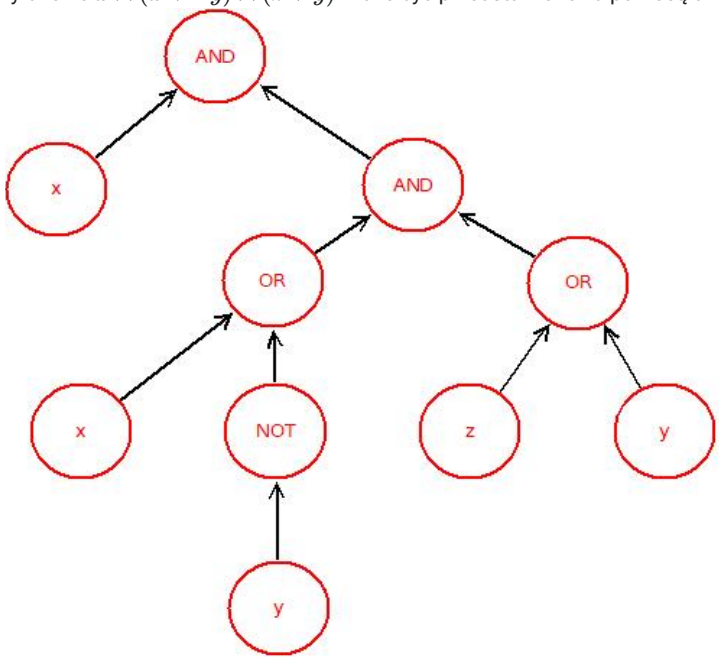
Duże zadanie zaliczeniowe z Javy

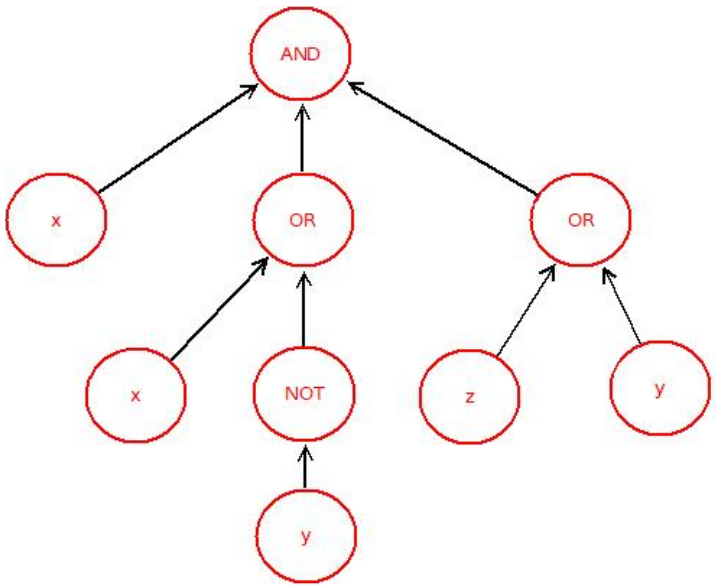
Wymagane do: poniedziałek, 2 grudnia 2024, 23:59

Obwody współbieżne

Obwody boolowskie reprezentują wyrażenia boolowskie przy pomocy grafów, np. wyrażenie $x \wedge (x \vee \neg y) \wedge (z \vee y)$ może być przedstawione za pomocą drzewa



lub, jeśli dopuścimy operatory dużych arności, za pomocą drzewa



Zwyczajowo obliczenia boolowskie są wyliczane od lewej do prawej, zatem w wyrażeniu $x \wedge y$ najpierw wyliczamy wartość x a następnie wartość y. Tak zwane

?

leniwe wyliczanie (lazy evaluation) może pominąć wyliczenie części podwyrażeń, jeśli wartość już obliczonych pozwala ustalić wartość całego wyrażenia. Na przykład wyrażenie $\mathrm{true} \vee x$ nie musi wyliczać wartości x by poznać wartość całego wyrażenia, które wylicza się do true. Zwróćmy uwagę, że jeśli wyrażenia nie generują efektów ubocznych to kolejność wyliczania podwyrażeń nie powinna wpływać na wartość całego wyrażenia, więc wyliczanie wartości podwyrażeń może odbywać się współbieżnie.

