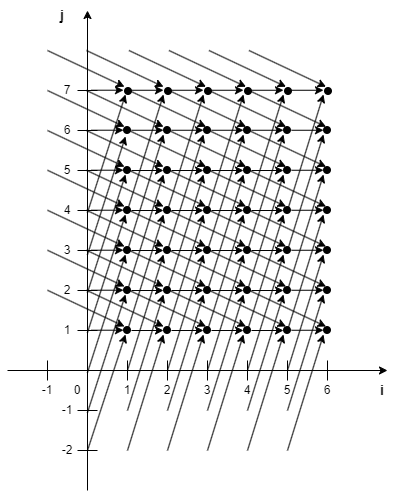
FOR i = 1, 6 DO

FOR j = 1, 7 DO (5)

{ x(i, j) := x(i - 1, j) + x(i - 1, j - 3) + x(i - 2, j + 1) }.



*FOR k* = 1, 7 *DO AUTON*

*FOR* *i* = 1, 6 *DO*

*FOR* *j* = *k*, *k* + [(6 - *i*) / 2] *DO*  (6)

{ *x*(*i*, *j*) := *x*(*i* - 1, *j*) + *x*(*i* - 1, *j* - 3) + *x*(*i* - 2, *j* + 1) }.

В наведеному тут вигляді поки що не є еквівалентною до (5). Крім цього, з ме­тою технічних спрощень ми вважаємо, що ітерації піра­мід, які не потрап­ляють в межі циклу, заміняються деякими фіктивними іте­раціями, що не змі­нюють значе­н­ня пам'яті. В нашому випадку такими є ітерації (1, 8), (2, 8) при *k =* 6, ітерації (1, 8), (1, 9), (2, 8), (2, 9), (3, 8), (4, 8) при *k =* 7*.* Можна вважати, що вони ма­­­ють вигляд:

*x*(*i*, *j*) := *x*(*i*, *j*).

Найбільш розповсюдженими є ОККАМ-подібні оператори: x ↑ k – пересла­ти змінну *x* у гілку (піраміду) з номером *k*, x ↓ k – зчитати змінну *x* із гілки з но­ме­ром *k*. Семантика цих операторів, якщо їх розгляда­ти кожен окремо, досить про­ста. Як тільки у гілку з номером *l* контроль до­ходить до оператора x ↑ k, зна­чен­ня змінної *x* присвоюється одноіменній змінній у гілці з номером *k*. Якщо в цій гілці змінна *x* досі не використовува­лась ні для читання, ні для запису, то вона за­водиться як вхідна. Це є екві­ва­лентним використанню оператора: ввід (*x*). Анало­гічно, спрацьовування x ↓ k полягає в зчитуванні в комірку *x* значення із одноіме­н­ної комірки гіл­ки з номером *k*.

Конструкцію можна переписати у вигляді:

{ *x*(*i*, *j*) ↑ (*k* + 3) = *x*(*i* – 1, *j*) + *x*(*i* – 1, *j* – 3) ↓ (*k* – 3) + *x*(*i* – 2, *j* + 1) }.