Оптимизация процесса непрерывного юнит-тестирования на основе статического и динамического анализа исходного кода

Александр Михайлов

Руководитель: Анатолий Никитин

СПбАУ, 2012

Непрерывное тестирование

- Чем позже обнаружен дефект – тем выше стоимость его исправления
- Большую часть времени компьютер простаивает
- НТ, используя простаивающие мощности пытается уменьшить время обнаружения ошибок в исходном коде, посредством фонового запуска юнит-тестов

Постановка задачи

- Создание системы обнаружения тестов, которые необходимо перезапустить. В рамках проекта dotCover
  - Поиск измененного кода
  - Определение тестов, покрывающих измененный код
- Платформа - .NET
- Анализируемый язык - CIL

Аналоги

nCrunch
Перезапуск всех тестов.
Приоритезация по данным динамического анализа

Continuous Tests
Используя комбинацию статического и динамического анализа определять минимальный набор тестов для перезапуска

Поиск измененного кода

- Минимальная единица изменения – метод
- Для каждого метода хранится:
  - сигнатура
  - число инструкций
  - хеш тела

Определение тестов, покрывающих измененный код

- Необходимо знать из какого теста какой метод достижим
  - Динамический анализ (профилирование)
  - Статический анализ

Динамический анализ

- Уже реализован в dotCover
- Для каждого метода хранится список тестов из которых эти методы посещались

- Высокая точность
- Профилирование существенно замедляет работу тестов

Статический анализ

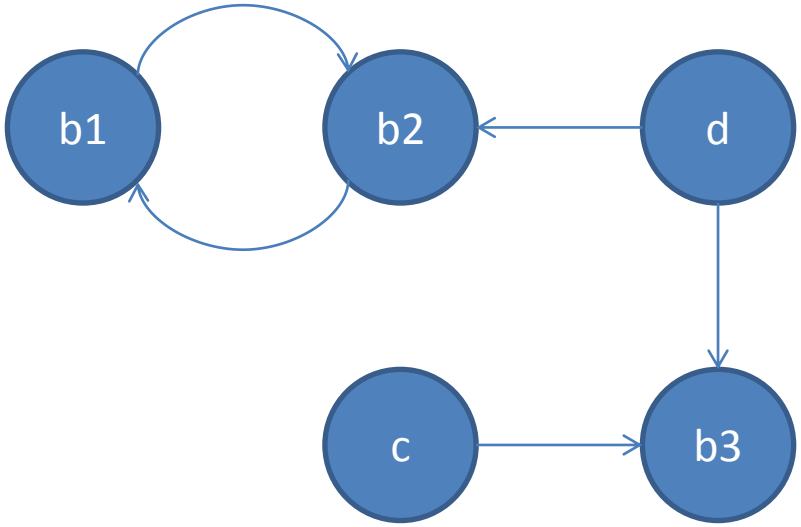
- Применяется до первого запуска динамического анализа
- Динамический анализ можно запустить не для всех тестов. Тогда для оставшихся будет применяться статический анализ
- Статический анализ не обнаруживает вызовы через reflection. Тесты, использующие reflection пропускаются

Rapid Type Analysis (RTA)

- Метод не может быть вызван, если не инстанцирован тип, его содержащий
- Алгоритм:
  - Обходим граф вызовов в глубину
  - Отслеживаем инстанцирование типов
  - Анализируются только те виртуальные вызовы, инстанцирование типов которых было обнаружено

