TLA+/PlusCal language basics

A. Jesse Jiryu Davis

TLA+ Syntax

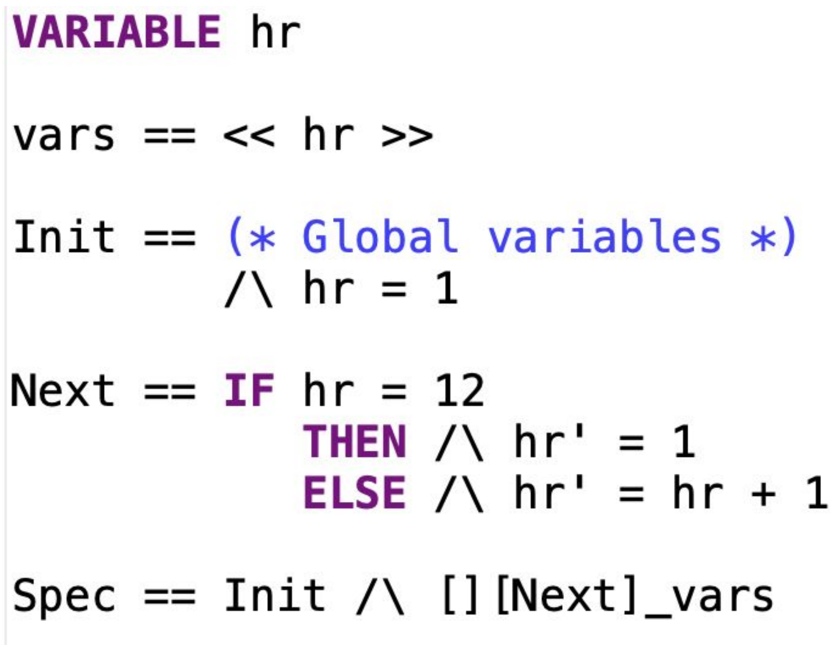
Cheatsheets

Keep these tabs open:

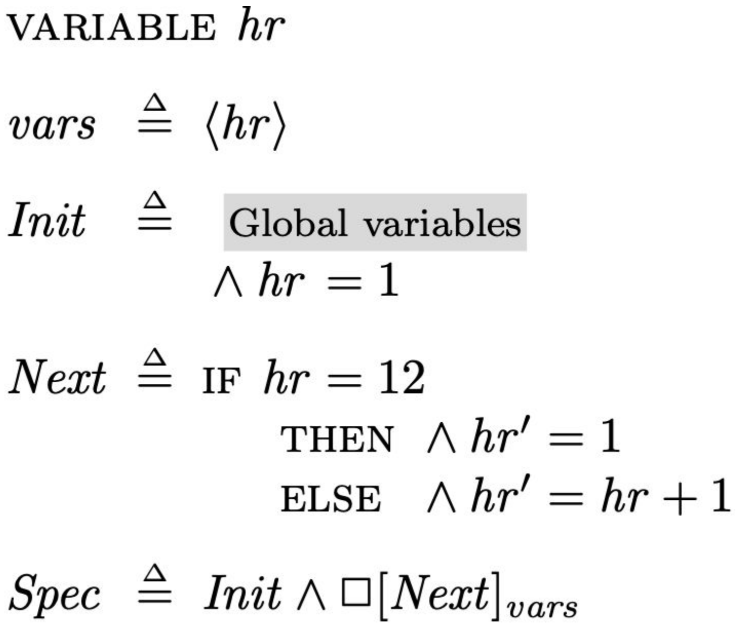
- Operators and values: learntla.com/core/operators.html
- Structured data: learntla.com/core/functions.html
- Lamport's summary: lamport.azurewebsites.net/tla/summary.pdf

  ^ uses the formatted representation, not ASCII

Formatted vs ASCII



```
VARIABLE hr

vars == << hr >>

Init == (* Global variables *)
        /\ hr = 1

Next == IF hr = 12
           THEN /\ hr' = 1
           ELSE /\ hr' = hr + 1

Spec == Init /\ [][Next]_vars
```



VARIABLE hr

$$vars \triangleq \langle hr \rangle$$

$Init \triangleq$ Global variables

$$\land\, hr = 1$$

$$Next \triangleq \text{IF } hr = 12 \text{ THEN } \land\, hr' = 1 \text{ ELSE } \land\, hr' = hr + 1$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

EXTENDS

```
EXTENDS Integers, Sequences
```

Sets

```
S == {1, 2, 3}

3 \in S = TRUE; 4 \in S = FALSE;

{2, 3} \subseteq S = TRUE; {2, 4} \subseteq S = FALSE; S \subset S = TRUE

S \union {3, 4} = {1, 2, 3, 4}

S \ {3, 4} = {1, 2}

UNION {{1}, {2}} = {1, 2}

SUBSET {1, 2} = {{}, {1}, {2}, {1, 2}}

{x*x : x \in S} = {1, 4, 9}

{x*y : x \in S, y \in S} = {1, 2, 3, 4, 6, 9}

{x \in S : x >= 2} = {3}
```

Functions

A function maps each element in its domain to at most one element in its range.

