Lecture 11 - February 10

Lab2 Solution Walkthrough, Model Checking

Generalizing rounds, Function, Macro
Postconditions: getAllSuffixes
LTL Grammar: Top-Down Derivation

Announcements/Reminders

- **ProgTest1** this Thursday during the lab session
  + Please arrange your commute accordingly.
  + Test will only be canceled if the university is closed.
- **Practice Test** questions and solutions released
- **Lab1** and **Lab2** solutions released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

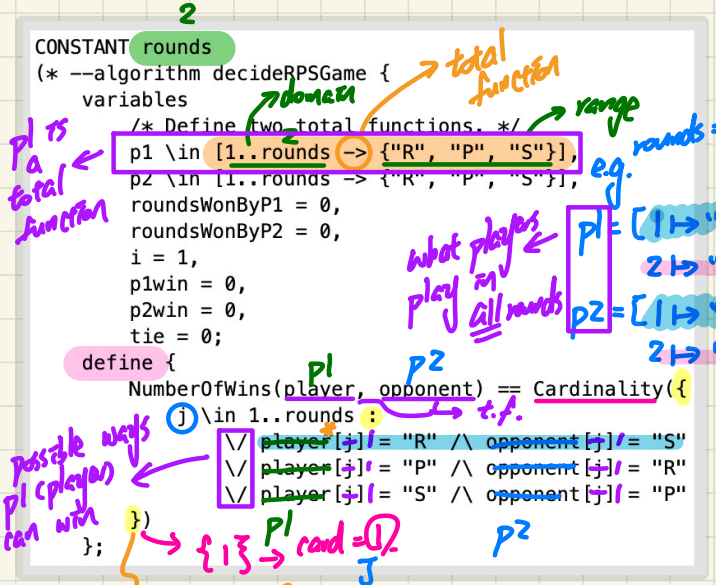
Lab2 Solution:
decideRPSGameResult
(Generalizing # of Rounds)

```
(*
--algorithm decideRPSGameResult {
variable
    p1r1 \in {"R", "P", "S"},
    p2r1 \in {"R", "P", "S"},
    numOfRoundsWonByPlayer1 = 0,
    numOfRoundsWonByPlayer2 = 0,
    p1win = FALSE,
    p2win = FALSE,
    tie = FALSE,
    i = 1,
    rounds = 2;
{
    while(i <= rounds){
        if (p1r1 = "R" /\ p2r1 = "P"){
            numOfRoundsWonByPlayer2 := numOfRoundsWonByPlayer2 + 1;
        };
        else if (p1r1 = "R" /\ p2r1 = "S"){
            numOfRoundsWonByPlayer1 := numOfRoundsWonByPlayer1 + 1;
        };
        ... /* Consider cases where p1r1 = "P" and p1r1 = "S" */

        \* Get prepared for the next round.

        choose_p1_next: with (x \in {"R", "P", "S"}) {
            p1r1 := x;
        };
        choose_p2_next: with (x \in {"R", "P", "S"}) {
            p2r1 := x;
        };
        i := i + 1;
    };

    if (numOfRoundsWonByPlayer1 = numOfRoundsWonByPlayer2){
            tie := TRUE;
    };
    else if (numOfRoundsWonByPlayer1 > numOfRoundsWonByPlayer2){
            p1win := TRUE;
    };
    else {
        p2win := TRUE;
    };
}
*)
```

initial non-deterministic choices for players in the 1st round

non-deterministic choices for players in the next round

Lab2 Solution:
decideRPSGameResult (Postcondition)




```
    {
        while (i <= rounds) {
            if (p1[i] = "R") {
                if (p2[i] = "P") {
                    roundsWonByP2 := roundsWonByP2 + 1;
                } else if (p2[i] = "S") {
                    roundsWonByP1 := roundsWonByP1 + 1;
                };
            }
            /* Consider p1[i] = "P" or p1[i] = "S"
            ...
            i := i + 1;
        };

        /* Set p1win, p2win, tie w.r.t. roundsWonByP1, roundsWonByP2.
        ...

        \* Assert consistency among variables.
        assert
            /\ roundsWonByP1 > roundsWonByP2 => (
                /\ p1win = 1
                /\ p2win = 0
                /\ tie = 0)
            /* Also consider _ < _ and _ = _
            ...

        \* Verify the correctness of win decisions.
        assert NumberOfWins(p1, p2) = roundsWonByP1;
        assert NumberOfWins(p2, p1) = roundsWonByP2;
    }
} *)
```

final value of prg. variable

calling the macro

Lab2 Solution: getAllSuffixes_v3 (1)

```
---------------------------- MODULE getAllSuffixes_v3 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v3 {
    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]);
    }
}
*)
```

concatenation

index 1 2 3

input: [23, 46, 69]

result:

