Technische Universität Berlin

Software and Embedded Systems Engineering Group
Prof. Dr. Sabine Glesner
www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2023/2024

Prof. Dr. Sabine Glesner
Julian Klein
Simon Schwan

Übungsaufgaben - Test 1a/1b

Dieser Aufgabenkatalog enthält Übungsaufgaben für den ersten Test von Softwaretechnik und Programmierparadigmen. Der Katalog ist kein Probetest und entspricht nicht dem üblichen Umfang eines Tests. Die Aufgaben dienen lediglich zur Übung und können sich in Art, Umfang und Thema von den Testaufgaben unterscheiden.

1. Codequalität und Testen

Angelehnt an einen Low-Pass-Filter ist die folgende Funktion `switchableLowPass(...)` gegeben, die lediglich Werte (`value`) unterhalb (und einschließlich) des `cutoff` durchlässt. Zusätzlich lässt sich der Filter explizit einschalten (`on`).

```
private int switchableLowPass(int value, int cutoff, boolean on) {
        int result = value;
        if (value > cutoff
                && on) {
                        result = cutoff;
        }
        return result;
}
```

a) Zeichnen Sie den Kontrollflussgraphen der gegebenen Funktion. Knoten müssen eindeutig anhand der zugehörigen Zeilennummer benannt werden.

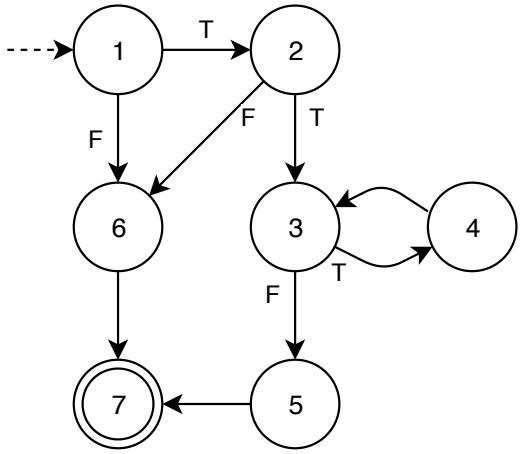
b) Bilden Sie die Äquivalenzklassen für die Eingabeparameter `value` und `on`.

value:

on:

c) Bestimmen Sie eine Menge an Testfällen mit minimaler Anzahl, um vollständige Zweigüberdeckung für das o.g. Programm zu erreichen. Testfälle müssen in folgendem Format aufgeschrieben werden:
$(value, cutoff, on) \rightarrow result$

d) Berechnen Sie für den gegebenen Kontrollflussgraphen die zyklomatische Komplexität nach McCabe. Es kann eine Anzahl von Komponenten $p = 1$ angenommen werden.



$v(G) =$

2. Codequalität und Testen

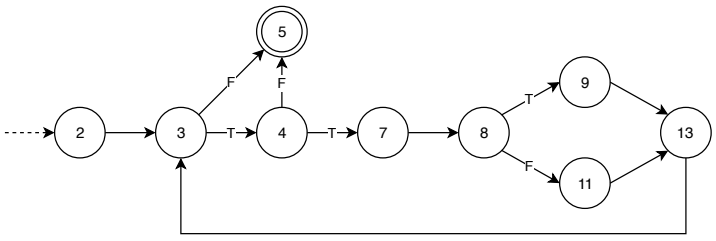
Gegeben ist die folgende Funktion `collatz(...)`, welche die Collatz-Folge ohne Zyklen berechnet. Dabei wird die Folge ausgehend vom letzten Element der übergebenen Liste `values` berechnet. Die Vermutung des Collatz-Problems ist, dass jede natürliche Zahl $n > 0$ in einem Zyklus der Zahlenfolge $4 \to 2 \to 1$ endet, wenn nachfolgende Zahlen wie folgt berechnet werden:

$$C(n) = \begin{cases} n/2 & \text{wenn } n \text{ gerade ist,} \\ 3n+1 & \text{wenn } n \text{ ungerade ist.} \end{cases}$$

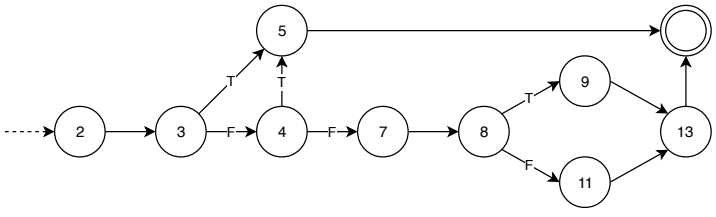
Beispiel für $n = 5$ (ohne Zyklen): $5 \to 16 \to 8 \to 4 \to 2 \to 1$

```
private void collatz(List<Integer> values) {
        int size = values.size();
        if (size == 0 ||
                values.get(size-1) <= 1) {
                return;
        } else {
                int last = values.get(size-1);
                if (last % 2 == 0) {
                        values.add(last/2);
                } else {
                        values.add(3*last + 1);
                }
                collatz(values);
        }
}
```

