**Beschreibung:**

Der Operator == untersucht, falls zwei Matrix gleich sind (falls sie die selben Elemente haben – nicht unbedingt auf die selben Positionen)

**Pre:** m ist ein Matrix, other ist ein anderes Matrix

**Post:** true – falls m = other, false andernfalls

**Pseudocode:**

function **operator ==**(m, other) **is:**

if m.nrLines != other.nrLines or m.nrCols != other.nrCols then  
 operator== <- false

end-if

for k<-0, m.m execute

found <- false  
p <- 0  
pos <- other.h(other, [m.T[k]].i \* 10 + [m.T[k]].j, p)

while p < other.m and [other.T[pos]].key != NIL and found != true execute  
 if [other.T[pos]].i = [m.T[k]].i and [other.T[pos]].j = [m.T[k]].i and [other.T[pos]].key = [m.T[k]].key then

found <- true  
 end-if

p <- p + 1  
 pos <- other.h(m, [m.T[k]].i \* 10 + [m.T[k]].j, p)  
end-while

if found != true then

operator== <- false

end-if

end-for

for k<-0, other.m execute

found <- false  
p <- 0  
pos <- m.h(other, [other.T[k]].i \* 10 + [other.T[k]].j, p)

while p < m.m and [m.T[pos]].key != NIL and found != true execute  
 if [m.T[pos]].i = [other.T[k]].i and [m.T[pos]].j = [other.T[k]].i and [m.T[pos]].key = [other.T[k]].key then

found <- true  
 end-if

p <- p + 1  
 pos <- m.h(m, [other.T[k]].i \* 10 + [other.T[k]].j, p)  
end-while

if found != true then

operator== <- false

end-if

end-for

end-function

**Komplexitaten:**

**Best Case: Θ(1)** – falls nrLines / nrCols sind verschieden oder das erste Element aus m existiert nicht in other Matrix

**Average Case: Θ(max(m.m, other.m))**

**Worst Case: Θ(max(m.m, other.m))** – falls m = other oder das letzte Element aus m / other verschieden zu den anderen ist

**Allgemeine Komplexitat: O(max(m.m, other.m))**