Proporcja równoważnych mutantów

Klucz do analizy równoważności mutacji [1]

Tomasz Homoncik

4 listopada 2021

Równoważność mutantów

Motywacja i wcześniejsze podejścia.

Redundancja

-Eliksir nieśmiertelności mutantów

- Zakładamy ustaloną strategię generowania mutantów (określony zbiór operatorów)
- Jaka własność programu czyni go podatnym na generowanie równoważnych mutantów?
- Zwracanie tego samego wyniku mimo obecności usterek - wcześniejsze badania wskazują na redundancję
- Definicja *REM* - Ratio of Equivalent Mutants jako proporcja oczekiwanej wartości równoważnych mutantów wygenerowanych na podstawie programu bazowego do wszystkich mwygenerowanych mutantów

Redundancja

-State redundancy

- Różnica między deklarowanym a używanym stanem (przykładowo użycie 32 bitów do reprezentacji miesiąca daje nadmiar 32 - 5 bitów)
- S zmienna losowa reprezentująca zadeklarowany stan, σ używany stan
- σ jest funkcją S
- $SR_I = \frac{H(S|\sigma_I)}{H(S)} = \frac{H(S)-H(\sigma_I}{H(S)}$
- $SR_F = \frac{H(S|\sigma_F)}{H(S)} = \frac{H(S)-H(\sigma_F}{H(S)}$
- Im więcej przetwarzanej informacji, która nie wpłynie na wynik, tym większa szansa, że mutant nie zmieni zachowania programu

Redundancja

-State redundancy

```
int gcd (int a, b) {
    assert(a>0 && b>0);
    while (a!=b) {
        if (a>b) a=a-b;
        else b=b-a;
    }
    return a;
}
```

- $SR_I = \frac{H(S)-H(\sigma_I}{H(S)} = \frac{64-62}{64} = 0.03125$
- Na koniec $a = b \Rightarrow SR_F = \frac{H(S)-H(\sigma_F}{H(S)} = \frac{64-31}{64} = 0.51562$

Redundancja

-Functional redundancy

- Program jako funkcja z X w Y
- $FR = \frac{H(X|Y}{H(X)} = \frac{H(X)-H(Y)}{H(X)}$, bo Y jest funkcją X
- Im mniej opcji na wynik, tym trudniej będzie mutantowi zmienić zachowanie programu

Redundancja

-Functional redundancy

```
int gcd (int a, b) {
    assert(a>0 && b>0);
    while (a!=b) {
        if (a>b) a=a-b;
        else b=b-a;
    }
    return a;
}
```

- $FR = \frac{62-31}{62} = 0.5$

Redundancja

-Non injectivity

- Jak daleko funkcja jest on iniektywności
- Entoripia warunkowa początkowego (aktualnego/używanego?) stanu, znając końcowy (aktualny/używany?) stan
- $NI = \frac{H(\sigma_I|\sigma_F}{H(\sigma_I} = \frac{H(\sigma_I)-H(\sigma_F)}{H(\sigma_I)}$, bo σ_F jest funkcją σ_I
- Zwiększona szansa na zamaskowanie defektu przy nieiniektywnej funkcji

Redundancja

-Non-determinacy

- Sposób na mierzenie niedeterministyczności wyroczni używanej do testowania równoważności
- Autorzy wspominają, że ich model może być używany bez tej metryki
- $ND = \frac{H(\sigma_F^P|\sigma_F^M)}{H(\sigma_F^P)}$
- Używana wyrocznia wpływa na określenie równoważności

Redundancja

-Co sprzyja tworzeniu równoważnych mutantów?

- Mutacja nie zmienia stanu programu
- Mutacja zmienia stan, ale nie powoduje defektu
- Mutacja powoduje defekt, ale jest on maskowany
- Mutacja powoduje defekt, który nie jest maskowany, ale nie powoduje to błędu
- Mutacja powoduje defekt, który nie jest maskowany, powoduje to błąd, ale to ta sama klasa co prawidłowy wynik

Redundancja

-Co sprzyja tworzeniu równoważnych mutantów?

Table 1
Redundancy metrics as drivers of mutant equivalence.

Circumstance of mutant equivalence	Redundancy attribute	Metric
The Mutation does not alter the program's state.	Initial state redundancy	SR_I
The Mutation does alter the program's state, but it is not a fault (the altered state is correct).	Final state redundancy	SR_F
The mutation is a fault, but the errors it causes are masked.	Non Injectivity	*NI*
The mutation is a fault, the errors it causes are propagated, but they cause no failure.	Functional Redundancy	*FR*
The mutation is a fault, the errors it causes are propagated they do cause failures, but the failures fall within the tolerance of the oracle of equivalence.	Non Determinacy	*ND*