Waszym zadaniem jest implementacja programu pozwalającego na współbieżne wyliczanie wartości wyrażeń boolowskich. Program powinien pozwalać na równoczesne wyliczanie wielu wyrażeń i wyliczać wartości pojedynczych wyrażeń współbieżnie.

Wyrażenie boolowskie

Wyrażenie boolowskie jest zdefiniowane indukcyjnie. Stałe true oraz false są wyrażeniami boolowskimi. $\mathrm{NOT}\ a$, negacja wyrażenia boolowskiego a, jest wyrażeniem boolowskim. Koniunkcja $\mathrm{AND}(a1, a2, \ldots)$ oraz alternatywa $\mathrm{OR}(a1, a2, \ldots)$ pewnej liczby wyrażeń boolowskich (co najmniej dwu) są wyrażeniami boolowskimi. Instrukcja warunkowa $\mathrm{IF}(a, b, c)$ jest wyrażeniem boolowskim. Również wyrażenia progowe $\mathrm{GTx}(a_1, a_2, \ldots, a_n)$ oraz $\mathrm{LTx}(a_1, a_2, \ldots, a_n)$, gdzie $n \geq 1$ i $x \geq 0$ są liczbami całkowitymi, są wyrażeniami boolowskimi.

Semantyka

Dla wyrażenia a, przez $[a]$ oznaczamy wartość wyrażenia a.

- $[true] = \mathrm{true}$
- $[false] = \mathrm{false}$
- $[\mathrm{AND}(a_1, a_2, \ldots, a_n)] = \mathrm{true}$, jeśli każde wyrażenie a_i, $1 \leq i \leq n$, spełnia $[a_i] = \mathrm{true}$, oraz $[\mathrm{AND}(a_1, a_2, \ldots, a_n)] = \mathrm{false}$ w przeciwnym przypadku.
- $[\mathrm{OR}(a_1, a_2, \ldots, a_n)] = \mathrm{true}$, jeśli istnieje wyrażenie a_i, $1 \leq i \leq n$, spełnia $[a_i] = \mathrm{true}$, oraz $[\mathrm{OR}(a_1, a_2, \ldots, a_n)] = \mathrm{false}$ w przeciwnym przypadku.
- $[\mathrm{GTx}(a_1, a_2, \ldots, a_n)] = \mathrm{true}$, jeśli co najmniej $x + 1$ wyrażeń a_i, $1 \leq i \leq n$, spełnia $[a_i] = \mathrm{true}$, oraz $[\mathrm{GTx}(a_1, a_2, \ldots, a_n)] = \mathrm{false}$ w przeciwnym przypadku.
- $[\mathrm{LTx}(a_1, a_2, \ldots, a_n)] = \mathrm{true}$, jeśli co najwyżej $x - 1$ wyrażeń a_i, $1 \leq i \leq n$, spełnia $[a_i] = \mathrm{true}$, oraz $\mathrm{LTx}(a_1, a_2, \ldots, a_n)] = \mathrm{false}$ w przeciwnym przypadku.
- $[\mathrm{IF}(a, b, c)] = [b]$, jeśli $[a] = \mathrm{true}$, oraz $[\mathrm{IF}(a, b, c)] = [c]$ w przeciwnym przypadku.

Specyfikacja

W rozwiązaniu obwody są reprezentowane poprzez obiekty klasy `Circuit`, a ich wartości wyliczane są poprzez obiekty implementujące interfejs `CircuitSolver`, zwane solverami.

```
public interface CircuitSolver {
    public CircuitValue solve(Circuit c);

    public void stop();
}
```

gdzie `CircuitValue` ma następujący interfejs.

?

```
public interface CircuitValue {
    public boolean getValue() throws InterruptedException;
}
```

Metoda `solve(Circuit c)` niezwłocznie zwraca specjalny obiekt typu `CircuitValue` reprezentujacy wartość obwodu. Wartość tę można pobrać wywołując metodę `CircuitValue.getValue()`, która czeka, aż wartość będzie wyliczona. Solvery powinny pozwalać na współbieżną obsługę wielu próśb (wywołań `solve()`) oraz o możliwie współbieżne obliczenie wartości obwodu.

Wywołanie metody `stop()` powinno spowodować zaprzestanie przyjmowania nowych zleceń `solve()` oraz niezwłocznie zakończyć wszystkie obecnie trwające obliczenia tego solvera. Od tego momentu, obiekty `CircuitValue` będące wynikiem nowych i przerwanych obliczeń mogą zwracać `InterruptedException` gdy wywołamy na nich `getValue()`. Pozostałe obiekty powinny zwracać poprawnie wyliczone wartości obwodów.