[[23, 46, 69], 1

[46, 69], 2

[69]] 3

Len(r[j]) = Len(input) + 1 - j

j + Len(r[j]) = Len(input) + 1

j	Len(result[j])		Len(input)
1	3	4	3
2	2	4	3
3	1	4	3

Postcondition

(1) Understand the quantification expression on some concrete example.

(2) Practice writing from scratch.

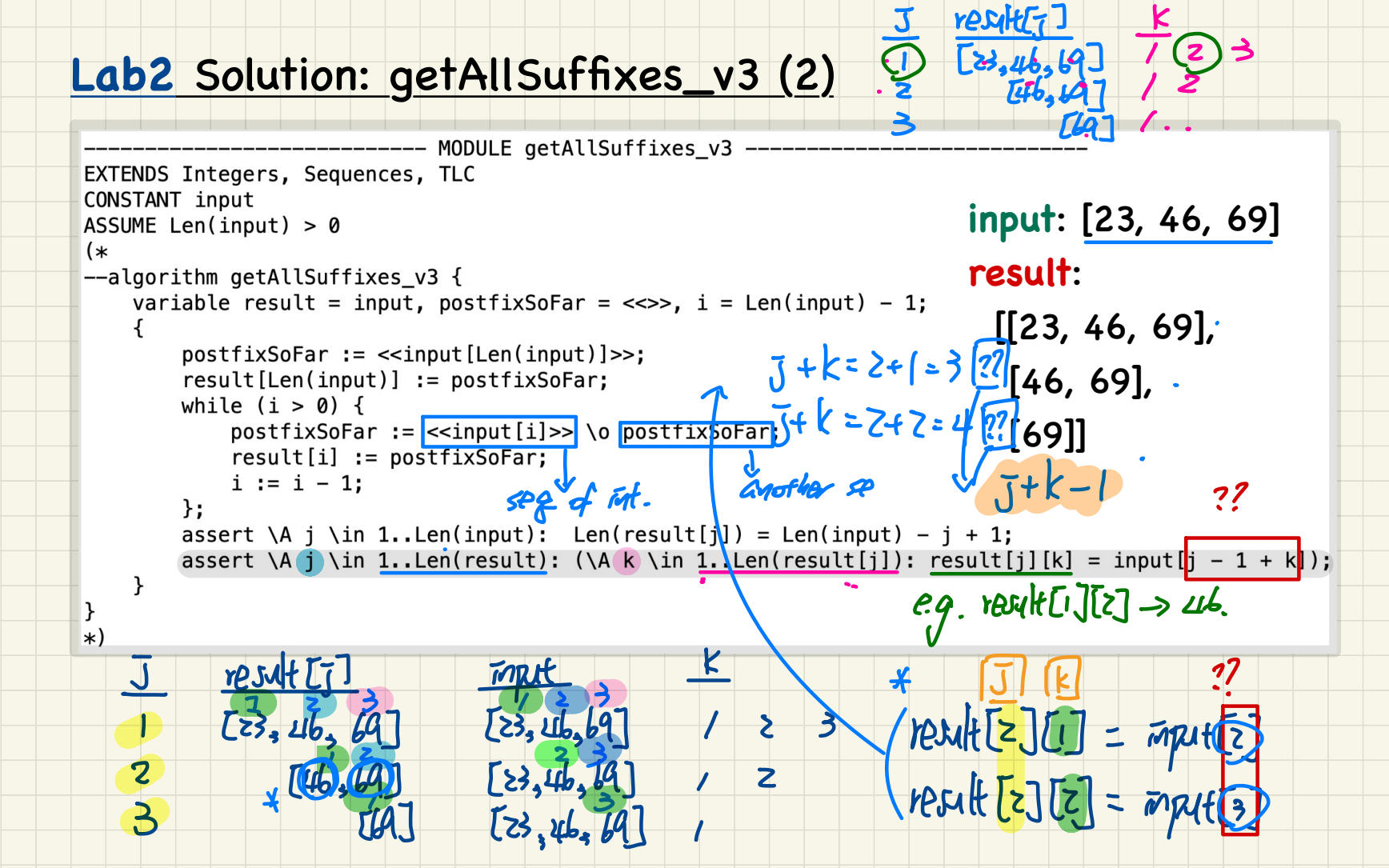
Lab2 Solution: getAllSuffixes_v3 (2)

```
---------------------------- MODULE getAllSuffixes_v3 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v3 {
    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]

result:

[[23, 46, 69],
[46, 69],
[69]]



LTL Syntax: Context-Free Grammar

$$
\begin{array}{lll}
\phi & ::= & \top \quad [\,\textit{true}\,] \\
& | & \bot \quad [\,\textit{false}\,] \\
& | & p \quad [\,\texttt{propositional atom}\,] \\
& | & (\neg\phi) \quad [\,\texttt{logical negation}\,] \\
& | & (\phi \wedge \phi) \quad [\,\texttt{logical conjunction}\,] \\
& | & (\phi \vee \phi) \quad [\,\texttt{logical disjunction}\,] \\
& | & (\phi \Rightarrow \phi) \quad [\,\texttt{logical implication}\,] \\
& | & (\mathbf{X}\,\phi) \quad [\,\texttt{neXt state}\,] \\
& | & (\mathbf{F}\,\phi) \quad [\,\texttt{some Future state}\,] \\
& | & (\mathbf{G}\,\phi) \quad [\,\texttt{all future states (Globally)}\,] \\
