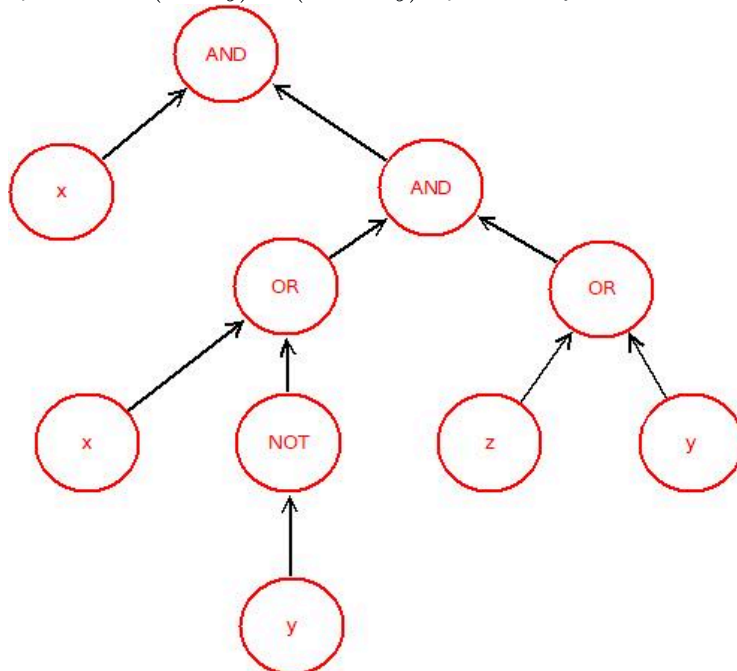


Big Java final assignment

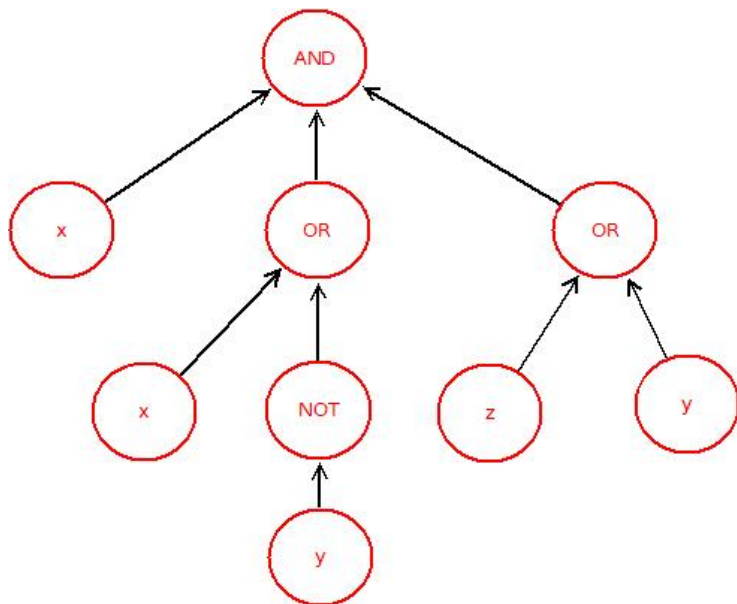
Due: Monday, 2 December 2024, 11:59 PM

Concurrent circuits

Boolean circuits represent Boolean expressions using graphs, e.g. the expression $x \wedge (x \vee \neg y) \wedge (z \vee y)$ can be represented by a tree



or, if we allow large arity operators, by a tree.



By convention, Boolean computations are evaluated from left to right, so in the

?

expression $x \wedge y$ first we calculate the value x and then the value y . The so-called lazy evaluation can skip the evaluation of some subexpressions if the value of the already evaluated ones allows to determine the value of the whole expression. For example, the expression $\text{true} \vee x$ doesn't have to calculate the value x to find out the value of the entire expression that is evaluated to true . Note that if the expressions do not generate side effects, the order of evaluation of subexpressions should not affect the value of the entire expression, so the evaluation of subexpressions can be done concurrently.

Your task is to implement a program that allows for the concurrent evaluation of Boolean expressions. The program should allow for the simultaneous evaluation of multiple expressions and the evaluation of single expressions concurrently.

Boolean expression

A boolean expression is defined inductively. Constant true and false are Boolean expressions. Negation of a boolean expression a , is a boolean expression. Conjunction $\text{AND}(a_1, a_2, \dots)$ and alternative $\text{OR}(a_1, a_2, \dots)$ a number of Boolean expressions (at least two) are Boolean expressions. The conditional statement $\text{IF}(a, b, c)$ is a boolean expression. Also threshold expressions $\text{GT}_x(a_1, a_2, \dots, a_n)$ and $\text{LT}_x(a_1, a_2, \dots, a_n)$ where $n \geq 1$ and $x \geq 0$ are integers, are boolean expressions.

Semantics

For the sake of expression a , $\text{By}[a]$ we denote the value of the expression a .

