

Vorlesung Softwaretechnik I

Übung 6

SWT I – Sommersemester 2019

Walter F. Tichy, Sebastian Weigelt, Tobias Hey

IPD Tichy, Fakultät für Informatik



Aufgabe 1b: Parallelisierung von HDrize

```
public class CameraCurveParallel extends CameraCurve {
    private final Random random = new Random(42);
    private final int threads;
    // ...

    public CameraCurveParallel(EnhancedImage[] images, int samples, double lambda,
        IMatrixCalculator<Matrix> mtxCalc, int threads) {
        super(images, samples, lambda, mtxCalc);
        this.threads = threads;
    }

    @Override
    public void calculate() {
        this.calculate(3);
    }

    private void calculate(int remainingTries) {
        // siehe nächste Folie
    }
}
```

CameraCurve Sichtbarkeiten angepasst.
Alternativ: Quelltext duplizieren

Aufgabe 1b: Parallelisierung von HDrize

```
public class CameraCurveParallel extends CameraCurve {  
    // ...  
    private void calculate(int remainingTries) {  
        // Abbruchkriterien aus der ÜB2 ML  
        System.out.println("Try to calculate response curves ... ");  
  
        int imagewidth = this.images[0].getWidth();  
        int imageHeight = this.images[0].getHeight();  
        int[] x = new int[this.samples];  
        int[] y = new int[this.samples];  
        boolean[][] taken = new boolean[imageHeight][imagewidth];  
  
        for (int n = 0; n < this.samples; n++) {  
            x[n] = this.random.nextInt(imagewidth);  
            y[n] = this.random.nextInt(imageHeight);  
            if (taken[y[n]][x[n]]) {  
                n--;  
                continue;  
            }  
            taken[y[n]][x[n]] = true;  
        }  
        this.respCurves = new IMatrix[CameraCurve.CHANNELS];  
        // siehe nächste Folie  
    }  
}
```

Unveränderter Teil

Aufgabe 1b: Parallelisierung von HDrize

```
private void calculate(int remainingTries) {  
    // ...  
    if (this.threads <= 1) {  
        for (int channel = 0; channel < CameraCurve.CHANNELS; channel++) {  
            this.calculateChannel(channel, x, y);  
        }  
    } else if (this.threads == 2) {  
        Thread redAndGreen = new Thread(() -> {  
            this.calculateChannel(0, x, y);  
            this.calculateChannel(1, x, y);  
        });  
        Thread blue = new Thread(() -> this.calculateChannel(2, x, y));  
        redAndGreen.start();  
        blue.start();  
        // try-catch  
        redAndGreen.join();  
        blue.join();  
    } else {  
        // siehe nächste Folie  
    }  
}
```

Zu wenig Fäden → sequenziell

2 Fäden → Teile Kanäle auf

Aufgabe 1b: Parallelisierung von HDrize

```
private void calculate(int remainingTries) {  
    // ...  
} else {  
    Thread red = new Thread(() -> this.calculateChannel(0, x, y));  
    Thread green = new Thread(() -> this.calculateChannel(1, x, y));  
    Thread blue = new Thread(() -> this.calculateChannel(2, x, y));  
    red.start();  
    green.start();  
    blue.start();  
    // try-catch  
    red.join();  
    green.join();  
    blue.join();  
}  
for (IMatrix rc : this.respCurves) {  
    if (rc == null) {  
        this.respCurves = null;  
        this.calculate(remainingTries - 1);  
    }  
}  
}
```