& | & (\phi\,\mathbf{U}\,\phi) \quad [\,\texttt{Until}\,] \\
& | & (\phi\,\mathbf{W}\,\phi) \quad [\,\texttt{Weak-until}\,] \\
& | & (\phi\,\mathbf{R}\,\phi) \quad [\,\texttt{Release}\,]
\end{array}
$$

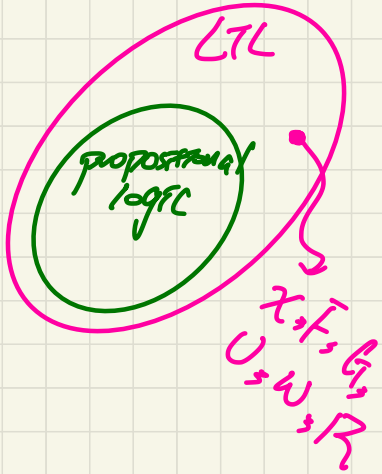
base cases

recursive cases

propositional logic

implicitly use $\forall$ and/or $\exists$

LTL



$$
\begin{array}{lll}
\phi & ::= & \top \quad \text{[true]} \\
& | & \bot \quad \text{[false]} \\
& | & p \quad \text{[propositional atom]} \\
& | & (\neg\phi) \quad \text{[logical negation]} \\
& | & (\phi \wedge \phi) \quad \text{[logical conjunction]} \\
& | & (\phi \vee \phi) \quad \text{[logical disjunction]} \\
& | & (\phi \Rightarrow \phi) \quad \text{[logical implication]} \\
& | & (\mathbf{X}\phi) \quad \text{[neXt state]} \\
& | & (\mathbf{F}\phi) \quad \text{[some Future state]} \\
& | & (\mathbf{G}\phi) \quad \text{[all future states (Globally)]} \\
& | & (\phi \mathbf{U} \phi) \quad \text{[Until]} \\
& | & (\phi \mathbf{W} \phi) \quad \text{[Weak-until]} \\
& | & (\phi \mathbf{R} \phi) \quad \text{[Release]}
\end{array}
$$

start symbol

(top-down) derivation:

$p \wedge \chi_\top$

syntactically simple

$$
\begin{aligned}
\phi &= \phi \wedge \phi \\
&= p \wedge \phi \\
&= p \wedge \chi\phi \\
&= p \wedge \chi_\top
\end{aligned}
$$