TLA+ Tools:
практичный инструмент
формальной верификации алгоритмов

(или что ещё разработчику на Golang точно надо изучить во время карантина)

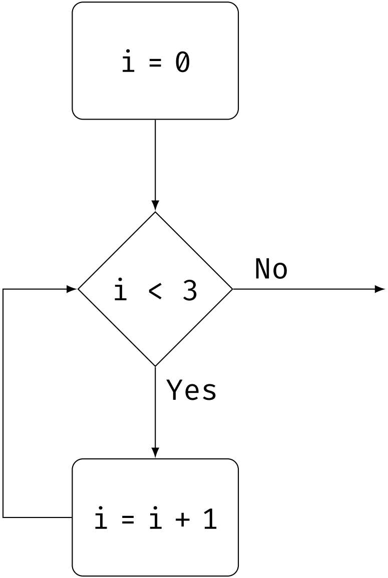
GopherCon Russia 2020
Alexey Naidyonov
ITooLabs
Mar 28 2020

ITOOLABS

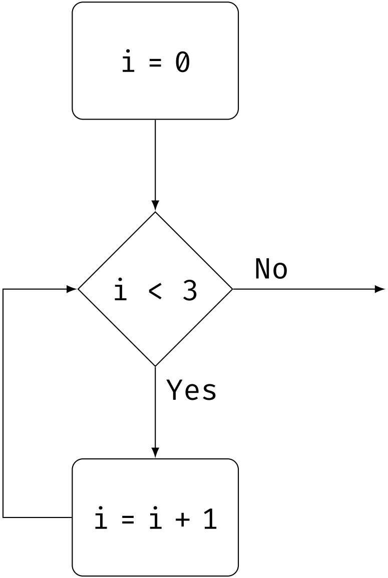
Что такое программа?

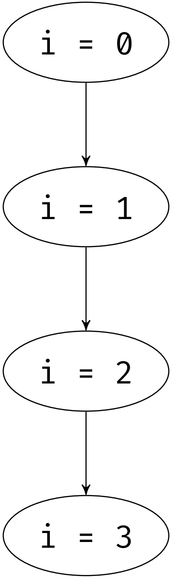
```
var i = 0
for i < 3 {
   i = i + 1
}
```

```
var i = 0
for i < 3 {
   i = i + 1
}
```



```
var i = 0
for i < 3 {
   i = i + 1
}
```





Process 1

```
a = b + 1
b = a + 1
```

```
a = 0
b = 0
```

Process 2

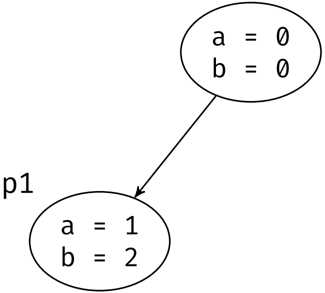
```
b = a + 1
a = b + 1
```

Process 1

```
a = b + 1
b = a + 1
```

Process 2

```
b = a + 1
a = b + 1
```

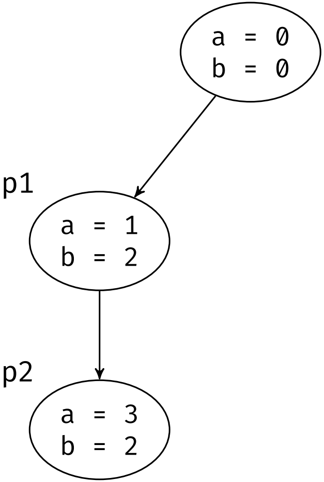


Process 1

```
a = b + 1
b = a + 1
```

Process 2

```
b = a + 1
a = b + 1
```

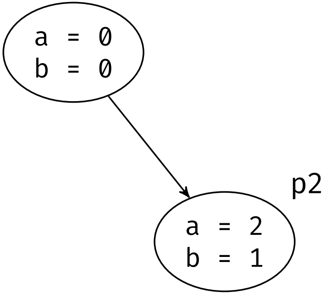


Process 1

```
a = b + 1
b = a + 1
```

Process 2

```
b = a + 1
a = b + 1
```

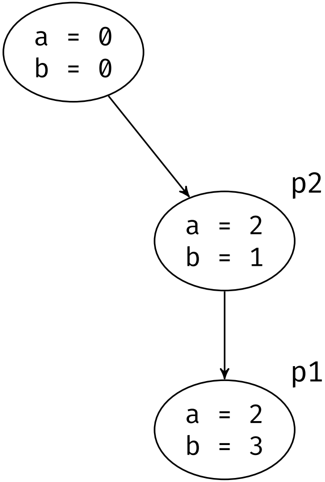


Process 1

```
a = b + 1
b = a + 1
```

Process 2

```
b = a + 1
a = b + 1
```

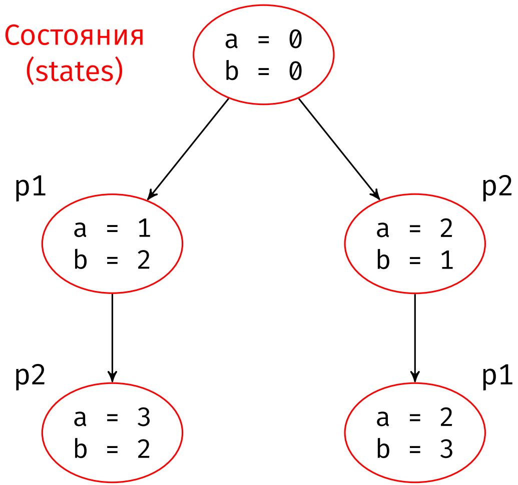


Process 1

```
a = b + 1
b = a + 1
```

Process 2

```
b = a + 1
a = b + 1
```

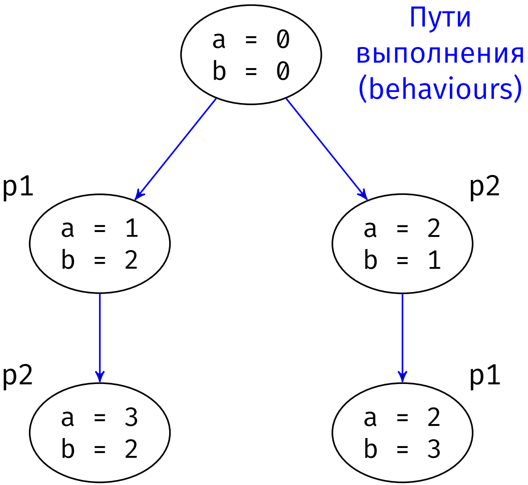


Process 1

```
1 a = b + 1
2 b = a + 1
```

Process 2

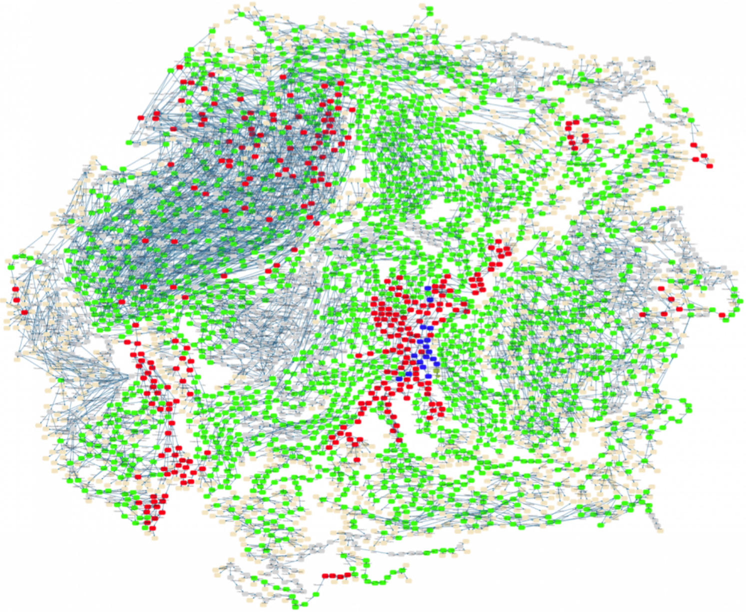
```
1 b = a + 1
2 a = b + 1
```



$$(m \cdot n)\frac{(m \cdot n)!}{(n!)^m},$$

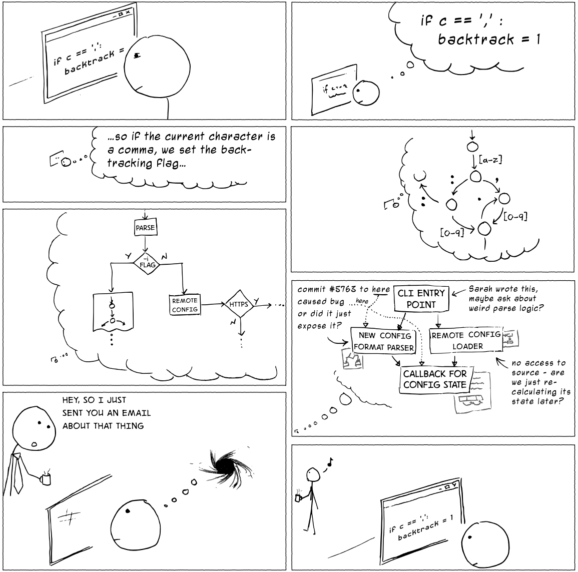
где

m - количество процессов
n - длина пути выполнения



CAN Bus state space visualization
https://www3.hhu.de/stups/prob/index.php/State_space_visualization_examples

THIS IS WHY YOU SHOULDN'T INTERRUPT A PROGRAMMER

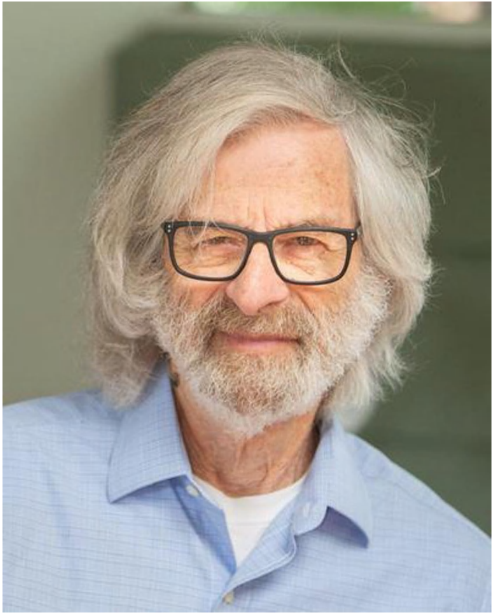


“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

— Leslie Lamport, May 1987

Лесли Лэмпорт
(Leslie Lamport)