Szacowanie metryk redundancji

- Jak automatyzować liczenie wymienionych metryk
- (SR_I, SR_F, FR, NI) odnoszą się do bazowego programu, (ND) do wyroczni używanej do stwierdzenia równoważności
- Model regresyjny $\rho(SR_I, SR_F, FR, NI)$ powstaje przy używaniu SR_I, SR_F, FR, NI jako niezależne zmienne i REM jako zależna (ND jako 0)
- $REM = \rho(SR_I, SR_F, FR, NI) + ND \times (1 - \rho(SR_I, SR_F, FR, NI))$
- Można tez używać tylko metryk zależnych od programu, wtedy $REM = \rho(SR_I, SR_F, FR, NI)$

Szacowanie metryk redundancji

-Metryki zależne od programu

Aby wyznaczyć metryki dla danej metody, potrzebne są entropie:

- $H(S)$ - przestrzeni stanowej
- $H(\sigma_I)$ - używanego początkowego stanu
- $H(\sigma_F)$ - używanego końcowego stanu
- $H(X)$ - wejściowej przestrzeni
- $H(Y)$ - wyjściowej przestrzeni

Szacowanie metryk redundancji

-Entropia zadeklarowanych przestrzeni

$H(S), H(X), H(Y)$ są powiązane z typem. Zakładając równe prawdopodobieństwo każdej z wartości jest będą to ilości bitów.

Data type	Entropy	Data type	Entropy
bool	1 bit	int	32 bits
byte	8 bits	float	32 bits
char	16 bits	long	64 bits
short	16 bits	double	64 bits

Dla Stringa i tablic używane domyślne rozmiary.

Szacowanie metryk redundancji

-Entropia właściwych przestrzeni stanowych

W przypadku, gdy nie wiemy nic o właściwym początkowym stanie metody, możemy założyć, że może przyjąć dowolną wartość spośród zadeklarowanej przestrzeni. $H(\sigma_I) = H(S)$ W przeciwnym wypadku $H(\sigma_I) = H(S) - \delta H(A)$, gdzie A jest asercją na S, definiującą warunki wstępne.

Szacowanie metryk redundancji

- $\delta H(true) = 0$
- $\delta H(false) = H(S)$
- $\delta H(E1 == E2) =$ entropia z typu danych $E1$ i $E2$
- $\delta H(E1 > E2) = H(E1 < E2) = H(E1 \leqslant E2) = H(E1 \geqslant E2)$ $= 1$ bit
- $\delta H(E1 \neq E2) = 0$, w przybliżeniu nic nie zmienia
- $\delta H(A1 \&\& A2) = \delta H(A1) + \delta H(A2)$
- $\delta H(A1 || A2) = min(\delta H(A1), \delta H(A2))$

Szacowanie metryk redundancji

-Entropia właściwych przestrzeni stanowych

Śledzenie zależności zmiennych od siebie, wprowadzona dependency matrix. Przejście przez przypadki

- Deklaracja - macierz z true na przekątnej, false wpp
- Inicializacja - wiersz odpowiadający zmiennej ustawiany na same wartości false
  - do stałej
- Przypisanie - OR ze wszystkich wierszy zmiennych użytych w wyrażeniu
- If-Then-Else - wybieramy macierz z większą entropią
- While - macierz z ciała pętli

Szacowanie proporcji równoważnych mutantów

- Funkcje z *Apache Commons Mathematics Library* i *Apache Commons Lang3 Library*
- Każda funkcja ma przypisane dane testowe
- PiTest wykorzystany do generowania mutantów
- Maven do zarządzania wykonaniami, testami i porównaniami
- Dla każdej metody użyte kalklulatory metryk redundancji, kótre szacują metryki zależne od programu
- Na koniec szacowanie *REM* dla każdej funkcji *P*

Model regresji

Określenie ilościowe redundacji mutacji

-Jak określić ile klas równoważności jest w zbiorze N mutantów P?

- Mając zbiór N ellementów, gdzie każda para jest równoważna z prawdopodobieństwem REM, jaka jest oczekiwana liczba klas równoważności?
- $NEC(N, REM) = \Sigma_{k=1}^{N} k \times p(N, REM, k)$
- $p(N, REM, k)$ to prawdopodobieństwo, że liczba klas równoważności jest równa k

Określenie ilościowe redundacji mutacji

-Jak określić ile klas równoważności jest w zbiorze N mutantów P?

- $p(N, REM, 1) = REM^{N-1}$ - wszystkie elementy są równoważne
- $p(N, REM, N) = (1 - REM)^{\frac{N \times (N-1)}{2}}$ - żadne elementy nie są równoważne
- $p(N, REM, k) = (1 - (1 - REM)^k) \times p(N - 1, REM, k) + (1 - REM)^{k-1} \times p(N - 1, REM, k - 1)$ - dwa przypadki dodając element do zbioru rozmiaru $N - 1$
  - Nowy element wpada do jednej z istniejących klas równoważności
  - Nowy element definiuje nową klasę

Mutation score

- $MS(M, X) = \frac{X}{M}$ - klasyczna miara efektywności zbioru testów (M - liczba mutantów, X - liczba zabitych)
- *essential mutation score* $EMS(N, X)$ - stosunek różnych mutantów w X do różnych mutantów w N (N to mutanty różne od bazowego programu)

$$EMS = \frac{NEC(X,REM)}{NEC(N,REM)}$$

Minimalny zbiór mutantów

Jak możemy wyznaczyć zbiór różnych mutantów, który zawiera przedstawiciela każdej klasy równoważności i zawiera nie więcej niż jednego przestawiciela z każdej klasy?