Fäden $\geq 3 \rightarrow$ ein Faden pro Kanal

Versuche erneut falls nicht alles
berechnet

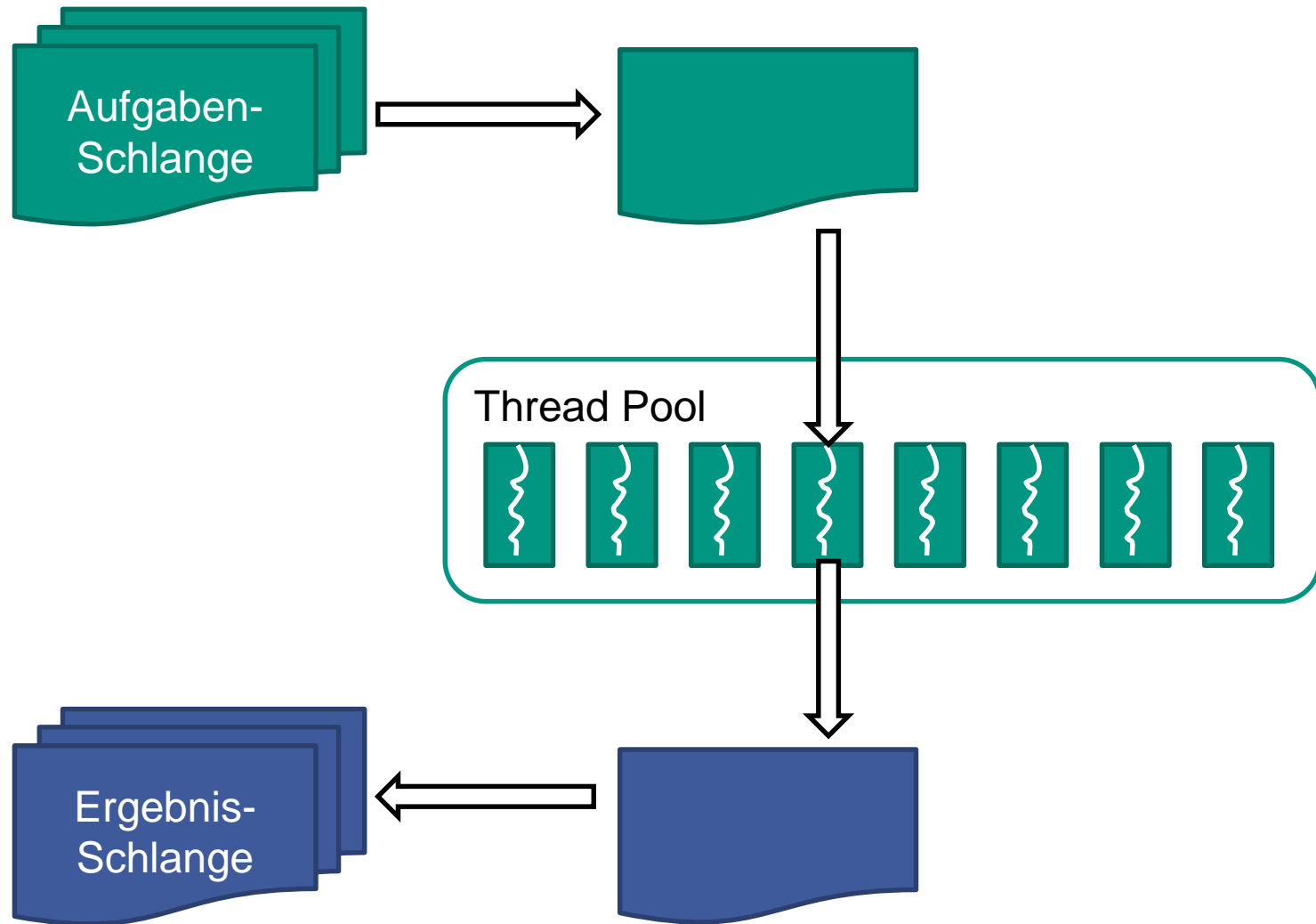
Aufgabe 1c: Parallelisierung von HDrize

```
public class MatrixCalculatorParallel extends MatrixCalculator {  
    private final ExecutorService threadPool;  
    private final int threads;  
  
    public MatrixCalculatorParallel(int threads) {  
        this.threads = threads;  
        if (threads <= 1) {  
            this.threadPool = null;  
  
        } else {  
            this.threadPool = Executors.newFixedThreadPool(threads);  
  
        }  
  
        // ... Nächste Folie  
    }  
}
```

Wenn die Anzahl der Fäden ≤ 1 ist, dann
lege keinen Thread Pool an

Thread Pool mit fester Größe erzeugen

Aufgabe 1c: Thread Pool



Aufgabe 1c: Parallelisierung von HDrize

```
public class MatrixCalculatorParallel extends MatrixCalculator {
    private final ExecutorService threadPool;
    private final int threads;

    @Override
    public Matrix multiply(Matrix a, Matrix b) {
        if (a.cols() != b.rows()) {
            return null;
        }
        double[][] mtxA = a.copy();
        double[][] mtxB = b.copy();
        double[][] mtxC = new double[a.rows()][b.cols()];

        int rows = mtxA.length;
        if (3 * rows <= this.threads || this.threads <= 1) {
            this.multiply(0, rows, mtxA, mtxB, mtxC);
        } else {
            // try-catch
            this.multiplyParallel(rows, mtxA, mtxB, mtxC);
            return new Matrix(mtxC);
        }
        // ...
    }
}
```

Führe parallele Berechnung nur bei
geeigneter Matrix Größe aus

Aufgabe 1c: Parallelisierung von HDrize

```
public class MatrixCalculatorParallel extends MatrixCalculator {  
    private final ExecutorService threadPool;  
    private final int threads;  
  
    private void multiplyParallel(int rows, double[][] mtxA, double[][] mtxB, double[][] mtxC)  
        throws InterruptedException {  
        List<Callable<Object>> allCallable = new ArrayList<>();  
        for (int i = 0; i < this.threads; i++) {  
            final int pos = i;  
            allCallable.add(Executors.callable(() -> this.multiply(  
                (rows * pos) / this.threads, (rows * (pos + 1)) / this.threads, mtxA, mtxB, mtxC)));  
        }  
        threadPool.invokeAll(allCallable);  
    }  
} // ... Nächste Folie
```

Implementiere parallele
Matrixmultiplikation in privater Methode

Rufe alle Fäden im Thread Pool auf

Aufgabe 1c: Parallelisierung von HDrize

```
public class MatrixCalculatorParallel extends MatrixCalculator {  
    private final ExecutorService threadPool;  
    private final int threads;  
  
    private void multiply(int iStart, int iEnd, double[][] mtxA, double[][] mtxB,  
        double[][] mtxC) {  
        for (int i = iStart; i < iEnd; i++) {  
            for (int j = 0; j < mtxB[0].length; j++) {  
                double sum = 0;  
                for (int k = 0; k < mtxB.length; k++) {  
                    sum = sum + mtxA[i][k] * mtxB[k][j];  
                }  
                mtxC[i][j] = sum;  
            }  
        }  
    }  
}
```



ijk-Algorithmus

Aufgabe 1d: Parallelisierung von HDrize

```
public class HDrizeParallel implements IHDrizeParallel {
    private final HDRCombineParallel combine;
    private final int threads;

    public HDrizeParallel() {
        this(Runtime.getRuntime().availableProcessors());
    }

    public HDrizeParallel(int threads) {
        this.combine = new HDRCombineParallel(threads);
        this.threads = threads;
    }

    // siehe nächste Folie
}
```