- Lamport timestamps
- Bakery algorithm
- PAXOS
- LaTeX
- **TLA+**



Temporal Logic of Actions

Математическая логика

Математическая логика

$A \wedge B$	Коньюнкция

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \to B$	Импликация

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \to B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always
$A \leadsto B$	Leads to

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always
$A \leadsto B$	Leads to
$\Box\Diamond A$	

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always
$A \leadsto B$	Leads to
$\Box\Diamond A$	Infinitely often

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always
$A \leadsto B$	Leads to
$\Box\Diamond A$	Infinitely often
$\Diamond\Box A$	

Математическая логика

$A \wedge B$	Коньюнкция
$A \vee B$	Дизьюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\Diamond A$	Eventually
$\Box A$	Always
$A \leadsto B$	Leads to
$\Box \Diamond A$	Infinitely often
$\Diamond \Box A$	Eventually always

Математическая логика

$A \wedge B$	A/\B	Коньюнкция
$A \vee B$	A\/B	Дизьюнкция
$A \to B$	A=>B	Импликация
$\neg A$	~A	Отрицание

Темпоральная логика

$\Diamond A$	<>A	Eventually
$\Box A$	[ ]A	Always
$A \leadsto B$	~>A	Leads to
$\Box \Diamond A$	[ ]<>A	Infinitely often
$\Diamond \Box A$	<>[ ]A	Eventually always

MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

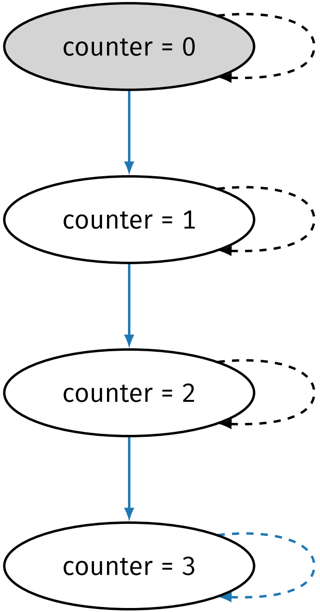
$$\text{Invariant} \triangleq \text{counter} \in \text{MinValue}\,..\,\text{MaxValue}$$

$$\text{Success} \triangleq \Diamond\Box(\text{counter} = \text{MaxValue})$$

$$\text{Init} \triangleq \text{counter} = \text{MinValue}$$

$$\text{Next} \triangleq \text{counter}' = \textsc{if}\ \text{counter} < \text{MaxValue}\ \textsc{then}\ \text{counter} + 1\ \textsc{else}\ \text{counter}$$

$$\text{Spec} \triangleq \land\ \text{Init} \\ \land\ \Box[\text{Next}]_{\text{counter}} \\ \land\ \text{WF}_{\text{counter}}(\text{Next})$$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME $MinValue < MaxValue$
VARIABLE counter

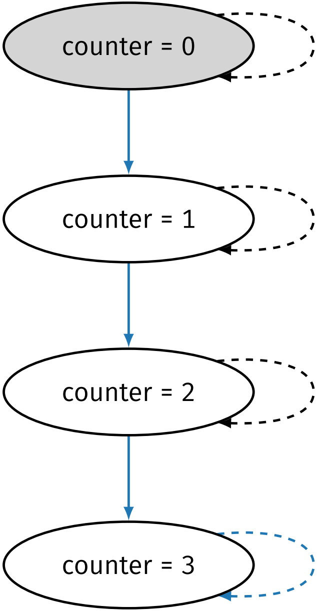
$Invariant \triangleq counter \in MinValue \,..\, MaxValue$

$Success \triangleq \Diamond\Box(counter = MaxValue)$

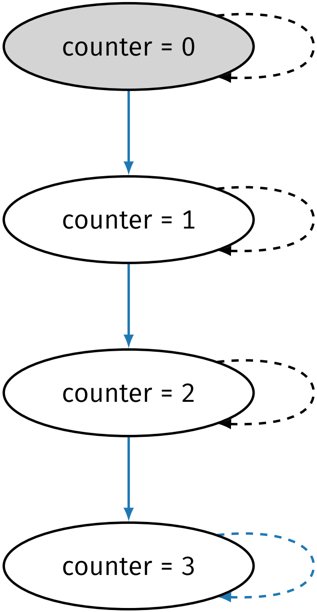
$Init \triangleq counter = MinValue$

$$Next \triangleq counter' = \textsc{if}\ counter < MaxValue\ \textsc{then}\ counter + 1\ \textsc{else}\ counter$$

$$Spec \triangleq \land\ Init \;\; \land\ \Box[Next]_{counter} \;\; \land\ WF_{counter}(Next)$$



```
─────────────── MODULE 00_Counter ───────────────
EXTENDS    Naturals
CONSTANTS  MinValue, MaxValue
ASSUME     MinValue < MaxValue
VARIABLE   counter

Invariant ≜ counter ∈ MinValue . . MaxValue

Success  ≜ ◇□(counter = MaxValue)

Init  ≜ counter = MinValue

Next ≜ counter′ = IF counter < MaxValue
                  THEN counter + 1
                  ELSE counter
Spec ≜ ∧ Init
       ∧ □[Next]_counter
       ∧ WF_counter(Next)
─────────────────────────────────────────────────
```



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue < MaxValue
VARIABLE counter

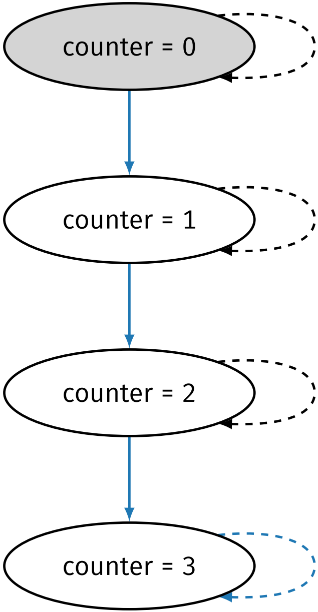
$$\text{Invariant} \triangleq \text{counter} \in \text{MinValue} \,..\, \text{MaxValue}$$

$$\text{Success} \triangleq \Diamond\Box(\text{counter} = \text{MaxValue})$$

$$\text{Init} \triangleq \text{counter} = \text{MinValue}$$

$$\text{Next} \triangleq \text{counter}' = \textsc{if}\ \text{counter} < \text{MaxValue}\ \textsc{then}\ \text{counter} + 1\ \textsc{else}\ \text{counter}$$

$$\text{Spec} \triangleq \land\ \text{Init} \quad \land\ \Box[\text{Next}]_{\text{counter}} \quad \land\ \text{WF}_{\text{counter}}(\text{Next})$$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

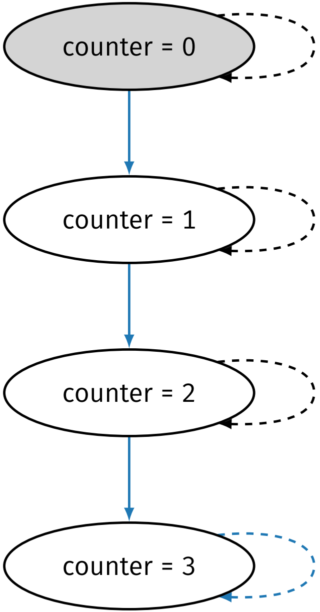
$Invariant \triangleq counter \in MinValue \,..\, MaxValue$

$Success \triangleq \Diamond\Box(counter = MaxValue)$

$Init \triangleq counter = MinValue$

$$
\begin{aligned}
Next \triangleq counter' = \ & \textsf{IF } counter < MaxValue \\
& \textsf{THEN } counter + 1 \\
& \textsf{ELSE } counter
\end{aligned}
$$

$$
\begin{aligned}
Spec \triangleq \ & \land\ Init \\
& \land\ \Box[Next]_{counter} \\
& \land\ WF_{counter}(Next)
\end{aligned}
$$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

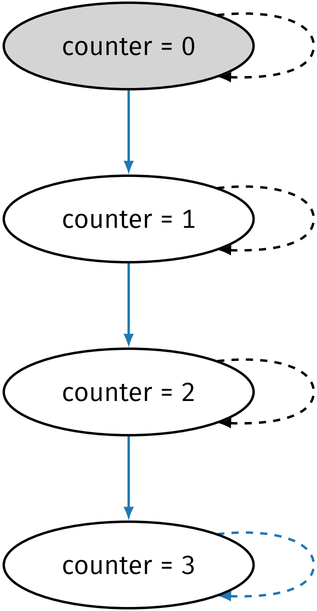
Invariant $\triangleq$ counter $\in$ MinValue . . MaxValue

Success $\triangleq \Diamond\Box$(counter $=$ MaxValue)

Init $\triangleq$ counter $=$ MinValue

Next $\triangleq$ counter$'$ $=$ IF counter $<$ MaxValue
THEN counter + 1
ELSE counter

Spec $\triangleq$ $\land$ Init
$\land$ $\Box$[Next]$_{\text{counter}}$
$\land$ WF$_{\text{counter}}$(Next)



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

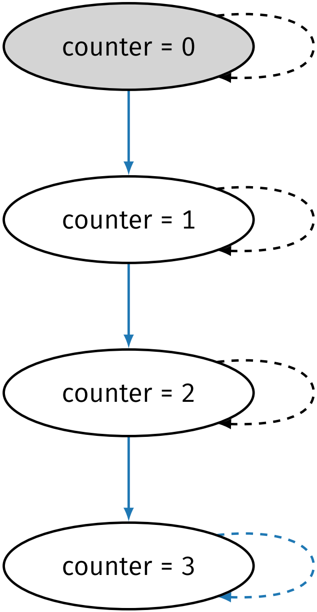
Invariant $\triangleq$ counter $\in$ MinValue . . MaxValue

Success $\triangleq \Diamond\Box$(counter $=$ MaxValue)

Init $\triangleq$ counter $=$ MinValue

Next $\triangleq$ counter$'$ $=$ IF counter $<$ MaxValue
THEN counter + 1
ELSE counter

