Optimizing Programs with Intended Semantics

Interaktywna optymalizacja programów

Krzysztof Niemkiewicz

26 kwietnia 2010

Spis treści

Spis treści

Problemy z optymalizacją

Problemy z optymalizacją - prosty przykład

```
...
i = 0;
new EmptyConstructor();
j = i * i;
...
```

Problemy z optymalizacją - przykład ciekawszy

```
...
String format;
for (String s : stringList){
    this.printPrefix();
    format = formatField;
    this.printWithFormat(s,format);
};
...
```

Problemy z optymalizacją

Problemy z optymalizacją - prosty przykład

```
...
i = 0;
new EmptyConstructor();
j = i * i;
...
```

Problemy z optymalizacją - przykład ciekawszy

```
...
String format;
for (String s : stringList){
    this.printPrefix();
    format = formatField;
    this.printWithFormat(s,format);
};
...
```

Rozwiązanie - adnotacje

Nie będzie wyjątku

```
...
i = 0;
@neverThrows(OutOfMemoryError)
new EmptyConstructor();
j = i * i;
...
```

Uważny czytelnik pewnie zauważył

- adnotacja na pojedynczej linii kodu !!
- JSR 308 ?

Rozwiązanie - adnotacje

Nie będzie wyjątku

```
...
i = 0;
@neverThrows(OutOfMemoryError)
new EmptyConstructor();
j = i * i;
...
```

Uważny czytelnik pewnie zauważył

- adnotacja na pojedynczej linii kodu !!
- JSR 308 ?

Rozwiązanie - adnotacje

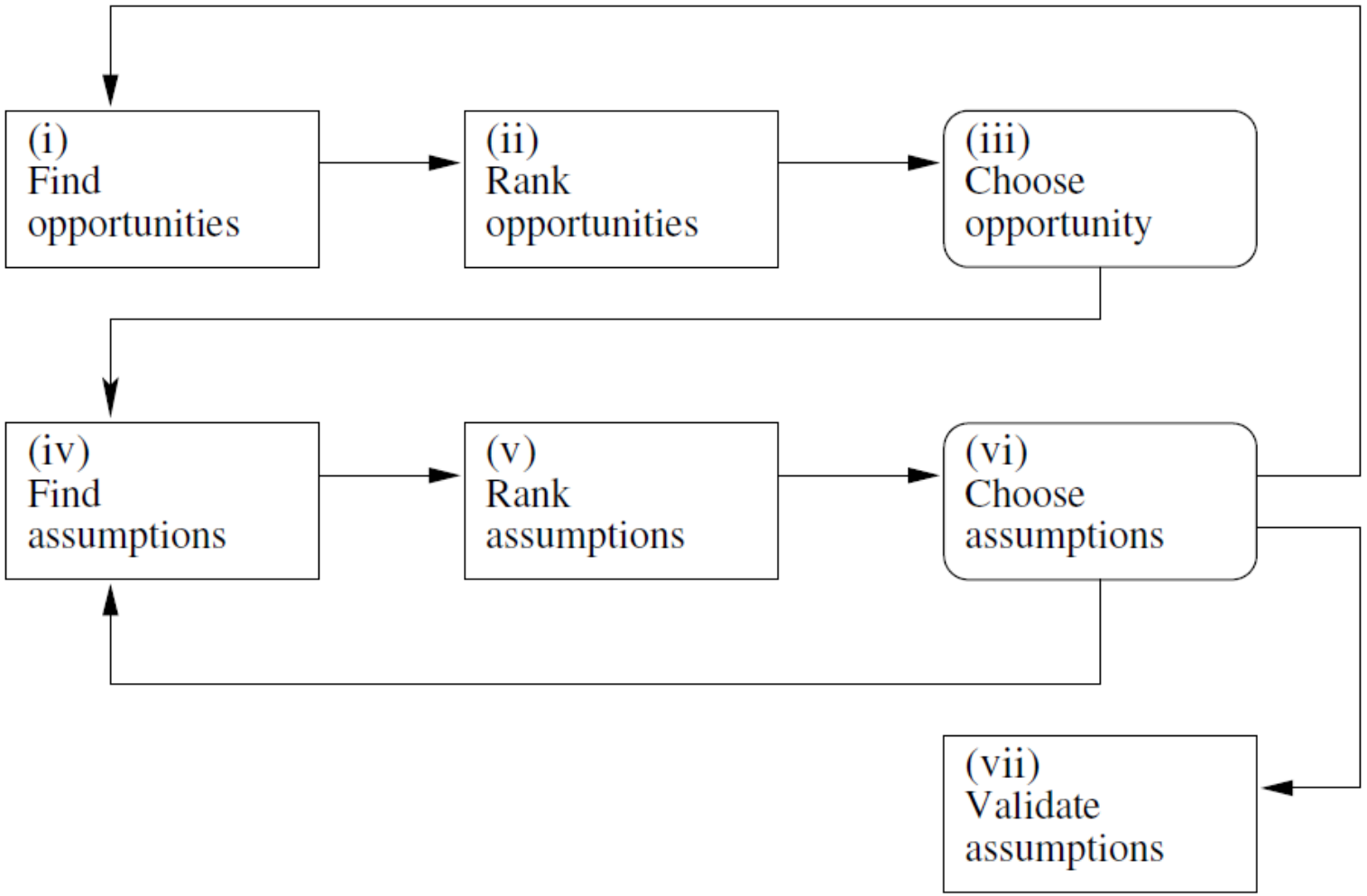
Nie będzie wyjątku

```
...
i = 0;
@neverThrows(OutOfMemoryError)
new EmptyConstructor();
j = i * i;
...
```