Aufgabe 1d: Parallelisierung von HDrize


```
public class HDrizeParallel implements IHDrizeParallel {
    private final HDRCombine combine;
    private final int threads;

    @Override
    public BufferedImage createRGB(InputStream[] images, int samples, double lambda) {
        Objects.requireNonNull(images, "images cannot be null");
        EnhancedImage[] enhanced = new EnhancedImage[images.length];
        for (int i = 0; i < enhanced.length; i++) {
            enhanced[i] = new EnhancedImage(images[i]);
        }

        ICameraCurve curve = this.createCurve(enhanced, samples, lambda,
            new MatrixCalculatorParallel(this.threads / 3));

        if (curve == null) {
            return null;
        }

        return HDRImageIO.createRGB(this.combine.createHDR(curve, enhanced),
            HDRImageIO.ToneMapping.SRGBGamma);
    }
}
```



Bestimme optimale
Zuteilung

Aufgabe 1d: Parallelisierung von HDrize

```
public class HDrizeParallel implements IHDrizeParallel {
    private final HDRCombine combine;
    private final int threads;

    private ICameraCurve createCurve(EnhancedImage[] images, int samples, double lambda,
        IMatrixCalculator<Matrix> mtxCalc) throws ImageReadException, IOException {
        if (lambda <= 0 || lambda > 100) {
            throw new IllegalArgumentException("Lambda has to be in (0,100]");
        }
        if (samples < 1 || samples > 1000) {
            throw new IllegalArgumentException("samples has to be in [1,1000]");
        }

        CameraCurveParallel cc = new CameraCurveParallel(images, samples, lambda, mtxCalc,
            this.threads);
        cc.calculate();
        return cc.isCalculated() ? cc : null;
    }
}
```

Nutze parallele
Kamerakurvenberechnung

Aufgabe 2: Parallelisierungswettbewerb

- Der Wettbewerb um den Titel „bester Parallelisierer des Monats“ bestimmt die beste Skalierbarkeit
- Unsere Testsystem
 - Intel Core i7-6700K CPU @ 4.00GHz(4 Kerne +HT)
 - 16GB RAM
 - Ubuntu 19.04 LTS (Disco Dingo)
- Gewinner
 - Muss noch bestimmt werden
 - Voraussetzung: Korrekte Implementierung
 - Siegerehrung: In der letzten Vorlesung

Aufgabe 3: Quelltext

```
public interface IKonto {  
    private BigDecimal kontostand;  
    //...  
  
    /**  
     * Hebt den spezifizierten Betrag ab, sofern der aktuelle Kontostand  
     * und der angegebene Kreditrahmen dies erlauben.  
     *  
     * @param betrag  
     *         der abzuhebende Betrag ( $\geq 0$ )  
     * @param kreditRahmen  
     *         der Kreditrahmen des aktuellen Kontos ( $\geq 0$ )  
     * @return wahr falls eine Abhebung durchgeführt werden konnte, unwahr sonst  
     */  
    public boolean hebeAb(BigDecimal betrag, BigDecimal kreditRahmen);  
    //...  
}
```

Aufgabe 3: Äquivalenzklassen

■ Eingabe:

- Direkt: betrag, kreditRahmen : BigDecimal

ÄK	Beschreibung	Mögliche Werte
$AK_{B,positiv}$	Gültige Eingabe von Betrag	$betrag \geq 0$
$AK_{B,negativ}$	Ungültige Eingabe von Betrag	$betrag < 0$
$AK_{KR,positiv}$	Gültige Eingabe für Kreditrahmen	$kreditRahmen \geq 0$
$AK_{KR,negativ}$	Ungültige Eingabe für Kreditrahmen	$kreditRahmen < 0$

Aufgabe 3: Äquivalenzklassen

■ Eingabe:

- Direkt: betrag, kreditRahmen : BigDecimal
- Indirekt (Zustand des Objekts): kontostand : BigDecimal

ÄK	Beschreibung	Mögliche Werte
$AK_{B,positiv}$	Gültige Eingabe von Betrag	$betrag \geq 0$
$AK_{B,negativ}$	Ungültige Eingabe von Betrag	$betrag < 0$
$AK_{KR,positiv}$	Gültige Eingabe für Kreditrahmen	$kreditRahmen \geq 0$
$AK_{KR,negativ}$	Ungültige Eingabe für Kreditrahmen	$kreditRahmen < 0$
$AK_{K,positiv}$	Kontostand ohne Kreditrahmen groß genug für Abhebung	$betrag \leq kontostand$
$AK_{K,negativ}$	Kontostand nur mit Kreditrahmen groß genug für Abhebung	$(betrag - kreditrahmen \leq kontostand) \wedge (kontostand < betrag)$
$AK_{K,Fehler}$	Kontostand mit Kreditrahmen zu gering für Abhebung	$kontostand < betrag - kreditrahmen$

Aufgabe 3: Äquivalenzklassen - Grenzwerte

■ Eingabe:

- Direkt: betrag, kreditRahmen : BigDecimal
- Indirekt (Zustand des Objekts): kontostand : BigDecimal

ÄK	Beschreibung	Mögliche Werte
$AK_{B,positiv}$	Gültige Eingabe von Betrag	0,00
$AK_{B,negativ}$	Ungültige Eingabe von Betrag	- 0,01
$AK_{KR,positiv}$	Gültige Eingabe für Kreditrahmen	0,00
$AK_{KR,negativ}$	Ungültige Eingabe für Kreditrahmen	- 0,01

$\pm \infty$ auch testen?
Nicht möglich!
BigDecimal wird
„beliebig“ groß.

