EVENT: Start with the library `"interpreter"`.

DEFINITION: $\text{value}(alist, key) = \text{cdr}(\text{assoc}(key, alist))$

DEFINITION: $\text{ct}(i) = \text{cons}(\texttt{'ct}, i)$

DEFINITION: $\text{cf}(i) = \text{cons}(\texttt{'cf}, i)$

DEFINITION: $\text{temp}(i) = \text{cons}(\texttt{'temp}, i)$

DEFINITION:
c-element (old, new, a, b, c)
= **if** value (old, a) ↔ value (old, b)
  then (value (new, c) ↔ value (old, a)) ∧ changed (old, new, list (c))
  else changed (old, new, **nil**) **endif**

DEFINITION:
nor-gate (old, new, a, b, c)
= ((value (new, c) ↔ (¬ (value (old, a) ∨ value (old, b))))
  ∧ changed (old, new, list (c)))

DEFINITION:
fifo-node (i)
= list (list (`'c-element`, ct ($1 + i$), temp (i), ct (i)),
    list (`'c-element`, cf ($1 + i$), temp (i), cf (i)),
    list (`'nor-gate`, ct ($i - 1$), cf ($i - 1$), temp (i)))

DEFINITION:
empty-node ($state$, i)
= ((¬ value ($state$, ct (i))) ∧ (¬ value ($state$, cf (i))))

DEFINITION:
true-node ($state$, i) = (value ($state$, ct (i)) ∧ (¬ value ($state$, cf (i))))

DEFINITION:
false-node ($state$, i) = ((¬ value ($state$, ct (i))) ∧ value ($state$, cf (i)))

DEFINITION:
in-node (old, new, i)
= **if** value (old, temp (i)) ↔ empty-node (old, i)
  then if empty-node (old, i)
    then changed (old, new, **nil**)
      ∨ ((true-node (new, i) ∨ false-node (new, i))
        ∧ (value (new, `'input`)
          = cons (true-node (new, i),

value (*old*, `'input`)))
$\land$ changed (*old*, *new*, list (ct (*i*), cf (*i*), `'input`)))
else changed (*old*, *new*, **nil**)
$\lor$ (empty-node (*new*, *i*)
$\land$ changed (*old*, *new*, list (ct (*i*), cf (*i*)))) **endif**
else changed (*old*, *new*, **nil**) **endif**

DEFINITION:
out-node (*old*, *new*)
$=$ ((value (*new*, ct (`0`)) $\leftrightarrow$ value (*old*, ct (`1`)))
$\land$ (value (*new*, cf (`0`)) $\leftrightarrow$ value (*old*, cf (`1`)))
$\land$ **if** empty-node (*old*, `0`) $\land$ ($\neg$ empty-node (*new*, `0`))
then value (*new*, `'output`)
$=$ cons (true-node (*new*, `0`), value (*old*, `'output`))
else value (*new*, `'output`) $=$ value (*old*, `'output`) **endif**
$\land$ changed (*old*, *new*, list (ct (`0`), cf (`0`), `'output`)))

DEFINITION:
internal-nodes (*n*)
$=$ **if** $n \simeq$ `0` **then nil**
else append (fifo-node (*n*), internal-nodes ($n - 1$)) **endif**

DEFINITION:
external-nodes (*n*)
$=$ list (list (`'in-node`, *n*),
list (`'out-node`),
list (`'nor-gate`, ct ($n - 1$), cf ($n - 1$), temp (*n*)))

DEFINITION:
fifo-queue (*n*) $=$ append (external-nodes (*n*), internal-nodes ($n - 1$))

THEOREM: member-internal-nodes
(*statement* $\in$ internal-nodes (*n*))
$=$ ((*statement* $\in$ fifo-node (cdadddr (*statement*)))
$\land$ ($n \not<$ cdadddr (*statement*))
$\land$ (cdadddr (*statement*) $\not\simeq$ `0`))

THEOREM: member-fifo-queue
(`1` $< n$)
$\rightarrow$ ((*statement* $\in$ fifo-queue (*n*))
$=$ ((*statement* $\in$ internal-nodes ($n - 1$))
$\lor$ (*statement* $\in$ external-nodes (*n*))))

THEOREM: listp-fifo-queue
listp (fifo-queue (*n*))

EVENT: Disable fifo-queue.

