Model-checking programów w Javie

Sławomir Rudnicki

Niezawodność systemów współbieżnych i obiektowych

20 stycznia 2010

1. **Wprowadzenie**
   - O model-checkingu raz jeszcze
2. **Proste metody**
   - Modelowanie
   - Translacja
3. **JPF**
   - Podstawy działania
   - Techniki
   - JPF w działaniu
   - Na chwilę obecną…
4. **Podsumowanie**
   - Porównanie metod
   - Bibliografia

Cel

Dla danego programu, chcemy sprawdzić:

1. brak zakleszczenia
2. spełnienie asercji
3. brak nieobsłużonych wyjątków
4. ...

Sposób działania

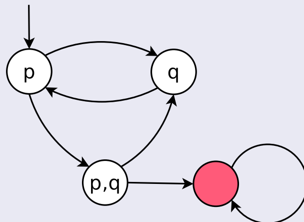
- Weryfikacja przez wyliczenie stanów programu.

Struktura Kripkego

$$M = (S, S_0, R, L),$$

gdzie

- S – zbiór stanów modelu,
- $S_0 \subseteq S$ – stany początkowe,
- $R \subseteq S \times S$ – funkcja przejścia,
- $L : S \to 2^P$ – etykietowanie predykatami.



Podstawowe problemy

- Eksplozja stanów
- Skomplikowany język do analizy:
  - odwołania do zewnętrznych bibliotek
  - dynamiczna alokacja pamięci
  - obsługa wątków
- Duży rozmiar weryfikowanych programów

Modelowanie

Sposób działania:

- Na podstawie projektu programu stwórz jego model w innym języku programowania.
- Użyj model-checkera dla tego języka.

Modelowanie – zalety i problemy

Zalety

- Abstrakcja:
  - wywołań funkcji bibliotecznych
  - wywołań niskopoziomowych

Problemy

- Ograniczony zakres języka docelowego
- Możliwe błędy przy tworzeniu modelu
- Model może nie odzwierciedlać końcowego programu.

Translacja

Sposób działania:

- Przetłumacz automatycznie kod Javy do prostszego języka
- Użyj istniejącego model-checkera

Implementacje:

- **Java Pathfinder (1999)**
  translacja do Promeli
- **Java Model Checker (Stanford, 2000)**
  translacja bajtkodu do języka SAL

Algorytm Petersona w Promeli

```
bool turn, flag[2];
byte cnt;
active [2] proctype P1()
{
        pid i, j;
        i = _pid;
        j = 1 - _pid;
again:  flag[i] = true;
        turn = i;
        !(flag[j] && turn == i) ->
                cnt++;
                assert(cnt == 1);
                cnt--;
        flag[i] = false;
        goto again
}
```

Translacja: problemy

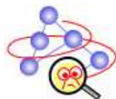
- Język docelowy nie pokrywa źródłowego

Java → Promela

- dynamiczna alokacja pamięci
- obsługa liczb zmiennoprzecinkowych
- ...

- Ograniczenie jakością docelowego model-checkera
- Potrzebne źródła tłumaczonego programu.

Java Pathfinder (JPF)



- Projekt *Robust Software Engineering Group* przy NASA

Historia

1999 Translator Java $\rightarrow$ Promela

2000 Odrębny model-checker Javy

2003 Implementacja mechanizmu rozszerzeń

2006 JPF4

- W 2005 roku udostępniono źródła.

JPF4

- Zastępuje maszynę wirtualną Javy
  - śledzenie stanów programu
  - możliwość powrotu w punktach wyboru
- Ograniczony zakres weryfikacji
- Nacisk na rozszerzalność