Aufgabe 3: Äquivalenzklassen - Grenzwerte

■ Eingabe:

- Direkt: betrag, kreditRahmen : BigDecimal
- Indirekt (Zustand des Objekts): kontostand : BigDecimal

ÄK	Beschreibung	Mögliche Werte
$AK_{B,positiv}$	Gültige Eingabe von Betrag	0,00
$AK_{B,negativ}$	Ungültige Eingabe von Betrag	- 0,01
$AK_{KR,positiv}$	Gültige Eingabe für Kreditrahmen	0,00
$AK_{KR,negativ}$	Ungültige Eingabe für Kreditrahmen	- 0,01
$AK_{K,positiv}$	Kontostand ohne Kreditrahmen groß genug für Abhebung	betrag = kontostand -1, betrag = kontostand +1
$AK_{K,negativ}$	Kontostand nur mit Kreditrahmen groß genug für Abhebung	betrag-kreditrahmen = kontostand-1 \wedge kontostand < betrag, betrag-kreditrahmen = kontostand+1 \wedge kontostand < betrag
$AK_{K,Fehler}$	Kontostand mit Kreditrahmen zu gering für Abhebung	betrag-kreditrahmen = kontostand-1, betrag-kreditrahmen = kontostand+1

Konkrete Werte auch möglich, sofern korrekt.

Aufgabe 3: Äquivalenzklassen - Grenzwerte

- Grenzwerte für die letzten 3 ÄK fallen zusammen



$AK_{K,positiv}$	Kontostand ohne Kreditrahmen groß genug für Abhebung	$\text{betrag} = \text{kontostand} - 1$, $\text{betrag} = \text{kontostand} + 1$
$AK_{K,negativ}$	Kontostand nur mit Kreditrahmen groß genug für Abhebung	$\text{betrag-kreditrahmen} = \text{kontostand} - 1$ $\wedge \text{kontostand} < \text{betrag}$, $\text{betrag-kreditrahmen} = \text{kontostand} + 1$ $\wedge \text{kontostand} < \text{betrag}$
$AK_{K,Fehler}$	Kontostand mit Kreditrahmen zu gering für Abhebung	$\text{betrag-kreditrahmen} = \text{kontostand} - 1$, $\text{betrag-kreditrahmen} = \text{kontostand} + 1$

Aufgabe 4: KFG-orientiertes Testen

■ Vorgehen

- Quelltext in Zwischensprache überführen (for, foreach, while, ... auflösen)
- Dabei Zeilennummern übernehmen (für Sprungziele), ggf. ergänzen
- Ende der Blöcke zeichnen und Sprünge durch Pfeile ersetzen
- Prüfen: Gibt es Pfeile, die in die Mitte eines Blockes zeigen
→ davor aufteilen
- Prüfen: Gibt es leere Blöcke
→ mit Vorgänger/Nachfolger verschmelzen
- Prüfen: Gibt es unbedingte Sprünge (Pfeile)
→ Knoten verschmelzen

Aufgabe 4: KFG-orientiertes Testen – Quelltext

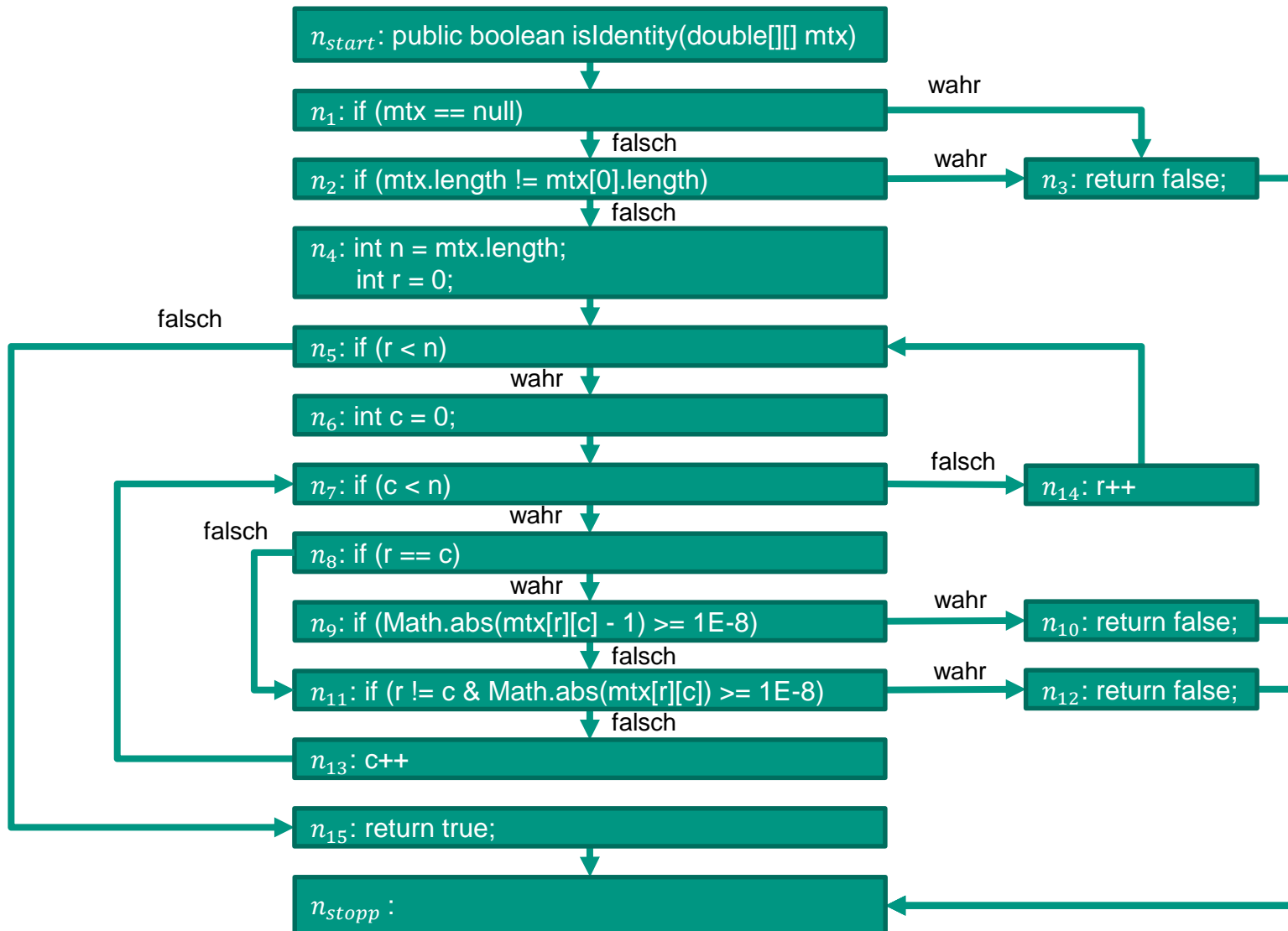
```
1. public boolean isIdentity(double[][] mtx) {
2.     if (mtx == null || mtx.length != mtx[0].length) {
3.         return false;
4.     }
5.     int n = mtx.length;
6.
7.     for (int r = 0; r < n; r++) {
8.         for (int c = 0; c < n; c++) {
9.             if (r == c && Math.abs(mtx[r][c] - 1) >= 1E-8) {
10.                return false;
11.            } else if (r != c & Math.abs(mtx[r][c]) >= 1E-8) {
12.                return false;
13.            }
14.        }
15.    }
16.    return true;
17.}
```