EVENT: Disable *1*fifo-queue.

DEFINITION:
proper-node (*state*, *i*)

$$
\begin{array}{l}
= \ ((((\neg \text{ empty-node}(state, i)) \wedge \text{empty-node}(state, i-1)) \\
\quad \rightarrow \ \text{value}(state, \text{temp}(i))) \\
\quad \wedge \ ((\text{empty-node}(state, i) \wedge (\neg \text{ empty-node}(state, i-1))) \\
\quad\quad \rightarrow \ (\neg \text{ value}(state, \text{temp}(i)))) \\
\quad \wedge \ (\text{true-node}(state, i) \\
\quad\quad \vee \ \text{false-node}(state, i) \\
\quad\quad \vee \ \text{empty-node}(state, i)) \\
\quad \wedge \ ((\neg \text{ empty-node}(state, i)) \\
\quad\quad \rightarrow \ (\text{empty-node}(state, i-1) \\
\quad\quad\quad \vee \ \textbf{if } \text{true-node}(state, i) \\
\quad\quad\quad\quad \textbf{then } \text{true-node}(state, i-1) \\
\quad\quad\quad\quad \textbf{else } \text{false-node}(state, i-1) \textbf{ endif})))
\end{array}
$$

DEFINITION:
proper-nodes (*state*, *n*)

$$
\begin{array}{l}
= \ \textbf{if } n \simeq 0 \ \textbf{then t} \\
\quad \textbf{else } \text{proper-node}(state, n) \wedge \text{proper-nodes}(state, n-1) \textbf{ endif}
\end{array}
$$

THEOREM: proper-nodes-implies

$$(\text{proper-nodes}(state, n) \wedge (n \not< i) \wedge (i \not\simeq 0)) \rightarrow \text{proper-node}(state, i)$$

THEOREM: proper-nodes-preserved-general

$$
\begin{array}{l}
(\text{n}(old, new, statement) \\
\ \wedge \ \text{proper-nodes}(old, n) \\
\ \wedge \ (statement \in \text{fifo-queue}(n)) \\
\ \wedge \ (n \not< i) \\
\ \wedge \ (1 < n)) \\
\rightarrow \ \text{proper-nodes}(new, i)
\end{array}
$$

THEOREM: proper-nodes-preserved

$$
\begin{array}{l}
(\text{n}(old, new, statement) \\
\ \wedge \ (statement \in \text{fifo-queue}(n)) \\
\ \wedge \ \text{proper-nodes}(old, n) \\
\ \wedge \ (1 < n)) \\
\rightarrow \ \text{proper-nodes}(new, n)
\end{array}
$$

EVENT: Disable proper-nodes-preserved-general.

THEOREM: proper-nodes-unless-false
$(1 < n)$
$\rightarrow$ unless (`(proper-nodes state ',n), '(false), fifo-queue (n))

THEOREM: proper-nodes-invariant
$((1 < n)$
$\wedge$ initial-condition (`(proper-nodes state ',n), fifo-queue $(n)))$
$\rightarrow$ invariant (`(proper-nodes state ',n), fifo-queue (n))

THEOREM: proper-node-invariant
$((1 < n)$
$\wedge$ $(i \not\simeq 0)$
$\wedge$ $(n \not< i)$
$\wedge$ initial-condition (`(proper-nodes state ',n), fifo-queue $(n)))$
$\rightarrow$ invariant (`(proper-node state ',i), fifo-queue (n))

