

Code slicing: Droga do lepszego rozumienia kodu źródłowego (część pierwsza)

Bartosz Bogacki

Poznan University of Technology, Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Bartosz.Bogacki@cs.put.poznan.pl

Streszczenie. Kod źródłowy skomplikowanych programów komputerowych może być lepiej zrozumiany przez programistów gdy zostanie podzielony na mniejsze fragmenty. Slicing kodu to metoda służąca do wykonania podziału kodu z zachowaniem jego poprawności w ramach rozważanej funkcjonalności. Poniższy tekst przedstawia podstawowe pojęcia z zakresu statycznej analizy kodu koncentrując się na slicingu kodu. Przedstawiony jest zarys historyczny, znane metody, narzędzia oraz praktyczne zastosowania slicingu kodu. W części pierwszej artykułu przedstawiono podstawy oraz charakterystykę tej popularnej techniki, podczas gdy w części drugiej zaprezentowane będą praktyczne aspekty, narzędzia oraz przykładowe case-study z wykorzystaniem narzędzi Indus/Kaveri.

1 Wprowadzenie

Slicing kodu (ang. *code slicing*) jest techniką ułatwiającą rozumienie kodu źródłowego programu komputerowego poprzez uproszczenie jego struktury tak, aby programista mógł skoncentrować się na wyrażeniach istotnych z punktu widzenia analizowanych obliczeń. Technika ta pozwala na dekompozycję kodu źródłowego ułatwiającą zrozumienie wzajemnego wpływu poszczególnych wyrażeń (ang. *statements*) oraz zmiennych (ang. *variables*). Nieformalnie wycinek kodu (ang. *slice*) dostarcza odpowiedzi na pytanie: „Które z wyrażeń programu mogą mieć wpływ na obliczenie wartości zmiennej V w wyrażeniu S ”. Jeśli zachowanie takie można sformułować jako wartości zmiennych z podzbioru pewnego zbioru zmiennych w obszarze występowania wyrażeń z podzbioru pewnego zbioru wyrażeń, to takie sformułowanie nosi nazwę kryterium slicingu (ang. *slicing criterion*). Oznacza to, że kryterium slicingu definiowane jako $\langle S, V \rangle$ specyfikuje wpływ zmiennych ze zbioru V na wyrażenia w obszarze S .

Slicing kodu po raz pierwszy został przedstawiony przez Marka Weisera w 1979 roku w pracy [7]. Prawdziwą popularność metoda zdobyła jednak 2 lata później, gdy została opublikowana przez IEEE w pracy [8]. Od tej pory na świecie prowadzone są liczne badania mające na celu rozwój metody, czego dowodem jest ogromna liczba

prac publikowanych w najważniejszych czasopismach branżowych. W pracy [8] Weiser przedstawił metodę obecnie noszącą nazwę wykonywalnego slicing statycznego wstecz (ang. *executable backward static slicing*). Nazwa pochodzi od założeń, które autor przyjął definiując wycinki kodu oraz sposobu tworzenia wycinka. Weiser przyjął, że każdy wycinek musi pozostać wykonywalnym programem a jego tworzenie polega na usuwaniu niepotrzebnych wyrażeń z kodu od rozważanego wyrażenia wstecz. Wycinek utworzony tą metodą zawiera więc informację na temat wpływu wyrażeń poprzedzających wyrażenie zadane w kryterium slicing na jego osiągalność oraz na wartości zmiennych przetwarzanych przez to wyrażenie. Przykładem obrazującym tę metodę są dwa poniższe rysunki. Rysunek 1 przedstawia przykładowy kod 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.