Spec $\triangleq$ $\land$ Init
$\land$ $\Box$[Next]$_{\text{counter}}$
$\land$ WF$_{\text{counter}}$(Next)



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

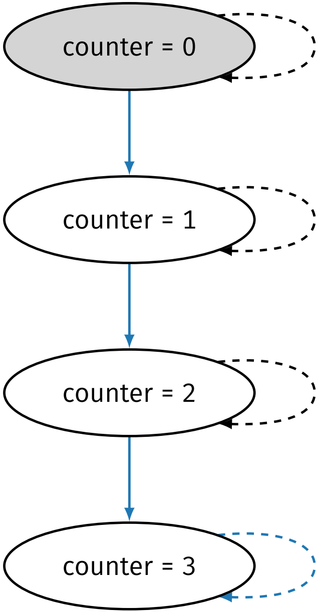
$Invariant \triangleq counter \in MinValue \,..\, MaxValue$

$Success \triangleq \Diamond\Box(counter = MaxValue)$

$Init \triangleq counter = MinValue$

$$
\begin{aligned}
Next \triangleq counter' = \ & \textsc{if}\ counter < MaxValue \\
& \textsc{then}\ counter + 1 \\
& \textsc{else}\ counter
\end{aligned}
$$

$$
\begin{aligned}
Spec \triangleq\ & \land\ Init \\
& \land\ \Box[Next]_{counter} \\
& \land\ WF_{counter}(Next)
\end{aligned}
$$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue < MaxValue
VARIABLE counter

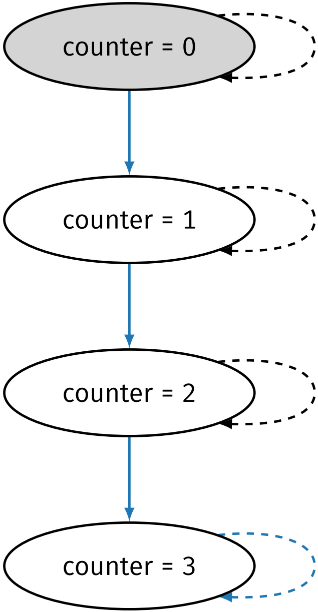
$\text{Invariant} \triangleq \text{counter} \in \text{MinValue} \,..\, \text{MaxValue}$

$\text{Success} \triangleq \Diamond\Box(\text{counter} = \text{MaxValue})$

$\text{Init} \triangleq \text{counter} = \text{MinValue}$

$\text{Next} \triangleq \text{counter}' = \textsc{if}\ \text{counter} < \text{MaxValue}\ \textsc{then}\ \text{counter} + 1\ \textsc{else}\ \text{counter}$

$\text{Spec} \triangleq \land\ \text{Init} \quad \land\ \Box[\text{Next}]_{\text{counter}} \quad \land\ \text{WF}_{\text{counter}}(\text{Next})$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

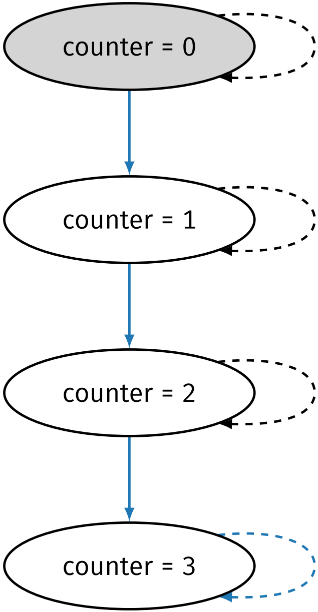
$$\text{Invariant} \triangleq \text{counter} \in \text{MinValue} \,..\, \text{MaxValue}$$

$$\text{Success} \triangleq \Diamond\Box(\text{counter} = \text{MaxValue})$$

$$\text{Init} \triangleq \text{counter} = \text{MinValue}$$

$$\begin{aligned}\text{Next} \triangleq \text{counter}' = \ & \textsc{if}\ \text{counter} < \text{MaxValue} \\ & \textsc{then}\ \text{counter} + 1 \\ & \textsc{else}\ \text{counter}\end{aligned}$$

$$\begin{aligned}\text{Spec} \triangleq\ & \land\ \text{Init} \\ & \land\ \Box[\text{Next}]_{\text{counter}} \\ & \land\ \text{WF}_{\text{counter}}(\text{Next})\end{aligned}$$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

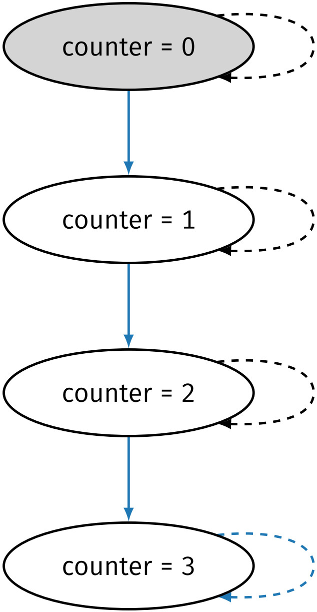
$Invariant \triangleq counter \in MinValue \,..\, MaxValue$

$Success \triangleq \Diamond\Box(counter = MaxValue)$

$Init \triangleq counter = MinValue$

$Next \triangleq counter' = \text{IF } counter < MaxValue \text{ THEN } counter + 1 \text{ ELSE } counter$

$Spec \triangleq \land\ Init$
$\qquad\quad \land\ \Box[Next]_{counter}$
$\qquad\quad \land\ WF_{counter}(Next)$



MODULE 00_Counter

EXTENDS Naturals
CONSTANTS MinValue, MaxValue
ASSUME MinValue $<$ MaxValue
VARIABLE counter

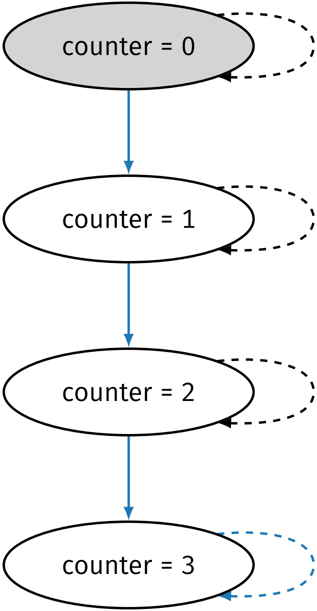
$$\text{Invariant} \triangleq \text{counter} \in \text{MinValue} \,..\, \text{MaxValue}$$

$$\text{Success} \triangleq \Diamond\Box(\text{counter} = \text{MaxValue})$$

$$\text{Init} \triangleq \text{counter} = \text{MinValue}$$

$$\text{Next} \triangleq \text{counter}' = \textsc{if}\ \text{counter} < \text{MaxValue}\ \textsc{then}\ \text{counter} + 1\ \textsc{else}\ \text{counter}$$

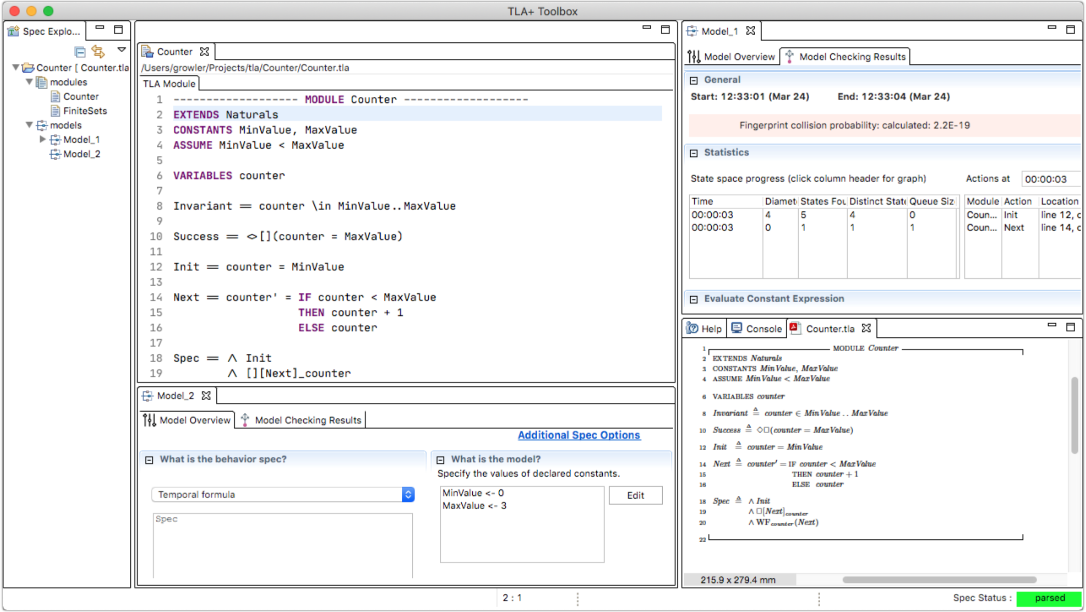
$$\text{Spec} \triangleq \land\ \text{Init} \quad \land\ \Box[\text{Next}]_{\text{counter}} \quad \land\ \text{WF}_{\text{counter}}(\text{Next})$$