DEFINITION:
queue-values (*state*, n)
= **if** $n \simeq 0$ **then nil**
elseif ($\neg$ empty-node (*state*, n)) $\wedge$ empty-node (*state*, $n - 1$)
then cons (true-node (*state*, n), queue-values (*state*, $n - 1$))
else queue-values (*state*, $n - 1$) **endif**

THEOREM: empty-empty-empty
$((1 < n)$
$\wedge$ n (*old*, *new*, *statement*)
$\wedge$ (*statement* $\in$ fifo-queue (n))
$\wedge$ empty-node (*old*, $1 + i$)
$\wedge$ empty-node (*old*, i)
$\wedge$ $(i \in \mathbf{N})$
$\wedge$ $(i < n))$
$\rightarrow$ empty-node (*new*, i)

THEOREM: full-full-full
$((1 < n)$
$\wedge$ n (*old*, *new*, *statement*)
$\wedge$ (*statement* $\in$ fifo-queue (n))
$\wedge$ ($\neg$ empty-node (*old*, $1 + i$))
$\wedge$ ($\neg$ empty-node (*old*, i))
$\wedge$ ((true-node (*old*, $1 + i$) $\leftrightarrow$ true-node (*old*, i))
$\vee$ (false-node (*old*, $1 + i$) $\leftrightarrow$ false-node (*old*, i)))
$\wedge$ $(i \in \mathbf{N})$
$\wedge$ $(i < n))$
$\rightarrow$ ($\neg$ empty-node (*new*, i))

THEOREM: queue-empty-preserved

$$
\begin{aligned}
&((1 < n) \\
&\land\ \text{n}\,(\mathit{old},\ \mathit{new},\ \mathit{statement}) \\
&\land\ (\mathit{statement} \in \text{fifo-queue}\,(n)) \\
&\land\ \text{proper-nodes}\,(\mathit{old},\ n) \\
&\land\ (\text{queue-values}\,(\mathit{old},\ 1 + i) = \textbf{nil}) \\
&\land\ (i < n)) \\
&\rightarrow\ (\text{queue-values}\,(\mathit{new},\ i) = \textbf{nil})
\end{aligned}
$$

THEOREM: value-moves-forward

$$
\begin{aligned}
&((1 < n) \\
&\land\ \text{n}\,(\mathit{old},\ \mathit{new},\ \mathit{statement}) \\
&\land\ (\mathit{statement} \in \text{fifo-queue}\,(n)) \\
&\land\ \text{proper-nodes}\,(\mathit{old},\ n) \\
&\land\ (\neg\ \text{empty-node}\,(\mathit{old},\ 1 + i)) \\
&\land\ \text{empty-node}\,(\mathit{old},\ i) \\
&\land\ (i < n) \\
&\land\ (i \not\simeq 0)) \\
&\rightarrow\ (((\neg\ \text{empty-node}\,(\mathit{new},\ 1 + i)) \land \text{empty-node}\,(\mathit{new},\ i)) \\
&\qquad \lor\ ((\neg\ \text{empty-node}\,(\mathit{new},\ i)) \land \text{empty-node}\,(\mathit{new},\ i - 1)))
\end{aligned}
$$

THEOREM: not-listp-queue-values-equals

$$\text{listp}\,(\text{queue-values}\,(\mathit{state},\ n)) = (\text{queue-values}\,(\mathit{state},\ n) \neq \textbf{nil})$$

THEOREM: full-empty-rest-unless-moves-forward

$((1 < n) \land (i < n) \land (i \not\simeq 0))$

→ unless (

```
`(and
   (proper-nodes state ',n)
   (and
    (not (empty-node state ',(add1 i)))
    (and
     (empty-node state ',i)
     (not (listp (queue-values state ',i)))))),
 `(and
   (not (empty-node state ',i))
   (and
    (empty-node state ',(sub1 i))
    (not
     (listp (queue-values state ',(sub1 i)))))),
```

fifo-queue (n))