Minimalny zbiór mutantów

-Podejście pierwsze

```
void min1(N) {
    minset = emptyset;
    forall (i in N) {
        bool equiv=false;
        forall (j in minset) {
            equiv = equiv || equivalent(i,)
        }
        if (!equiv) minset = minset+{i};
    }
}
```

Minimalny zbiór mutantów

-Podejście po poprawkach, $K = NEC(N, REM)$

```
void min2(N, K) {
    minset = emptyset;
    while ( |minset| < K) {
        bool equiv=false;
        i=nextelement(N);
        forall (j in minset) {
            equiv = equiv || equivalent(i,j)
        }
        if (!equiv) minset=minset+{i};
    }
}
```

Minimalny zbiór mutantów

-Number Of Inspections

- $NOI(N, K)$ - szacowana liczba elementów zbioru rozmiaru N podzielonego na K klas równoważności, który musimy zbadać, aby uzyskać co najmniej jednego przestawiciela każdej klasy.
- d_i, dla $1 \leqslant i \leqslant K$ - liczba dodatkowych „remisów” dla i-tej klasy
  - $d_1 = 1$
  - $D_K = \Sigma_{i=1}^{K} d_i$
- Przy dużej wartości N względem K prawdopodobieństwo pokrycia nowej klasy nie zmienia się po każdym remisie. Każde d_i jest geometryczną zmienną losową z parametrem $p_i = \frac{K-i+1}{K}$ i wartością oczekiwaną $\frac{1}{p_i}$
- $E(D_K) = 1 + \frac{K}{K-1} + \frac{K}{K-2} + \frac{K}{K-3} + \cdots + K$

Minimalny zbiór mutantów

Jeśli pozwalamy na zmianę prawdopodobieństw po każdym remisie, to d_i zależy od wyniku $d_1, \ldots, d_{i-1}$

- $P(d_2 = 1) = \frac{N-N/K}{N-1} \times \frac{N-N/K}{N-2} = \frac{N-N/K}{N-1} \times \frac{N/K-1}{N-2}$
- $\ldots$
- $P(d_2 = N/K) = \frac{N-N/K}{N-1} \times \frac{N/K-1}{N-2} \times \ldots \frac{N/K-(N/K-1)}{N-N/K}$

Minimalny zbiór mutantów

-Szacowanie zysku z modyfikacji algorytmu

$H = NOI(N, K)$

- Liczba odwołań do $equivalent(i, j)$ to $1 + \cdots + 1 + 2 + \cdots + 2 + \cdots + (K-1) + \cdots + (K-1) + K$ i składa się z H wyrażeń, gdzie liczba jedynek oznacza liczbę iteracji gdzie minimalny zbiór miał 1 element itd.
- Można to przybliżyć przez $\frac{H \times K}{2}$
- Drugi algorytm kończy się, gdy minimalny zbiór osiąga rozmiar K, ilość odwołań można przybliżyć przez $\frac{H \times K}{2} + (N - H) \times K$
- Udało się poprawić algorytm $\frac{2 \times N - H}{H}$ razy
  - Dla $N = 3000, H = 253$ daje to 22.72 2272% poprawa)

Minimalny zbiór mutantów

Można dobrać $H' = n * H$ przy n przykładowo $1.2, 1.5$ czy 2.0, aby mieć pewność, że nie zgubimy jakiejś klasy, i nie przechodziić przez całość jak przy pierwszym algorytmie.

Wpływ sposobu generacji mutantów

-Na podstawie podstawowych operacji możemy szacować wpływ całej strategii

Table 6
Mutation operators.

Op1	Increments_mutator;
Op2	Math_mutator;
Op3	Negate_conditionals_mutator;
Op4	Conditionals_ boundary_mutator;
Op5	Void_method_call_mutator;
Op6	Return_vals_mutator;
Op7	Invert_negs_mutator
Op8	Constructor_Calls_Mutator

- $REM = \frac{REM_1 \times M_1 + REM_2 \times M_2 + \cdots + REM_k \times M_k}{M_1 + M_2 + \cdots + M_k}$
- Przy braku informacji o rozmiarach M_i, zakładamy, że są równe i wtedy $REM = \frac{1}{k}\Sigma_{i=1}^{k} REM_i$

Źródła I

Imen Marsit, Amani Ayad, David Kim, Monsour Latif, JiMeng Loh, Mohamed Nazih Omri, and Ali Mili.
The ratio of equivalent mutants: A key to analyzing mutation equivalence.
Journal of Systems and Software, 181:111039, 2021.