

TLA+ Tools: практичный инструмент формальной верификации алгоритмов

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



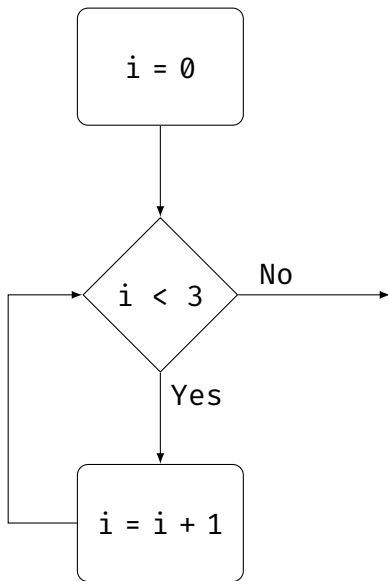
GopherCon Russia 2020
Alexey Naidyonov
IToolLabs
Mar 28 2020

ITOOOLABS 

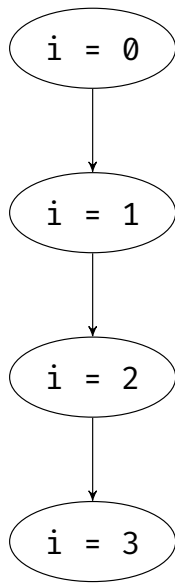
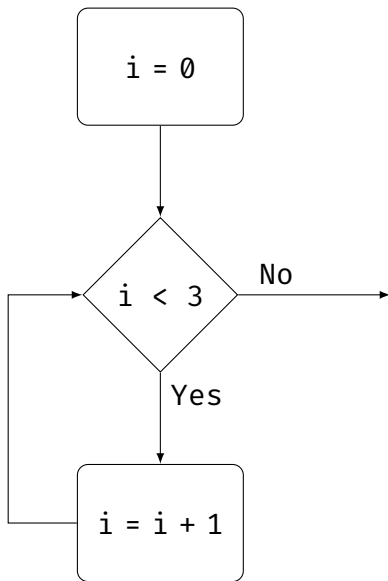
Что такое программа?

```
1 var i = 0
2 for i < 3 {
3     i = i + 1
4 }
```

```
1 var i = 0
2 for i < 3 {
3     i = i + 1
4 }
```



```
1 var i = 0
2 for i < 3 {
3   i = i + 1
4 }
```



