Математические модели последовательных вычислений

mk.cs.msu.ru → Лекционные курсы
→ Математические модели последовательных вычислений

Блок 12

Проблема эквивалентности программ
Схемы программ

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2022/2023, весенний семестр

Проблема эквивалентности программ

В широком смысле проблема эквивалентности программ формулируется так:

Для заданной произвольной пары программ выяснить, имеют ли эти программы одинаковое поведение

Чтобы задать конкретный (строго поставленный) вариант проблемы эквивалентности, требуется определить,

- какого вида программы рассматриваются,
- что именно считается их поведением и
- какие поведения считаются одинаковыми,

Проблема эквивалентности программ

Чтобы поставить проблему эквивалентности программ, достаточно определить только

- синтаксис рассматриваемых программ
  - (то, как они записываются)
- и их семантику
  - (то, какой смысл они имеют)

(Теоретическая) трудность или простота проблемы эквивалентности может служить индикатором трудности или простоты устройства программ в целом: сложность проблемы эквивалентности — это сложность определения смысла программы по её форме, и схожий уровень сложности можно ожидать для других задач анализа поведения программ

Например, проблема R-эквивалентности сетей Петри неразрешима, и многие другие проблемы анализа их поведения оказались если и разрешимы, то всё равно очень трудны

Проблема эквивалентности программ

При этом проблема эквивалентности лежит в основе многих прикладных задач программирования — например:

- Трансляция (компиляция)
- Оптимизация
- Рефакторинг (реорганизация)
- Распараллеливание
- Верификация
- Обфускация
- Обнаружение вредоносных программ

Проблема эквивалентности программ

Теорема Райса-Успенского (1953). В каждой «естественной» системе программирования любое нетривиальное семантическое свойство программ неразрешимо

«Естественная» = «как машины Тьюринга»: алгоритмически полная, эффективно интерпретируемая и допускающая трансляцию в неё любых эффективно интерпретируемых программ

Нетривиальное = хотя бы одна программа обладает этим свойством и хотя бы одна не обладает

Семантическое = зависящее только от семантики программы (вычисляемой ей функции преобразования входных данных в выходные), но не от её структуры

Функциональная эквивалентность программ означает равенство функций, вычисляемых этими программами

Следствие. В каждой «естественной» системе программирования функциональная эквивалентность программ неразрешма

Проблема эквивалентности программ

Два естественных способа «обойти» неразрешимость функциональной эквивалентности устроены так:

1. Решить проблему эквивалентности не для всех программ, а только для некоторого подкласса, устроенного согласно синтаксическим ограничениям, делающим рассматриваемый класс программ алгоритмически неполным
2. Исследовать другие виды эквивалентности, взаимосвязанные с функциональной, но при этом устроенные проще

Увы, способ 1 применим только к очень узким классам программ

Проблема эквивалентности программ

Например, если под программмой понимать машину Тьюринга, то вот такая очень маленькая машина очень трудна для анализа, так как *в некотором смысле* способна моделировать поведение произвольной машины Тьюринга на произвольном слове (Wolfram, Smithm, 2007 и 2020):

	a	**b**
q_0	q_1, **b**, L	q_2, **a**, R
q_1	q_2, **a**, R	q_2, **b**, L
q_2	q_1, **a**, R	q_0, **a**, L

То есть проверка эквивалентности даже маленьких машин Тьюринга, не содержащих ничего «сверхъестественного», очень трудна

Проблема эквивалентности программ

Другой пример

```
integer f(positive_integer x) {
  A[0] = 0; A[1] = 0; A[2] = x; i = 2;
  do
    if (A[i] % 2 == 0) A[i + 1] = A[i] / 2;
    else A[i + 1] = 3 * A[i] + 1;
    i++;   while (A[i] != A[i-3]);
  return A[i];
}
```

```
integer g(positive_integer x) {
  if (x == 1 || x == 2) return x;
  else return 4;
}
```