Klasa reprezentująca obwody `Circuit` oraz jej klasy pomocnicze zostaną dostarczone w archiwum. Interfejs `Circuit` wygląda następująco.

```
public class Circuit {
    private final CircuitNode root;

    public final CircuitNode getRoot();
}
```

Wyrażenia umieszczone w polu root mają drzewiastą strukurę reprezentowaną przez klasy, które zawierają m.in. następujące pola (szczegółowy interfejs jest dotępny w archiwum umieszczonym w sekcji Rozwiązanie).

```
public class CircuitNode{
  private final NodeType type;
  private final CircuitNode[] args;

  public CircuitNode[] getArgs();
  public NodeType getType();
}

public class ThresholdNode extends CircuitNode{
        public int getThreshold();
}

public class LeafNode extends CircuitNode{
  public boolean getValue();
}
```

gdzie `NodeType` jest typem wyliczeniowym

```
public enum NodeType {
  LEAF, GT, LT, AND, OR, NOT, IF
}
```

z naturalną interpretacją symboli.

Współbieżność: żywotność i bezpieczeństwo.

Program powinien pozwalać na wiele równoczesnych zapytań `solve()`. Wyniki wywołań `solve()` powinny być zwracane możliwie szybko a wartości obliczonych wyrażeń nie muszą być wykorzystywane zgodnie z kolejnością wywołań. Zarówno wartości liści jak i wartości operatorów powinny być wyliczane współbieżnie. W szczególności, należy przyjąć, że wywołania `LeafNode.getValue()` oraz `getArgs()` mogą wyliczać się dowolnie długo, ale nie powodują efektów ubocznych i poprawnie obługują przerwania. Można założyć, że zewnętrzne (tj. nie wynikające z Państwa implementacji) wywołania przerwań, np. wywołanie metody `Thread.interrupt()`, będzie ograniczone do wątków w czasie wywołań metody `CircuitValue.getValue()`.

?

W rozwiązaniu można założyć, że każdy węzeł w drzewiastej strukturze wyrażenia jest unikalny oraz, że zbiory węzłów struktur drzewiastych obwodów są parami rozłączne. W szczególności każde wywołanie `solve()` otrzymuje inną instancję `Circuit`. Dodatkowo, w implementacji można założyć, że na każdym utworzonym obiekcie klasy `CircuitSolver` zostanie wywołana metoda `stop()`.

Rozwiązanie

Rozwiązaniem jest archiwum ab123456.zip gdzie

- ab123456 zostało zastąpione identyfikatorem studenta,
- a w pakiecie (package) `cp2024.solution` znajduje się implementacja solvera.

Wymagania formalne

- Rozwiązanie należy przesłać poprzez umieszczenie w odpowiednim miejscu na Moodle.
- Pliki pakietu `cp2024.solution` mogą być modyfikowane, i można w nich umieszczać pliki pomocnicze rozwiązania, np. nowe definicje klas. Zmiany w pozostałych pakietach (katalogach archiwum) zostaną zignorowane, w szczególności...
- Rozwiązania będą uruchamiane przez skopiowanie plików `*.java` z cp2024/solution, wszelkie pozostałe zmiany będą zignorowane (tj. nie należy tworzyć podpakietów ani zmieniać interfejsów).
- Rozwiązanie musi kompilować i uruchamiać się na maszynie `students.mimuw.edu.pl` z katalogu src/ poleceniem `javac -d ../bin/ cp2024/*/*.java && java --class-path ../bin/ cp2024.demo.Demo` (po użyciu `cp2024.solution.ParallelCircuitSolver` w `Demo.java`).
- Implementacja nie powinna wypisywać niczego na standardowe wyjścia (`System.out` i `System.err`).
- W plikach źródłowych nie należy używać nieanglojęzycznych znaków (w szczególności polskich znaków) poza komentarzami.
- Styl kodu powinien być konsekwentny i spójny. Można np. przyjąć konwencje z Google Java Style Guide.
- Rozwiązanie nie może korzystać z klasy `java.util.concurrent.CompletableFuture<T>` ani jej pochodnych.

Wszelkie pytania i uwagi powinny być umieszczone na forum Moodle dedykowanym zadaniu.