Uważny czytelnik pewnie zauważył

- adnotacja na pojedynczej linii kodu !!
- JSR 308 ?

Kolejne kroki alogrytmu



Kolejny przykład

Zagadka dla optymalizatora...

```
...
boolean func(a, b){
    Object x, temp1, temp2;
    x = new Object();
    int counter = 0;
    while ( counter < b.length ) {
        temp1 = b[ counter ];
        temp2 = a.field ;
        if (temp1 == temp2){
            return true ;
        }
        counter++;
    }
}
...
```

Uruchamiamy IOpt

Przykładowe propozycje

- loop-invariant code motion (LICM) dla temp2 = a.field ;
- dead-store elimination dla x = new Object();

Przyczyny niepowodzenia optymalizacji

```
temp2 = a. field may not be executed
Explanation :
    b[ counter ] may be executed
and
    b[ counter ] may throw an exception
```

```
new Object() may exit the method
Explanation :
    new Object() may throw an exception
and
    new Object() may change some state
Explanation :
    new Object() is non-analyzed code
and
    non-analyzed code may modify any field
```

Uruchamiamy IOpt

Przykładowe propozycje

- loop-invariant code motion (LICM) dla temp2 = a.field ;
- dead-store elimination dla x = new Object();

Przyczyny niepowodzenia optymalizacji

```
temp2 = a. field may not be executed
Explanation :
    b[ counter ] may be executed
and
    b[ counter ] may throw an exception
```

```
new Object() may exit the method
Explanation :
    new Object() may throw an exception
and
    new Object() may change some state
Explanation :
    new Object() is non-analyzed code
and
    non-analyzed code may modify any field
```

Kolejne kroki

Kolejne kroki

- wybór jednej z propozycji, w tym przypadku np. LICM
- generowanie propozycji założeń/adnotacji
- walidacja propozycji wybranych przez programistę

Przykładowe założenia

```
temp2 = a. field may not be executed
-> Assume temp2 = a.field is always executed

b[ counter ] may be executed
-> Assume b[counter] is never executed

b[ counter ] may throw an exception
-> Assume b[counter] never throws an exception
```

Dodatkowo, przez uogólnienie:

```
Assume any statement never throws an exception
```

Kolejne kroki

- wybór jednej z propozycji, w tym przypadku np. LICM
- generowanie propozycji założeń/adnotacji
- walidacja propozycji wybranych przez programistę

Przykładowe założenia

```
temp2 = a. field may not be executed
-> Assume temp2 = a.field is always executed

b[ counter ] may be executed
-> Assume b[counter] is never executed

b[ counter ] may throw an exception
-> Assume b[counter] never throws an exception
```

Dodatkowo, przez uogólnienie:

```
Assume any statement never throws an exception
```

Wewnętrzny język zdefiniowany przez autorów

Język w którym wyrażane są warunki na zastosowanie optymalizacji

- property(a, x1, ... , xn)
- operatory and, or
- operator forall x in y z

Przyczyny niepowodzeń analiz

- przyczyny podstawowe - failed(analysis, x1, ... , xn)
- analogiczne operatory - and, or