Process 1

1 $a = b + 1$
2 $b = a + 1$

$a = 0$
 $b = 0$

Process 2

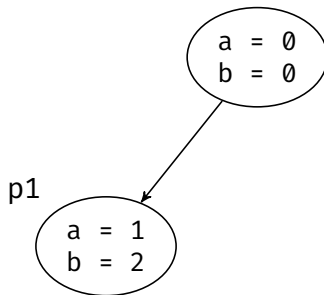
1 $b = a + 1$
2 $a = b + 1$

Process 1

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

Process 2

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

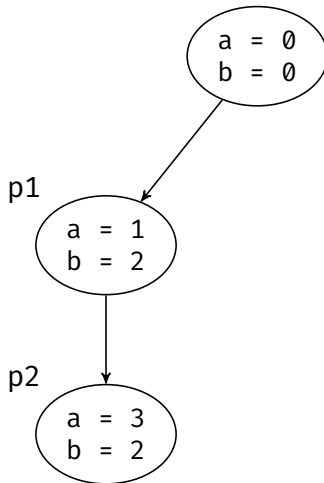


Process 1

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

Process 2

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

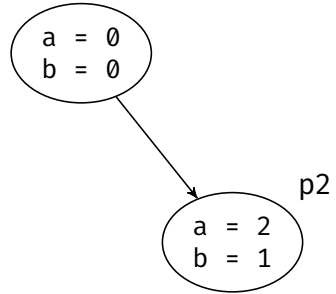


Process 1

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

Process 2

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

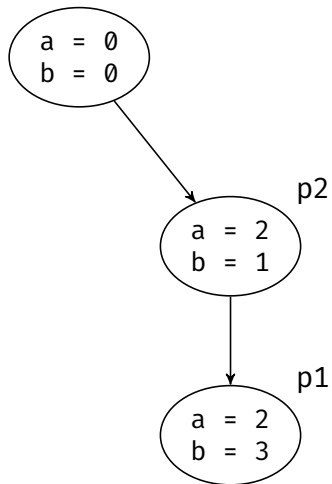


Process 1

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

Process 2

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

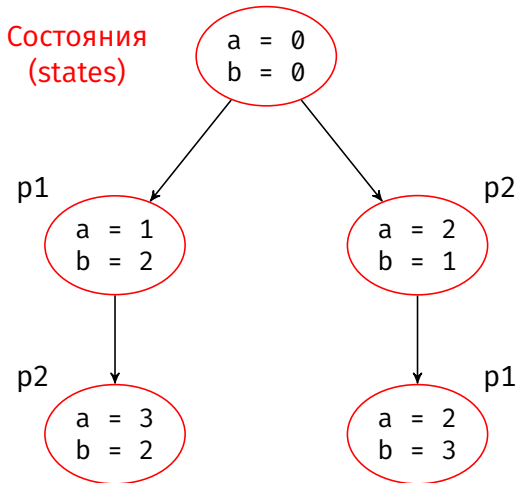


Process 1

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

Process 2

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

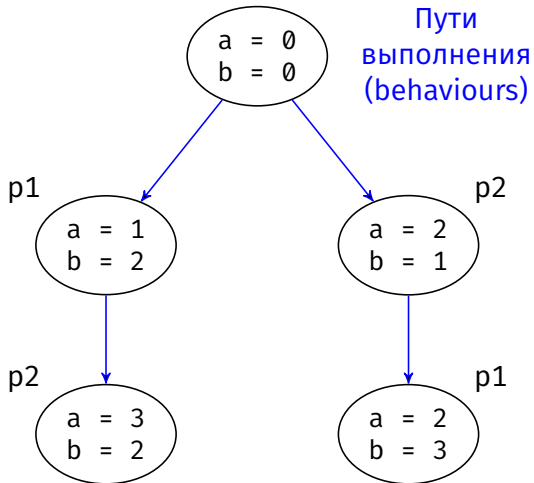


Process 1

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

Process 2

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

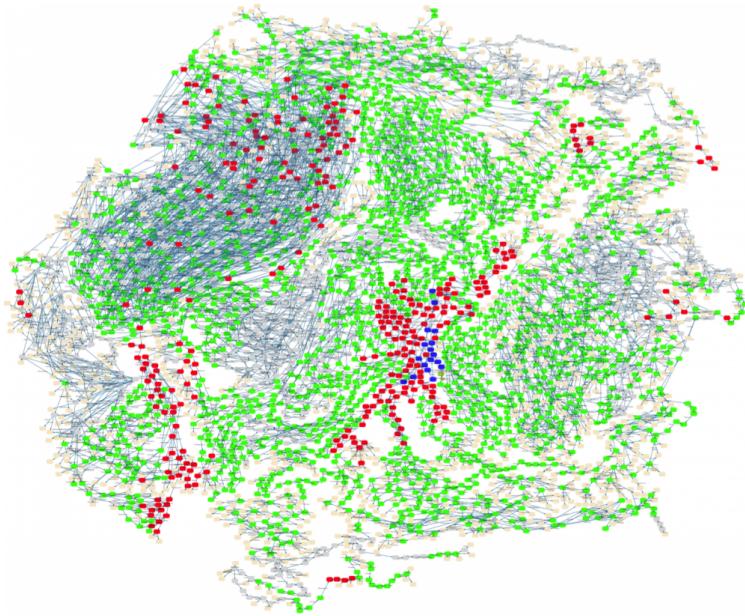


$$(m \cdot n) \frac{(m \cdot n)!}{(n!)^m},$$

где

m - количество процессов

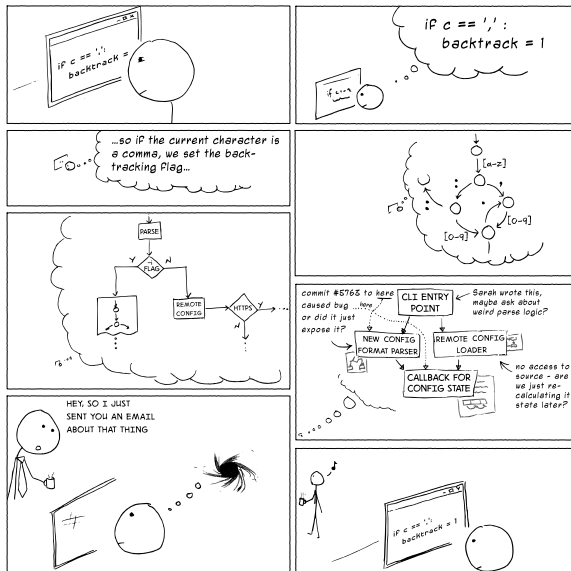
n - длина пути выполнения



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



© Jason Heeris 2013 LICENSE: CC BY-NC-ND 2.5 AU

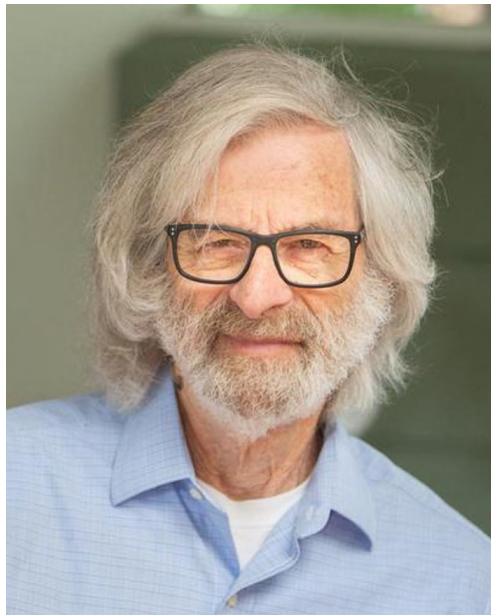
heeris.id.au

“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

$A \wedge B$

Конъюнкция

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика



©Universal Pictures Amblin Entertainment

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика

$\diamond A$

Eventually

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика

$\diamond A$

Eventually

$\square A$

Always

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика

$\diamond A$

Eventually

$\square A$

Always

$A \rightsquigarrow B$

Leads to

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика

$\diamond A$

Eventually

$\square A$

Always

$A \rightsquigarrow B$

Leads to

$\square \diamond A$

Математическая логика

$A \wedge B$	Конъюнкция
$A \vee B$	Дизъюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\diamond A$	Eventually
$\square A$	Always
$A \rightsquigarrow B$	Leads to
$\square \diamond A$	Infinitely often

Математическая логика

$A \wedge B$

Конъюнкция

$A \vee B$

Дизъюнкция

$A \rightarrow B$

Импликация

$\neg A$

Отрицание

Темпоральная логика

$\diamond A$

Eventually

$\square A$

Always

$A \rightsquigarrow B$

Leads to

$\square \diamond A$

Infinitely often

$\diamond \square A$

Математическая логика

$A \wedge B$	Конъюнкция
$A \vee B$	Дизъюнкция
$A \rightarrow B$	Импликация
$\neg A$	Отрицание

Темпоральная логика

$\diamond A$	Eventually
$\square A$	Always
$A \rightsquigarrow B$	Leads to
$\square \diamond A$	Infinitely often
$\diamond \square A$	Eventually always

Математическая логика

$A \wedge B$	$A \setminus B$	Конъюнкция
$A \vee B$	$A \setminus / B$	Дизъюнкция
$A \rightarrow B$	$A \Rightarrow B$	Импликация
$\neg A$	$\sim A$	Отрицание

Темпоральная логика

$\diamond A$	$\langle \rangle A$	Eventually
$\square A$	$[] A$	Always
$A \rightsquigarrow B$	$\sim \rangle A$	Leads to
$\square \diamond A$	$[] \langle \rangle A$	Infinitely often
$\diamond \square A$	$\langle \rangle [] A$	Eventually always

MODULE 00_Counter

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

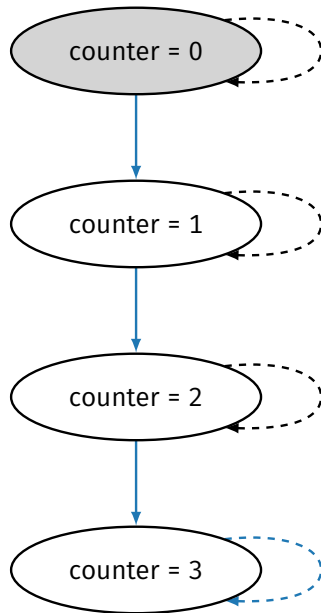
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

EXTENDS **Naturals**

CONSTANTS MinValue, MaxValue

ASSUME MinValue < MaxValue

VARIABLE counter

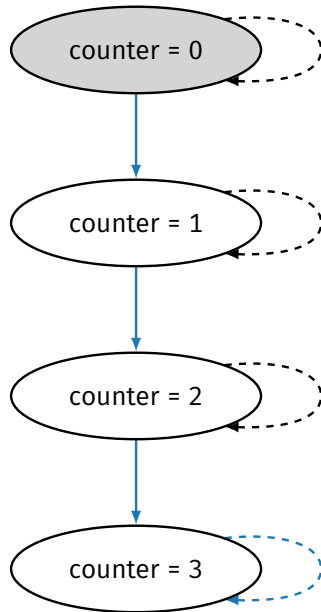
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

EXTENDS Naturals

CONSTANTS MinValue, MaxValue

ASSUME MinValue < MaxValue

VARIABLE counter

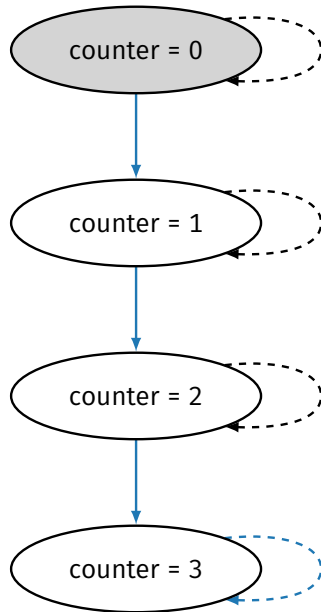
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

EXTENDS Naturals
 CONSTANTS MinValue, MaxValue
 ASSUME $\text{MinValue} < \text{MaxValue}$
 VARIABLE counter

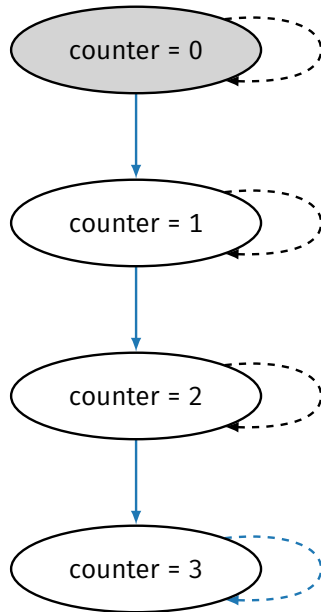
Invariant $\triangleq \text{counter} \in \text{MinValue}.. \text{MaxValue}$

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

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

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

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



MODULE 00_Counter

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

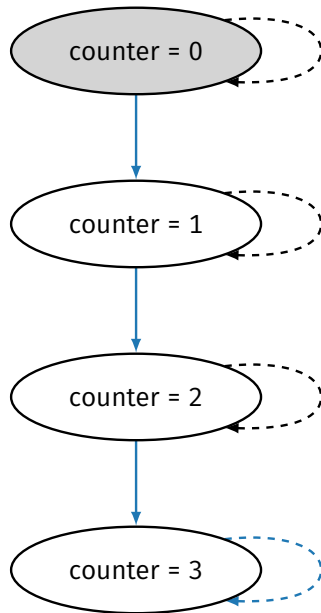
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