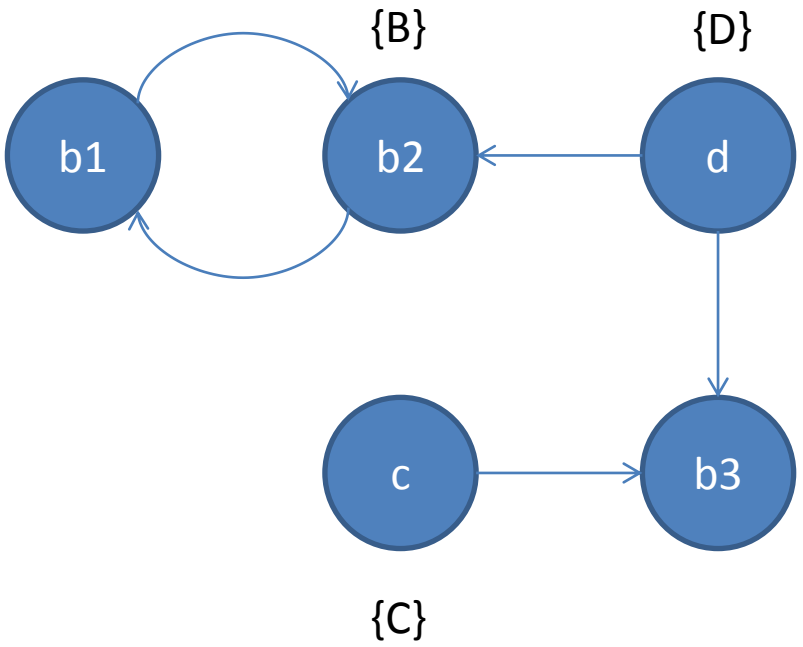
Variable Type Analysis (VTA)

```
B b1, b2, b3;
D d = new D();
C c = new C();
b2 = new B();

b1 = d;
b3 = c;
b2 = b1;
b3 = d;
b1 = b2;
```



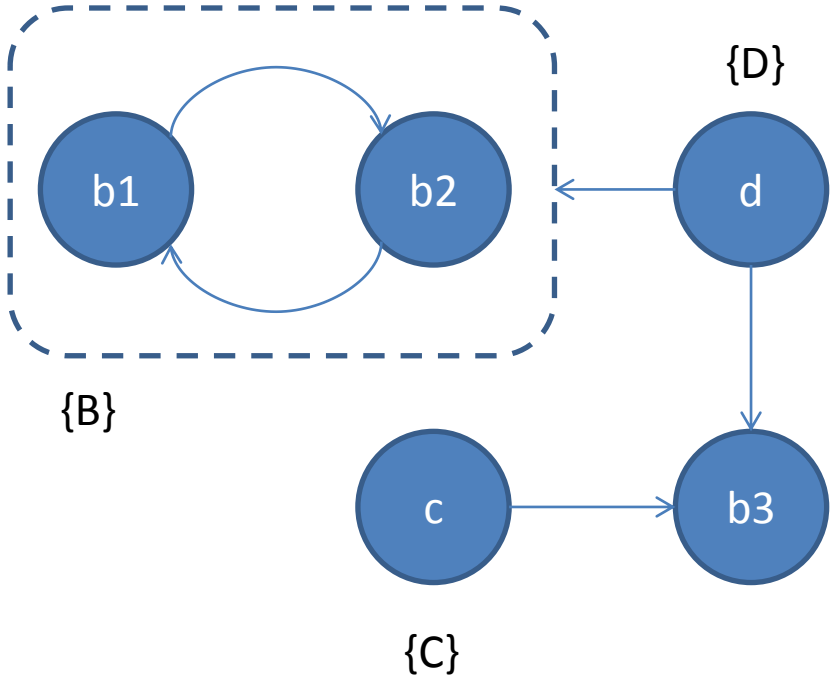
Variable Type Analysis (VTA)

```
B b1, b2, b3;
D d = new D();
C c = new C();
b2 = new B();

b1 = d;
b3 = c;
b2 = b1;
b3 = d;
b1 = b2;
```