Function F with domain S:

```
F == [x \in S |-> expr]
```

The domain and range are sets. For every x, the value of F is "expr".

Functions can act like structs or hashtables (stay tuned).

Functions are powerful

```
S = 1..10  \* set of numbers 1 through 10 inclusive

F == [x \in S |-> x + 1]

F[ 4 ] = 5

F == [x \in S, y \in S |-> x + y]

F[2, 3] = 5

\* F's domain is cross-product of S and S

DOMAIN F = S \X S
```

Tuples/Sequences (are functions)

```
T == <<"a", "b", "c", "d">>

T[ 1 ] = "a"

DOMAIN T = 1..4  \* domain is the set 1 through 4 inclusive

Len(T) = 4

Head(T) = "a"

Tail(T) = <<"b", "c", "d">>
```

Tuples/Sequences (are functions)

```
T == <<"a", "b", "c", "d">>

<<"e">> \o T = <<"e", "a", "b", "c", "d">>

Append(T, "e") = <<"a", "b", "c", "d", "e">>

SubSeq(T, 2, 3) = <<"b", "c">>

SelectSeq(T, LAMBDA x: x \in {"a", "d"}) = <<"a", "d">>
```

Structures (are functions)

```
S == [ from |-> 1 , val |-> 2 ]

S == [ from:1 , val:2 ]

DOMAIN S = { "from", "val"}

S[ "from" ] = 1

S.b = 2
```

Structures are especially useful to represent messages between nodes

Operators

```
EXTENDS Integers

MinutesToSeconds(m) == m * 60
```

Operators can be higher-order

```
Sum(a, b) == a + b

Do(op(,), a, b) == op(a, b)

Do(Sum, 1, 2) = 3

IsCommutative(f(_, _), a, b) == f(a, b) = f(b, a)

IsCommutative(Sum, 23, 42) = TRUE
```

Operators can be higher-order

```
RECURSIVE IsOdd(_)

IsOdd(n) == CASE n = 1 -> TRUE
            [] n = 0 -> FALSE
            [] OTHER -> IsOdd(n-2)

IsOdd(3) = TRUE
```

Macros

```
define Add(x, y) == x + y
```

PlusCal

```
---- MODULE TwoProcesses ----

EXTENDS TLC \* for "print"

(* --algorithm TwoProcesses {

process (pid \in {1, 2}) {
start_process:
    print <<"Process ID: ", self>>;
}

} -- *)
```

```
VARIABLE pc
vars == << pc >>
ProcSet == ({1, 2})
Init == /\ pc = [self \in ProcSet |-> "start_process"]
start_process(self) == /\ pc[self] = "start_process"
                       /\ PrintT(<<"Process ID: ", self>>)
                       /\ pc' = [pc EXCEPT ![self] = "Done"]

pid(self) == start_process(self)

Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == (\E self \in {1, 2}: pid(self))
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

```
VARIABLE pc
vars == << pc >>
ProcSet == ({1, 2})
Init == /\ pc = [self \in ProcSet |-> "start_process"]
start_process(self) == /\ pc[self] = "start_process"
                       /\ PrintT(<<"Process ID: ", self>>)
                       /\ pc' = [pc EXCEPT ![self] = "Done"]

pid(self) == start_process(self)

Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == (\E self \in {1, 2}: pid(self))
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

```
VARIABLE pc
vars == << pc >>
ProcSet == ({1, 2})
Init == /\ pc = [self \in ProcSet |-> "start_process"]
start_process(self) == /\ pc[self] = "start_process"
                       /\ PrintT(<<"Process ID: ", self>>)
                       /\ pc' = [pc EXCEPT ![self] = "Done"]

pid(self) == start_process(self)

Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == (\E self \in {1, 2}: pid(self))
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