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

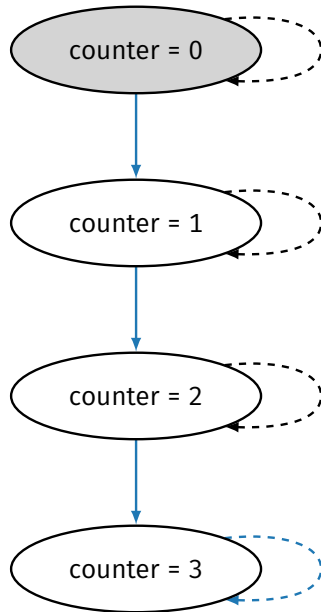
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

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

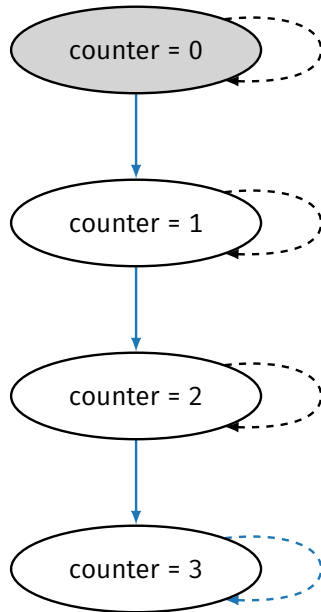
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

EXTENDS Naturals
 CONSTANTS MinValue, MaxValue
 ASSUME $\text{MinValue} < \text{MaxValue}$
 VARIABLE counter

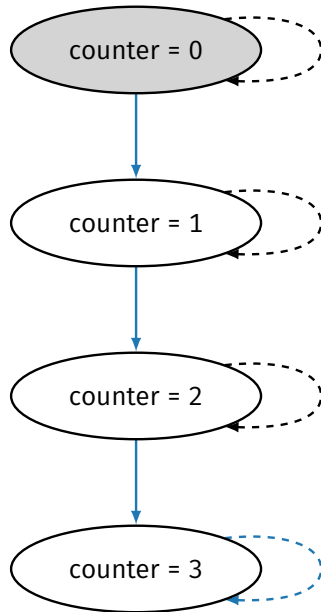
Invariant $\triangleq \text{counter} \in \text{MinValue}.. \text{MaxValue}$

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

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

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

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



MODULE 00_Counter

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

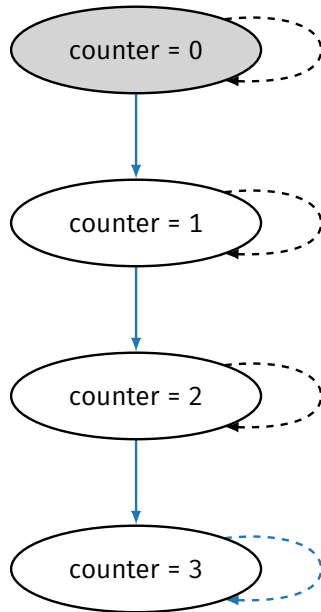
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

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

Invariant \triangleq counter \in MinValue..MaxValue

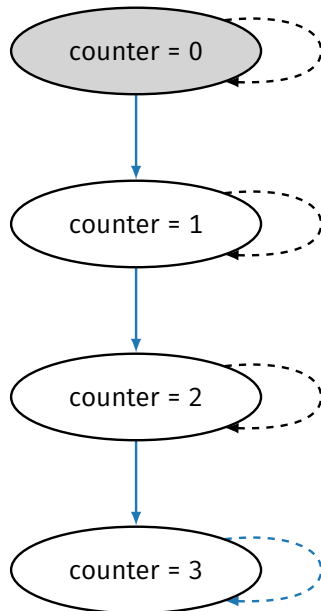
Success \triangleq $\Diamond\Box(\text{counter} = \text{MaxValue})$

Init \triangleq counter = MinValue

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

Spec \triangleq \wedge Init

$\wedge \Diamond\Box[\text{Next}]_{\text{counter}}$
 $\wedge \text{WF}_{\text{counter}}(\text{Next})$



MODULE 00_Counter

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

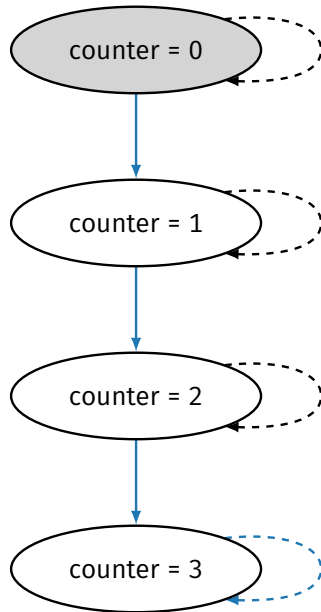
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



MODULE 00_Counter

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

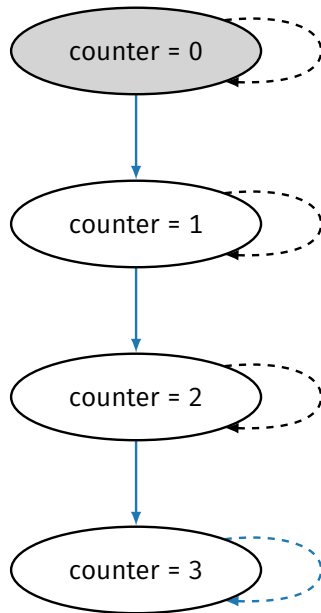
Invariant \triangleq counter \in MinValue..MaxValue

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

Init \triangleq counter = MinValue

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

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



TLC

Explicit State Model Checker for TLA+

TLA+ Toolbox by Simon Zambrovski, et al.

The screenshot displays the TLA+ Toolbox interface, which is divided into several panes. On the left, the 'Spec Explorer' pane shows a tree view of the project structure, including 'Counter.tla', 'modules', 'Counter', 'FiniteSets', 'models', 'Model_1', and 'Model_2'. The main editor pane shows the 'Counter.tla' specification, which is a TLA+ module defining a counter. The specification includes constants for 'MinValue' and 'MaxValue', a variable 'counter', an invariant, a success condition, an initialization action, and a next-state action. The right pane is titled 'Model_1' and contains a 'Model Overview' and 'Model Checking Results' section. The 'Model Overview' section shows the start and end times of the model checking run (12:33:01 to 12:33:04 on Mar 24) and a fingerprint collision probability of 2.2E-19. The 'Model Checking Results' section includes a table of statistics and a table of actions. The bottom pane shows the 'Additional Spec Options' section, which allows users to specify the values of declared constants and the temporal formula to be checked. The status bar at the bottom indicates that the specification is 'parsed'.

TLA+ Toolbox

Spec Explorer

Counter [Counter.tla]

- modules
 - Counter
- FiniteSets
- models
 - Model_1
 - Model_2

Counter.tla

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant = counter \in MinValue..MaxValue
9
10 Success = <>[](counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15             THEN counter + 1
16             ELSE counter
17
18 Spec = & Init
19       & [][Next]_counter
```

Model_1

Model Overview

Model Checking Results

General

Start: 12:33:01 (Mar 24) End: 12:33:04 (Mar 24)

Fingerprint collision probability: calculated: 2.2E-19

Statistics

State space progress (click column header for graph) Actions at 00:00:03

Time	Diamet	States	Fo	Distinct	Stat	Queue	Siz	Module	Action	Location
00:00:03	4	5		4		0		Coun...	Init	line 12, c
00:00:03	0	1		1		1		Coun...	Next	line 14, c

Evaluate Constant Expression

Help Console Counter.tla

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant = counter \in MinValue..MaxValue
9
10 Success = <>[](counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15             THEN counter + 1
16             ELSE counter
17
18 Spec = & Init
19       & [][Next]_counter
20
21
22
```

Additional Spec Options

What is the behavior spec?