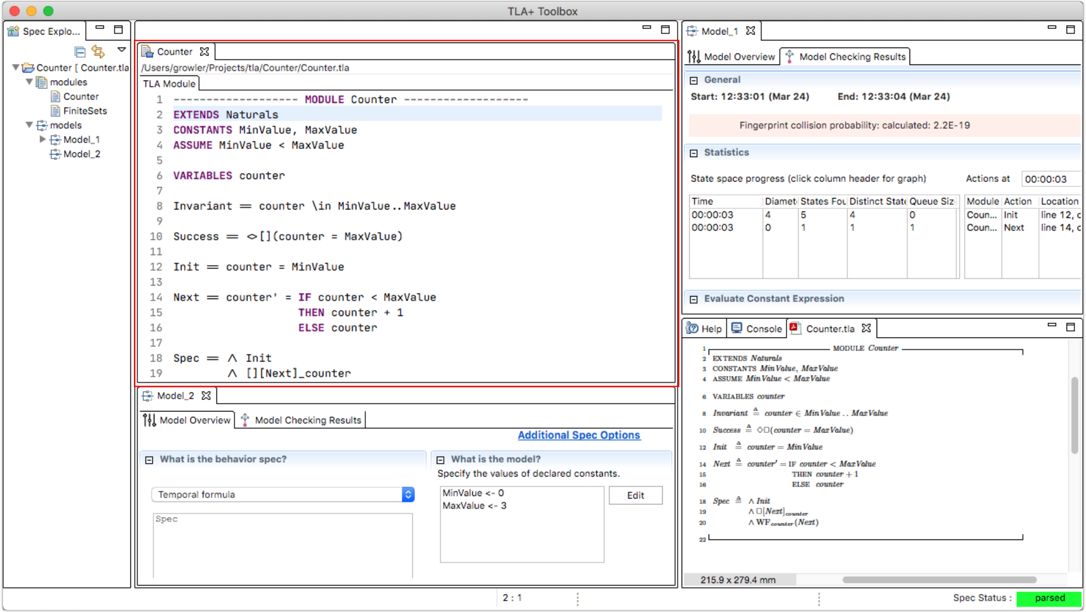
TLC

Explicit State Model Checker for TLA+

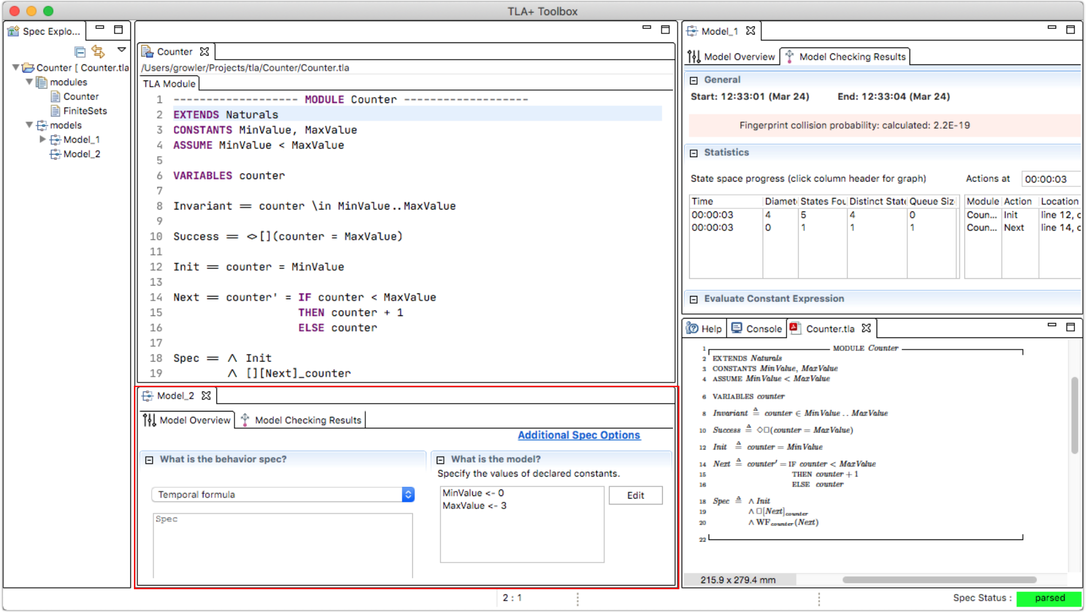
TLA+ Toolbox by Simon Zambrovski, et al.



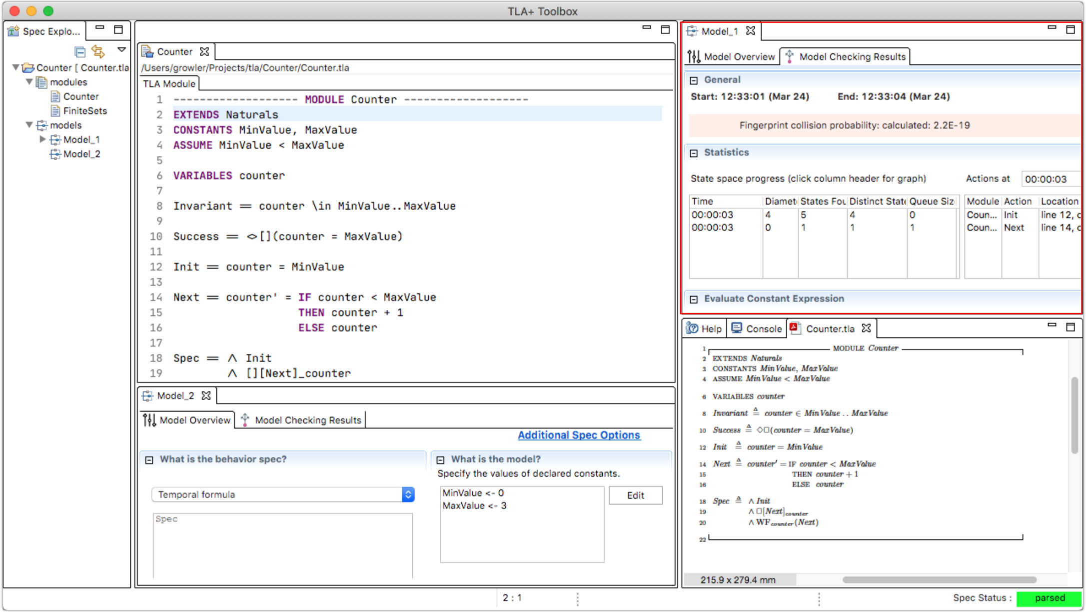
TLA+ Toolbox by Simon Zambrovski, et al.



TLA+ Toolbox by Simon Zambrovski, et al.



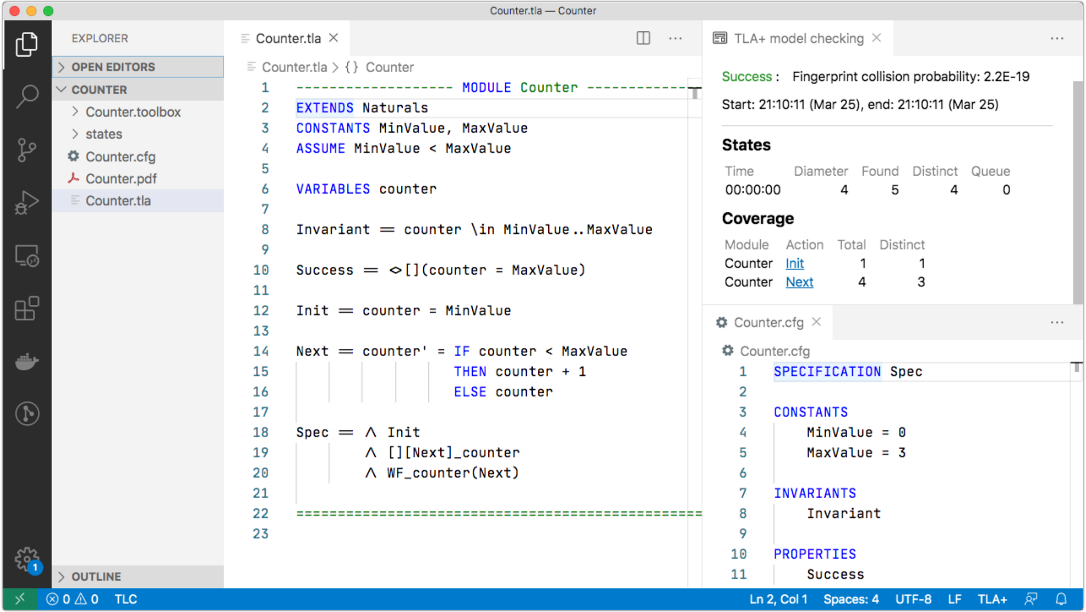
TLA+ Toolbox by Simon Zambrovski, et al.



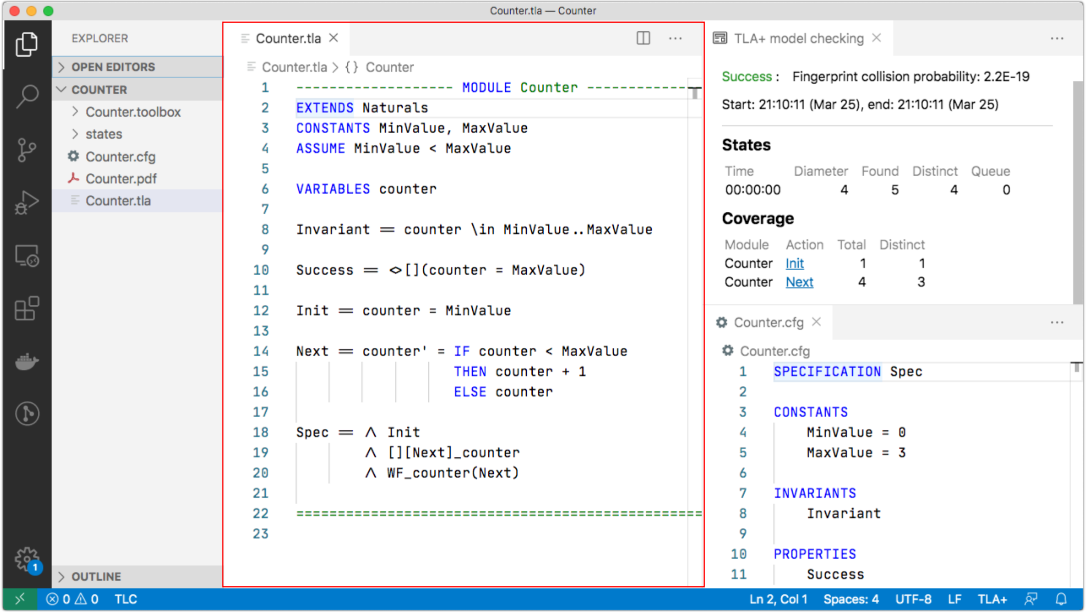
Time	Diamet	States Fou	Distinct Stat	Queue Siz
00:00:03	4	5	4	0
00:00:03	0	1	1	1