```
VARIABLE pc
vars == << pc >>
ProcSet == ({1, 2})
Init == /\ pc = [self \in ProcSet |-> "start_process"]
start_process(self) == /\ pc[self] = "start_process"
                       /\ PrintT(<<"Process ID: ", self>>)
                       /\ pc' = [pc EXCEPT ![self] = "Done"]

pid(self) == start_process(self)

Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == (\E self \in {1, 2}: pid(self))
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

```
VARIABLE pc
vars == << pc >>
ProcSet == ({1, 2})
Init == /\ pc = [self \in ProcSet |-> "start_process"]
start_process(self) == /\ pc[self] = "start_process"
                       /\ PrintT(<<"Process ID: ", self>>)
                       /\ pc' = [pc EXCEPT ![self] = "Done"]

pid(self) == start_process(self)

Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == (\E self \in {1, 2}: pid(self))
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

Pro tips for TLA+/PlusCal modeling

Labels are important

Labels are not cosmetic; they determine the granularity of steps

- Each chunk of PlusCal between labels becomes one TLA+ action
- Don't think "label", think "action"
- TLA+ conversion of PlusCal uses the labels: If you forget a label where it is syntactically needed, compiler will complain
- TLA+ does not allow for updating the same variable twice in one action
- Consider what's atomic in real life: e.g., a node can't send a message to two peers at once

Write a TypeOK invariant

- TLA+ is untyped, it's just math
- As interpreted by TLC, TLA+ is dynamic-typed
- Writing a TypeOK invariant can catch nefarious bugs quickly

"Bait invariant" is your friend

If you get no errors, maybe your spec does nothing, or your invariant is trivially true, or some other bug.

"Always be suspicious of success" —Lamport

Sometimes you want to get an error trace to check if the model is running as you intended to. For this, write a fake invariant, and see it violated.

Limit the model-checking domain

In order to limit the model checking domain, use a constant value to limit some spec variable.

We use this a lot when model-checking MongoDB replication specs:

MCMongoReplReconfig.cfg:

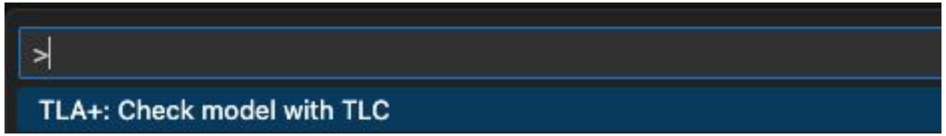
```
CONSTANTS
\* The number of election terms to
\* simulate during model-checking.
MaxTerm = 3

\* The longest oplog any server can reach
\* during model-checking.
MaxLogLen = 2

\* The number of reconfigs allowed during
\* model-checking.
MaxConfigVersion = 3

\* The number of commit points advanced
\* during model-checking.
MaxCommittedEntries = 3