Temporal formula

Spec

What is the model?

Specify the values of declared constants.

MinValue <- 0

MaxValue <- 3

Edit

215.9 x 279.4 mm

Spec Status: **parsed**

TLA+ Toolbox by Simon Zambrovski, et al.

The screenshot displays the TLA+ Toolbox interface with the following components:

- Spec Explorer:** A tree view on the left showing the project structure: Counter [Counter.tla], modules (Counter, FiniteSets), and models (Model_1, Model_2).
- Counter.tla Editor:** The main text area showing the TLA+ specification for a counter. The code is as follows:

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant = counter \in MinValue..MaxValue
9
10 Success = <>[] (counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15           THEN counter + 1
16           ELSE counter
17
18 Spec = & Init
19       & [] [Next]_counter
```
- Model_1 Overview:** A panel on the right showing model checking results for Model_1. It includes a general summary, statistics, and a table of actions.

General
Start: 12:33:01 (Mar 24) End: 12:33:04 (Mar 24)
Fingerprint collision probability: calculated: 2.2E-19

Statistics
State space progress (click column header for graph) Actions at 00:00:03

Time	Diamet	States	Fo	Distinct	Stat	Queue	Siz	Module	Action	Location
00:00:03	4	5		4		0		Coun...	Init	line 12, c
00:00:03	0	1		1		1		Coun...	Next	line 14, c
- Model_2 Overview:** A panel on the right showing model checking results for Model_2. It includes a general summary, statistics, and a table of actions.

What is the behavior spec?
Temporal formula
Spec

What is the model?
Specify the values of declared constants.
MinValue <- 0
MaxValue <- 3
Edit
- Counter.tla Editor (Bottom Right):** A panel showing the full specification code, including the module declaration, constants, variables, and the specification formula.

The status bar at the bottom indicates the specification is **parsed**.

TLA+ Toolbox by Simon Zambrovski, et al.

The screenshot displays the TLA+ Toolbox interface, which is divided into several panes. On the left, the 'Spec Explorer' pane shows a tree view of the project structure, including 'Counter.tla', 'modules', 'Counter', 'FiniteSets', 'models', 'Model_1', and 'Model_2'. The main editor pane shows the 'Counter.tla' module code, which defines a counter module with constants, variables, and a specification. The code is as follows:

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant = counter \in MinValue..MaxValue
9
10 Success =  $\Diamond[(\text{counter} = \text{MaxValue})]$ 
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15           THEN counter + 1
16           ELSE counter
17
18 Spec =  $\wedge$  Init
19        $\wedge$   $[\text{Next}]_{\text{counter}}$ 
```

The 'Model_1' pane shows the 'Model Overview' and 'Model Checking Results' tabs. The 'Model Overview' tab displays the 'General' information, including the start and end times of the model checking run (Start: 12:33:01 (Mar 24), End: 12:33:04 (Mar 24)) and the fingerprint collision probability (calculated: 2.2E-19). The 'Statistics' tab shows the state space progress (click column header for graph) and the actions at 00:00:03. The 'Evaluate Constant Expression' tab is also visible.

The 'Model_2' pane shows the 'Model Overview' and 'Model Checking Results' tabs. The 'Model Overview' tab displays the 'What is the behavior spec?' and 'What is the model?' sections. The 'What is the behavior spec?' section shows the temporal formula and the specification. The 'What is the model?' section shows the values of the declared constants (MinValue <- 0, MaxValue <- 3) and an 'Edit' button. The 'Additional Spec Options' section is also visible.

The bottom status bar shows the 'Spec Status' as 'parsed'.

TLA+ Toolbox by Simon Zambrovski, et al.

The screenshot displays the TLA+ Toolbox interface, which is divided into several panes. On the left, the 'Spec Explorer' pane shows a tree view of the project structure, including 'Counter.tla', 'modules', 'Counter', 'FiniteSets', 'models', 'Model_1', and 'Model_2'. The main central pane shows the 'Counter.tla' specification file, which defines a counter module with constants, variables, invariants, and actions. The right pane is titled 'Model_1' and contains a 'Model Overview' and 'Model Checking Results' section. This section displays the start and end times of the model checking process, the fingerprint collision probability, and a table of statistics. Below the statistics table is an 'Evaluate Constant Expression' section. The bottom pane shows the 'Counter.tla' specification file again, but with a different view, possibly a simplified or annotated version. The status bar at the bottom indicates the specification is 'parsed'.

Counter.tla

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant = counter \in MinValue..MaxValue
9
10 Success =  $\Diamond[](\text{counter} = \text{MaxValue})$ 
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15           THEN counter + 1
16           ELSE counter
17
18 Spec =  $\wedge$  Init
19        $\wedge [][\text{Next}]_{\text{counter}}$ 
```

Model_1 Model Overview

Start: 12:33:01 (Mar 24) End: 12:33:04 (Mar 24)

Fingerprint collision probability: calculated: 2.2E-19

Statistics

State space progress (click column header for graph) Actions at 00:00:03

Time	Diamet	States	Fol	Distinct	Stat	Queue	Siz	Module	Action	Location
00:00:03	4	5		4		0		Coun...	Init	line 12, c
00:00:03	0	1		1		1		Coun...	Next	line 14, c

Evaluate Constant Expression

Counter.tla

```
1 ----- MODULE Counter -----
2 EXTENDS NaturalS
3 CONSTANTS MinValue, MaxValue
4 ASSUME MinValue < MaxValue
5
6 VARIABLES counter
7
8 Invariant  $\triangleq$  counter  $\in$  MinValue .. MaxValue
9
10 Success  $\triangleq$   $\Diamond\Box(\text{counter} = \text{MaxValue})$ 
11
12 Init  $\triangleq$  counter = MinValue
13
14 Next  $\triangleq$  counter' = IF counter < MaxValue
15           THEN counter + 1
16           ELSE counter
17
18 Spec  $\triangleq$   $\wedge$  Init
19            $\wedge \Box[\text{Next}]_{\text{counter}}$ 
20            $\wedge \text{WF}_{\text{counter}}(\text{Next})$ 
21
22
```

2 : 1

Spec Status : **parsed**

VS Code / TLA+ Support by Андрей Лыгин @alygin

The screenshot displays the VS Code interface with the TLA+ model checking extension. The main editor shows the `Counter.tla` file, and the right sidebar shows the model checking results.

Counter.tla

```
1  ----- MODULE Counter -----
2  EXTENDS Naturals
3  CONSTANTS MinValue, MaxValue
4  ASSUME MinValue < MaxValue
5
6  VARIABLES counter
7
8  Invariant = counter \in MinValue..MaxValue
9
10 Success = <>[](counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15 | THEN counter + 1
16 | ELSE counter
17
18 Spec = & Init
19 | & [][Next]_counter
20 | & WF_counter(Next)
21
22 =====
```

TLA+ model checking

Success: Fingerprint collision probability: 2.2E-19
Start: 21:10:11 (Mar 25), end: 21:10:11 (Mar 25)

States

Time	Diameter	Found	Distinct	Queue
00:00:00	4	5	4	0

Coverage

Module	Action	Total	Distinct
Counter	Init	1	1
Counter	Next	4	3

Counter.cfg

```
1  SPECIFICATION Spec
2
3  CONSTANTS
4  | MinValue = 0
5  | MaxValue = 3
6
7  INVARIANTS
8  | Invariant
9
10 PROPERTIES
11 | Success
```

VS Code status bar: Ln 2, Col 1 Spaces: 4 UTF-8 LF TLA+

VS Code / TLA+ Support by Андрей Лыгин @alygin

VS Code interface showing TLA+ support. The main editor displays the Counter.tla file, which defines a Counter module. The right sidebar shows the TLA+ model checking results, including success status, start/end times, states, coverage, and the Counter.cfg configuration file.

Counter.tla

