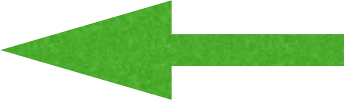
```clojure
(defn pattern-left-join
  [graph partial-result pattern]
  (let [cols (:cols (meta partial-result))
        total-cols (calc-new-columns cols pattern)
        pattern->left (matching-vars pattern cols)]

    ;; iterate over partial-result, lookup pattern
    (with-meta
      (for [left-row partial-result
                   ;; convert bindings in left-row into the
                   ;; pattern to lookup in the graph
            :let [lookup (modify-pattern left-row
                                             pattern->left
                                             pattern)]
            right-row (gr/resolve-pattern graph lookup)]
        (concat left-row right-row))
      {:cols total-cols})))
```



```
(defn pattern-left-join
  [graph partial-result pattern]
  (let [cols (:cols (meta partial-result))
        total-cols (calc-new-columns cols pattern)
        pattern->left (matching-vars pattern cols)]

    ;; iterate over partial-result, lookup pattern
    (with-meta
      (for [left-row partial-result
                 ;; convert bindings in left-row into the
                 ;; pattern to lookup in the graph
            :let [lookup (modify-pattern left-row
                                         pattern->left
                                         pattern)]
            right-row (gr/resolve-pattern graph lookup)]
        (concat left-row right-row))
      {:cols total-cols})))
```

```
def patternLeftJoin(graph, partialResult, pattern):
  cols, leftData = partialResult
  totalCols = calNewColumns(cols, pattern)
  patternToLeft = matchingVarMapping(pattern, cols)
  result = []
  for leftRow in leftData:
    lookup = modifyPattern(leftRow, patternToLeft, pattern)
    for rightRow in graph.resolvePattern(lookup):
      result.append(leftRow + rightRow)
  return (totalCols, result)
```