Lecture 12 - March 2

Model Checking

Path Satisfaction: Nested LTL Operators

F $\phi 1 \Rightarrow$ ***FG*** $\phi 2$

Nesting "Global" and "Future" in LTL Formulas

model sat.

$$s \models \boxed{\mathbf{F}\phi 1} \Rightarrow \boxed{\mathbf{FG}\phi_2}$$

Each path π starting with s is s.t. if eventually ϕ1 holds on π, then ϕ2 eventually holds on π continuously.

Q. Formulate the above nested pattern of LTL operators.

$* \forall \pi \bullet \pi = s \rightarrow \cdots \Rightarrow$

$\big((\exists i_1 \bullet i_1 \geq 1 \wedge \pi^{i_1} \models \phi_1)$

$** \Rightarrow$

$(\exists i_2 \bullet i_2 \geq 1 \wedge (\forall j \bullet j \geq i_2 \Rightarrow \pi^j \models \phi_2)) \big)$

Q. How to prove the above nested pattern of LTL operators?

① Consider all path patterns ② a. T ⇒ T b. F ⇒ _ c. _ ⇒ T

Q. How to disprove the above nested pattern of LTL operators?

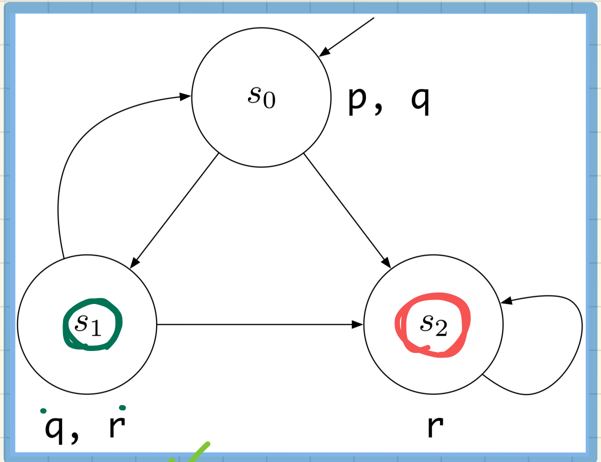
① Find a witness path ② T ⇒ F .

Path Satisfaction: Exercises (5.2)

alt to *: s_2 satisfies $\neg q \wedge r$, then from s_2 r is cont. T

④ $s_0 \to s_1 \to s_0 \to s_1 \to \cdots \to s_2 \to \cdots$

⑤ $s_0 \to s_1 \to s_0 \to s_1 \to \cdots \to s_0 \to s_2 \to \cdots$



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

③ $s_0 \to s_1 \to s_2 \to \cdots$ (exercise)

* $s_0 \models F(\neg q \wedge r) \Rightarrow FG\ r$ T

① $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

↳ no state in this path satisfies $\neg q \wedge r$

↳ $F(\neg q \wedge r)$ is false

false $\Rightarrow P \equiv$ True

② $s_0 \to s_2 \to s_2 \to \cdots$

↳ satisfies $\neg q \wedge r$

↳ $F(\neg q \wedge r)$ is T

starting from s_2, Gr is satisfied. T $\Rightarrow$ T $\equiv$ T

$s_0 \models F(\neg q \vee r) \Rightarrow FG\ r$ F

T

Witness: $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

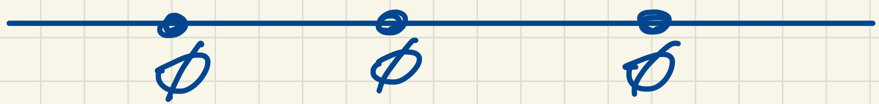
↳ satisfies $F(\neg q \vee r)$: s_1

↳ violates: $FG\ r$: s_0 does not satisfy r.

Exercise: What if we change the LHS to s_2?

FG $\emptyset$

GE $\emptyset$



Lab2 Solution: getAllSuffixes (V2: Tuple of Tuples)

```
---------------------------- MODULE getAllSuffixes_v2 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v2 {
    variable result = input, postfixSoFar = <<>>, i = Len(input) - 1;
    {
        postfixSoFar := <<input[Len(input)]>>;
        result[Len(input)] := postfixSoFar;
        while (i > 0) {
            postfixSoFar := <<input[i]>> \o postfixSoFar;
            result[i] := postfixSoFar;
            i := i - 1;
        };
        assert \A j \in 1..Len(input):  Len(result[j]) = Len(input) - j + 1;
        assert \A j \in 1..Len(result): \A k \in 1..Len(result[j]): result[j][k] = input[j - 1 + k]);
    }
}
*)
```

input vars will be constants

↳ need to be instantiated for model checking

outputs will be variables

algo

\E

input: [23, 46, 69]

result: Len(input)

[[23, 46, 69], 3

[46, 69], 2

[69]]

I	Len(i)	J	Len(r[J])
	3	1	3
	3	2	2
	3	3	1

```
----------------------------- MODULE getAllSuffixes_v2 -----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v2 {
    variable result = input, postfixSoFar = <<>>, i = Len(input) - 1;
    {
        postfixSoFar := <<input[Len(input)]>>;
        result[Len(input)] := postfixSoFar;
        while (i > 0) {
            postfixSoFar := <<input[i]>> \o postfixSoFar;
            result[i] := postfixSoFar;
            i := i - 1;
        };
        assert \A j \in 1..Len(input):  Len(result[j]) = Len(input) - j + 1;
        assert \A j \in 1..Len(result): (\A k \in 1..Len(result[j]): result[j][k] = input[j - 1 + k]);
    }
}
*)
```

input: [23, 46, 69]

2 3

result:

use k to refer to an item in tuple.

[[23, 46, 69],

1 2

[46, 69],

[69]]

use j to refer to a suffix

an item in some result tuple

offset

j

1 k = 1, 2, 3

2 k = 1, 2

3 k = 1

(Len(result[1]))

Lab2 Solution: getRightShifts

$B \Rightarrow P$

$\equiv$ $\wedge \; \neg B \Rightarrow Q$

```
------------------------------ MODULE getRightShifts ------------------------------
EXTENDS Integers, Naturals, Sequences, TLC
CONSTANT input, n
ASSUME /\ Len(input) > 0
       /\ n >= 0

(*
--algorithm getRightShifts {
    variable result = input, nos = n % Len(input) (* number of shifts *), i = 1;
    {
        while (i <= Len(input)) {
            if (((i + Len(input)) - nos) % Len(input) = 0) {
                result[i] := input[Len(input)];
            } else {
                result[i] := input[((i + Len(input)) - nos) % Len(input)];
            };
            i := i + 1;
        };
        \* version 1 of postcondition: for each index in the result, what is the corresponding index in the input?
        assert \A j \in 1..Len(result): IF ((j + Len(result)) - (n % Len(result))) % Len(result) = 0
                                        THEN result[j] = input[Len(input)]
                                        ELSE result[j] = input[((j + Len(input)) - (n % Len(input))) % Len(input)];
        \* version 2 of postcondition: for each index in the input, what is the corresponding index in the result?
        assert \A j \in 1..Len(input): IF (j + (n % Len(input))) % Len(input) = 0
                                       THEN result[Len(input)] = input[j]
                                       ELSE result[((j + (n % Len(input))) % Len(input))] = input[j];
    }
}
*)
```

IF B

THEN P

ELSE Q

shift to R by one pos.

	1	2	3
input:	23	46	69
result:	69	23	46

input	+1	output
1	2	2
2	3	3
3	4	1

4 % 3

Assertion: explicit about the variables
that can be used.

```
<<1, 2, 3>> \o <<4, 5, 6>>

output1 \o <<i>>
```