```
1  ----- MODULE Counter -----
2  EXTENDS Naturals
3  CONSTANTS MinValue, MaxValue
4  ASSUME MinValue < MaxValue
5
6  VARIABLES counter
7
8  Invariant = counter \in MinValue..MaxValue
9
10 Success =  $\diamond []$ (counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15 | THEN counter + 1
16 | ELSE counter
17
18 Spec =  $\wedge$  Init
19 |  $\wedge []$ [Next]_counter
20 |  $\wedge$  WF_counter(Next)
21
22 =====
23
```

TLA+ model checking

Success: Fingerprint collision probability: 2.2E-19
Start: 21:10:11 (Mar 25), end: 21:10:11 (Mar 25)

States

Time	Diameter	Found	Distinct	Queue
00:00:00	4	5	4	0

Coverage

Module	Action	Total	Distinct
Counter	Init	1	1
Counter	Next	4	3

Counter.cfg

```
1  SPECIFICATION Spec
2
3  CONSTANTS
4  | MinValue = 0
5  | MaxValue = 3
6
7  INVARIANTS
8  | Invariant
9
10 PROPERTIES
11 | Success
```

VS Code / TLA+ Support by Андрей Лыгин @alygin

The screenshot displays the VS Code interface with the TLA+ model checking extension. The main editor shows the `Counter.tla` file, and the right sidebar shows the `TLA+ model checking` results. A red box highlights the `Counter.cfg` configuration file.

Counter.tla

```
1  ----- MODULE Counter -----
2  EXTENDS Naturals
3  CONSTANTS MinValue, MaxValue
4  ASSUME MinValue < MaxValue
5
6  VARIABLES counter
7
8  Invariant = counter \in MinValue..MaxValue
9
10 Success = <>[](counter = MaxValue)
11
12 Init = counter = MinValue
13
14 Next = counter' = IF counter < MaxValue
15 | THEN counter + 1
16 | ELSE counter
17
18 Spec = & Init
19 | & [][Next]_counter
20 | & WF_counter(Next)
21
22 =====
```

TLA+ model checking

Success: Fingerprint collision probability: 2.2E-19
Start: 21:10:11 (Mar 25), end: 21:10:11 (Mar 25)

States

Time	Diameter	Found	Distinct	Queue
00:00:00	4	5	4	0

Coverage

Module	Action	Total	Distinct
Counter	Init	1	1
Counter	Next	4	3

Counter.cfg

```
1  SPECIFICATION Spec
2
3  CONSTANTS
4  | MinValue = 0
5  | MaxValue = 3
6
7  INVARIANTS
8  | Invariant
9
10 PROPERTIES
11 | Success
```

VS Code / TLA+ Support by Андрей Лыгин @alygin

The screenshot displays the VS Code interface with a TLA+ file named `Counter.tla` open. The Explorer sidebar on the left shows the project structure, including `Counter.tla`, `Counter.cfg`, `Counter.pdf`, and a `COUNTER` folder containing `Counter.toolbox`, `states`, and `Counter.tla`. The main editor shows the TLA+ code for the Counter module, which includes constants for `MinValue` and `MaxValue`, a variable `counter`, an invariant, a success condition, and a specification with `Init` and `Next` actions.

On the right, the **TLA+ model checking** panel is open, showing the following results:

Success: Fingerprint collision probability: 2.2E-19
Start: 21:10:11 (Mar 25), end: 21:10:11 (Mar 25)

States

Time	Diameter	Found	Distinct	Queue
00:00:00	4	5	4	0

Coverage

Module	Action	Total	Distinct
Counter	Init	1	1
Counter	Next	4	3

Below the model checking results, the `Counter.cfg` file is open, showing the configuration for the model checker, including the specification, constants, invariants, and properties.

At the bottom of the VS Code window, the status bar indicates the current line and column (Ln 2, Col 1), the number of spaces (Spaces: 4), the encoding (UTF-8), the language (LF), and the active language (TLA+).

ДЕМО

00_Counter.tla

PlusCal

```
4 (*--algorithm 01_PCDemo
5
6 variables
7     a = 0,
8     b = 0;
9
10 define
11     Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16     a := b + 1;
17     b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22     b := a + 1;
23     a := b + 1;
24 end process;
25
26 end algorithm; *)
```

PlusCal

Алгоритм

```
4  (*--algorithm 01_PCDemo
5
6  variables
7      a = 0,
8      b = 0;
9
10 define
11     Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16     a := b + 1;
17     b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22     b := a + 1;
23     a := b + 1;
24 end process;
25
26 end algorithm; *)
```


PlusCal

Переменные

```
4 (*--algorithm 01_PCDemo
5
6 variables
7     a = 0,
8     b = 0;
9
10 define
11     Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16     a := b + 1;
17     b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22     b := a + 1;
23     a := b + 1;
24 end process;
25
26 end algorithm; *)
```

PlusCal

Определения TLA+

```
4 (*--algorithm 01_PCDemo
5
6 variables
7     a = 0,
8     b = 0;
9
10 define
11     Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16     a := b + 1;
17     b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22     b := a + 1;
23     a := b + 1;
24 end process;
25
26 end algorithm; *)
```

PlusCal

Процессы

```
4 (*--algorithm 01_PCDemo
5
6 variables
7   a = 0,
8   b = 0;
9
10 define
11   Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16   a := b + 1;
17   b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22   b := a + 1;
23   a := b + 1;
24 end process;
25
26 end algorithm; *)
```

PlusCal

Метки

```
4 (*--algorithm 01_PCDemo
5
6 variables
7   a = 0,
8   b = 0;
9
10 define
11   Success == <>[](a+b = 5)
12 end define;
```

```
14 fair process proc1 = 1
15 begin p1:
16   a := b + 1;
17   b := a + 1;
18 end process;
19
20 fair process proc2 = 2
21 begin p2:
22   b := a + 1;
23   a := b + 1;
24 end process;
25
26 end algorithm; *)
```

PlusCal

Трансляция

```
27 \* BEGIN TRANSLATION
28 VARIABLES a, b, pc
29
30 (* define statement *)
31 Success == <>[](a+b = 5)
32
33
34 vars == << a, b, pc >>
35
36 ProcSet == {1} \cup {2}
37
38 Init == (* Global variables *)
39     /\ a = 0
40     /\ b = 0
41     /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
42                                     [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
27 \* BEGIN TRANSLATION
28 VARIABLES a, b, pc
29
30 (* define statement *)
31 Success == <>[](a+b = 5)
32
33
34 vars == << a, b, pc >>
35
36 ProcSet == {1} \cup {2}
37
38 Init == (* Global variables *)
39     /\ a = 0
40     /\ b = 0
41     /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
42             [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
27 \* BEGIN TRANSLATION
28 VARIABLES a, b, pc
29
30 (* define statement *)
31 Success == <>[](a+b = 5)
32
33
34 vars == << a, b, pc >>
35
36 ProcSet == {1} \cup {2}
37
38 Init == (* Global variables *)
39     /\ a = 0
40     /\ b = 0
41     /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
42                                     [] self = 2 -> "p2"]
```

PlusCal

Трансляция

```
27 \* BEGIN TRANSLATION
28 VARIABLES a, b, pc
29
30 (* define statement *)
31 Success == <>[](a+b = 5)
32
33
34 vars == << a, b, pc >>
35
36 ProcSet == {1} \cup {2}
37
38 Init == (* Global variables *)
39     /\ a = 0
40     /\ b = 0
41     /\ pc = [self \in ProcSet |-> CASE self = 1 -> "p1"
42                                     [] self = 2 -> "p2"]
```


PlusCal

Трансляция