\* Constrain the model to be finite
CONSTRAINT StateConstraint
```

MCMongoReplReconfig.tla:

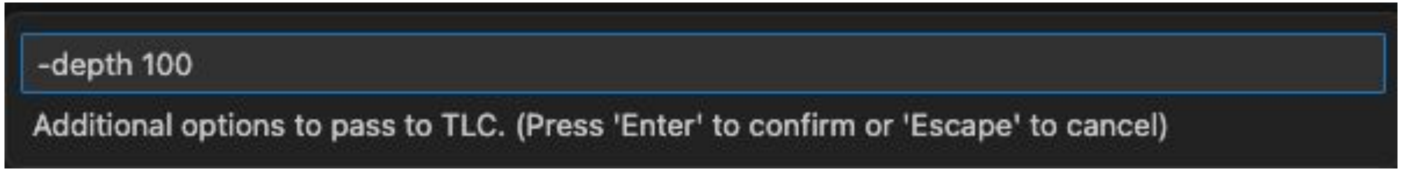
```
StateConstraint == \A s \in Server :
  /\ currentTerm[s] <= MaxTerm
  /\ Len(log[s]) <= MaxLogLen
  /\ configVersion[s] <= MaxConfigVersion
  /\ Cardinality(immediatelyCommitted)
         <= MaxCommittedEntries
```

Limit the model-checking domain

VSCode:



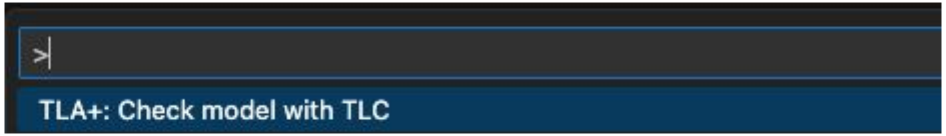
```
>
TLA+: Check model with TLC
```



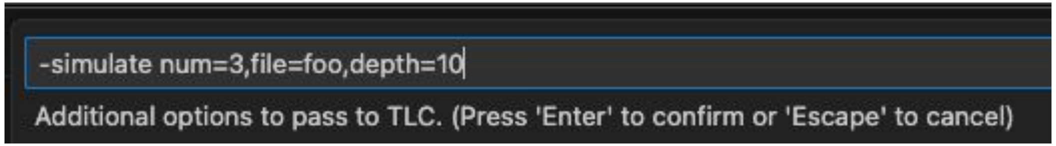
```
-depth 100
Additional options to pass to TLC. (Press 'Enter' to confirm or 'Escape' to cancel)
```

Limit the model-checking domain

VSCode:



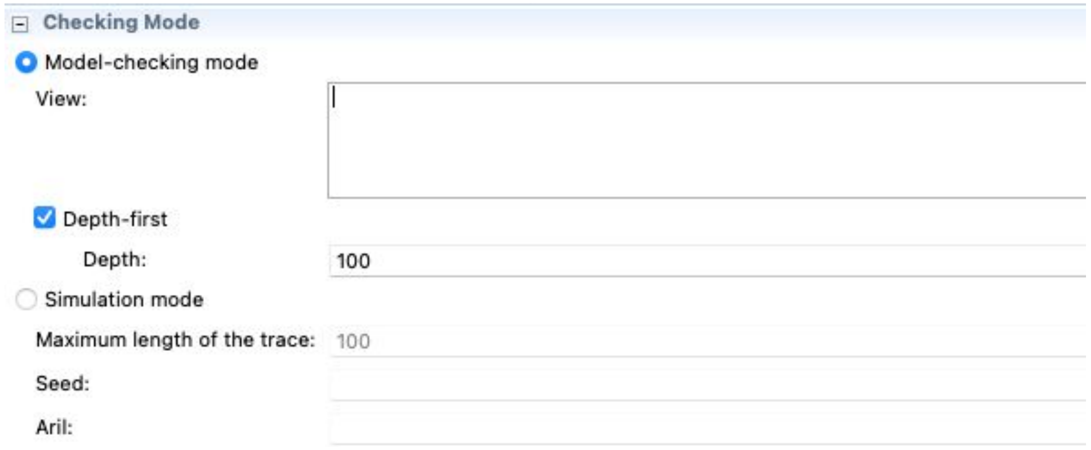
```
>
TLA+: Check model with TLC
```



```
-simulate num=3,file=foo,depth=10
Additional options to pass to TLC. (Press 'Enter' to confirm or 'Escape' to cancel)
```

Limit the model-checking domain

TLA+ Toolbox:



Checking Mode

[● Model-checking mode]

View:

[✓] Depth-first

Depth: 100

○ Simulation mode

Maximum length of the trace: 100

Seed:

Aril:

Warning about the "states" directory

The "states" directory contains all states for all runs of the model checker, can easily grow to gigabytes.

Delete your "states" directories in your TLA+ projects occasionally to reclaim disk.

The TLA+ way

TLA+ is math. It's powerful and very general; gives you too many options.

You need to study examples to learn about the right ways of doing things.

The TLA+ way

TLA+ is math. It's powerful and very general; gives you too many options.

You need to study examples to learn about the right ways of doing things.

Novices write procedural/imperative rather than declarative/math model, which is very slow to model-check.

- Write with macros, instead of PlusCal procedures.
- For message passing, use structure-typed messages which are written just once to a shared message board; other nodes can react to them by just reading, not consuming them.
- Reduce the state footprint of the model to avoid state explosion problem.

Read lots of examples

https://github.com/mongodb/mongo/blob/master/src/mongo/tla_plus/

https://code.amazon.com/packages/TLA_Seminar_Examples

https://muratbuffalo.blogspot.com/search?q=tla

https://github.com/tlaplus/Examples

Discussion

- What did you find surprising about TLA+/Pluscal syntax?
- What did you find surprising about TLA+/Pluscal model/paradigm?
- What will you choose as an entry project to model in TLA+/Pluscal?

Resources

Slack channel: #tla

https://learntla.com

https://groups.google.com/g/tlaplus

TLA+ model examples:

- https://code.amazon.com/packages/TLA_Seminar_Examples
- https://muratbuffalo.blogspot.com/search?q=tla
- https://github.com/tlaplus/Examples
- https://github.com/mongodb/mongo/blob/master/src/mongo/tla_plus/