Module	Action	Location
Coun...	Init	line 12, c
Coun...	Next	line 14, c

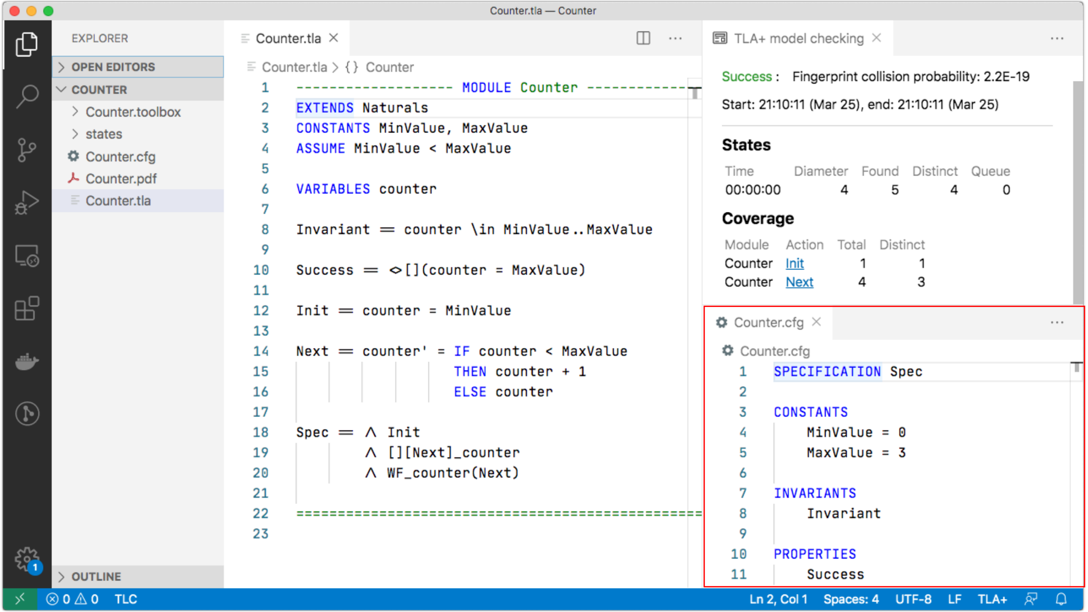
VS Code / TLA+ Support by Андрей Лыгин @alygin



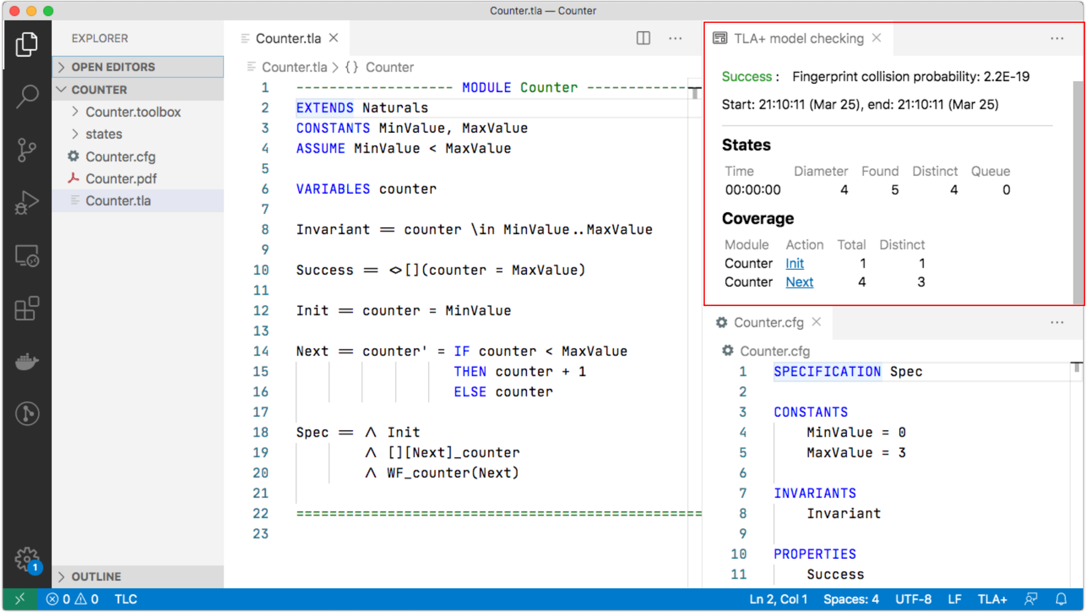
VS Code / TLA+ Support by Андрей Лыгин @alygin



VS Code / TLA+ Support by Андрей Лыгин @alygin



VS Code / TLA+ Support by Андрей Лыгин @alygin



ДЕМО

`00_Counter.tla`

PlusCal

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;

fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal

Алгоритм

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;
```

```
fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal

Переменные

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;
```

```
fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal

Определения TLA+

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;

fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal

Процессы

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;

fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal

Метки

```
(*--algorithm 01_PCDemo

variables
  a = 0,
  b = 0;

define
  Success == <>[](a+b = 5)
end define;

fair process proc1 = 1
begin p1:
  a := b + 1;
  b := a + 1;
end process;

fair process proc2 = 2
begin p2:
  b := a + 1;
  a := b + 1;
end process;

end algorithm; *)
```

PlusCal
Трансляция

```
\* BEGIN TRANSLATION
VARIABLES a, b, pc

(* define statement *)
Success == <>[](a+b = 5)

vars == << a, b, pc >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
        /\ a = 0
        /\ b = 0
        /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
                                        [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
\* BEGIN TRANSLATION
VARIABLES a, b, pc

(* define statement *)
Success == <>[](a+b = 5)

vars == << a, b, pc >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
        /\ a = 0
        /\ b = 0
        /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
                                        [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
\* BEGIN TRANSLATION
VARIABLES a, b, pc

(* define statement *)
Success == <>[](a+b = 5)

vars == << a, b, pc >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
        /\ a = 0
        /\ b = 0
        /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
                                       [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
\* BEGIN TRANSLATION
VARIABLES a, b, pc

(* define statement *)
Success == <>[](a+b = 5)

vars == << a, b, pc >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
        /\ a = 0
        /\ b = 0
        /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
                                        [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
p1 == /\ pc[1] = "p1"
      /\ a' = b + 1
      /\ b' = a' + 1
      /\ pc' = [pc EXCEPT ![1] = "Done"]

proc1 == p1

p2 == /\ pc[2] = "p2"
      /\ b' = a + 1
      /\ a' = b' + 1
      /\ pc' = [pc EXCEPT ![2] = "Done"]

proc2 == p2
```

PlusCal

Трансляция

```
p1 == /\ pc[1] = "p1"
      /\ a' = b + 1
      /\ b' = a' + 1
      /\ pc' = [pc EXCEPT ![1] = "Done"]

proc1 == p1

p2 == /\ pc[2] = "p2"
      /\ b' = a + 1
      /\ a' = b' + 1
      /\ pc' = [pc EXCEPT ![2] = "Done"]

proc2 == p2
```

PlusCal

Трансляция

```
p1 == /\ pc[1] = "p1"
      /\ a' = b + 1
      /\ b' = a' + 1
      /\ pc' = [pc EXCEPT ![1] = "Done"]

proc1 == p1

p2 == /\ pc[2] = "p2"
      /\ b' = a + 1
      /\ a' = b' + 1
      /\ pc' = [pc EXCEPT ![2] = "Done"]

proc2 == p2
```

PlusCal

Трансляция

```
(* Allow infinite stuttering to prevent deadlock on termination.
 ↪  *)
Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == proc1 \/ proc2
           \/ Terminating

Spec == /\ Init /\ [][Next]_vars
        /\ WF_vars(proc1)
        /\ WF_vars(proc2)

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

PlusCal

Трансляция

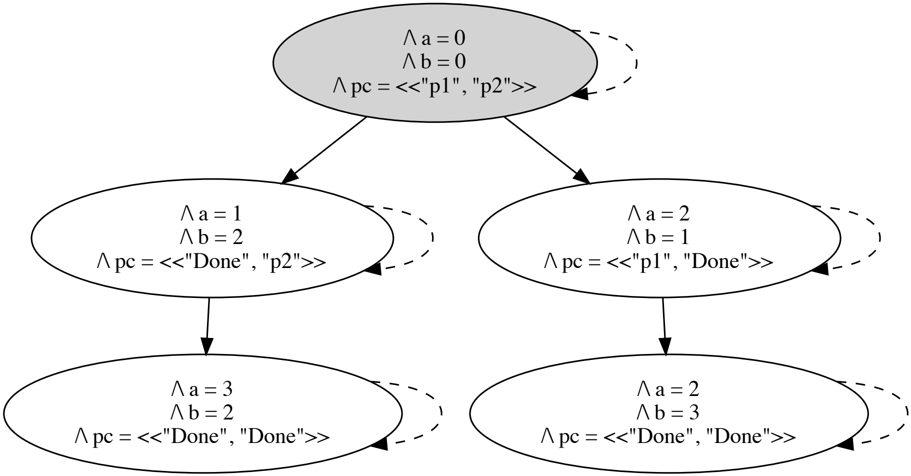
```
(* Allow infinite stuttering to prevent deadlock on termination.
 ↪  *)
Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
               /\ UNCHANGED vars

Next == proc1 \/ proc2
           \/ Terminating

Spec == /\ Init /\ [][Next]_vars
        /\ WF_vars(proc1)
        /\ WF_vars(proc2)

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

ДЕМО

`01_PCDemo.tla`

TLA+ Toolbox CLI

```
growler@macbook:$ tlc -dump dot,colorized 01_PCDemo.dot 01_PCDemo.tla
Computing initial states...
Finished computing initial states: 1 distinct state generated.
Checking 2 branches of temporal properties for the complete state space
  with 10 total distinct states
Finished checking temporal properties
Model checking completed. No error has been found.
  Estimates of the probability that TLC did not check all reachable states
  because two distinct states had the same fingerprint:
  calculated (optimistic):  val = 5.4E-19
7 states generated, 5 distinct states found, 0 states left on queue.
growler@macbook:$ dot -Tsvg 01_PCDemo.dot -o 01_PCDemo.svg
growler@macbook:$
```


/\ a = 0
/\ b = 0
/\ pc = <<"p1", "p2">>
/\ a = 1
/\ b = 2
/\ pc = <<"Done", "p2">>
/\ a = 2
/\ b = 1
/\ pc = <<"p1", "Done">>
/\ a = 3
/\ b = 2
/\ pc = <<"Done", "Done">>
/\ a = 2
/\ b = 3
/\ pc = <<"Done", "Done">>

PlusCal

Атомарность

```
(*--algorithm A

variables
    a = 0,
    b = 0;

define
    Success == <>[](
        (a + b) = 5
    )
end define;

fair process proc1 = 1
begin
    p1_1: a := b + 1;
    p1_2: b := a + 1;
end process;

fair process proc2 = 2
begin
    p2_1: b := a + 1;
    p2_2: a := b + 1;
end process;

end algorithm; *)
```