```
44 p1 == /\ pc[1] = "p1"  
45      /\ a' = b + 1  
46      /\ b' = a' + 1  
47      /\ pc' = [pc EXCEPT ![1] = "Done"]  
48  
49 proc1 == p1  
50  
51 p2 == /\ pc[2] = "p2"  
52      /\ b' = a + 1  
53      /\ a' = b' + 1  
54      /\ pc' = [pc EXCEPT ![2] = "Done"]  
55  
56 proc2 == p2
```

PlusCal

Трансляция

```
44 p1 == /\ pc[1] = "p1"  
45     /\ a' = b + 1  
46     /\ b' = a' + 1  
47     /\ pc' = [pc EXCEPT ![1] = "Done"]  
48  
49 proc1 == p1  
50  
51 p2 == /\ pc[2] = "p2"  
52     /\ b' = a + 1  
53     /\ a' = b' + 1  
54     /\ pc' = [pc EXCEPT ![2] = "Done"]  
55  
56 proc2 == p2
```

PlusCal

Трансляция

```
44 p1 == /\ pc[1] = "p1"  
45      /\ a' = b + 1  
46      /\ b' = a' + 1  
47      /\ pc' = [pc EXCEPT ![1] = "Done"]  
48  
49 proc1 == p1  
50  
51 p2 == /\ pc[2] = "p2"  
52      /\ b' = a + 1  
53      /\ a' = b' + 1  
54      /\ pc' = [pc EXCEPT ![2] = "Done"]  
55  
56 proc2 == p2
```

PlusCal

Трансляция

```
58 (* Allow infinite stuttering to prevent deadlock on termination.  
   ↪ *)  
59 Terminating == /\ \A self \in ProcSet: pc[self] = "Done"  
60                /\ UNCHANGED vars  
61  
62 Next == proc1 \/ proc2  
63         \/ Terminating  
64  
65 Spec == /\ Init /\ [][Next]_vars  
66         /\ WF_vars(proc1)  
67         /\ WF_vars(proc2)  
68  
69 Termination == <>(\A self \in ProcSet: pc[self] = "Done")
```

PlusCal

Трансляция

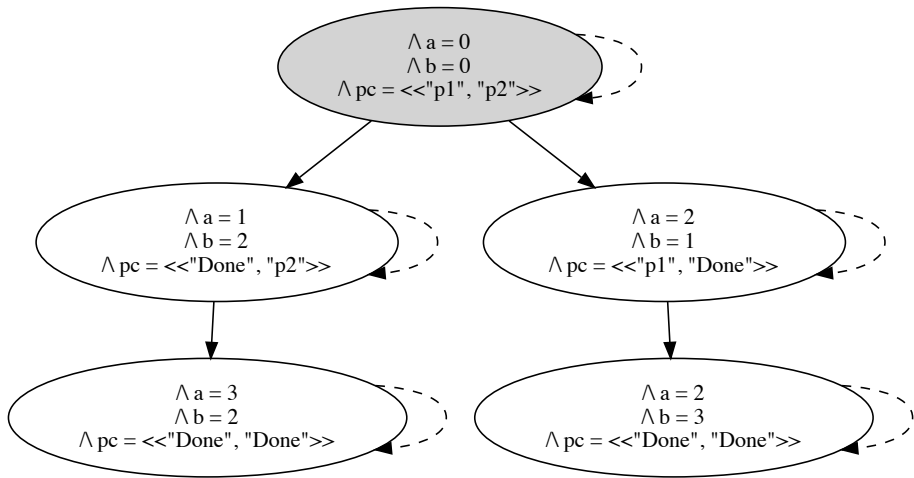
```
58 (* Allow infinite stuttering to prevent deadlock on termination.  
   ↪ *)  
59 Terminating == /\ \A self \in ProcSet: pc[self] = "Done"  
60                /\ UNCHANGED vars  
61  
62 Next == proc1 \/ proc2  
63         \/ Terminating  
64  
65 Spec == /\ Init /\ [][Next]_vars  
66         /\ WF_vars(proc1)  
67         /\ WF_vars(proc2)  
68  
69 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:$
```



PlusCal

Атомарность

```
4 (*--algorithm A
5
6 variables
7     a = 0,
8     b = 0;
9
10 define
11     Success == <>[](
12         (a + b) = 5
13     )
14 end define;
```

```
16 fair process proc1 = 1
17 begin
18     p1_1: a := b + 1;
19     p1_2: b := a + 1;
20 end process;
21
22 fair process proc2 = 2
23 begin
24     p2_1: b := a + 1;
25     p2_2: a := b + 1;
26 end process;
27
28 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

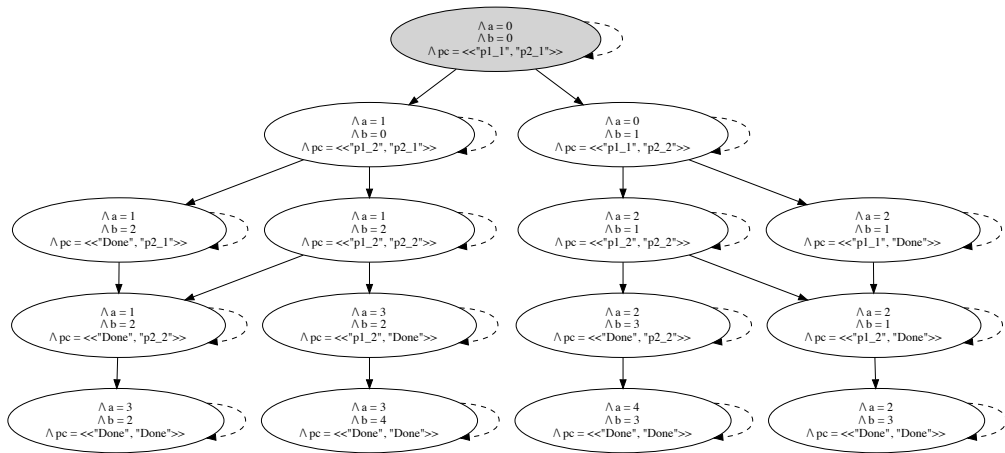
/\ pc = <<"p1_2", "Done">>

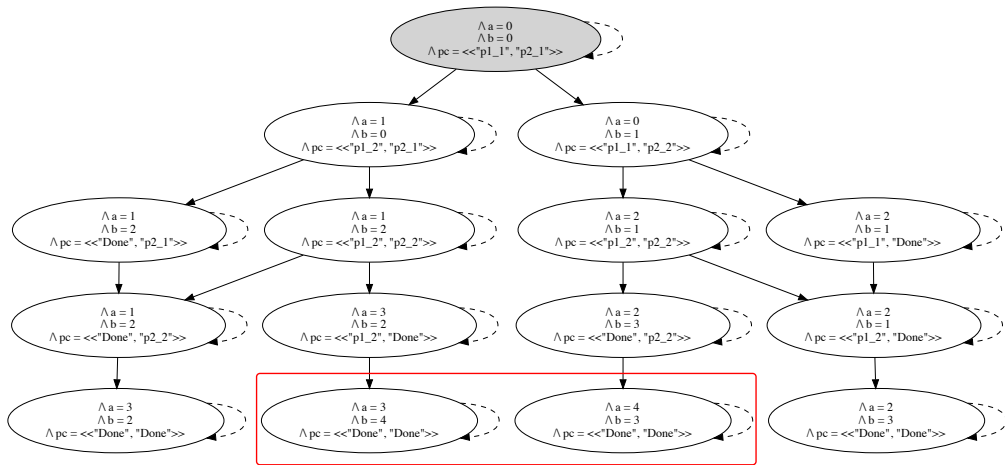
State 5: <p1_2 line 53, col 9 to line 56, col 17 of module 02_PCDemo>