THEOREM: output-only-adds-boolean

$(1 < n)$

→ unless (

```
`(equal (value state 'output) ',k),
 `(or
```

```
        (equal
         (value state 'output)
         (cons (true) ',k))
        (equal
         (value state 'output)
         (cons (false) ',k))),
      fifo-queue (n))
```

THEOREM: input-only-adds-boolean

```
 (1 < n)
 →  unless (`(equal (value state 'input) ',k),
      `(or
        (equal
         (value state 'input)
         (cons (true) ',k))
        (equal
         (value state 'input)
         (cons (false) ',k))),
      fifo-queue (n))
```

THEOREM: output-never-shortens

```
 (1 < n)
 →  unless (`(equal (length (value state 'output)) ',k),
      `(lessp ',k (length (value state 'output))),
      fifo-queue (n))
```

THEOREM: total-sufficient-fifo-queue

```
 (1 < n)
 →  total-sufficient (statement,
              fifo-queue (n),
              old,
              if car (statement) = 'c-element
              then if value (old, cadr (statement))
                      ↔  value (old, caddr (statement))
                   then update-assoc (cadddr (statement),
                                      value (old, cadr (statement)),
                                      old)
                   else old endif
              elseif car (statement) = 'nor-gate
              then update-assoc (cadddr (statement),
                                 ¬ (value (old, cadr (statement))
                                    ∨  value (old,
                                              caddr (statement))),
                                 old)
              elseif car (statement) = 'in-node
```

```
then if (value (old, temp (cadr (statement)))
           =   empty-node (old, cadr (statement)))
          ∧   (¬ empty-node (old, cadr (statement)))
        then update-assoc (ct (cadr (statement)),
                           f,
                           update-assoc (cf (cadr (statement)),
                                         f,
                                         old))
        else old endif
elseif car (statement) = 'out-node
then update-assoc (ct (0),
                   value (old, ct (1)),
                   update-assoc (cf (0),
                                 value (old, cf (1)),
                                 if empty-node (old, 0)
                                    ∧   (¬ empty-node (old,
                                                       1))
                                 then update-assoc ('output,
                                                    cons (true-node (old,
                                                                     1),
                                                          value (old,
                                                                 'output)),
                                                    old)
                                 else old endif))
else old endif)
```

THEOREM: total-fifo-queue
$(1 < n) \rightarrow$ total (fifo-queue (n))

THEOREM: output-grows-immediately

```
(initial-condition (‘(proper-nodes state ',n), fifo-queue (n))
 ∧   (1 < n))
→   leads-to (‘(and
                (not (empty-node state 1))
                (and
                 (empty-node state 0)
                 (equal (length (value state 'output)) ',k))),
              ‘(lessp ',k (length (value state 'output))),
              fifo-queue (n))
```

THEOREM: true-empty-temp-moves

```
(initial-condition (‘(proper-nodes state ',n), fifo-queue (n))
 ∧   (1 < n)
 ∧   (i ≄ 0)
 ∧   (i < n))
```

$\rightarrow$ leads-to (`` `(and ``
`(true-node state ',(add1 i))`
`(value state (temp ',i)))`,
`` `(not (empty-node state ',i)) ``,
fifo-queue (n))

THEOREM: false-empty-temp-moves
(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(i \not\simeq 0)$
$\land$ $(i < n))$
$\rightarrow$ leads-to (`` `(and ``
`(false-node state ',(add1 i))`
`(value state (temp ',i)))`,
`` `(not (empty-node state ',i)) ``,
fifo-queue (n))

THEOREM: not-empty-empty-temp-moves
(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(i \not\simeq 0)$
$\land$ $(i < n))$
$\rightarrow$ leads-to (`` `(and ``
`(not (empty-node state ',(add1 i)))`
`(value state (temp ',i)))`,
`` `(not (empty-node state ',i)) ``,
fifo-queue (n))

THEOREM: empty-empty-sets-temp
(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(i \in \mathbf{N})$
$\land$ $(i < n))$
$\rightarrow$ leads-to (`` `(and ``
`(empty-node state ',(add1 i))`
`(empty-node state ',i))`,
`` `(value state (temp ',(add1 i))) ``,
fifo-queue (n))