ДЕМО

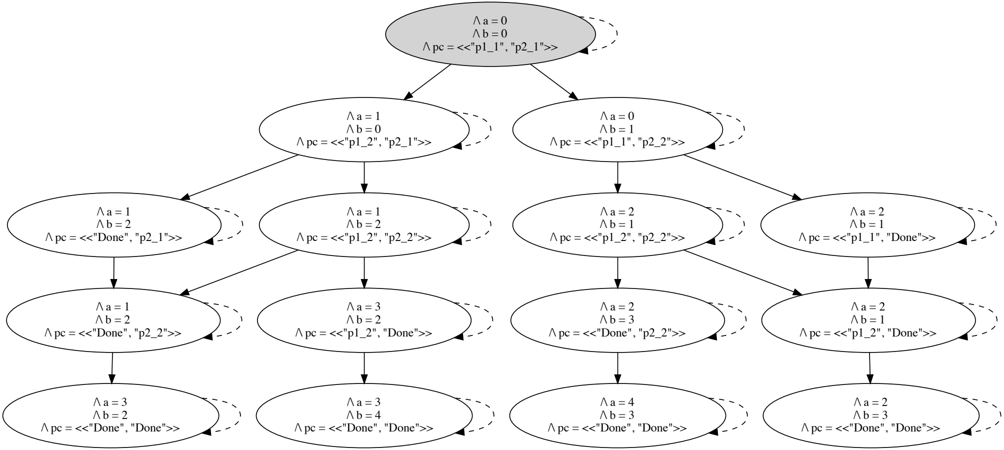
`02_PCDemo.tla`

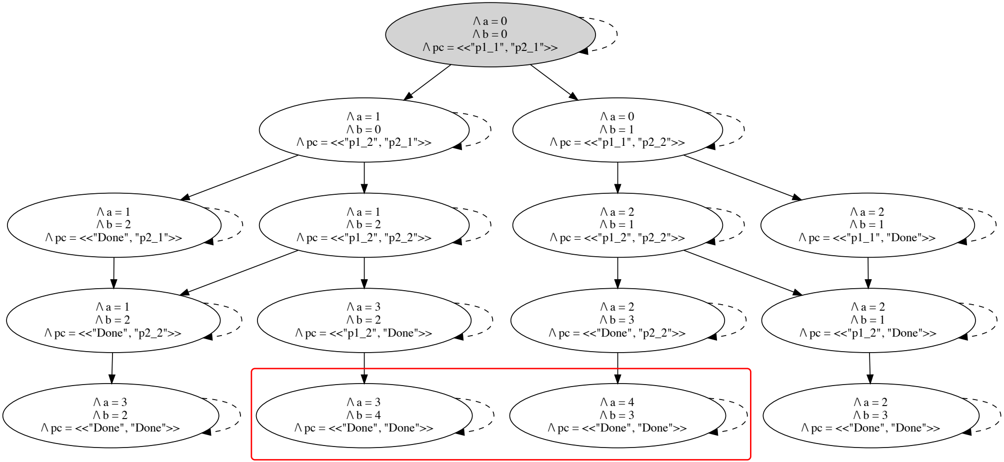
```
Error: Temporal properties were violated.

Error: The following behavior constitutes a counter-example:

State 1: <Initial predicate>
/\ a = 0
/\ b = 0
/\ pc = <<"p1_1", "p2_1">>

State 2: <p1_1 line 48, col 9 to line 51, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 0
/\ pc = <<"p1_2", "p2_1">>

State 3: <p2_1 line 60, col 9 to line 63, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 2
/\ pc = <<"p1_2", "p2_2">>

State 4: <p2_2 line 65, col 9 to line 68, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 2
/\ pc = <<"p1_2", "Done">>

State 5: <p1_2 line 53, col 9 to line 56, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 4
/\ pc = <<"Done", "Done">>
```

```
Error: Temporal properties were violated.

Error: The following behavior constitutes a counter-example:

State 1: <Initial predicate>
/\ a = 0
/\ b = 0
/\ pc = <<"p1_1", "p2_1">>

State 2: <p1_1 line 48, col 9 to line 51, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 0
/\ pc = <<"p1_2", "p2_1">>

State 3: <p2_1 line 60, col 9 to line 63, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 2
/\ pc = <<"p1_2", "p2_2">>

State 4: <p2_2 line 65, col 9 to line 68, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 2
/\ pc = <<"p1_2", "Done">>

State 5: <p1_2 line 53, col 9 to line 56, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 4
/\ pc = <<"Done", "Done">>
```

```
Error: Temporal properties were violated.

Error: The following behavior constitutes a counter-example:

State 1: <Initial predicate>
/\ a = 0
/\ b = 0
/\ pc = <<"p1_1", "p2_1">>

State 2: <p1_1 line 48, col 9 to line 51, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 0
/\ pc = <<"p1_2", "p2_1">>

State 3: <p2_1 line 60, col 9 to line 63, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 2
/\ pc = <<"p1_2", "p2_2">>

State 4: <p2_2 line 65, col 9 to line 68, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 2
/\ pc = <<"p1_2", "Done">>

State 5: <p1_2 line 53, col 9 to line 56, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 4
/\ pc = <<"Done", "Done">>
```

```
Error: Temporal properties were violated.

Error: The following behavior constitutes a counter-example:

State 1: <Initial predicate>
/\ a = 0
/\ b = 0
/\ pc = <<"p1_1", "p2_1">>

State 2: <p1_1 line 48, col 9 to line 51, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 0
/\ pc = <<"p1_2", "p2_1">>

State 3: <p2_1 line 60, col 9 to line 63, col 17 of module 02_PCDemo>
/\ a = 1
/\ b = 2
/\ pc = <<"p1_2", "p2_2">>

State 4: <p2_2 line 65, col 9 to line 68, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 2
/\ pc = <<"p1_2", "Done">>

State 5: <p1_2 line 53, col 9 to line 56, col 17 of module 02_PCDemo>
/\ a = 3
/\ b = 4
/\ pc = <<"Done", "Done">>
```


/\ a = 0
/\ b = 0
/\ pc = <<"p1_1", "p2_1">>
/\ a = 1
/\ b = 0
/\ pc = <<"p1_2", "p2_1">>
/\ a = 0
/\ b = 1
/\ pc = <<"p1_1", "p2_2">>
/\ a = 1
/\ b = 2
/\ pc = <<"Done", "p2_1">>
/\ a = 1
/\ b = 2
/\ pc = <<"p1_2", "p2_2">>
/\ a = 2
/\ b = 1
/\ pc = <<"p1_2", "p2_2">>
/\ a = 2
/\ b = 1
/\ pc = <<"p1_1", "Done">>
/\ a = 1
/\ b = 2
/\ pc = <<"Done", "p2_2">>
/\ a = 3
/\ b = 2
/\ pc = <<"p1_2", "Done">>
/\ a = 2
/\ b = 3
/\ pc = <<"Done", "p2_2">>
/\ a = 2
/\ b = 1
/\ pc = <<"p1_2", "Done">>
/\ a = 3
/\ b = 2
/\ pc = <<"Done", "Done">>
/\ a = 3
/\ b = 4
/\ pc = <<"Done", "Done">>
/\ a = 4
/\ b = 3
/\ pc = <<"Done", "Done">>
/\ a = 2
/\ b = 3
/\ pc = <<"Done", "Done">>



/\ a = 0 /\ b = 0 /\ pc = <<"p1_1", "p2_1">>
/\ a = 1 /\ b = 0 /\ pc = <<"p1_2", "p2_1">>
/\ a = 0 /\ b = 1 /\ pc = <<"p1_1", "p2_2">>
/\ a = 1 /\ b = 2 /\ pc = <<"Done", "p2_1">>
/\ a = 1 /\ b = 2 /\ pc = <<"p1_2", "p2_2">>
/\ a = 2 /\ b = 1 /\ pc = <<"p1_2", "p2_2">>
/\ a = 2 /\ b = 1 /\ pc = <<"p1_1", "Done">>
/\ a = 1 /\ b = 2 /\ pc = <<"Done", "p2_2">>
/\ a = 3 /\ b = 2 /\ pc = <<"p1_2", "Done">>
/\ a = 2 /\ b = 3 /\ pc = <<"Done", "p2_2">>
/\ a = 2 /\ b = 1 /\ pc = <<"p1_2", "Done">>
/\ a = 3 /\ b = 2 /\ pc = <<"Done", "Done">>
/\ a = 3 /\ b = 4 /\ pc = <<"Done", "Done">>
/\ a = 4 /\ b = 3 /\ pc = <<"Done", "Done">>
/\ a = 2 /\ b = 3 /\ pc = <<"Done", "Done">>

Sets (Множества)

```
Procs == {1, 5, 8}
Ops == {"inc", "dec"}
Vals == 1..MaxValue
```

Sets (Множества)

```
Procs == {1, 5, 8}
Ops == {"inc", "dec"}
Vals == 1..MaxValue
Cmd == Ops \union {NULL}
{1, 2} \intersect {2, 3} = {2}
Var \in 1..10
ASSUME Var \notin Procs
```

Sets (Множества)

```
Procs == {1, 5, 8}
Ops == {"inc", "dec"}
Vals == 1..MaxValue
Cmd == Ops \union {NULL}
{1, 2} \intersect {2, 3} = {2}
Var \in 1..10
ASSUME Var \notin Procs
TRUE \in BOOLEAN
"string" \in STRING
UNION {{1, 2}, {2, 3}} = {1, 2, 3}
```

Tuples (Кортежи)

```
a == <<1, 5, 8>>
a[2] = 5
```

Tuples (Кортежи)

```
a == <<1, 5, 8>>
a[2] = 5
Len(a) = 3
Head(a) = 1
Tail(a) = <<5, 8>>
SubSeq(a, 1, 2) = <<1, 5>>
```

Tuples (Кортежи)

```
a == <<1, 5, 8>>
a[2] = 5
Len(a) = 3
Head(a) = 1
Tail(a) = <<5, 8>>
SubSeq(a, 1, 2) = <<1, 5>>
<<1, 2>> \o <<4, 8>> = <<1, 2, 4, 8>>
a := Append(a, 5)
a[i] := i
```

Records (Записи)

```
s == [rdy|->0, ack|->0, val|->0]
s.rdy = 0
s["ack"] = 0
```

