Т.е. определяется возрастание $C_{k_i} \trianglelefteq C_g$ по вложению.
- Нет гарантии, что возрастающая последовательность $C_{g_1} \trianglelefteq C_{g_2} \trianglelefteq \dots \trianglelefteq C_{g_i} \trianglelefteq \dots$ когда-нибудь стабилизируется.

В алгебрах, содержащих ассоциативные операции, существуют бесконечные возрастающие последовательности, не обладающие нётеровостью, например:
$\mathtt{f}(x_0\,x_0),\ \mathtt{f}(x_1\,x_0\,x_0\,x_1),\ \dots,\ \mathtt{f}(x_{k+1}\,x_k\ \dots\ x_0\,x_0\ \dots\ x_k\,x_{k+1}),\ \dots$
Над данными в свободной алгебре (древесными термами) таких последовательностей не существует, если возрастание $T_i \trianglelefteq T_j$ определяется наличием подстановки σ такой, что $T_j\sigma = T_i$.

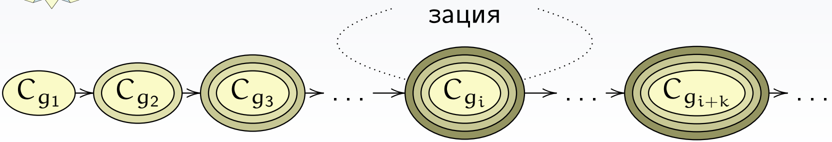
Нётеровость

- Нет гарантии, что возрастающая последовательность $C_{g_1} \trianglelefteq C_{g_2} \trianglelefteq \dots \trianglelefteq C_{g_i} \trianglelefteq \dots$ когда-нибудь стабилизируется.

Алгоритм обобщения корректен, если порождаемые им состояния являются нётеровыми относительно отношения вложения путей. Свойство нётеровости гарантирует, что вместо обобщения рано или поздно выполнится вложение.

Эквивалентно артиновости (фундированности, Well-Founded-Ordering) обратного отношения (C_{k_i} к C_g).

WQO + WFO



Алгоритм обобщения корректен, если порождаемые им состояния являются нётеровыми относительно отношения вложения путей. Свойство нётеровости гарантирует, что вместо обобщения рано или поздно выполнится вложение.

Эквивалентно артиновости (фундированности, Well-Founded-Ordering) обратного отношения (C_{k_i} к C_g).

Два кита индуктивного динамического анализа:

- WQO (шаг);
- WFO (индуктивный переход).