Rysunek 2 przedstawia wycinek kodu utworzony zgodnie z kryterium slicing <15, z>, gdzie „15” jest linią kodu dla której konstruowany będzie wycinek a „z” jest rozważaną zmienną.

```

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>.

Łatwo dostrzec, że utworzony wycinek kodu zgodnie z definicją zawiera jedynie wyrażenia, które mogą mieć wpływ na wartość zmiennej „z” w linii 15 kodu.

Ponieważ wyrażenia znajdujące się w liniach 4, 6, 7, 9, 12 oraz 14 z pewnością nie mają wpływu na tę wartość, więc wycinek ich nie zawiera. Wyrażenia z linii 3 oraz 8 pozostały w wycinku, gdyż wykonanie przypisania do zmiennej „z” w linii 11 jest zależnie od wartości zmiennej „x” (linia 8 oraz 10). Wartość zmiennej „x” wynika natomiast z wartości parametru `args[0]` (linia 3).

Wspomnieć należy również, iż oprócz slicingu wstecznego istnieje również slicing „do przodu” (ang. *forward slicing*). Metoda ta służy do określenia wpływu zadanego w kryterium slicingu wyrażenia i zmiennych na część kodu występującą „później” z punktu widzenia przetwarzania programu. W dalszej części artykułu metoda ta jednak nie będzie szerzej omawiana a opisane zagadnienia rozważane są jedynie w kontekście slicingu wstecznego.

2 Odmiany slicingu

2.1 Statyczny slicing (ang. *Static Slicing*)

Na przestrzeni lat powstało wiele odmian slicingu. Statyczny slicing został zaprezentowany jako pierwszy i jest jedną z najprostszych metod tworzenia wycinków kodu. Pozostałe metody powstawały później będąc jej rozszerzeniami. Wycinek jest tworzony poprzez usuwanie z programu tych wyrażen, które nie są istotne z punktu widzenia wartości wybranych zmiennych w obszarze zadanym w kryterium slicingu. Przykład statycznego slicingu został przedstawiony we wprowadzeniu (patrz rysunek 2). Niestety w wyniku stosowania tej metody dla dobrze zaprojektowanego programu otrzymane wycinki będą dość duże (niewiele mniejsze niż oryginalny fragment programu). Wynika to z faktu, że w programach o dużej spójności funkcjonalnej wartość zadanej zmiennej będzie wynikała z dużej liczby innych zmiennych użytych w rozważanym fragmencie kodu źródłowego.

Dokładny opis statycznego slicing jest dostępny w pracach [2], [3], [4] oraz [8].

2.2 Dynamiczny slicing (ang. *Dynamic Slicing*)

Dzięki metodzie dynamicznego slicing można uzyskać znacznie mniejszy wycinek dla zadanego kryterium slicing. Wynika to z faktu, iż metoda ta w kryterium slicing uwzględnia dodatkowo wartości zmiennych, dla których tworzony jest wycinek kodu. W przeciwieństwie do statycznego slicing nie są tu rozpatrywane wszystkie możliwe wartości zmiennych. Kryterium dynamicznego slicing <S, V, I> składa się więc z trzech składowych:

- obszaru występowania (lokalizacji) wyrażeń S,
- zmiennych V,
- ustalonych wartości zmiennych I.

Wracając do przykładowego kodu źródłowego przedstawionego na rysunku 1, dla kryterium dynamicznego slicing sformułowanego jako: <15, z, <args[0]="1", args[1]="2", args[2]="3">>, wycinek oprócz części stałych takich jak deklaracja klasy oraz metody składa się z pojedynczego wyrażenia (`int z = 0;`). Rezultat przedstawia rysunek 3.

```
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">>

Ponieważ wartość zmiennej `args[0]` jest ustalona w kryterium, więc ścieżka wykonywania programu jest z góry znana. Z wartości zmiennej „x” wynika, iż w linii 8 oryginalnego kodu źródłowego (rysunek 1) warunek zawsze będzie spełniony i sterowanie nigdy nie zostanie przekazane do bloku „else”. Dlatego też w przeciwieństwie do wycinka utworzonego w oparciu o metodę statycznego slicing, wyrażenia z linii 8, 10, 11 oraz 13 mogą zostać usunięte. Powoduje to znaczne zmniejszenie rozmiaru wycinka i poprawienie czytelności istotnego fragmentu kodu.

Więcej informacji na temat dynamicznego slicing można znaleźć w pracy [1] a także w pracach [3] i [4].

2.3 Warunkowy slicing (ang. *Conditioned Slicing*)

Warunkowy slicing jest kolejną metodą, która dąży do zmniejszenia rozmiaru wycinka kodu. Zakłada ona jednak pewien stopień swobody podczas precyzowania wartości zmiennych. O ile w dynamicznym slicing wymagane są konkretne wartości, o

tyle tutaj w kryterium slicingu wystarczy sprecyzować pewne zależności pomiędzy zmiennymi. Kryterium warunkowego slicingu $\langle S, V, C \rangle$ składa się więc z trzech składowych:

- obszaru występowania (lokalizacji) wyrażeń S ,
- zmiennych V ,
- wartości zmiennych lub wyrażenia warunkowego C .

Przykładowo zakładając pewien stopień abstrakcji można zapisać kryterium warunkowego slicingu jako:

```
<15, z,  
  <args[1]="1", args[0] < args[1], args[2]=args[0] >  
>
```

Wycinek uzyskany zgodnie z tak zadaniem kryterium powinien być identyczny jak ten dla dynamicznego slicingu. Wycinek został przedstawiony na rysunku 4.