Aufgabe 4: KFG-orientiertes Testen – Zwischensprache

```
1. public boolean isIdentity(double[][] mtx) {  
2.     if (mtx == null) goto 4;  
3.     if not (mtx.length != mtx[0].length) goto 5;  
4.     return false;  
5.     int n = mtx.length;  
6.     int r = 0;  
7.     if not (r < n) goto 19;  
8.     int c = 0;  
9.     if not (c < n) goto 17;  
10.    if not (r == c) goto 13;  
11.    if not (Math.abs(mtx[r][c] - 1) >= 1E-8) goto 13;  
12.    return false;  
13.    if not (r != c & Math.abs(mtx[r][c]) >= 1E-8) goto 15;  
14.    return false;  
15.    c++;  
16.    goto 9;  
17.    r++;  
18.    goto 7;  
19.    return true;
```

Aufgabe 4: KFG-orientiertes Testen

Blöcke zeichnen



Aufgabe 4: KFG-orientiertes Testen

Anweisungsüberdeckung

■ Testfallmenge

$$P_{anw} = \{P_1(mtx = \{\{X\}, \{Y\}\}),$$

z.B. $\{\{0\}, \{0\}\}$

$$P_2(mtx = \{\{0\}\}),$$

$$P_3(mtx = \{\{1,1\}, \{X,Y\}\}),$$

z.B. $\{\{1,1\}, \{0,1\}\}$

$$P_4(mtx = \{\{1\}\})$$

Durchlaufene Pfade:

■ P_1 : Start $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow$ Stopp

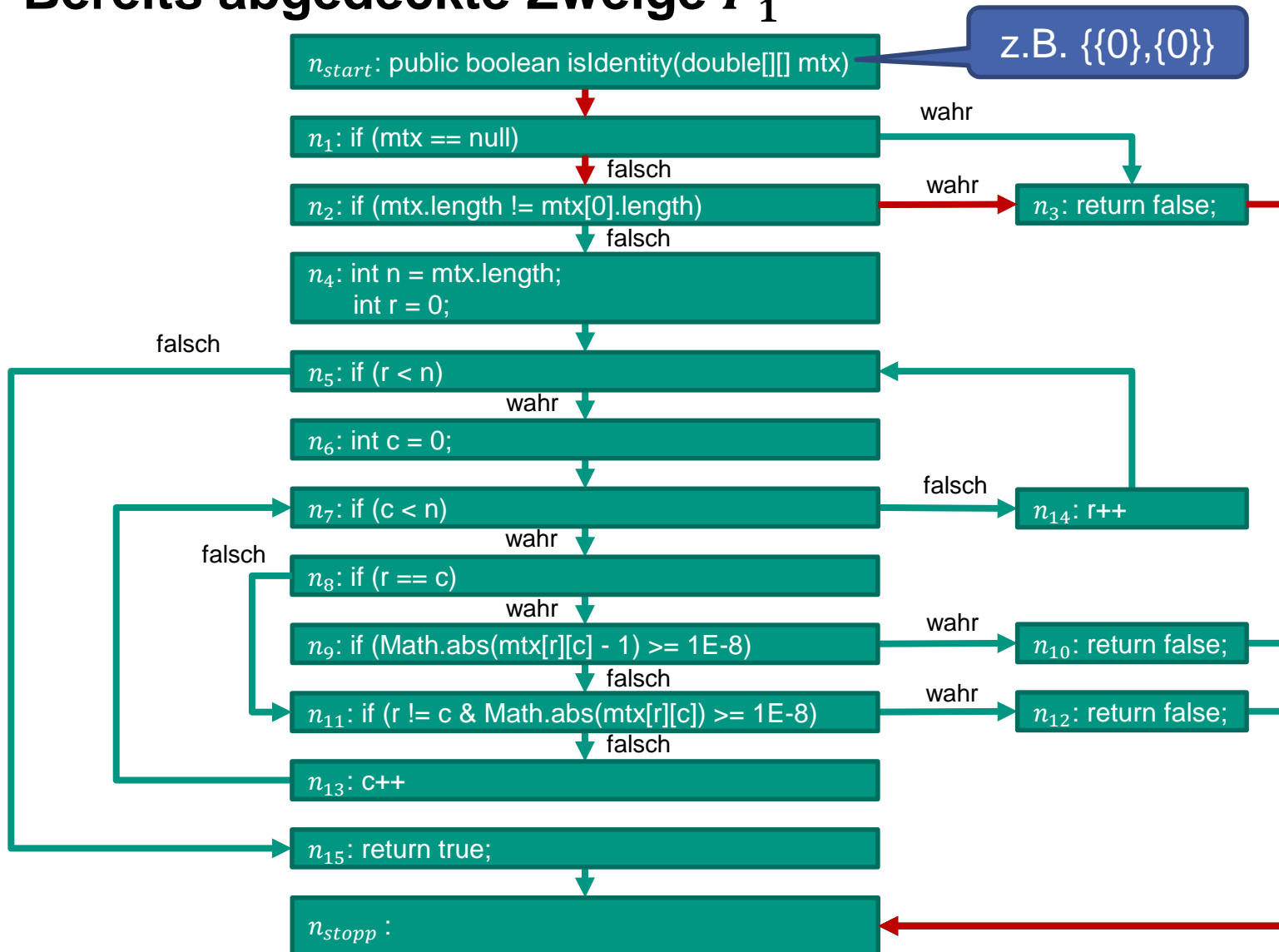
■ P_2 : Start $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow$ Stopp