THEOREM: full-full-unsets-temp
(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(i \in \mathbf{N})$
$\land$ $(i < n))$
$\rightarrow$ leads-to (`` `(and ``

```
                (not (empty-node state ',(add1 i)))
                (not (empty-node state ',i))),
             `(not (value state (temp ',(add1 i)))),
             fifo-queue (n))
```

THEOREM: empty-true-not-temp-moves

```
 (initial-condition (`(proper-nodes state ',n), fifo-queue (n))
  ∧   (1 < n)
  ∧   (i ≄ 0)
  ∧   (i < n))
 →   leads-to (`(and
                (empty-node state ',(add1 i))
                (and
                 (true-node state ',i)
                 (not (value state (temp ',i))))),
             `(empty-node state ',i),
             fifo-queue (n))
```

THEOREM: empty-false-not-temp-moves

```
 (initial-condition (`(proper-nodes state ',n), fifo-queue (n))
  ∧   (1 < n)
  ∧   (i ≄ 0)
  ∧   (i < n))
 →   leads-to (`(and
                (empty-node state ',(add1 i))
                (and
                 (false-node state ',i)
                 (not (value state (temp ',i))))),
             `(empty-node state ',i),
             fifo-queue (n))
```

THEOREM: empty-not-temp-moves

```
 (initial-condition (`(proper-nodes state ',n), fifo-queue (n))
  ∧   (1 < n)
  ∧   (i ≄ 0)
  ∧   (i < n))
 →   leads-to (`(and
                (empty-node state ',(add1 i))
                (not (value state (temp ',i)))),
             `(empty-node state ',i),
             fifo-queue (n))
```

THEOREM: empty-1-leads-to-empty-0

```
 (initial-condition (`(proper-nodes state ',n), fifo-queue (n))
  ∧   (1 < n))
```

$\rightarrow$ leads-to (`` `(empty-node state ',1) ``,
`` `(empty-node state ',0) ``,
fifo-queue (n))

Theorem: full-empty-empty-moves-forward

(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(n \not< (1 + (1 + i)))$
$\land$ $(i \in \mathbf{N})$)
$\rightarrow$ leads-to (

```
`(and
   (not (empty-node state ',(add1 (add1 i))))
   (and
    (empty-node state ',(add1 i))
    (and
     (empty-node state ',i)
     (not
      (listp (queue-values state ',(add1 i))))))),
`(and
   (not (empty-node state ',(add1 i)))
   (empty-node state ',i)),
```

fifo-queue (n))

Theorem: full-empty-full-moves-forward

(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(n \not< (1 + (1 + i)))$
$\land$ $(i \in \mathbf{N})$)
$\rightarrow$ leads-to (

```
`(and
   (not (empty-node state ',(add1 (add1 i))))
   (and
    (empty-node state ',(add1 i))
    (and
     (not (empty-node state ',i))
     (not
      (listp (queue-values state ',(add1 i))))))),
`(and
   (not (empty-node state ',(add1 i)))
   (empty-node state ',i)),
```

fifo-queue (n))

Theorem: full-empty-moves-forward

(initial-condition (`` `(proper-nodes state ',n) ``, fifo-queue (n))
$\land$ $(1 < n)$
$\land$ $(n \not< (1 + (1 + i)))$
$\land$ $(i \in \mathbf{N})$)

$\rightarrow$ leads-to (`(and
```
              (not (empty-node state ',(add1 (add1 i))))
              (and
               (empty-node state ',(add1 i))
               (not
                (listp (queue-values state ',(add1 i)))))),
           `(and
              (not (empty-node state ',(add1 i)))
              (empty-node state ',i)),
```
fifo-queue (n))