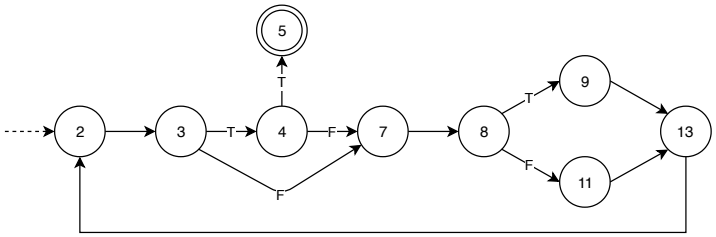
a) Identifizieren Sie den Kontrollflussgraphen der gegebenen Funktion `collatz(...)`. Kreuzen Sie die korrekte Antwort an!



☐



☐



☐

b) Berechnen Sie die zyklomatische Komplexität nach McCabe für den von Ihnen ausgewählten Kontrollflussgraphen aus Aufgabe 1.a). Nehmen Sie hierzu für das Programm eine Anzahl von Komponenten $p = 1$ an.

c) Bestimmen Sie eine Menge an Testfällen mit minimaler Anzahl, um vollständige Zweigüberdeckung für das o.g. Programm `collatz(...)` zu erreichen. Testfälle müssen in Form von Listen angegeben werden, die als Eingabewert für den `values`-Parameter der Funktion interpretiert werden.

d) Gegeben sind folgende Aussagen zum Thema Codequalität. Kreuzen Sie an!

	wahr	**falsch**
Refactoring ist ein Optimierungsprozess, bei dem rekursive Programme in iterative Programme überführt werden ohne eine Veränderung des Verhaltens.	☐	☐
Die *Halstead*-Metrik ist eine dynamische Messung des Fortschritts des Software-Entwicklungsprozesses.	☐	☐

3. Logische Programmierung

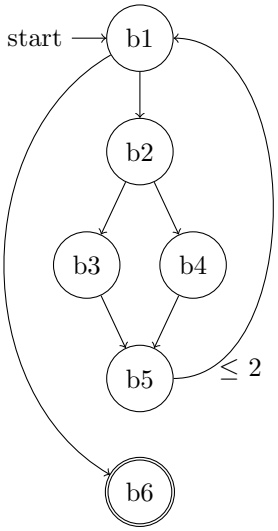
Gegeben ist ein Kontrollflussgraph in Form einer Datenbasis in Prolog. Er hat das gleiche Format wie das unten gegebene Beispiel. Zusätzlich enthält die Datenbasis *constraints*, mit denen z.B. die maximale Anzahl Iterationen einer Schleife angegeben werden kann.

Erstellen Sie die Prädikate so, dass sie korrekte Belegungen für alle Kontrollflussgraphen vom gegebenen Format finden. Außerdem dürfen Hilfsprädikate geschrieben und verwendet werden sowie Prädikate der anderen Teilaufgaben, auch wenn sie selbst nicht gelöst wurden. Vordefinierte Prädikate dürfen nicht verwendet werden.

```
start(b1).
end(b6).

edge(b1,b2).
edge(b1,b6).
edge(b2,b3).
edge(b2,b4).
edge(b3,b5).
edge(b4,b5).
edge(b5,b1).

constraint(b5,2).
```



a) Entscheiden Sie für folgende Queries, ob dafür mit der im Beispiel gegebenen Datenbasis Belegungen gefunden werden können (**true**), keine Belegungen gefunden werden (**false**) oder ein **Fehler** ausgegeben wird. Es ist nur eine Antwort richtig, mehr oder weniger Kreuze werden nicht bewertet.

Query	true	false	Fehler
`?- edge(b7,b8) = edge(b7,_).`	☐	☐	☐
`?- A=(constraint(b5,L),L>5).`	☐	☐	☐
`?- 5+6 is 11.`	☐	☐	☐
`?- constraint(b5,<=4).`	☐	☐	☐

b) Das Prädikat `node(X)` soll prüfen, ob `X` ein Knoten im Graph ist, d.h. ob er ein Start- oder Endknoten ist, oder ob er am Anfang oder am Ende einer Kante steht.
Beispiel-Query: `?- node(b4).`
Ausgabe: `true.`

```
node(X) :- ____________________________________________________________________.
```

c) Das Prädikat `isEndNode(X)` soll prüfen, ob `X` ein Knoten im Graph ist, der keine ausgehenden Kanten hat und als Endknoten gekennzeichnet ist.
Beispiel-Query: `?- isEndNode(b6).`
Ausgabe: `true.`

```
isEndNode(X) :- ____________________________________________________________________.
```