Swiss army knife of Java verification

Podstawy działania

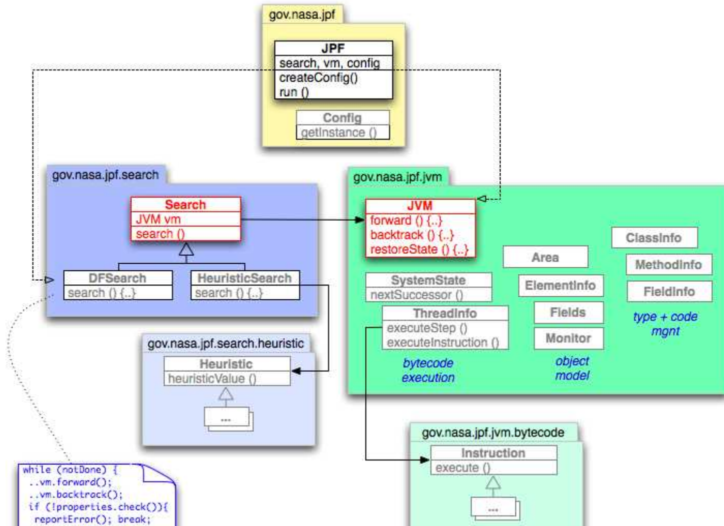
Główne elementy:

JVM – Reprezentacja maszyny wirtualnej. Odpowiada za:

- zarządzanie klasami, metodami i polami obiektów
- symulację wykonywania bajtkodu
- przechowywanie stanów obiektów

Search – Abstrakcja polityki przeszukiwania przestrzeni stanów.

Podstawy działania



Reprezentacja stanów

Motywacja:

- język obiektowy $\longrightarrow$ złożony opis stanu
- konieczność efektywnego porównywania stanów

Reprezentacja stanów

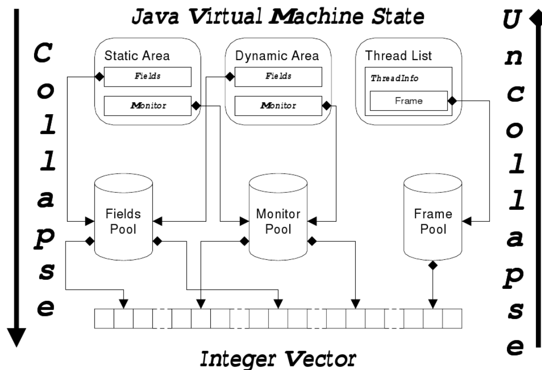
Elementy stanu:

- Stan wątku:
  - ramki stosu
  - zmienne lokalne
- Elementy statyczne:
  - klasy
  - zmienne statyczne
- Zmienne dynamiczne:
  - pola obiektów
  - stan monitorów

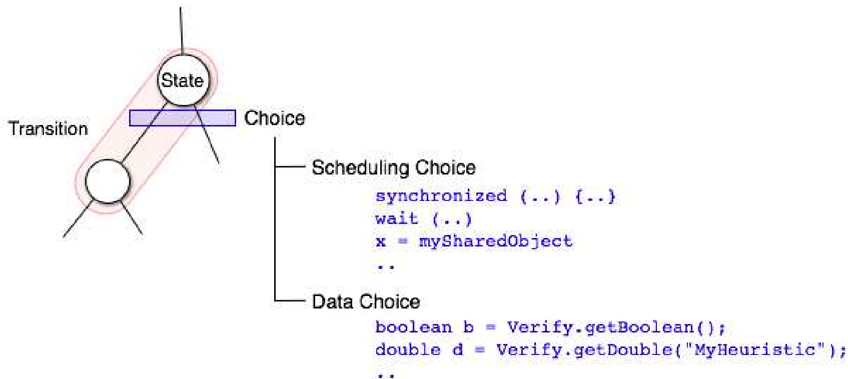
Przechowywanie:

- elementy w osobnych tablicach haszujących
- stan $\leftrightarrow$ ciąg indeksów

Reprezentacja stanów



Przeszukiwanie



Przeszukiwanie

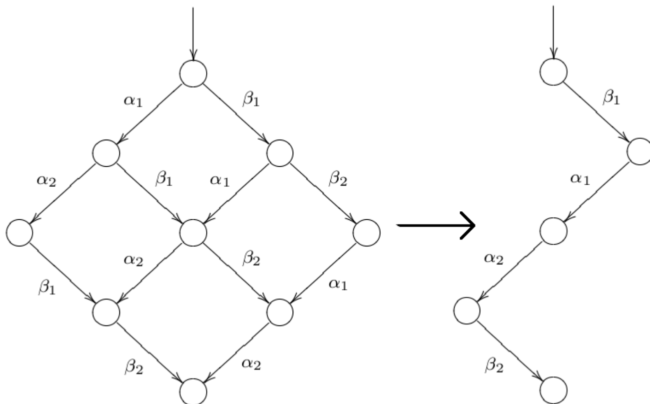
Generator wyboru:

- tworzony w punkcie wyboru
- wylicza możliwości

Przykładowe generatory wyboru

- ThreadChoiceGenerator
- BooleanGenerator
- DoubleThresholdGenerator
- . . .