■ P_3 : Start $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 11 \rightarrow 13 \rightarrow 7 \rightarrow 8 \rightarrow 9$
 $\rightarrow 11 \rightarrow 12 \rightarrow$ Stopp

■ P_4 : Start $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 11 \rightarrow 13 \rightarrow 7 \rightarrow 14 \rightarrow 5$
 $\rightarrow 15 \rightarrow$ Stopp

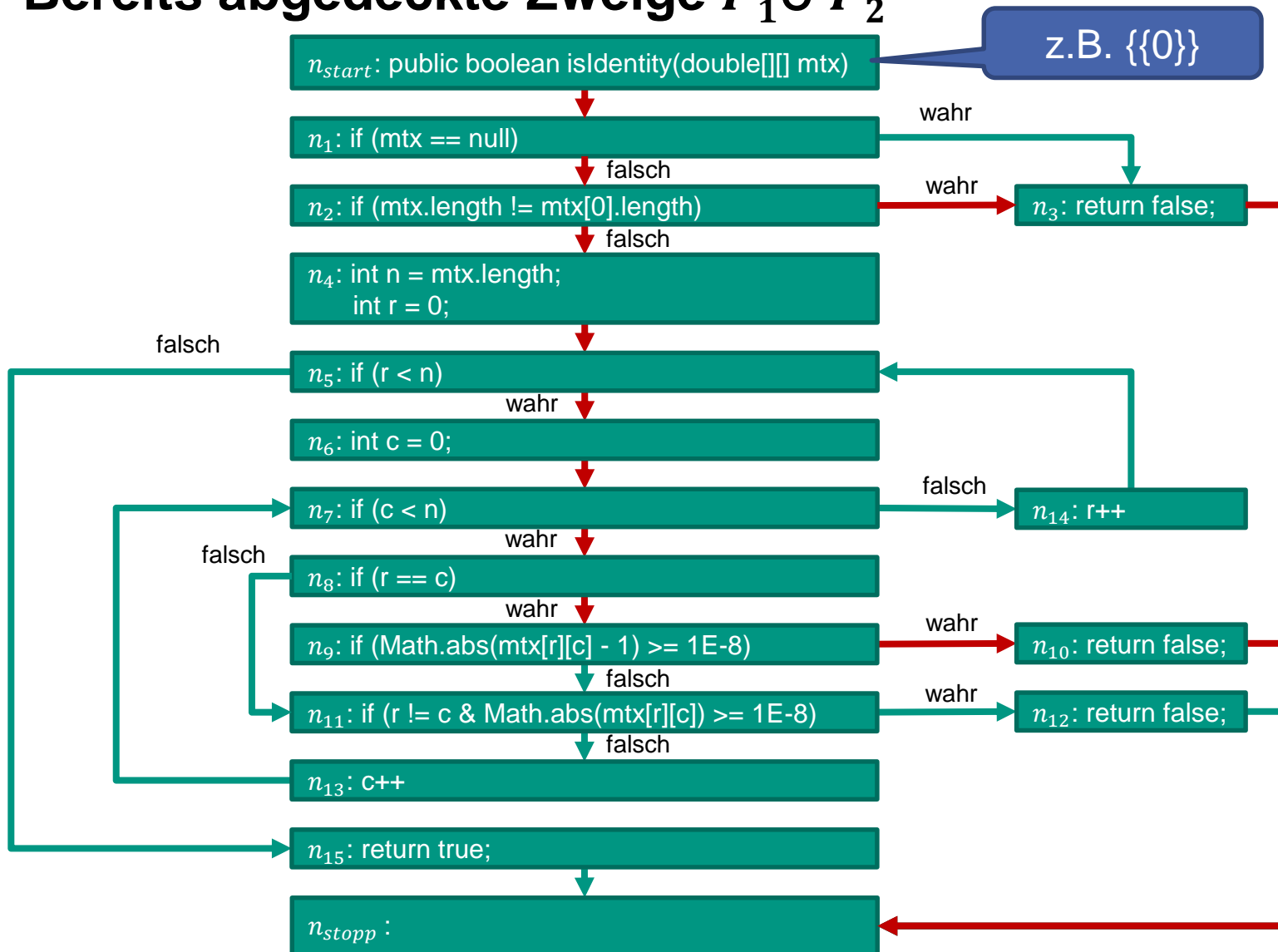
Aufgabe 4: KFG-orientiertes Testen

Bereits abgedeckte Zweige P_1



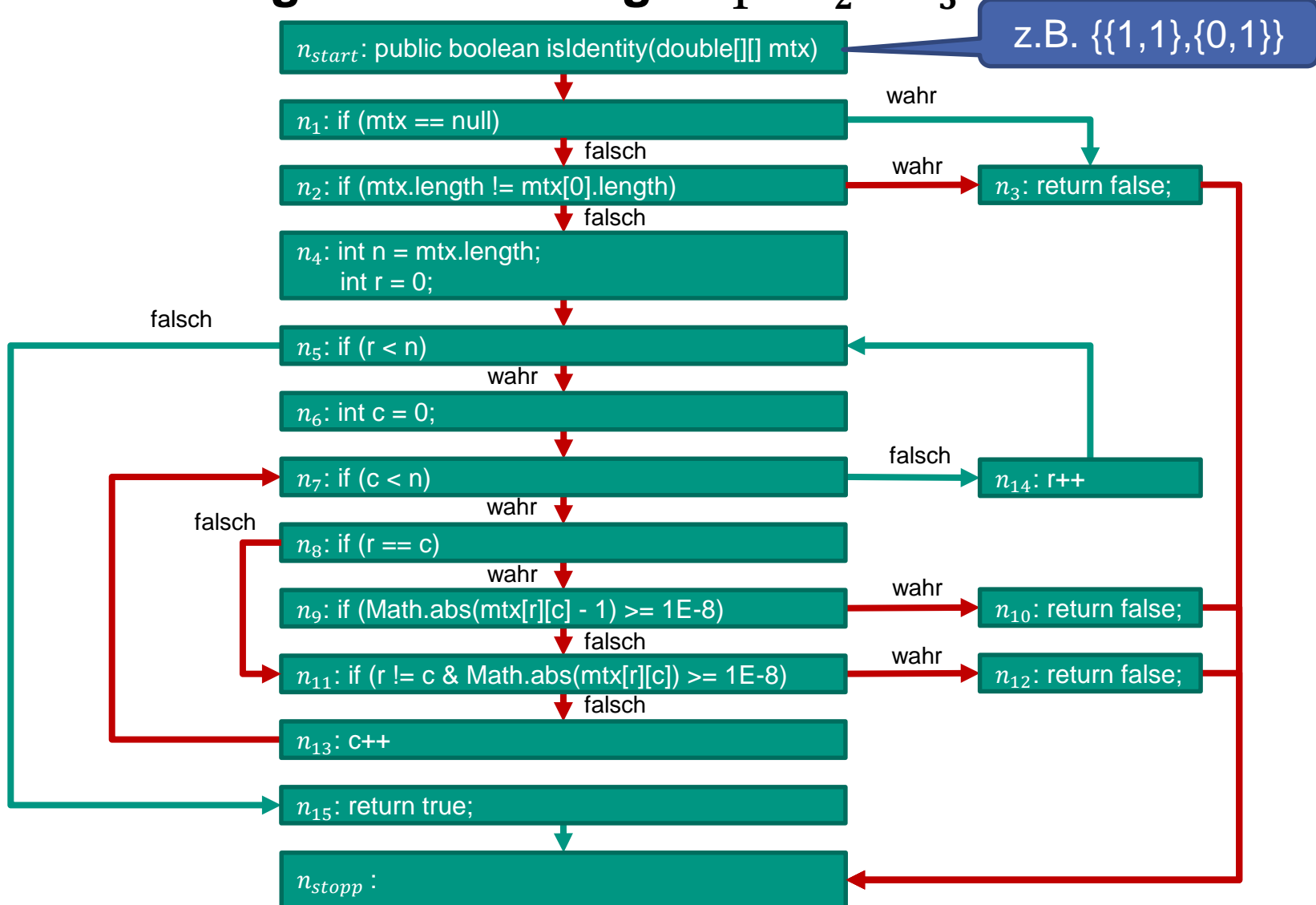
Aufgabe 4: KFG-orientiertes Testen

Bereits abgedeckte Zweige $P_1 \cup P_2$



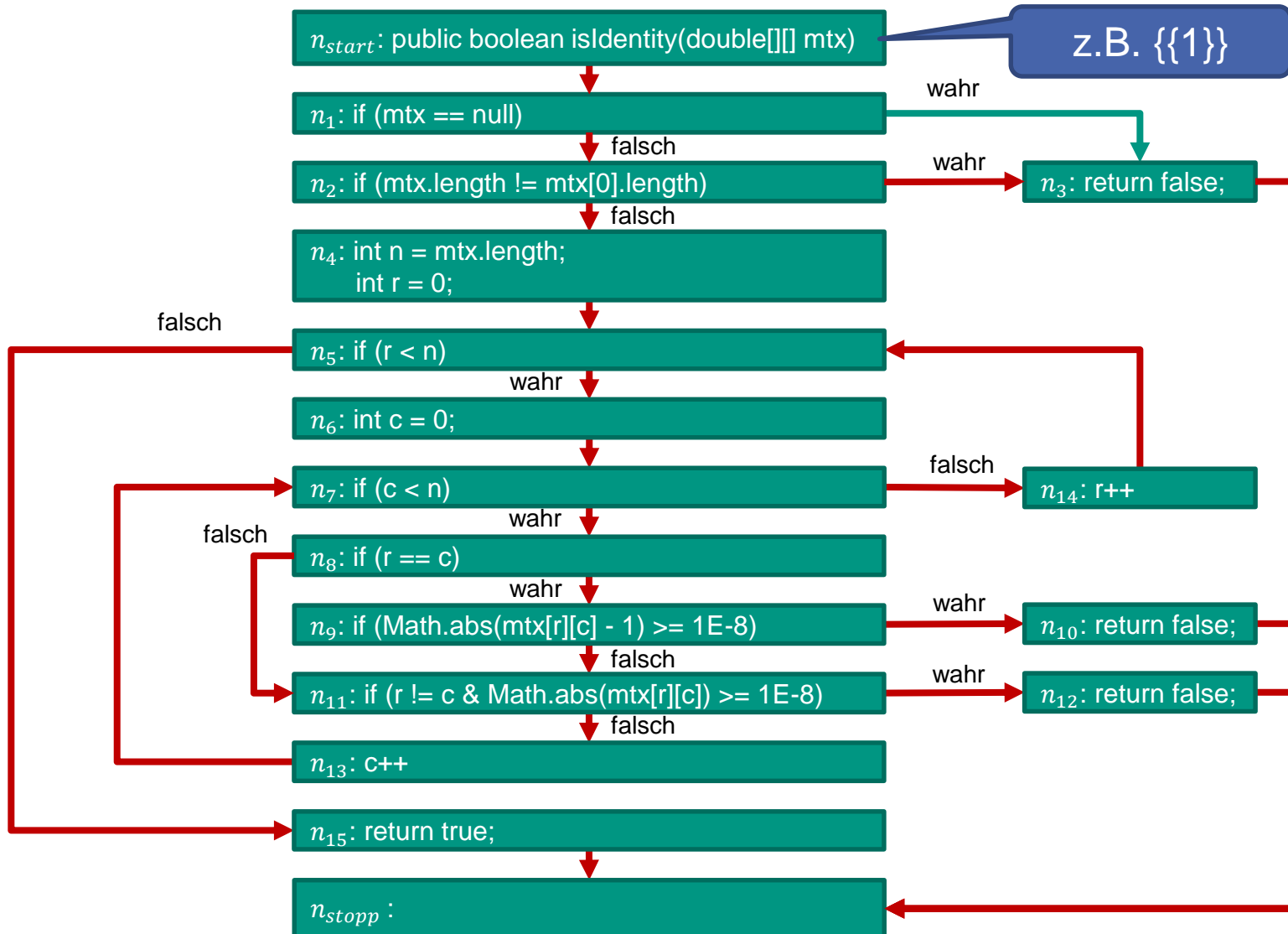
Aufgabe 4: KFG-orientiertes Testen

Bereits abgedeckte Zweige $P_1 \cup P_2 \cup P_3$



Aufgabe 4: KFG-orientiertes Testen

Bereits abgedeckte Zweige $P_1 \cup P_2 \cup P_3 \cup P_4$



Aufgabe 4: KFG-orientiertes Testen

Zweigüberdeckung

■ Testfallmenge

$$P_{\text{zweig}} = P_{\text{anw}} \cup \{P_5(mtx = null)\}$$

Durchlaufene Pfade:

■ P_5 : Start \rightarrow 1 \rightarrow 3 \rightarrow Stopp

Aufgabe 4: KFG-orientiertes Testen

Pfadüberdeckung (Bonus)

- Pfadüberdeckung bedeutet
 - Alle möglichen Pfade des Kontrollflusses werden betrachtet
 - Alle!

- Problem