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

Rysunek 4. Wycinek utworzony zgodnie z kryterium

```
<15, z, <args[1]="1", args[0] < args[1], args[2]=args[0] >>
```

Podobnie jak w przypadku dynamicznego slicingu można przewidzieć tu przepływ sterowania w programie. Z zależności podanych w kryterium slicingu można wywnioskować, że wartość `args[0]` a co za tym idzie wartość zmiennej „x” będzie zawsze mniejsza od 1. Oznacza to, że warunek `x <= 1` (linia 8 oryginalnego kodu źródłowego z rysunku 1) będzie zawsze spełniony. Na tej podstawie do wycinka, podobnie jak w dynamicznym slicingu, nie trafiają wyrażenia znajdujące się w liniach 8, 10, 11 oraz 13.

Więcej informacji na temat warunkowego slicingu można znaleźć w pracy [4].

2.4 Bezpostaciowy slicing (ang. *Amorphous Slicing*)

Bezpostaciowy slicing w przeciwieństwie do przedstawionych wcześniej metod przyjmujących usuwanie wyrażeń jako jedyny sposób na utworzenie wycinka (czyli tzw. metod zachowujących oryginalną składnię, ang. *syntax-preserving*) rezygnuje z tego ograniczenia. W celu dalszego uproszczenia wycinka przedstawianego programiście do analizy dokonywana jest modyfikacja struktury programu. Oczywiście modyfikowane fragmenty wycinka muszą być semantycznie równoważne oryginalnemu kodowi. Bardzo prosty przykład wycinka utworzonego za pomocą metody bezpostaciowego slicingu dla kryterium slicingu $\langle 15, z \rangle$ dla kodu źródłowego z rysunku 1 jest przedstawiony na rysunku 5.

```

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>

Wycinek z rysunku 5 jest równoważny wycinkowi wykonanemu za pomocą statycznego slicing (rysunek 2). Jediną różnicą jest modyfikacja wyrażenia warunkowego „if” tak, aby było one czytelniejsze dla programisty. W przypadku statycznego slicing wycinek zawierał całą logikę w bloku „else”. W przypadku bezpostaciowego slicing wyrażenie warunkowe zostało zmodyfikowane tak, aby sprawdzało warunek przeciwny do oryginalnego. Pozwoliło to usunąć blok „else” programu.

Więcej informacji na temat bezpostaciowego slicing można znaleźć w pracy [5].

3 Podsumowanie

W części pierwszej artykułu przedstawiono podstawy popularnej techniki usprawniającej proces zrozumienia i analizy kodu źródłowego zwanej slicingiem kodu. Wprowadzono pojęcie kryterium slicing oraz wycinka. Zademonstrowano cztery popularne metody slicing: statyczny, dynamiczny, warunkowy oraz bezpostaciowy. Metody te omówiono krótko na przykładzie kodu źródłowego napisanego w języku Java. W części drugiej artykułu przedstawiony będzie praktyczny aspekt slicing kodu. Omówione zostaną zastosowania slicing kodu m.in. podczas debugingu czy refactoringu. Zademonstrowane będzie działanie statycznego slicing kodu w praktyce z użyciem narzędzia Indus/Kaveri.

Literatura

1. Agrawal H., Horgan J.: Dynamic Program Slicing. Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation. pp. 246-256, White Plains, NY, June 1990.
2. Binkley, D., Gallagher, K.: Program Slicing. Advances in Computers, Volume 43, Academic Press San Diego, 1996.
3. De Lucia, A.: Program Slicing: Methods and Applications. Source Code Analysis and Manipulation, Proceedings. 2001.
4. Harman, M., Hierons R.: An overview of Program Slicing, <http://www.brunel.ac.uk/~csstmmh2/>.

5. Harman, M., Binkley D., Danicic S.: Amorphous program slicing. *Journal of Systems and Software*, Volume 68 , Issue 1, 2003.
6. Russel J.: *Program Slicing Literature Survey*. 2001.
7. Weiser, M.: *Program Slicing: Formal, Psychological and Practical Investigations of an Automatic Program Abstraction Method*. PhD thesis, The University of Michigan, Ann Arbor, Michigan, 1979.
8. Weiser, M.: *Program Slicing*. *Proceeding of the Fifth International Conference in Software Engineering*, pages 439-449, 1981.