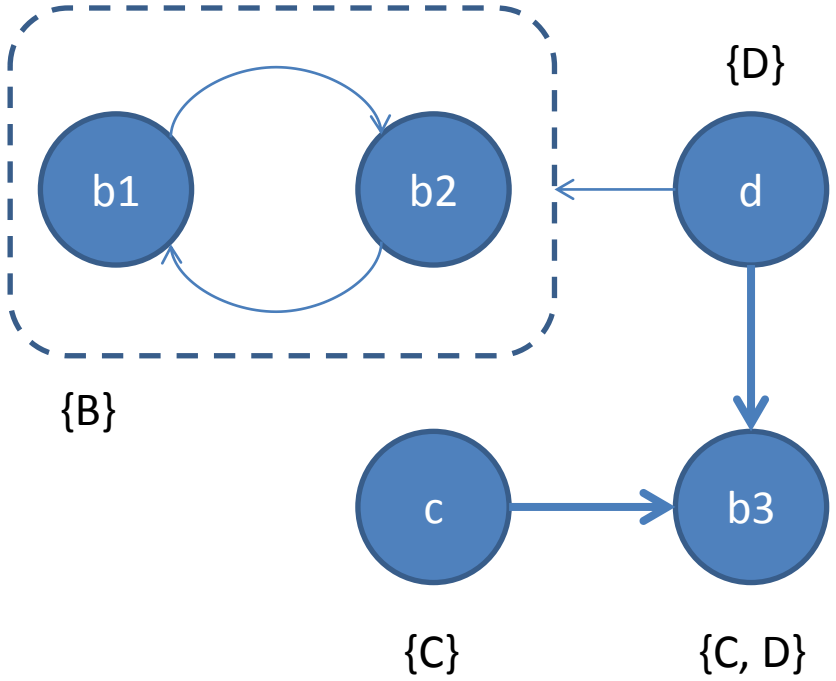
Variable Type Analysis (VTA)

```
B b1, b2, b3;
D d = new D();
C c = new C();
b2 = new B();

b1 = d;
b3 = c;
b2 = b1;
b3 = d;
b1 = b2;
```



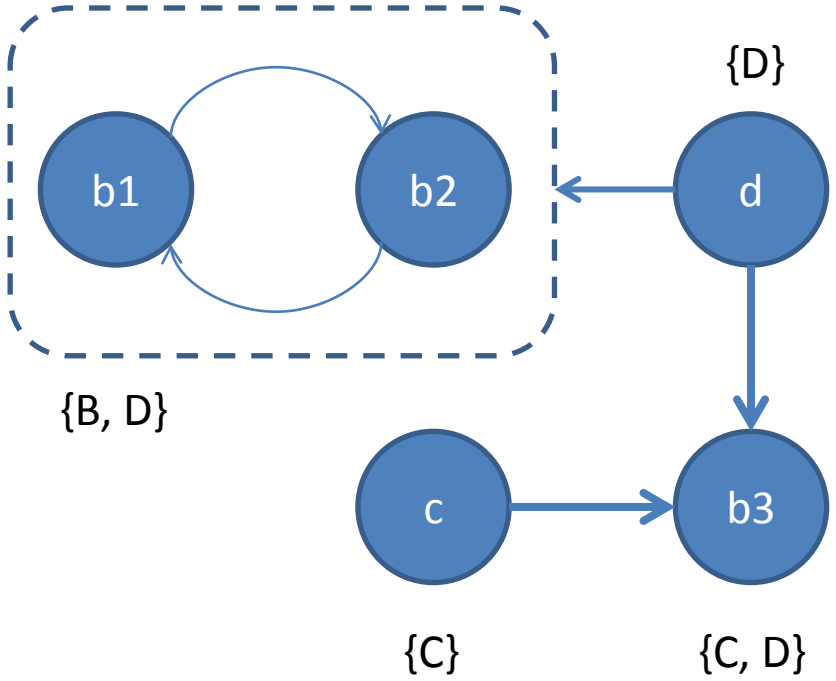
Variable Type Analysis (VTA)

```
B b1, b2, b3;
D d = new D();
C c = new C();
b2 = new B();

b1 = d;
b3 = c;
b2 = b1;
b3 = d;
b1 = b2;
```



Variable Type Analysis (VTA)

```
B b1, b2, b3;
D d = new D();
C c = new C();
b2 = new B();

b1 = d;
b3 = c;
b2 = b1;
b3 = d;
b1 = b2;
```



Тестовые данные

- Тестирование проводилось на исходном коде библиотеки nUnit (+System).
  - 575 тестов
  - 150 546 методов
  - 16 646 классов

Результаты

Алгоритм	Количество узлов	Количество ребер	Время работы, с
Динамический анализ	7 628	48 417	33
CHA	92 125	1 302 578	26,96
RTA	44 668	527 654	168,5
VTA	15 239	129 597	36,03

Резюме

- Реализован алгоритм поиска изменений исходного кода
- Реализовано несколько алгоритмов статического разрешения виртуальных вызовов:
  - CHA
  - RTA
  - **VTA**
- Проводятся работы по улучшению показателей VTA

Дальнейшее развитие

- Приоритезация тестов
- Компактизация Type Propagation и Call-графов
- Предварительная обработка системных библиотек
- Частичное обновление графов при перекомпиляции

Спасибо за внимание