/\ a = 3

/\ b = 4

/\ pc = <<"Done", "Done">>





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

```
9 func sender(ch chan *Data) {  
10     ch <- MESSAGE  
11     close(ch)  
12 }  
13  
14 func receiver(ch chan *Data) (result *Data) {  
15     for {  
16         if r := <-ch; r == nil {  
17             return  
18         } else {  
19             result = r  
20         }  
21     }  
22 }
```

Go Channel

```
1 ----- MODULE 03_GoChannel -----
2 INSTANCE TLC
3 CONSTANTS NULL, Message
4
5 (*--algorithm GoChannel
6 variables
7     ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
8     result = NULL;
9 define
10     Success == <>[](result = Message)
11
12     Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
13 end define;
```

Go Channel

```
1 ----- MODULE 03_GoChannel -----
2 INSTANCE TLC
3 CONSTANTS NULL, Message
4
5 (*--algorithm GoChannel
6 variables
7   ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
8   result = NULL;
9 define
10   Success == <>[](result = Message)
11
12   Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
13 end define;
```

Go Channel

```
1 ----- MODULE 03_GoChannel -----
2 INSTANCE TLC
3 CONSTANTS NULL, Message
4
5 (*--algorithm GoChannel
6 variables
7     ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
8     result = NULL;
9 define
10     Success == <>[](result = Message)
11
12     Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
13 end define;
```


Go Channel

```
1 ----- MODULE 03_GoChannel -----
2 INSTANCE TLC
3 CONSTANTS NULL, Message
4
5 (*--algorithm GoChannel
6 variables
7     ch = [open |-> TRUE, sent |-> FALSE, rcvd |-> FALSE, val |-> NULL],
8     result = NULL;
9 define
10     Success == <>[](result = Message)
11
12     Ordered == pc.sender = "Done" ~> pc.receiver = "Done"
13 end define;
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```


Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

Go Channel

```
15 fair process Sender = "sender"
16 begin
17     Send:
18         assert ch.open;
19         await ~ch.sent;
20         ch.val := Message || ch.sent :=
            ↪ TRUE;
21     WaitSent:
22         await ch.sent /\ ch.rcvd;
23         ch.sent := FALSE || ch.rcvd :=
            ↪ FALSE;
24     Close:
25         assert ch.open;
26         ch.open := FALSE;
27 end process;
```

```
29 fair process Receiver = "receiver"
30 begin Receive:
31     while TRUE do
32         either
33             await ch.sent /\ ~ch.rcvd;
34             result := ch.val || ch.val :=
                ↪ NULL || ch.rcvd := TRUE;
35         or
36             await ~ch.open;
37             goto Done;
38         end either;
39     end while;
40 end process;
41
42 end algorithm; *)
```

ДЕМО

03_GoChannel.tla

PlusCal

Макросы

```
14 macro send(ch, msg)
15 begin
16     assert ch.open;
17     await ~ch.sent;
18     ch.val := msg || ch.sent := TRUE;
19 end macro;
20
21 macro wait_sent(ch)
22 begin
23     await ch.sent /\ ch.rcvd;
24     ch.sent := FALSE || ch.rcvd :=
        ⇨ FALSE;
25 end macro;
```

```
27 macro recv(var, ch)
28 begin
29     await ~ch.open /\ (ch.sent /\
        ⇨ ~ch.rcvd);
30     if ~ch.open then var := NULL;
31     else
32         var := ch.val || ch.val := NULL
        ⇨ || ch.rcvd := TRUE;
33     end if;
34 end macro;
35
36 macro close(ch)
37 begin
38     assert ch.open;
39     ch.open := FALSE;
40 end macro;
```

PlusCal

Макросы

```
14 macro send(ch, msg)
15 begin
16     assert ch.open;
17     await ~ch.sent;
18     ch.val := msg || ch.sent := TRUE;
19 end macro;
20
21 macro wait_sent(ch)
22 begin
23     await ch.sent /\ ch.rcvd;
24     ch.sent := FALSE || ch.rcvd :=
25         ↪ FALSE;
26 end macro;
```

```
27 macro recv(var, ch)
28 begin
29     await ~ch.open /\ (ch.sent /\
30         ↪ ~ch.rcvd);
31     if ~ch.open then var := NULL;
32     else
33         var := ch.val || ch.val := NULL
34         ↪ || ch.rcvd := TRUE;
35     end if;
36 end macro;
37
38 macro close(ch)
39 begin
40     assert ch.open;
41     ch.open := FALSE;
42 end macro;
```

PlusCal

Макросы

```
14 macro send(ch, msg)
15 begin
16     assert ch.open;
17     await ~ch.sent;
18     ch.val := msg || ch.sent := TRUE;
19 end macro;
20
21 macro wait_sent(ch)
22 begin
23     await ch.sent /\ ch.rcvd;
24     ch.sent := FALSE || ch.rcvd :=
        ⇨ FALSE;
25 end macro;
```

```
27 macro recv(var, ch)
28 begin
29     await ~ch.open /\ (ch.sent /\
        ⇨ ~ch.rcvd);
30     if ~ch.open then var := NULL;
31     else
32         var := ch.val || ch.val := NULL
        ⇨ || ch.rcvd := TRUE;
33     end if;
34 end macro;
35
36 macro close(ch)
37 begin
38     assert ch.open;
39     ch.open := FALSE;
40 end macro;
```


PlusCal

Макросы

```
14 macro send(ch, msg)
15 begin
16     assert ch.open;
17     await ~ch.sent;
18     ch.val := msg || ch.sent := TRUE;
19 end macro;
20
21 macro wait_sent(ch)
22 begin
23     await ch.sent /\ ch.rcvd;
24     ch.sent := FALSE || ch.rcvd :=
        ⇨ FALSE;
25 end macro;
```

```
27 macro recv(var, ch)
28 begin
29     await ~ch.open /\ (ch.sent /\
        ⇨ ~ch.rcvd);
30     if ~ch.open then var := NULL;
31     else
32         var := ch.val || ch.val := NULL
        ⇨ || ch.rcvd := TRUE;
33     end if;
34 end macro;
35
36 macro close(ch)
37 begin
38     assert ch.open;
39     ch.open := FALSE;
40 end macro;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45   while nextId \in 1..MaxMsg do
```

```
46     i := nextId || sent :=
```

```
      ⇨ Append(sent, nextId) ||
```

```
      ⇨ nextId := nextId + 1;
```

```
47     Send_1: send(ch, i);
```

```
48     Send_2: wait_sent(ch);
```

```
49   end while;
```

```
50   close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56   while TRUE do
```

```
57     recv(j, ch);
```

```
58     if j /= NULL then
```

```
59       recvd := Append(recvd, j);
```

```
60     else
```

```
61       goto Done;
```

```
62     end if;
```

```
63   end while;
```

```
64 end process;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45   while nextId \in 1..MaxMsg do
```

```
46     i := nextId || sent :=
```

```
       ⇨ Append(sent, nextId) ||
```

```
       ⇨ nextId := nextId + 1;
```

```
47     Send_1: send(ch, i);
```

```
48     Send_2: wait_sent(ch);
```

```
49   end while;
```

```
50   close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56   while TRUE do
```

```
57     recv(j, ch);
```

```
58     if j /= NULL then
```

```
59       recvd := Append(recvd, j);
```

```
60     else
```

```
61       goto Done;
```

```
62     end if;
```

```
63   end while;
```

```
64 end process;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45   while nextId \in 1..MaxMsg do
```

```
46     i := nextId || sent :=
```

```
      ↪ Append(sent, nextId) ||
```

```
      ↪ nextId := nextId + 1;
```

```
47     Send_1: send(ch, i);
```

```
48     Send_2: wait_sent(ch);
```

```
49   end while;
```

```
50   close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56   while TRUE do
```

```
57     recv(j, ch);
```

```
58     if j /= NULL then
```

```
59       recvd := Append(recvd, j);
```

```
60     else
```

```
61       goto Done;
```

```
62     end if;
```

```
63   end while;
```

```
64 end process;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45     while nextId \in 1..MaxMsg do
```

