Template for arbitrary arity

```
(define (fn-for-element e)
  (local [(define (fn-for-element e)
            (... (fn-for-loe (element-subs e))))
          (define (fn-for-loe loe)
            (cond [(empty? loe) (...)]
                  [else
                   (... (fn-for-element (first loe))
                        (fn-for-loe (rest loe)))]))]
    (fn-for-element e)))
```

Template for arbitrary sized tree of boards

```
(define (solve bd)
  (local [(define (solve--board bd)
            (...
             (solve-lobd (board-subs bd))))
          (define (solve-lobd lobd)
            (cond [(empty? lobd) (...)]
                  [else
                   (... (solve—board (first lobd))
                        (solve-lobd (rest lobd)))]))]
    (solve-bd bd)))
```

Template for generative recursion

```
(define (solve p)
  (local [(define (genrec-fn p)
            (cond [(trivial? p) (trivial-answer p)]
                  [else
                   (... p
                        (genrec-fn (next-problem p)))]))]
    (genrec-fn p)))
```

```
(define (solve bd)
  (local [(define (solve-bd bd)
            (cond [(trivial? bd) (trivial-answer bd)]
                  [else
                   (... bd
                   (solve-bd (next-problems bd)))]))]
    (solve-bd bd)))
```

```
(define (solve bd)
  (local [(define (solve-bd bd)
            (cond [(trivial? bd) (trivial-answer bd)]
                  [else
                   (... bd
                        (solve-lobd (next-problems bd)))]))
          (define (solve-lobd lobd)
            (cond [(empty? lobd) (...)]
                  [else
                   (... (solve—bd (first lobd))
                        (solve-lobd (rest lobd)))]))]
    (solve-bd bd)))
```

blended templates…

```
(define (solve bd)
  (local [(define (solve-bd bd)
            (cond [(trivial? bd) (trivial-answer bd)]
                  [else
                   (... bd
                        (solve-lobd (next-problems bd)))]))
          (define (solve-lobd lobd)
            (cond [(empty? lobd) (...)]
                  [else
                   (... (solve—bd (first lobd))
                        (solve-lobd (rest lobd)))]))]
    (solve-bd bd)))
```

with backtracking…

```
(define (solve bd)
  (local [(define (solve-bd bd)
            (cond [(trivial? bd) (trivial-answer bd)]
                  [else
                   (... bd
                   (solve-lobd (next-problems bd)))]))
          (define (solve-lobd lobd)
            (cond [(empty? lobd) false]
                  [else
                   (local [(define try (solve—bd (first lobd)))]
                     (if (not (false? try))
                         try
                         (solve-lobd (rest lobd))))]))]
    (solve-bd bd)))
```