Przykładowa reguła

```
isDeadStore(s = e) :=
   (forall x in property ( usesForDefinition , s = e)
      property (isDeadCode, x))
      and
   property ( isStraightLineCode , s = e)
      and
   property (doesNotChangeState, e)
```

Wewnętrzny język zdefiniowany przez autorów

Język w którym wyrażane są warunki na zastosowanie optymalizacji

- property(a, x1, ... , xn)
- operatory and, or
- operator forall x in y z

Przyczyny niepowodzeń analiz

- przyczyny podstawowe - failed(analysis, x1, ... , xn)
- analogiczne operatory - and, or

Przykładowa reguła

```
isDeadStore(s = e) :=
   (forall x in property ( usesForDefinition , s = e)
      property (isDeadCode, x))
      and
   property ( isStraightLineCode , s = e)
      and
   property (doesNotChangeState, e)
```

Wewnętrzny język zdefiniowany przez autorów

Język w którym wyrażane są warunki na zastosowanie optymalizacji

- property(a, x1, ... , xn)
- operatory and, or
- operator forall x in y z

Przyczyny niepowodzeń analiz

- przyczyny podstawowe - failed(analysis, x1, ... , xn)
- analogiczne operatory - and, or

Przykładowa reguła

```
isDeadStore(s = e) :=
   (forall x in property ( usesForDefinition , s = e)
      property (isDeadCode, x))
      and
   property ( isStraightLineCode , s = e)
      and
   property (doesNotChangeState, e)
```

Inne elementy programu

Wyszukiwanie okazji do optymalizacji

- na podstawie wzorców
- dopasowywane są konstrukcje języka, elementy zmienne (np. dowolny typ) oraz ich ciągi
- wydaje się, że takie podejście generuje b. wiele propozycji i może być dość wolne

Zbiór reguł

- można go rozszerzać korzystając ze zdefiniowanych języków
- ważne jest, żeby z reguły dało się wygenerować określony powód uniemożliwiający optymalizację i propozycję adnotacji
- propozycje i przyczyny muszą być czytelne dla programisty

Wybór propozycji optymalizacji

Według ryzyka

Koszt określa liczba przyczyn, które uniemożliwiają optymalizację. Dodatkowo przyczyny szczegółowe są bardziej „ryzykowne”.

Według powszechności

Reguły o dużej liczbie wspólnych przyczyn są mniej ryzykowne, a usuwanie przyczyn jest bardziej opłacalne.

Według oczekiwanego przyspieszenia

Uwzględniane jest oczekiwane przyspieszenie podzielone przez liczbę wymaganych założeń.

Wybór propozycji założeń / adnotacji

Założenia

- Po wybraniu optymalizacji programista powinien wybrać założenia, które je umożliwiają lub ją odrzucić
- Podobnie jak w przypadku samych optymalizacji, chcemy zmaksymalizować efektywność programisty
- Musimy wybrać założenia na tyle ogólne, by umożliwiały wiele optymalizacji, ale jednocześnie na tyle szczegółowe, aby zachodziły

Parametry uwzględniane przy wyborze

- według użyteczności danego założenia (liczba optymalizacji które umożliwia)
- według ryzyka (odwrotnie niż dla optymalizacji, bardziej szczegółowe są wybierane)

Wybór propozycji założeń / adnotacji

Założenia

- Po wybraniu optymalizacji programista powinien wybrać założenia, które je umożliwiają lub ją odrzucić
- Podobnie jak w przypadku samych optymalizacji, chcemy zmaksymalizować efektywność programisty
- Musimy wybrać założenia na tyle ogólne, by umożliwiały wiele optymalizacji, ale jednocześnie na tyle szczegółowe, aby zachodziły

Parametry uwzględniane przy wyborze

- według użyteczności danego założenia (liczba optymalizacji które umożliwia)
- według ryzyka (odwrotnie niż dla optymalizacji, bardziej szczegółowe są wybierane)

Testowanie wybranych założeń

Testowanie

- Istnieje możliwość wyboru błędnego założenia
- Efektem będzie najprawdopodobniej nieprawidłowo działający program
- Abyt tego uniknąć, IOpt ma wbudowaną możliwość przetestowania założeń
- Opcja ta działa w oparciu o AspektJ i JUnit