 - Hängt die Anzahl an Pfaden von der Eingabe ab, kann sie unbeschränkt sein
 - (... oder nicht praktikabel hoch, bspw. `for (int r = 0; r < n; r++)` mit `n : int`)

- Folge: PÜ ist nicht erreichbar

Aufgabe 5: Codeinspektion

Indikator: Variablen- und Konstantendeklaration

- Regel 14. Gibt es lokale Variablen oder Instanz- und Klassenvariablen, die Konstanten sein sollten?

Wird aber nie benutzt ☹️

- z.B. `String klammerZu = "}"` sollte Konstante sein
- Regel 16. Haben alle Klassen- und Instanzvariablen die richtigen Zugriffsmodifizierer (default, private, protected, public)?
 - z.B. `final double[][] data;`
- Regel 18. Gibt es importierte Bibliotheken die nirgends verwendet werden?
 - z.B. `import org.ojalgo.matrix.store.PrimitiveDenseStore;`

Aufgabe 5: Codeinspektion

Indikator: Methoden- und Konstantendefinition

- Regel 20. Sind alle Namen sprechend gewählt und in Einklang mit der Java-Namenskonvention?
 - m, r, c, v
- Regel 21. Wird jeder Methodenparameter vor Gebrauch auf korrekte Eingabewerte geprüft?
 - Nur teilweise
- Regel 23. Haben alle Methoden und Konstruktoren die richtigen Zugriffsmodifizierer (default, private, protected, public)?
 - z.B. `private Matrix(double[][] m)`

