

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)

initial not deterministic choices for players in the 1st round

non-deterministic choices for players in the next round

```
(*  
--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)

2

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"
    });
}
set comprehension.
* p1[i] = "R" \& p2[i] = "S"
* p1[i] = "P" \& p2[i] = "R"
* p1[i] = "S" \& p2[i] = "P"
* p1[i] = "R" \& p2[i] = "R" none of the disjoint is satisfied
* p1[i] = "P" \& p2[i] = "P" with function application
```

total function
domain
range
e.g. rounds = 2
what plays
play in all rounds
 $p1 = [1 \mapsto "R", 2 \mapsto "P"]$
 $p2 = [1 \mapsto "S", 2 \mapsto "S"]$

possible ways
p1 (player)
can win
p2 (opponent)

```
{
    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 prog. 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]);
    }
}
*)

index 1 2 3
input: [23, 46, 69]
result: index
[[23, 46, 69], 1
[46, 69], 2
[69]] 3
```

$$\begin{aligned}len(r[j]) &= \\len(\text{input}) + 1 - j\end{aligned}$$

$$j *$$

$$j + len(r[j]) \\= \\len(\text{input}) + 1$$

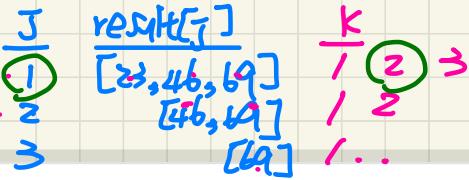
<u>J</u>	<u>len(result[j])</u>	<u>len(input)</u>
1	3	3
2	2	3
3	1	3

PostCondition

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

(2) Practise 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 $\forall j \in 1..Len(input): Len(result[j]) = Len(input) - j + 1$;

assert $\forall j \in 1..Len(result): (\forall k \in 1..Len(result[j]): result[j][k] = input[j - 1 + k])$;

}

*

$j+k = 2+1=3$ [??]
 $j+k = 2+2=4$ [??]
 $j+k-1$??

seg of int.

another sp

e.g. $result[1][2] \rightarrow 46$.

J
1
2
3

result[J]
[23, 46, 69]
* [46, 69]
[69]

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

K
1
2
3

*	J	K	?

$result[2][1] = input[2]$

$result[2][2] = input[3]$

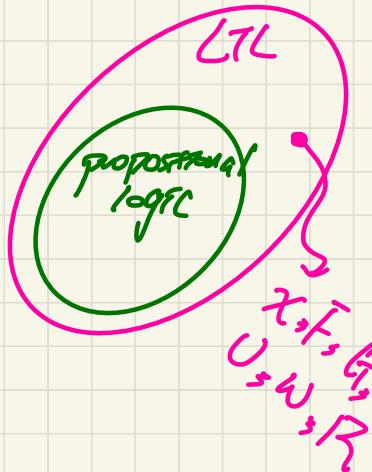
LTL Syntax: Context-Free Grammar

$\phi ::=$	T	[true]
	\perp	[false]
	p	[propositional atom]
	$(\neg\phi)$	[logical negation]
	$(\phi \wedge \phi)$	[logical conjunction]
	$(\phi \vee \phi)$	[logical disjunction]
	$(\phi \Rightarrow \phi)$	[logical implication]
	$(X\phi)$	[next state]
	$(F\phi)$	[some Future state]
	$(G\phi)$	[all future states (Globally)]
	$(\phi U \phi)$	[Until]
	$(\phi W \phi)$	[Weak-until]
	$(\phi R \phi)$	[Release]

implicitly use
and/or \exists

LTL

propositional
logic



↙

Start symbol

ϕ	::=	T	\top	$\textcircled{4}$	
		\perp			
$p(z)$					
$(\neg\phi)$		$\textcircled{1}$			
$(\phi \wedge \phi)$					[propositional atom]
$(\phi \vee \phi)$					[logical negation]
$(\phi \Rightarrow \phi)$					[logical conjunction]
$(X\phi)$		$\textcircled{2}$			[logical disjunction]
$(F\phi)$					[logical implication]
$(G\phi)$					[next state]
$(\phi U \phi)$					[some Future state]
$(\phi W \phi)$					[all future states (Globally)]
$(\phi R \phi)$					

				$[\text{true}]$
				$[\text{false}]$
				[propositional atom]
				[logical negation]
				[logical conjunction]
				[logical disjunction]
				[logical implication]
				[next state]
				[some Future state]
				[all future states (Globally)]
				[Until]
				[Weak-until]
				[Release]

(top-down) derivation:

$$P \wedge X_T$$

syntactically correct

$$\begin{aligned} \phi &= \phi \wedge \phi \\ &= P \wedge \phi \\ &= P \wedge X\phi \\ &= P \wedge X_T \end{aligned}$$