Redukcje częściowo-porządkowe



Redukcje częściowo-porządkowe

Motywacja:

- znaczne zmniejszenie liczby stanów
- niewiele instrukcji oddziałujących z innymi wątkami

Realizacja:

- wykonywana w locie przez JVM
- oparta o analizę statyczną kodu

Model Java Interface

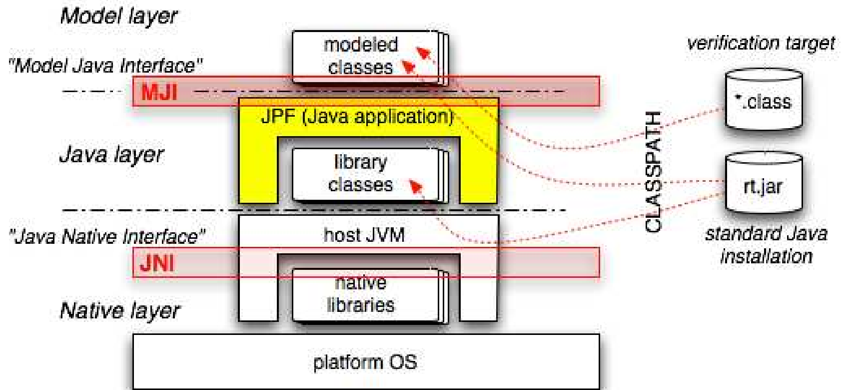
Motywacja:

- niektóre wywołania metod są “nieistotne”

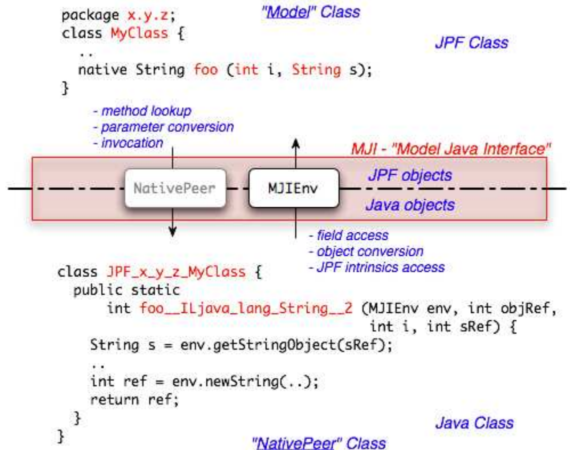
Realizacja:

- wykonanie tych metod delegowane do zwykłej maszyny wirtualnej Javy
  - ↪ szybsze wykonanie
  - ↪ redukcja liczby stanów
- dla modelowanej klasy należy utworzyć “natywną” klasę jej odpowiadającą (*NativePeer*)

Model Java Interface



Model Java Interface



Verify API

API używane do budowy modeli w Javie.

- niedeterministyczny wybór:
  `Verify.getBoolean()`
  `Verify.getDouble()...`
- wsparcie wyszukiwania:
  `Verify.ignoreIf(p)`
  `Verify.interesting()`
- kontrola atomowości:
  `Verify.beginAtomic()`
  `Verify.endAtomic()`

Definiowanie własności

- Asercje w kodzie

```
assert(node.next != null);
```

- Własności

```
public class MyProperty implements Property {
 boolean check (Search search, VM vm) {...};
 String getErrorMessage() {...};
}
```

Definiowanie własności

- Obserwatorzy:
  - wyszukiwania
  - maszyny wirtualnej

Przykłady zdarzeń

- odwiedzenie nowego stanu
- przełączenie wykonywanego wątku
- wykonanie instrukcji

- Możliwe łączenie obserwatorów i własności.

JPF w działaniu

- uczta filozofów
- analiza pokrycia kodu
- protokół sekcji krytycznej

Na chwilę obecną. . .