Records (Записи)

```
s == [rdy|->0, ack|->0, val|->0]
s.rdy = 0
s["ack"] = 0
"a" :> 1 = [a|->1]
[a|->1] @@ [b|->2] = [a|->1, b|->2]
```

Records (Записи)

```
s == [rdy|->0, ack|->0, val|->0]
s.rdy = 0
s["ack"] = 0
"a" :> 1 = [a|->1]
[a|->1] @@ [b|->2] = [a|->1, b|->2]
s.ack := 1
p.a := p.b || p.b = p.a
```

Functions (Функции)

```
Squares == [n \in 1..Nat |-> n * n]
Squares[2] = 4
DOMAIN Squares = 1..Nat
```

Functions (Функции)

```
Squares == [n \in 1..Nat |-> n * n]
Squares[2] = 4
DOMAIN Squares = 1..Nat
s == [rdy|->0, ack|->0, val|->0]
DOMAIN s = {"ack", "rdy", "val"}
```

Functions (Функции)

```
Squares == [n \in 1..Nat |-> n * n]
Squares[2] = 4
DOMAIN Squares = 1..Nat
s == [rdy|->0, ack|->0, val|->0]
DOMAIN s = {"ack", "rdy", "val"}
DOMAIN <<5, 6, 7>> = 1..3
```

Go Channel

```go
func sender(ch chan *Data) {
    ch <- MESSAGE
    close(ch)
}

func receiver(ch chan *Data) (result *Data) {
    for {
        if r := <-ch; r == nil {
            return
        } else {
            result = r
        }
    }
}
```

Go Channel

```
----- MODULE 03_GoChannel -----
INSTANCE TLC
CONSTANTS NULL, Message

(*--algorithm GoChannel
variables
    ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
    result = NULL;
define
    Success == <>[](result = Message)

    Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
end define;
```

Go Channel

```
----- MODULE 03_GoChannel -----
INSTANCE TLC
CONSTANTS NULL, Message

(*--algorithm GoChannel
variables
    ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
    result = NULL;
define
    Success == <>[](result = Message)

    Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
end define;
```

Go Channel

```
----- MODULE 03_GoChannel -----
INSTANCE TLC
CONSTANTS NULL, Message

(*--algorithm GoChannel
variables
    ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
    result = NULL;
define
    Success == <>[](result = Message)

    Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
end define;
```

Go Channel

```
----- MODULE 03_GoChannel -----
INSTANCE TLC
CONSTANTS NULL, Message

(*--algorithm GoChannel
variables
    ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
    result = NULL;
define
    Success == <>[](result = Message)

    Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
end define;
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent :=
     ↪  TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd :=
     ↪  FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val :=
       ↪  NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent :=
     ↪  TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd :=
     ↪  FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val :=
       ↪  NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent := TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

Go Channel

```
fair process Sender = "sender"
begin
  Send:
    assert ch.open;
    await ~ch.sent;
    ch.val := Message || ch.sent :=
     ↪  TRUE;
  WaitSent:
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd :=
     ↪  FALSE;
  Close:
    assert ch.open;
    ch.open := FALSE;
end process;
```

```
fair process Receiver = "receiver"
begin Receive:
  while TRUE do
    either
      await ch.sent /\ ~ch.rcvd;
      result := ch.val || ch.val :=
       ↪  NULL || ch.rcvd := TRUE;
    or
      await ~ch.open;
      goto Done;
    end either;
  end while;
end process;

end algorithm; *)
```

ДЕМО

`03_GoChannel.tla`

PlusCal

Макросы

```
macro send(ch, msg)
begin
    assert ch.open;
    await ~ch.sent;
    ch.val := msg || ch.sent := TRUE;
end macro;

macro wait_sent(ch)
begin
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
end macro;
```

```
macro recv(var, ch)
begin
    await ~ch.open \/ (ch.sent /\ ~ch.rcvd);
    if ~ch.open then var := NULL;
    else
        var := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    end if;
end macro;

macro close(ch)
begin
    assert ch.open;
    ch.open := FALSE;
end macro;
```

PlusCal

Макросы

```
macro send(ch, msg)
begin
    assert ch.open;
    await ~ch.sent;
    ch.val := msg || ch.sent := TRUE;
end macro;

macro wait_sent(ch)
begin
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
end macro;
```

```
macro recv(var, ch)
begin
    await ~ch.open \/ (ch.sent /\ ~ch.rcvd);
    if ~ch.open then var := NULL;
    else
        var := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    end if;
end macro;

macro close(ch)
begin
    assert ch.open;
    ch.open := FALSE;
end macro;
```

PlusCal

Макросы

```
macro send(ch, msg)
begin
    assert ch.open;
    await ~ch.sent;
    ch.val := msg || ch.sent := TRUE;
end macro;

macro wait_sent(ch)
begin
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd := FALSE;
end macro;
```

```
macro recv(var, ch)
begin
    await ~ch.open \/ (ch.sent /\ ~ch.rcvd);
    if ~ch.open then var := NULL;
    else
        var := ch.val || ch.val := NULL || ch.rcvd := TRUE;
    end if;
end macro;

macro close(ch)
begin
    assert ch.open;
    ch.open := FALSE;
end macro;
```

PlusCal

Макросы

```
macro send(ch, msg)
begin
    assert ch.open;
    await ~ch.sent;
    ch.val := msg || ch.sent := TRUE;
end macro;

macro wait_sent(ch)
begin
    await ch.sent /\ ch.rcvd;
    ch.sent := FALSE || ch.rcvd :=
     ↪  FALSE;
end macro;
```

```
macro recv(var, ch)
begin
    await ~ch.open \/ (ch.sent /\
     ↪  ~ch.rcvd);
    if ~ch.open then var := NULL;
    else
        var := ch.val || ch.val := NULL
         ↪  || ch.rcvd := TRUE;
    end if;
end macro;

macro close(ch)
begin
    assert ch.open;
    ch.open := FALSE;
end macro;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;

    Success == <>[](sent = recvd)

fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent :=
         ↪  Append(sent, nextId) ||
         ↪  nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;

    Success == <>[](sent = recvd)

fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent :=
         ↪  Append(sent, nextId) ||
         ↪  nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;
```

```
    Success == <>[](sent = recvd)
```

```
fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent :=
         ↪  Append(sent, nextId) ||
         ↪  nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;

    Success == <>[](sent = recvd)

fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent :=
         ↪  Append(sent, nextId) ||
         ↪  nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;

    Success == <>[](sent = recvd)

fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent :=
         ↪  Append(sent, nextId) ||
         ↪  nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

Go Channel

```
    sent = <<>>, recvd = <<>>;

    Success == <>[](sent = recvd)

fair process Sender = "sender"
variable i = 0;
begin Send:
    while nextId \in 1..MaxMsg do
        i := nextId || sent := Append(sent, nextId) || nextId := nextId + 1;
        Send_1: send(ch, i);
        Send_2: wait_sent(ch);
    end while;
    close(ch);
end process;
```

```
fair process Receiver = "receiver"
variable j = 0;
begin Receive:
    while TRUE do
        recv(j, ch);
        if j /= NULL then
            recvd := Append(recvd, j);
        else
            goto Done;
        end if;
    end while;
end process;
```

ДЕМО

`04_GoChannel.tla`

Go Channel Bug

05_channelbug_test.go

```go
func proc(to time.Duration) int {
  ch := make(chan int)
  go func() {
    // do some processing
    ch <- RESULT
  }()
  select {
  case rslt := <-ch:
    return rslt
  case <-time.After(to):
    return 0
  }
}
```

Go Channel Bug

05_channelbug_test.go

```go
func proc(to time.Duration) int {
  ch := make(chan int)
  go func() {
    // do some processing
    ch <- RESULT
  }()
  select {
  case rslt := <-ch:
    return rslt
  case <-time.After(to):
    return 0
  }
}
```

05_ChannelBug.tla

```
fair process processor = "processor"
begin
    Proc_1: send(ch, "result");
    Proc_2: wait_sent(ch);
end process;

fair process receiver = "receiver"
variable rslt = NULL;
begin Rec_1:
    either
        recv(rslt, ch);
    or
        skip;
    end either;
end process;
```

Go Channel Bug

05_channelbug_test.go

```go
func proc(to time.Duration) int {
  ch := make(chan int)
  go func() {
    // do some processing
    ch <- RESULT
  }()
  select {
  case rslt := <-ch:
    return rslt
  case <-time.After(to):
    return 0
  }
}
```

05_ChannelBug.tla

```
fair process processor = "processor"
begin
    Proc_1: send(ch, "result");
    Proc_2: wait_sent(ch);
end process;

fair process receiver = "receiver"
variable rslt = NULL;
begin Rec_1:
    either
        recv(rslt, ch);
    or
        skip;
    end either;
end process;
```

ДЕМО

`05_ChannelBug.tla`

```
Error: Deadlock reached.
Error: The behavior up to this point is:
State 1: <Initial predicate>
/\ ch = [open |-> TRUE, val |-> NULL, sent |-> FALSE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_1", receiver |-> "Rec_1"]
/\ rslt = NULL

State 2: <Proc_1 line 55, col 11 to line 59, col 25 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Rec_1"]
/\ rslt = NULL

State 3: <Rec_1 line 69, col 10 to line 78, col 52 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Done"]
/\ rslt = NULL

7 states generated, 6 distinct states found, 1 states left on queue.
```

```
Error: Deadlock reached.
Error: The behavior up to this point is:
State 1: <Initial predicate>
/\ ch = [open |-> TRUE, val |-> NULL, sent |-> FALSE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_1", receiver |-> "Rec_1"]
/\ rslt = NULL

State 2: <Proc_1 line 55, col 11 to line 59, col 25 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Rec_1"]
/\ rslt = NULL

State 3: <Rec_1 line 69, col 10 to line 78, col 52 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Done"]
/\ rslt = NULL

7 states generated, 6 distinct states found, 1 states left on queue.
```

```
Error: Deadlock reached.
Error: The behavior up to this point is:
State 1: <Initial predicate>
/\ ch = [open |-> TRUE, val |-> NULL, sent |-> FALSE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_1", receiver |-> "Rec_1"]
/\ rslt = NULL

State 2: <Proc_1 line 55, col 11 to line 59, col 25 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Rec_1"]
/\ rslt = NULL

State 3: <Rec_1 line 69, col 10 to line 78, col 52 of module 05_ChannelBug>
/\ ch = [open |-> TRUE, val |-> "result", sent |-> TRUE, rcvd |-> FALSE]
/\ pc = [processor |-> "Proc_2", receiver |-> "Done"]
/\ rslt = NULL

7 states generated, 6 distinct states found, 1 states left on queue.
```

