```
(defun add-expression-p (x)
  (and (equal (len x) 3)
          (equal (first x) 'add)
          (variable-or-numberp (second x))
          (variable-or-numberp (third x))))

(defun phi-expression-p (x)
  (and (consp x) (equal (len x) 1)
          (consp (car x)) (> (len (car x)) 2)
          (equal (caar x) 'phi) (phi-l (cdr (car x)))))

(defun phi-statement-p (x)
  (and (consp x) (equal (len x) 2)
          (symbolp (first x)) (first x)
          (phi-expression-p (cdr x))))

(defun evaluate-val (val bindings)
   (if (symbolp val)
        (cdr (assoc-equal val bindings)) val))

(defun run-ccdfg (pre loop post iterations init-state prev)
(let* ((state1 (run-block-set pre init-state nil prev))
         (state2 (run-blocks-iters loop state1 iterations (prefix loop)))
         (state3 (run-block-set post state2 nil (prefix post)))
state3))
```

```
(defun expression-p (x)
  (and (consp x)
          (or (load-expression-p x)
               (add-expression-p x)
               (xor-expression-p x)...)))

(defun assignment-statement-p (x)
  (and (equal (len x) 1)
          (and (equal (len (car x)) 2)
                  (first (car x)) (symbolp (first (car x)))
                  (expression-p (second (car x))))))

(defun choose (choices prev-bb)
  (if (or (equal (nth 1 (first choices)) prev-bb)
            (equal (symbol-name (nth 1 (first choices))) prev-bb))
                 (nth 0 (first choices))
                 (nth 0 (second choices))))
(defun execute-phi (stmt init-state prev-bb)
  (let* ((expr (cdr stmt))
            (var (first stmt))
            (val (evaluate-val (choose (cdr (car expr)) prev-bb)
                                           (car init-state))))
       (list (replace-var var val (variables-of init-state))
              (memory-of init-state)
              (pointers-of init-state))))
```