Code slicing

Bartłomiej Wołowiec

16 lutego 2011

Code / Program slicing

- Zaproponowany przez Mark Weiser w 1981,
- Ułatwia analizę dużych programów,
- Czytelniejsze od grafów zależności,
- Zmniejsza ilość kodu podczas weryfikacji,

- Początkowa metoda zaproponowana przez Weiser-a,
- Usuwanie całych wyrażeń, jeżeli nie mają one wpływu na wybrane zmienne
- Wybieramy zmienne użyte w wyrażeniu oraz jako lokalizacje
- Najprostsza odmiana slicingu
- Mało efektywna dla większości kawałków kodu

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int z = 0;
5          if (x <= 1) {
6          } else {
7              z = Integer.parseInt(args[2]);
8          }
9      }
10 }
```

Rysunek 2. Wycinek utworzony zgodnie z kryterium <15, z>.

- Dodajemy konkretne wartości niektórych zmiennych,
- Często duża redukcja kodu,
- Za duże ograniczenia na zmienne,

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int z = 0;
4      }
5  }
```

Rysunek 3. Wycinek utworzony zgodnie z kryterium
<15, z, <args[0]="1", args[1]="2", args[2]="3">>

- Dodajemy zależności między zmiennymi (np $x < y$)
- Przydatny przy dowodzeniu własności
- Trudny w realizacji

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```java
public class Example1 {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        int z = 0;
        int total = 0;
        int sum = 0;
        if (x <= 1) {
            sum = y;
        } else {
            z = Integer.parseInt(args[2]);
            total = x * y;
        }
        System.out.println("Result is: " + total + sum);
    }
}
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```java
public class Example1 {
    public static void main(String[] args) {
        int z = 0;
    }
}
```

- Dopuszczamy zmianę składni (nie tylko usuwanie)
- Zwiększona czytelność powstałego kodu (lub zmniejszona...)

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int y = Integer.parseInt(args[1]);
5          int z = 0;
6          int total = 0;
7          int sum = 0;
8          if (x <= 1) {
9              sum = y;
10         } else {
11             z = Integer.parseInt(args[2]);
12             total = x * y;
13         }
14         System.out.println("Result is: " + total + sum);
15     }
16 }
```

Rysunek 1. Przykładowy kod źródłowy w języku Java.

```
1  public class Example1 {
2      public static void main(String[] args) {
3          int x = Integer.parseInt(args[0]);
4          int z = 0;
5          if (x > 1) {
6              z = Integer.parseInt(args[2]);
7          }
8      }
9  }
```

Rysunek 5. Wycinek utworzony zgodnie z kryterium <15, z>

Backward slice

- Wszystkie ścieżki od początku do danej lokalizacji,
- Większość sprawdzeń poprawności programu w danym punkcie,

Forward slice

- Wszystkie ścieżki od danej lokalizacji do końca programu,
- Można badać wpływ zmiany w kodzie na dalszą część programu

Połączenie obu daje complete slice.

- Większość narzędzi sprawdza model, nie program,
- Duże znaczenie ma rozmiar,
- Większość programów jest duża i bardzo skomplikowana...
- ...ale często sprawdzamy proste własności dotyczące małego kawałka,
- Staramy się zautomatyzować jak największą część weryfikacji oprogramowania,

- Automatyczny program jest bezpieczniejszy od człowieka,
- Może działać na dużych programach,
- Pomaga znaleźć komponenty wymagające ręcznego napisania modelu,
- Redukuje rozmiar programu bez zmniejszania czytelności,

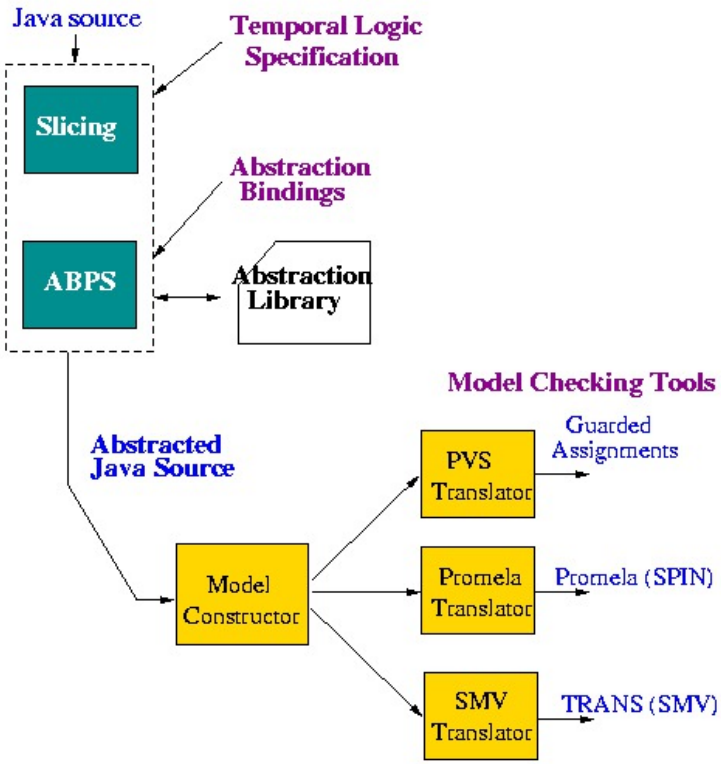
- Statyczny slicing z warunkiem zawierającym lokalizacje i zmienne z formuły LTL,
- Warunkowy slicing na podstawie reguły LTL,

Frama-C,

- Framework do weryfikacji programów w C,
- Początkowy etap analizy to slicing,
- Wspiera slicing statyczny i dynamiczny,

Bogor / Bandera (Indus),

- Framework
- Wsparcie języków OO z dynamicznymi obiektami, wątkami, itd,
- Integracja z Eclipse,



Indus - część projektu Bandera

- Wspiera statyczną analizę i przekształcanie kodu w języku Java,
- Backward, Forward slice,
- Wygenerowane wycinki można skompilować,
- Integracja z Eclipse,

CodeSurfer

- Komercyjny,
- Wspiera C / C++,
- Wspiera zmienną ilość argumentów funkcji, struktury, libc (sygnały, abort(), exec()),
- Umożliwia robienie wycinków z programów do 100K LOC,

- Bartosz Bogacki, Code slicing: Droga do lepszego rozumienia kodu źródłowego
- Mark Weiser. "Program slicing". Proceedings of the 5th International Conference on Software Engineering, pages 439–449, IEEE Computer Society Press, March 1981.
- John Hatcliff, Matthew B. Dwyer and Hongjun Zheng, Slicing Software for Model Construction