**Traversing a Singly-Listed List**

(G ( r XOR s) & G ( s -> X s)

G(r -> !end))

((G( s -> (end <-> X end)) AND ((G F r) -> F end) ) => F G end)

input = end

output = r,s

-----------------------------------

**Traversing a Tree:**

LTL formulas for StriX:

Assumptions:

true

Guarantees:

G((e & !p & !s) || (!e & p & !s) || (!e & !p & s)) & G(s -> (X s))

G(e -> !em)

G(p -> ha)

((!em && G(p->(X(!em & !ha))) && G(s -> ((em <-> (X em)) & (ha <-> (X ha))) )) AND ((G (F e)) -> (F(em & !ha)))) => ((F (G(em & !ha))) && G(ha -> (!e U p)))

Input propositions:

em, ha #memory is empty, current node’s children have not been put in memory

Output propositions:

e, p, s #extract a node from memory, for visiting, put all children of current node into memory, stop

-----------------------------------

**Traversing a Graph:**

(G(e XOR p XOR s) && G(s -> X (s)) && G(e -> !em) && G(p -> ha)) AND (((!em && G(p->X(!em && !ha)) && G(s -> ((em <-> X(em)) && (ha <-> X(ha))))) AND (( ((GF(e)) && (FG(!p))) -> FG(em) ) && ( (GF(p)) -> FG(!ha) ))) => (FG(em && !ha) && G(ha -> (!e U p))) )

input = em, ha

output = e, p, s

---------------

**Traversing a Doubly-Linked list**

G ( r XOR l XOR s) & G ( s -> X s)

G ( r -> !end) & G( l -> !beg)

(( ( (beg || end) && G( r -> X !beg) && G (l -> X !end) && G( s -> ((beg <-> X beg) & (end <-> X end))) ) AND ( ((G F r & F G !l ) -> F end) && ((G F l & F G !r ) -> F beg)) ) => ( ((beg -> F G end) &( end -> F G beg)) && ( G((l -> X !r) & ( r -> X !l))) ))

input = beg, end

output = r,l,s

-------------

**Minimum in a list**

LTL formulas for StriX:

Assumptions:

true

Guarantees:

G((r & !s & !u) || (!r & s & !u) || (!r & !s & u)) & G(s -> X(s))

G(r -> !e)

(G(u -> (X(!lt) & (e <-> X(e)))) & G(s -> ((e <-> X(e)) & (lt <-> X(lt)))) & G(F(r)) -> F(e)) -> (F(G(e)) & G(u <-> lt))

Input propositions:

e, lt #less then, end

Output propositions:

r, s, u #right, stop, update

---------------------------------------------------------------

**Membership in a tree**

LTL formulas for StriX:

Assumptions:

true

Guarantees:

G((e & !p & !s) || (!e & p & !s) || (!e & !p & s)) & G(s -> (X s))

G(e -> !empty)

G(p -> ha)

((!empty && G(p->(X(!empty & !ha))) && G(s -> ((empty <-> (X empty)) & (ha <-> (X ha)) & (equal <-> (X equal))) )) AND ((G (F e)) -> (F(empty & !ha)))) => ((F(G(equal))) || (F(G(empty & !ha))) & G((ha & !equal) -> ((!empty) U p)) & G(equal -> s))

Input propositions:

empty, ha , equal

Output propositions:

s, e, p

-------------------------------------------------------------------------

**COPY ONLY EVEN NUMBERS FROM A LIST**

LTL formulas for StriX:

Assumptions:

true

Guarantees:

G((r & !s & !a) || (!r & s & !a) || (!r & !s & a)) & G(s -> X(s))

G(r -> !e)

(G(a -> (X(hasput) & (even <-> X(even)) & (e <-> X(e)))) & G(s -> ((e <-> X(e)) & (hasput <-> X(hasput)) & (even <-> X(even)))) AND G(F(r)) -> F(e)) -> (F(G(e)) & G(a <-> (even & !hasput)))

Input propositions:

e, even, hasput #end, è pari, è stato inserito nella lista

Output propositions:

r, s, a #right, stop, add element in the new list

-------------------------------------------------------------------------

**Bubble sort**

LTL formulas for StriX:

Assumptions:

true

Guarantees:

G((e & !p & !s) || (!e & p & !s) || (!e & !p & s)) & G(s -> (X s))

G(e -> !empty)

G(p -> ha)

((!empty && G(p->(X(!empty & !ha))) && G(s -> ((empty <-> (X empty)) & (ha <-> (X ha)) & (equal <-> (X equal))) )) AND ((G (F e)) -> (F(empty & !ha)))) => ((F(G(equal))) || (F(G(empty & !ha))) & G((ha & !equal) -> ((!empty) U p)) & G(equal -> s))

Input propositions:

empty, ha , equal

Output propositions:

s, e, p

-------------------------------------------------------------------------

-------------------------------------------------

SUM (of positive numbers) IS GREATER THAN A CONSTANT sum > constant

G((r & !s & !a) || (!r & s & !a) || (!r & !s & a)) & G(s -> X(s))

G(r -> !end)

((G(a -> (X(summed) & (end <-> X(end)) & (isgreater <-> X(isgreater)))) & G(s -> ((end <-> X(end)) & (!summed <-> X(!summed)) & (isgreater <-> X(isgreater)))) & G(F(r)) -> F(end)) -> (F(G(end OR isgreater)) & G(a <-> !summed)))

summed , end, isgreater

r, s, a

-------------------------------------------------------

SUM (non funziona)

Strix:

G((r & !s & !a) || (!r & s & !a) || (!r & !s & a)) & G(s -> X(s))

G(r -> (!end & !isgreater & summed))

G(a -> (!summed & !isgreater))

(!summed AND G(a -> (X(summed) & (end <-> X(end)) )) & G(s -> ((end <-> X(end)) & (summed <-> X(summed)) & (isgreater <-> X(isgreater)))) & G(F(r)) -> F(end )) -> (F(isgreater OR (end AND summed)) )

input: summed , end, isgreater

output: r, s, a

LTL:

◻️((r ∧ ¬s ∧ ¬a) || (¬r ∧ s ∧ ¬a) || (¬r ∧ ¬s ∧ a)) ∧ ◻️(s → 𐩒(s))

◻️(r → (¬end ∧ ¬isgreater ∧ summed))

◻️(a → (¬summed ∧ ¬isgreater))

((◻️(a → (𐩒(summed) ∧ (end ↔ 𐩒(end)) )) ∧ ◻️(s → ((end ↔ 𐩒(end)) ∧ (summed ↔ 𐩒(summed)) ∧ (isgreater ↔ 𐩒(isgreater)))) ∧ ◻️(♢(r)) → ♢(end )) → (♢(◻️(end ∨ isgreater )) ∧ ◻️(isgreater → s) ))

----------------------------------------------------------------

SUM OF A LIST

G((r & !s & !add) || (!r & s & !add) || (!r & !s & add)) & G(s -> X(s))

G(r -> !end)

((G(add -> (X(summed) & (end <-> X(end)))) & G(s -> ((end <-> X(end)) & (!summed <-> X(!summed)))) & G(F(r)) -> F(end)) -> (F(G(end)) & G(add <-> !summed)))

summed , end

r, s, add

-----------------------------------------------------------------