Проверка эквивалентности этих двух программ — это проверка справедливости гипотезы Коллатца, которую математики не могут ни доказать, ни опровергнуть уже 80 лет

Проблема эквивалентности программ

Второй подход: не ограничивать класс программ, но упростить понятие эквивалентности, — можно расценивать как нахождение *приближённого* решения задачи

Этот подход широко применяется при решении вычислительных задач: если невозможно найти точное решение задачи, то можно, внеся достаточно небольшую погрешность, достаточно точно оценить это решение

На таком способе проверки эквивалентности — внесении *погрешности* в семантику программ и проверки получившейся *приближённой* эквивалентности, основан раздел математики, изначально посвящённый решению проблемы эквивалентности программ:

теория схем программ

Схемы программ

Исследование проблемы эквивалентности в теории схем программ обычно следует такой схеме

Этап 1

Формируется семейство $\mathcal{M}(\sigma)$ моделей программ, параметризованное семантикой σ базовых (примитивных) компонентов программ

Объекты этого семейства называются схемами программ

На основе семантики σ вводится понятие вычисления схемы и отношение эквивалентности $\sim_\sigma$

Схемы программ

Этап 2

Вводится отношение $\sqsubseteq$ аппроксимации на семействах моделей программ, такое что

$$\mathcal{M}(\sigma_1) \sqsubseteq \mathcal{M}(\sigma_2)$$
$$\Leftrightarrow$$
$$\text{для любых схем программ } \pi_1, \pi_2 \text{ верно } (\pi_1 \sim_{\sigma_1} \pi_2 \Leftarrow \pi_1 \sim_{\sigma_2} \pi_2)$$

Этап 3

Выделяется класс семантик Σ, такой что проверка эквивалентности $\sim_\sigma$ в модели $\mathcal{M}(\sigma)$ для любой семантики $\sigma \in \Sigma$ имеет достаточно эффективное решение

Для моделей рассматриваемых классов разрабатываются эффективные алгоритмы проверки эквивалентности схем программ

Схемы программ

Тогда для разработки *приближённого* решения проблемы эквивалентности в программной системе S достаточно сделать так:

- Описать модель $\mathcal{M}(\sigma_0)$, в точности соответствующую системе S: $\mathcal{M}(\sigma_0) = S$
- Выбрать модель $\mathcal{M}(\sigma_1)$, такую что $\mathcal{M}(\sigma_0) \sqsubseteq \mathcal{M}(\sigma_1)$
- Использовать алгоритм проверки эквивалентности схем программ в $\mathcal{M}(\sigma_1)$
- Если схемы эквивалентны в $\mathcal{M}(\sigma_1)$, то они обязательно эквивалентны и в S
  - Но обратное неверно, и *погрешность* состоит именно в этом

Для этого следует научиться

- строить семейства моделей $\mathcal{M}(\sigma)$
- проверять соотношение $\mathcal{M}(\sigma_0) \sqsubseteq \mathcal{M}(\sigma_1)$
- выделять классы моделей с разрешимой проблемой эквивалентности
- решать проблему эквивалентности для моделей этих классов

Схемы программ

Впервые этот подход был предложили и применили Алексей Андреевич Ляпунов и Юрий Иванович Янов в 1956–58 гг.

При применении этого подхода была придумана первая математическая модель программ (известная сейчас под названием «схемы Ляпунова-Янова») и показано, как можно применить методы алгебры и математической логики для анализа поведения программ

Затем в 1968 г. Майк Патерсон предложил более общую модель императивных программ, совместив схемы Ляпунова-Янова с понятиями логики предикатов первого порядка

Параллельно с этим Андрей Петрович Ершов привносит в схемы программ графовую нотацию (1968), давшую начало привычному сейчас графовому изображению программ (блок-схемы, граф потока управления, ...), успешно применяет схемы Ляпунова-Янова и Патерсона для оптимизирующей трансляции (компиляции) и предлагает для модели Патерсона название «стандартные схемы программ»