THEOREM: full-rest-empty-moves-forward
(initial-condition (`(proper-nodes state ',n), fifo-queue (n))
$\wedge$ $(1 < n)$
$\wedge$ $(i < n)$
$\wedge$ $(i \not\simeq 0)$)
$\rightarrow$ leads-to (`(and
```
              (not (empty-node state ',(add1 i)))
              (and
               (empty-node state ',i)
               (not (listp (queue-values state ',i))))),
           `(and
              (not (empty-node state ',i))
              (and
               (empty-node state ',(sub1 i))
               (not
                (listp (queue-values state ',(sub1 i)))))),
```
fifo-queue (n))

THEOREM: full-rest-empty-output-grows
(initial-condition (`(proper-nodes state ',n), fifo-queue (n))
$\wedge$ $(1 < n)$
$\wedge$ $(i < n)$
$\wedge$ $(i \in \mathbf{N})$)
$\rightarrow$ leads-to (`(and
```
              (not (empty-node state ',(add1 i)))
              (and
               (empty-node state ',i)
               (and
                (not (listp (queue-values state ',i)))
                (equal
                 (length (value state 'output))
                 ',k)))),
           `(lessp ',k (length (value state 'output))),
```
fifo-queue (n))

Definition:
first-empty-subqueue $(state,\ n)$
$=$ **if** $n \simeq$ `0` **then** `0`
elseif listp (queue-values $(state,\ n)$)
then first-empty-subqueue $(state,\ n-1)$
else n **endif**

Theorem: first-empty-subqueue-is-empty
$\neg$ listp (queue-values ($state$, first-empty-subqueue $(state,\ n)$))

Theorem: queue-not-empty-implies
listp (queue-values $(state,\ n)$)
$\rightarrow$ (($\neg$ empty-node ($state$, $1 +$ first-empty-subqueue $(state,\ n)$))
$\wedge$ empty-node ($state$, first-empty-subqueue $(state,\ n)$))

Theorem: not-lessp-first-empty-subqueue
$n \not< $ first-empty-subqueue $(state,\ n)$

Theorem: lessp-first-empty-subqueue
listp (queue-values $(state,\ n)$) $\rightarrow$ (first-empty-subqueue $(state,\ n) < n$)

Theorem: output-grows
(initial-condition (`‘(proper-nodes state ’,n)`, fifo-queue (n))
$\wedge$ (`1` $< n$))
$\rightarrow$ leads-to (`‘(and`
`(listp (queue-values state n))`
`(equal (length (value state ’output)) ’,k))`,
`‘(lessp ’,k (length (value state ’output)))`,
fifo-queue (n))

Theorem: in-node-leaves-rest-of-queue-unchanged
(n (old, new, list (`’in-node`, n)) $\wedge$ (`1` $< n$) $\wedge$ $(i < n)$ $\wedge$ $(i \in \mathbf{N})$)
$\rightarrow$ ((value (new, `’output`) = value (old, `’output`))
$\wedge$ (queue-values $(new,\ i)$ = queue-values $(old,\ i)$))

Theorem: in-node-preserves-values
(n (old, new, list (`’in-node`, n))
$\wedge$ proper-node $(old,\ n)$
$\wedge$ (value (old, `’input`)
$=$ append (queue-values $(old,\ n)$, value (old, `’output`)))
$\wedge$ (`1` $< n$))
$\rightarrow$ (value (new, `’input`)
$=$ append (queue-values $(new,\ n)$, value (new, `’output`)))

Theorem: out-node-preserves-queue-values

$(\mathrm{n}\,(old,\ new,\ \texttt{'(out-node)}) \land (\texttt{1} < n) \land (n \not< i) \land (i \not\simeq \texttt{0}))$
$\rightarrow \quad ((\mathrm{value}\,(new,\ \texttt{'input}) = \mathrm{value}\,(old,\ \texttt{'input}))$
$\quad\quad \land \quad (\mathrm{append}\,(\text{queue-values}\,(new,\ i),\ \mathrm{value}\,(new,\ \texttt{'output}))$
$\quad\quad\quad\quad = \quad \mathrm{append}\,(\text{queue-values}\,(old,\ i),\ \mathrm{value}\,(old,\ \texttt{'output}))))$