d) Mit dem Prädikat `count` soll ermittelt werden, wie oft `X` in der Liste `L` vorkommt.
Beispiel-Query: `?- count(3,[1,2,3,3,3,2,4],C).`
Ausgabe: `C = 3.`

e) Mit dem Prädikat `path(X,Y,V,P)` sollen nun Pfade ermittelt werden, die von einem Knoten `X` zu `Y` gehen. Dabei sollen Zyklen nicht ganz vermieden werden - stattdessen soll mithilfe von den Loop-Bounds `constraint(K,N)` jeder Knoten `K` nur höchstens `N` mal im Pfad auftauchen, falls dies entsprechend eingeschränkt ist. Die ermittelten Pfade sollten vollständig und in der richtigen Reihenfolge sein. Zur Vereinfachung darf davon ausgegangen werden, dass immer `X` ein Startknoten und `Y` ein Endknoten ist.
Beispiel-Query/Ausgabe:

```
?- path(b1,b6,[],P).
P = [b1, b6] ;
P = [b1, b2, b3, b5, b1, b6] ;
P = [b1, b2, b3, b5, b1, b2, b3, b5, b1|...] ;
P = [b1, b2, b3, b5, b1, b2, b4, b5, b1|...] ;
P = [b1, b2, b4, b5, b1, b6]
....
```

4. Logische Programmierung

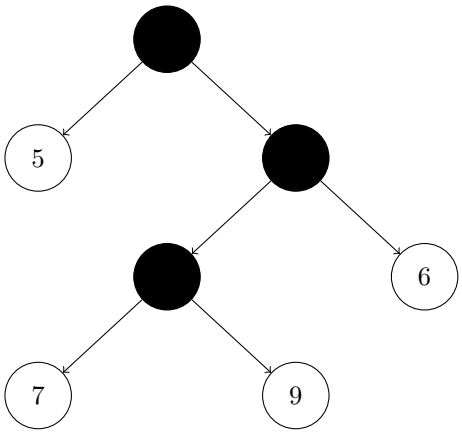
Gegeben ist ein **vollständiger Binärbaum** in Form einer Datenbasis in Prolog. Er hat das gleiche Format wie das nachfolgende Beispiel. Die nachfolgenden Prädikate müssen in einer **allgemeingültigen Form** gelöst werden, d.h. die Beispielanfragen sollen auch für andere vollständige Binärbäume im gegebenen Format korrekte Ergebnisse liefern. Prädikate aus anderen Teilaufgaben dürfen verwendet werden, auch ohne dass sie selbst korrekt gelöst sind. **Vordefinierte Prädikate dürfen nicht verwendet werden.**

```
node(n1).
node(n2).
node(n3).

leaf(l1,5).
leaf(l2,6).
leaf(l3,7).
leaf(l4,9).

left(n1,l1).
left(n2,n3).
left(n3,l3).

right(n1,n2).
right(n2,l2).
right(n3,l4).
```



a) Entscheiden Sie für folgende Queries, ob dafür mit der im Beispiel gegebenen Datenbasis Belegungen gefunden werden können (**true**), oder ob **false** oder ein **Fehler** ausgegeben wird. Es ist nur eine Antwort richtig, mehr oder weniger Kreuze werden nicht bewertet.

Query	true	false	Fehler
`?- leaf(N4,N5).`	☐	☐	☐
`?- leaf(l1,X) = leaf(X,5).`	☐	☐	☐
`?- left(n2) = leaf(X) ; X=l1.`	☐	☐	☐
`?- left(n2,left(n3,X)).`	☐	☐	☐

b) Das Prädikat `child(X,Y)` soll prüfen, ob X ein Elternknoten von Y im Baum ist.
Beispiel-Query: `?- child(n1,l1).`
Ausgabe: `true.`

```
child(X,Y) :- ____________________________________________________________.
```

c) Das Prädikat `root(X)` soll prüfen, ob X die Wurzel des Baums, d.h. ein Knoten im Baum ohne Elternknoten, ist. (1P)
Beispiel-Query: `?- root(n2).`
Ausgabe: `false.`

```
root(X) :- ____________________________________________________________.
```

d) In einer gültigen Belegung für das Prädikat `mergeSorted(L1,L2,L)` erhält L alle Elemente von L1 und L2, so dass die Elemente aufsteigend sortiert sind. Es ist davon auszugehen, dass L1 und L2 Zahlen enthalten und selbst jeweils aufsteigend sortiert sind.
Beispiel-Query: `?- mergeSorted([1,2,6,9],[2,3,7],L).`
Ausgabe: `L = [1, 2, 2, 3, 6, 7, 9] .`

e) Mit dem Prädikat `toList(X,L)` soll ein Teilbaum mit der Wurzel `X` in eine sortierte Liste umgeformt werden, die alle Blattwerte enthält.
Beispiel-Query: `?- toList(n2,L).`
Ausgabe: `L = [6, 7, 9] .`