Badanie efektywności IOpt-a

Dwa rodzaje testów

- Najpierw przetestowane zostały wybrane heurystyki wyboru optymalizacji
- Dodatkowo przetestowano całą procedurę optymalizacji na dwóch wybranych programach

Testy heurystyk

- Publikacja zawiera dane dotyczące optymalizacji LICM
- Wykazano, że wybór optymalizacji według ryzyka dobrze rozdziela propozycje
- Wybór według liczby wspólnych przyczyn również został uzasadniony przez dane

Badanie efektywności IOpt-a

Dwa rodzaje testów

- Najpierw przetestowane zostały wybrane heurystyki wyboru optymalizacji
- Dodatkowo przetestowano całą procedurę optymalizacji na dwóch wybranych programach

Testy heurystyk

- Publikacja zawiera dane dotyczące optymalizacji LICM
- Wykazano, że wybór optymalizacji według ryzyka dobrze rozdziela propozycje
- Wybór według liczby wspólnych przyczyn również został uzasadniony przez dane

Testy IOpt na prawdziwych aplikacjach

Symulator inteligetnych sieci radiowych

- Początkowo aplikacja była niezdolna do symulacji dużych sieci
- Użyto optymalizacji: Strength Reduction(zastąpienie metod przez prostsze obliczenia), Method Part Precomputation oraz LICM
- Początkowo autorzy nie znali kodu, całość pracy zajęła około 30 min

Wyniki

Optymalizacja	Brak	SR	MPP	LICM	Razem
Czas	100%	72-90%	40-45%	80-86%	23-35%

Testy IOpt na prawdziwych aplikacjach

Symulator inteligetnych sieci radiowych

- Początkowo aplikacja była niezdolna do symulacji dużych sieci
- Użyto optymalizacji: Strength Reduction(zastąpienie metod przez prostsze obliczenia), Method Part Precomputation oraz LICM
- Początkowo autorzy nie znali kodu, całość pracy zajęła około 30 min

Wyniki

Optymalizacja	Brak	SR	MPP	LICM	Razem
Czas	100%	72-90%	40-45%	80-86%	23-35%

Testy IOpt na prawdziwych aplikacjach

jess - zaawansowany silnik do przetwarzania reguł

- Najbardziej realistyczny z dostarczonych przykładów
- Użyto optymalizacji: Partial Redundancy Elimination(usunięcie niektórych operacji **new**), Remove Rewriting(dotyczy metody Collection.remove()) oraz LICM
- całość pracy zajęła około 15 min, autorzy znali kod

Wyniki

Optymalizacja	Brak	PRE	RR	LICM	Razem
Czas	100%	38-51%	94-97%	96-98%	36-47%

Testy IOpt na prawdziwych aplikacjach

jess - zaawansowany silnik do przetwarzania reguł

- Najbardziej realistyczny z dostarczonych przykładów
- Użyto optymalizacji: Partial Redundancy Elimination(usunięcie niektórych operacji **new**), Remove Rewriting(dotyczy metody Collection.remove()) oraz LICM
- całość pracy zajęła około 15 min, autorzy znali kod

Wyniki

Optymalizacja	Brak	PRE	RR	LICM	Razem
Czas	100%	38-51%	94-97%	96-98%	36-47%

Podsumowanie

Według autorów publikacji

- Niektóre możliwości języka, w szczególności dynamiczne ładowanie klas i wyjątki, bardzo utrudniają pracę standardowych optymalizatorów
- Praca opisuje nowe rozwiązanie oparte na znajomości zamierzonej semantyki programu
- Został przedstawiony program umożliwiający interaktywne dospecyfikowanie semantyki programu i optymalizację
- Program został z powodzeniem przetestowany na powszechnie dostępnych przykładowych programach

Podsumowanie

Według autora prezentacji

- Bardzo ciekawy pomysł
- Niestety program nie jest dostępny w Internecie i jego przetestowanie nie jest możliwe
- Wydawałoby się, że przy podanym sposobie wyboru, programowi/programiście znacznie dłużej zajmie znalezienie efektywnych optymalizacji
- Właściwie programista mógłby samodzielnie poprawić program zamiast wstawiać założenia