Go Unbound Queue

```go
func buffer1(inpCh, outCh chan int) {
  var oC chan int
  var out int
  var buf []int
  defer close(outCh)
  for {
    select {
    case inp, ok := <-inpCh:
      if !ok {
        return
      } else if oC != nil {
        buf = append(buf, inp)
      } else {
        out = inp
        oC = outCh
      }
    case oC <- out:
      if len(buf) > 0 {
        out, buf = headtail(buf)
      } else {
        oC = nil
      }
    }
  }
}
```

```go
  inp := make(chan int)
  out := make(chan int)
  go buffer1(inp, out)
```

Go Unbound Queue

```
fair process Buffer = "buffer"
variable k = NULL, buffer = <<>>;
begin BufferProcess:
  while TRUE do
    either
      recv(k, inpCh);
    Buffer_1:
      if k /= NULL then
        buffer := Append(buffer, k);
      else
        close(outCh);
        goto Done;
      end if;
    or
      when Len(buffer) > 0;
      Buffer_2:
        send(outCh, Head(buffer));
      Buffer_3:
        wait_sent(outCh);
        buffer := Tail(buffer);
    end either;
  end while;
end process;
```

ДЕМО

`06_UnboundQueue.tla`

```
Error: Temporal properties were violated.
Error: The following behavior constitutes a counter-example:
...
State 30: <Consume line 138, col 12 to line 146, col 66 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Consumer_1"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 31: <Consumer_1 line 148, col 15 to line 154, col 72 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Done"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 32: Stuttering
406 states generated, 213 distinct states found, 0 states left on queue.
```

```
Error: Temporal properties were violated.
Error: The following behavior constitutes a counter-example:
...
State 30: <Consume line 138, col 12 to line 146, col 66 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Consumer_1"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 31: <Consumer_1 line 148, col 15 to line 154, col 72 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Done"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 32: Stuttering
406 states generated, 213 distinct states found, 0 states left on queue.
```

```
Error: Temporal properties were violated.
Error: The following behavior constitutes a counter-example:
...
State 30: <Consume line 138, col 12 to line 146, col 66 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Consumer_1"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 31: <Consumer_1 line 148, col 15 to line 154, col 72 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Done"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 32: Stuttering
406 states generated, 213 distinct states found, 0 states left on queue.
```

```
Error: Temporal properties were violated.
Error: The following behavior constitutes a counter-example:
...
State 30: <Consume line 138, col 12 to line 146, col 66 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Consumer_1"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 31: <Consumer_1 line 148, col 15 to line 154, col 72 of module 06_UnboundQueue>
/\ buffer = <<3>>
/\ i = 3
/\ j = NULL
/\ k = NULL
/\ pc = [buffer |-> "Done", producer |-> "Done", consumer |-> "Done"]
/\ recvd = <<1, 2>>
/\ inpCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ outCh = [sent |-> FALSE, open |-> FALSE, val |-> NULL, rcvd |-> FALSE]
/\ sent = <<1, 2, 3>>
/\ id = 4

State 32: Stuttering
406 states generated, 213 distinct states found, 0 states left on queue.
```

Go Unbound Queue (fixed)

```
fair process Buffer = "buffer"
variable k = NULL, buffer = <<>>,
         exit = FALSE;
begin BufferProcess:
  while TRUE do
  either
      when ~exit;
      recv(k, inpCh);
    Buffer_1:
      if k /= NULL then
        buffer := Append(buffer, k);
      else
        exit := TRUE;
      end if;
    or
      when Len(buffer) > 0;
    Buffer_2:
      send(outCh, Head(buffer));
    Buffer_3:
      wait_sent(outCh);
      buffer := Tail(buffer);
    or
      when Len(buffer) = 0 /\ exit;
      close(outCh);
      goto Done;
    end either;
  end while;
end process;
```

ДЕМО

`07_UnboundQueue.tla`

Go Unbound Queue (fixed)

```go
func buffer2(inpCh, outCh chan int) {
  var exit bool
  var oC chan int
  var out int
  var buf []int
  defer close(outCh)
  for {
    select {
    case inp, ok := <-inpCh:
      if !ok {
        if oC == nil {
          return
        } else {
          exit = true
          inpCh = nil
        }
      } else if oC != nil {
        buf = append(buf, inp)
      } else {
        out = inp
        oC = outCh
      }
    case oC <- out:
      if len(buf) > 0 {
        out, buf = headtail(buf)
      } else if !exit {
        oC = nil
      } else {
        return
      }
    }
  }
}
```

А на практике?

Amazon Web Services успешно использует TLA+[1] с 2011 года

> *"At AWS, formal methods have been a big success. They have helped us prevent subtle, serious bugs from reaching production, bugs that we would not have found via any other technique."*

1. http://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf

Amazon Web Services успешно использует TLA+[1] с 2011 года

> *"At AWS, formal methods have been a big success. They have helped us prevent subtle, serious bugs from reaching production, bugs that we would not have found via any other technique."*

Microsoft использует TLA+во многих проектах[2], включая Azure CosmosDB[3]

> *"TLA+ is not yet a prerequisite for our hiring. However, a candidate's knowledge of TLA+ is given significant weight in our evaluation. To us, it is a great indicator of those who really care about quality and correctness."*

1. http://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf
2. https://www.microsoft.com/en-us/research/search/?q=TLA%2B
3. https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

MongoDB для поиска[1] ошибок в протоколе[2] репликации

> *"We expect that formally modeling our system upfront could have saved 100s of hours of engineering time."*

1. conf.tlapl.us/07_-_TLAConf19_-_William_Schultz_-_Fixing_a_MongoDB_Replication_Protocol_Bug_with_TLA.pdf
2. github.com/will62794/mongo-repl-tla-models

MongoDB для поиска[1] ошибок в протоколе[2] репликации

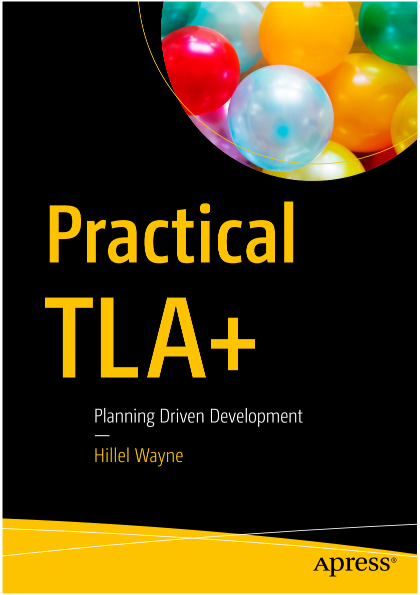
> *“We expect that formally modeling our system upfront could have saved 100s of hours of engineering time.”*

CockroachDB для спецификации[3] протокола распределенных транзакций[4]

> *“We found that the process of writing this specification gave us more confidence in the Parallel Commit protocol itself and in its integration into CockroachDB.”*

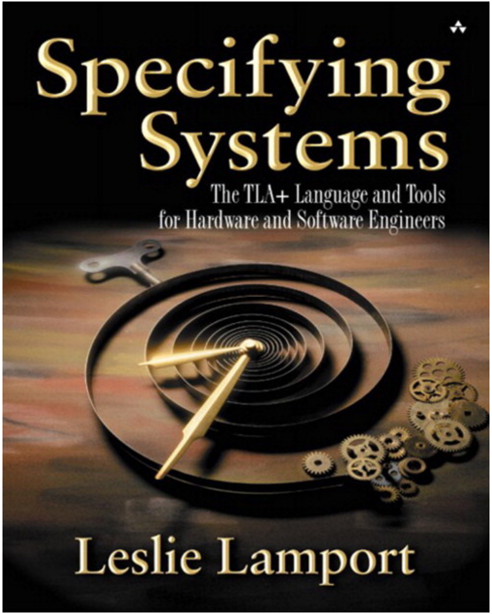
1. conf.tlapl.us/07_-_TLAConf19_-_William_Schultz_-_Fixing_a_MongoDB_Replication_Protocol_Bug_with_TLA.pdf
2. github.com/will62794/mongo-repl-tla-models
3. www.cockroachlabs.com/blog/parallel-commits
4. github.com/cockroachdb/cockroach/tree/master/docs/tla-plus

С чего начать?



Practical TLA+
Planning Driven Development
by Hillel Wayne
Apress, 2018

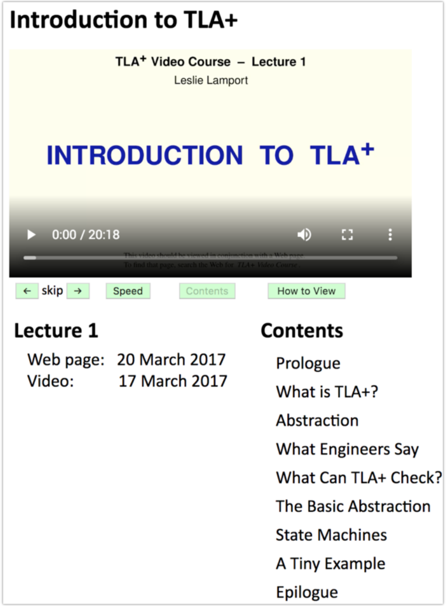
PlusCal, идеально для вхождения в тему!



Specifying Systems
The TLA+ Language and Tools for Hardware and Software Engineers
by Leslie Lamport
Addison-Wesley Professional, 2002

TLA+, обязательно иметь под рукой

(последняя версия всегда доступна на сайте Лэмпорта, бесплатно!)



TLA+ Video Course

by Leslie Lamport

`lamport.azurewebsites.net/video/videos.html`

3.5 часа TLA+, обязательно посмотреть

(Лэмпорт прекрасный лектор!)

Lamport TLA+ Site: https://lamport.azurewebsites.net/tla/tla.html

Learn TLA+: https://learntla.com

google group: https://groups.google.com/forum/#!forum/tlaplus

reddit: https://www.reddit.com/r/tlaplus/

TLA+ Tools: http://www.tlaplus.net/

Спасибо!

(Презентация и код: github.com/growler/gophercon-russia-2020-talk)