- Programy do ok. 10 000 linii kodu
- Brak wsparcia:
  - `java.net`
  - `java.awt`
- Ograniczone wsparcie:
  - `java.io`
- Brak wsparcia dla logik temporalnych.

Rozszerzenia

jpf-aprop

Weryfikacja zgodnie z adnotacjami:

@Nonnull	@Requires
@Const	@Ensures
@NonShared	@Invariant
@SandBox	

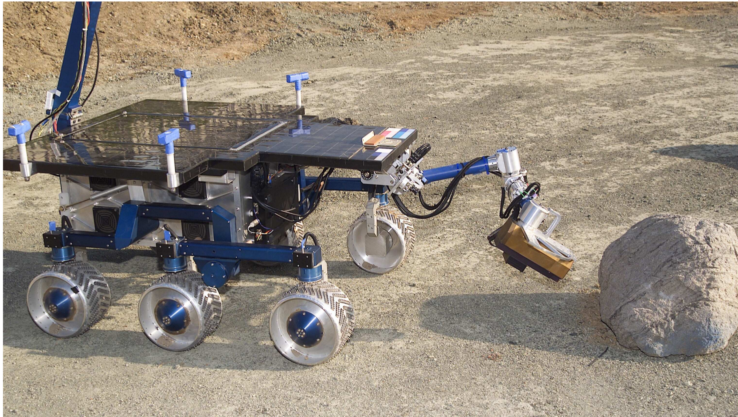
jpf-shell

Graficzny front-end dla JPF.

rtembed

Weryfikacja systemów wbudowanych (RT).

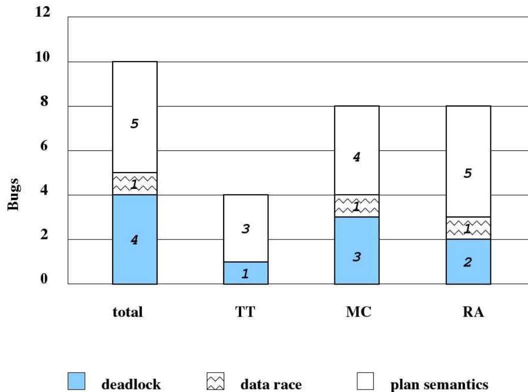
K9 Mars Rover (2005)



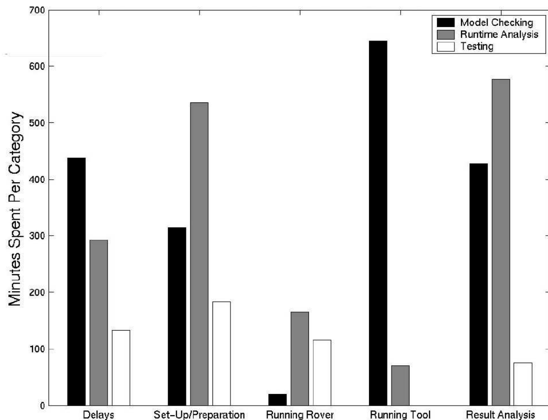
Eksperyment weryfikacyjny

- Testowanie ręczne
- Analiza statyczna – *PolySpace Verifier*
- Runtime checking – *JPaX, DBRover*
- Model-checking – *JPF*

Eksperyment: statystyki



Eksperyment: statystyki



Eksperyment: wnioski

- testowanie tradycyjne słabo spisuje się przy współbieżności
- analiza statyczna: niezbyt skuteczna w danym zastosowaniu
- runtime checking: niewielki narzut, dobre wyniki
- model-checking: skuteczny, ale kosztowny

Wynik eksperymentu:

- X9: runtime checking + model-checking

Bibliografia

- *Java Pathfinder* – strona projektu
  `http://babelfish.arc.nasa.gov/trac/jpf`
- Brat, Havelund, Lerda, Park, Visser, *Model Checking Programs* [w:] *Automated Software Engineering Journal* vol. 10, nr 2, kwiecień 2003
- Havelund, Visser et al., *Experimental evaluation of Verification and Validation tools on Martian Rover Software* [w:] *Formal Methods in Systems Design Journal*, vol. 25, wrzesień 2005