THEOREM: external-nor-gates-preserves-values
$(\mathrm{n}\,(old,\ new,\ \mathrm{list}\,(\texttt{'nor-gate},\ \mathrm{ct}\,(n-1),\ \mathrm{cf}\,(n-1),\ \mathrm{temp}\,(n)))$
$\land \quad (\texttt{1} < n)$
$\land \quad (n \not< i)$
$\land \quad (i \in \mathbf{N}))$
$\rightarrow \quad ((\mathrm{value}\,(new,\ \texttt{'input}) = \mathrm{value}\,(old,\ \texttt{'input}))$
$\quad\quad \land \quad (\mathrm{value}\,(new,\ \texttt{'output}) = \mathrm{value}\,(old,\ \texttt{'output}))$
$\quad\quad \land \quad (\text{queue-values}\,(new,\ i) = \text{queue-values}\,(old,\ i)))$

THEOREM: internal-nodes-preserves-rest-of-queue
$(\mathrm{n}\,(old,\ new,\ statement)$
$\land \quad (statement \in \text{fifo-node}\,(i))$
$\land \quad (j \in \mathbf{N})$
$\land \quad (j < i))$
$\rightarrow \quad (\text{queue-values}\,(new,\ j) = \text{queue-values}\,(old,\ j))$

THEOREM: internal-nodes-preserve-values
$(\mathrm{n}\,(old,\ new,\ statement)$
$\land \quad (statement \in \text{fifo-node}\,(i))$
$\land \quad \text{proper-node}\,(old,\ 1 + i)$
$\land \quad \text{proper-node}\,(old,\ i)$
$\land \quad (i \not\simeq \texttt{0})$
$\land \quad (i < j))$
$\rightarrow \quad ((\mathrm{value}\,(new,\ \texttt{'input}) = \mathrm{value}\,(old,\ \texttt{'input}))$
$\quad\quad \land \quad (\mathrm{value}\,(new,\ \texttt{'output}) = \mathrm{value}\,(old,\ \texttt{'output}))$
$\quad\quad \land \quad (\text{queue-values}\,(new,\ j) = \text{queue-values}\,(old,\ j)))$

THEOREM: values-preserved
$(\mathrm{n}\,(old,\ new,\ statement)$
$\land \quad \text{proper-nodes}\,(old,\ n)$
$\land \quad (\texttt{1} < n)$
$\land \quad (\mathrm{value}\,(old,\ \texttt{'input})$
$\quad\quad = \quad \mathrm{append}\,(\text{queue-values}\,(old,\ n),\ \mathrm{value}\,(old,\ \texttt{'output})))$
$\land \quad (statement \in \text{fifo-queue}\,(n)))$
$\rightarrow \quad (\mathrm{value}\,(new,\ \texttt{'input})$
$\quad\quad = \quad \mathrm{append}\,(\text{queue-values}\,(new,\ n),\ \mathrm{value}\,(new,\ \texttt{'output})))$

THEOREM: values-invariant
$(\texttt{1} < n)$
$\rightarrow \quad \mathrm{unless}\,(\texttt{`(and}$

```
            (proper-nodes state ',n)
            (equal
             (value state 'input)
             (append
              (queue-values state ',n)
              (value state 'output)))),
          '(false),
          fifo-queue (n))
```

THEOREM: queue-values-invariant

```
(initial-condition (`(and
                      (proper-nodes state ',n)
                      (equal
                       (value state 'input)
                       (append
                        (queue-values state ',n)
                        (value state 'output)))),
                    fifo-queue (n))
 ∧   (1 < n))
 →   invariant (`(equal
                  (value state 'input)
                  (append
                   (queue-values state ',n)
                   (value state 'output))),
                fifo-queue (n))
```

Index