- $[\text{true}] = \text{true}$
- $[\text{false}] = \text{false}$
- $[\text{AND}(a_1, a_2, \dots, a_n)] = \text{true}$, if each expression a_i , $1 \leq i \leq n$, meets $[a_i] = \text{true}$, and $[\text{AND}(a_1, a_2, \dots, a_n)] = \text{false}$ otherwise.
- $[\text{OR}(a_1, a_2, \dots, a_n)] = \text{true}$, if there is an expression a_i , $1 \leq i \leq n$, meets $[a_i] = \text{true}$, and $[\text{OR}(a_1, a_2, \dots, a_n)] = \text{false}$ otherwise.
- $[\text{GT}_x(a_1, a_2, \dots, a_n)] = \text{true}$, if at least $x + 1$ expressions a_i , $1 \leq i \leq n$, meets $[a_i] = \text{true}$, and $[\text{GT}_x(a_1, a_2, \dots, a_n)] = \text{false}$ otherwise.
- $[\text{LT}_x(a_1, a_2, \dots, a_n)] = \text{true}$, if at most $x - 1$ expressions a_i , $1 \leq i \leq n$, meets $[a_i] = \text{true}$, and $[\text{LT}_x(a_1, a_2, \dots, a_n)] = \text{false}$ otherwise.
- $[\text{IF}(a, b, c)] = [b]$, if $[a] = \text{true}$, and $[\text{IF}(a, b, c)] = [c]$ otherwise.

Specification

In the solver, circuits are represented by class objects **Circuit**, and their values are computed by objects implementing the interface **CircuitSolver**, called solvers.

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

    public void stop();
}
```

where **CircuitValue** has the following interface.

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

The method `solve(Circuit c)` immediately returns a special object of the type `CircuitValue` representing the circuit value. This value can be retrieved by calling the method `CircuitValue.getValue()`, which waits until the value is calculated. Solvers should allow for concurrent handling of multiple requests (calls `solve()`) and for possibly concurrent calculation of the circuit value.

Calling the method `stop()` should cause the solver to stop accepting new requests `solve()` and immediately end all currently ongoing calculations of this solver. From this point on, objects `CircuitValue` resulting from new and interrupted calculations can return `InterruptedException` when we call `getValue()`. Other objects should return correctly calculated circuit values.

The class representing circuits `Circuit` and its auxiliary classes will be provided in the archive. The interface `Circuit` looks as follows. Expressions placed in the root field have a tree structure represented by classes that contain, among others, the following fields (the detailed interface is available in the archive placed in the Solution section). where is an enumeration type with a natural interpretation of symbols.

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

    public final CircuitNode getRoot();
}
```

```
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();
}
```

NodeType

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

Concurrency: liveness and security.

The program should allow for multiple simultaneous queries `solve()`. The results of the calls `solve()` should be returned as quickly as possible and the values of the calculated expressions do not have to be used in the order of the calls. Both leaf values and operator values should be computed concurrently. In particular, it should be assumed that calls to `LeafNode.getValue()` and `getArgs()` may compute for an arbitrarily long time, but they do not cause side effects and handle interrupts correctly. It can be assumed that external (i.e. not resulting from your implementation) interrupt calls, e.g. method calls `Thread.interrupt()`, will be limited to threads during method calls `CircuitValue.getValue()`.

In the solution, it can be assumed that each node in the tree structure of the expression is unique and that the sets of nodes of the tree structures of the circuits are pairwise disjoint. In particular, each call `solve()` receives a different

instance of `Circuit`. Additionally, in the implementation, it can be assumed that `CircuitSolver` the method `stop()`.

Solution

The solution is the [ab123456.zip](#) archive where

- `ab123456` has been replaced with student id,
- and in the package `cp2024.solution` there is an implementation of the solver.

Formal requirements

- The solution should be submitted by posting it in the appropriate place on Moodle.
- Package files `cp2024.solution` can be modified, and solution support files, such as new class definitions, can be placed in them. Changes in other packages (archive directories) will be ignored, in particular...
- Solutions will be run by copying files `*.java` from `cp2024/solution`, any remaining changes will be ignored (i.e. do not create subpackages or change interfaces).
- The solution must compile and run on the machine `students.mimuw.edu.pl` with the `src/` directory using the command `javac -d ../bin/ cp2024/*/*.java && java --class-path ../bin/ cp2024.demo.Demo` (after using `cp2024.solution.ParallelCircuitSolverW Demo.java`).
- The implementation should not write anything to standard output (`System.out` and `System.err`).
- Please do not use non-English characters (especially Polish characters) in your source files except for comments.
- Code style should be consistent and coherent. For example, you can adopt conventions from [Google Java Style Guide](#).
- The solution cannot use the class `java.util.concurrent.CompletableFuture<T>` or its derivatives.


All questions and comments should be posted on the Moodle [forum](#) dedicated to the task.

Last content change: 27/11/2024 18:30

Edit submission

Remove submission

Submission status

Submission status	Submitted for grading
Grading status	Not graded
Time remaining	Assignment was submitted 1 day 4 hours early
Last modified	Sunday, 1 December 2024, 7:36 PM
File submissions	<div><div> mk459179.zip</div><div>1 December 2024, 7:36 PM</div></div>