Aufgabe 5: Codeinspektion

Indikator: Kommentare

- Regel 30. Haben alle Klassen sowie alle nicht-privaten Konstruktoren, Methoden, Konstanten, Klassen- und Instanzvariablen komplette JavaDoc-Kommentare?
 - Zwei Konstruktoren nicht

Indikator: Layout

- Regel 40. Ist die Einrückung korrekt und konsistent?
 - z.B. Einrückung der Getter-Methoden und der toString-Methode
- Regel 41. Haben alle Blöcke geschweifte Klammern?

```
35: for (int c = 0; c < this.cols; c++)
36:     this.set(r, c, m[r][c]);
```

Indikator: Leistung

- Regel 51. Wird jeder berechnete Wert auch verwendet?
 - `String klammerZu = "}"` wird nirgends verwendet

Aufgabe 5: Codeinspektion

Regelnummer	Zeilennummer(n)	Kurzbeschreibung
14	89	klammerZu könnte auch Konstante sein
16	15	data sollte private sein
18	4	org.ojalgo.matrix.store.PrimitiveDenseStore wird nie verwendet
20	21, 29, 76, 82	Methodennamen bzw. Parameternamen nicht sprechend
21	21, 29, 76, 82	Eingabeparameter werden nicht geprüft
23	29	Konstruktor sollte public sein
30	28, 45	Parameter fehlen in JavaDoc
40	68, 73, 92-104	Zeilen nicht korrekt eingerückt
41	35-36,104	for ohne { Klammern } und } zu viel
51	89	klammerZu wird nirgends verwendet