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;
    };
}
*)
```

Lab2 Solution: decideRPSGameResult (Postcondition)

```
CONSTANT rounds
(* --algorithm decideRPSGame {
    variables
        /* Define two total functions. */
        p1 \in [1..rounds -> {"R", "P", "S"}],
        p2 \in [1..rounds -> {"R", "P", "S"}],
        roundsWonByP1 = 0,
        roundsWonByP2 = 0,
        i = 1,
        p1win = 0,
        p2win = 0,
        tie = 0;
    define {
        NumberOfWins(player, opponent) == Cardinality({
            j \in 1..rounds :
                \/ player[j] = "R" /\ opponent[j] = "S"
                \/ player[j] = "P" /\ opponent[j] = "R"
                \/ player[j] = "S" /\ opponent[j] = "P"
        })
    };
```

```
    {
        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;
    }
} *)
```

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]);
    }
}
*)
```

input: [23, 46, 69]

result:

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

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]]