```
46         i := nextId || sent :=
```

```
            ↪ Append(sent, nextId) ||
```

```
            ↪ nextId := nextId + 1;
```

```
47         Send_1: send(ch, i);
```

```
48         Send_2: wait_sent(ch);
```

```
49     end while;
```

```
50     close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56     while TRUE do
```

```
57         recv(j, ch);
```

```
58         if j /= NULL then
```

```
59             recvd := Append(recvd, j);
```

```
60         else
```

```
61             goto Done;
```

```
62         end if;
```

```
63     end while;
```

```
64 end process;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45     while nextId \in 1..MaxMsg do
```

```
46         i := nextId || sent :=
```

```
            ↪ Append(sent, nextId) ||
```

```
            ↪ nextId := nextId + 1;
```

```
47         Send_1: send(ch, i);
```

```
48         Send_2: wait_sent(ch);
```

```
49     end while;
```

```
50     close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56     while TRUE do
```

```
57         recv(j, ch);
```

```
58         if j /= NULL then
```

```
59             recvd := Append(recvd, j);
```

```
60         else
```

```
61             goto Done;
```

```
62         end if;
```

```
63     end while;
```

```
64 end process;
```

Go Channel

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

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

```
42 fair process Sender = "sender"
```

```
43 variable i = 0;
```

```
44 begin Send:
```

```
45   while nextId \in 1..MaxMsg do
```

```
46     i := nextId || sent :=
```

```
      ⇨ Append(sent, nextId) ||
```

```
      ⇨ nextId := nextId + 1;
```

```
47     Send_1: send(ch, i);
```

```
48     Send_2: wait_sent(ch);
```

```
49   end while;
```

```
50   close(ch);
```

```
51 end process;
```

```
53 fair process Receiver = "receiver"
```

```
54 variable j = 0;
```

```
55 begin Receive:
```

```
56   while TRUE do
```

```
57     recv(j, ch);
```

```
58     if j /= NULL then
```

```
59       recvd := Append(recvd, j);
```

```
60     else
```

```
61       goto Done;
```

```
62     end if;
```

```
63   end while;
```

```
64 end process;
```

ДЕМО

04_GoChannel.tla

Go Channel Bug

05_channelbug_test.go

```
10 func proc(to time.Duration) int {
11     ch := make(chan int)
12     go func() {
13         // do some processing
14         ch <- RESULT
15     }()
16     select {
17     case rslt := <-ch:
18         return rslt
19     case <-time.After(to):
20         return 0
21     }
22 }
```

Go Channel Bug

05_channelbug_test.go

```
10 func proc(to time.Duration) int {
11     ch := make(chan int)
12     go func() {
13         // do some processing
14         ch <- RESULT
15     }()
16     select {
17     case rslt := <-ch:
18         return rslt
19     case <-time.After(to):
20         return 0
21     }
22 }
```

05_ChannelBug.tla

```
24 fair process processor = "processor"
25 begin
26     Proc_1: send(ch, "result");
27     Proc_2: wait_sent(ch);
28 end process;
29
30 fair process receiver = "receiver"
31 variable rslt = NULL;
32 begin Rec_1:
33     either
34         recv(rslt, ch);
35     or
36         skip;
37     end either;
38 end process;
```

Go Channel Bug

05_channelbug_test.go

```
10 func proc(to time.Duration) int {
11     ch := make(chan int)
12     go func() {
13         // do some processing
14         ch <- RESULT
15     }()
16     select {
17     case rslt := <-ch:
18         return rslt
19     case <-time.After(to):
20         return 0
21     }
22 }
```

05_ChannelBug.tla

```
24 fair process processor = "processor"
25 begin
26     Proc_1: send(ch, "result");
27     Proc_2: wait_sent(ch);
28 end process;
29
30 fair process receiver = "receiver"
31 variable rslt = NULL;
32 begin Rec_1:
33     either
34         recv(rslt, ch);
35     or
36         skip;
37     end either;
38 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

```
8 func buffer1(inpCh, outCh chan int) {
9     var oC chan int
10    var out int
11    var buf []int
12    defer close(outCh)
13    for {
14        select {
15        case inp, ok := <-inpCh:
16            if !ok {
17                return
18            } else if oC != nil {
19                buf = append(buf, inp)
20            } else {
21                out = inp
22                oC = outCh
23            }
24        }
```

```
24        case oC <- out:
25            if len(buf) > 0 {
26                out, buf = headtail(buf)
27            } else {
28                oC = nil
29            }
30        }
31    }
32 }

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


Go Unbound Queue

```
56 fair process Buffer = "buffer"
57 variable k = NULL, buffer = <<>>;
58 begin BufferProcess:
59   while TRUE do
60     either
61       recv(k, inpCh);
62     Buffer_1:
63       if k /= NULL then
64         buffer := Append(buffer, k);
65       else
66         close(outCh);
67         goto Done;
68       end if;
```

```
69   or
70     when Len(buffer) > 0;
71     Buffer_2:
72       send(outCh, Head(buffer));
73     Buffer_3:
74       wait_sent(outCh);
75       buffer := Tail(buffer);
76     end either;
77   end while;
78 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)

```
54 fair process Buffer = "buffer"
55 variable k = NULL, buffer = <<>>,
56     exit = FALSE;
57 begin BufferProcess:
58     while TRUE do
59         either
60             when ~exit;
61             recv(k, inpCh);
62         Buffer_1:
63             if k /= NULL then
64                 buffer := Append(buffer, k);
65             else
66                 exit := TRUE;
67             end if;
```

```
68     or
69         when Len(buffer) > 0;
70     Buffer_2:
71         send(outCh, Head(buffer));
72     Buffer_3:
73         wait_sent(outCh);
74         buffer := Tail(buffer);
75     or
76         when Len(buffer) = 0 /\ exit;
77         close(outCh);
78         goto Done;
79     end either;
80 end while;
81 end process;
```

ДЕМО

07_UnboundQueue.tla

Go Unbound Queue (fixed)

```
5 func buffer2(inpCh, outCh chan int) {  
6     var exit bool  
7     var oC chan int  
8     var out int  
9     var buf []int  
10    defer close(outCh)  
11    for {  
12        select {  
13        case inp, ok := <-inpCh:  
14            if !ok {  
15                if oC == nil {  
16                    return  
17                } else {  
18                    exit = true  
19                    inpCh = nil  
20                }  
11
```

```
21        } else if oC != nil {  
22            buf = append(buf, inp)  
23        } else {  
24            out = inp  
25            oC = outCh  
26        }  
27        case oC <- out:  
28            if len(buf) > 0 {  
29                out, buf = headtail(buf)  
30            } else if !exit {  
31                oC = nil  
32            } else {  
33                return  
34            }  
35        }  
36    }  
37 }
```

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 CosmosDB³

“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.”

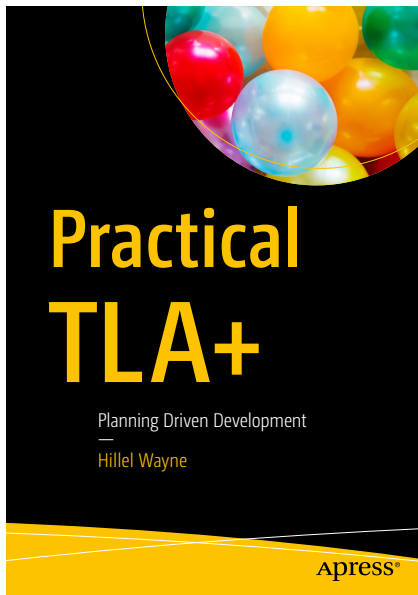
1. conf.tlapl.us/07_-_TLAConf19_-_William_Schultz_-_Fixing_a_MongoDB_Replication_Protocol_Bug_with_TLA.pdf

2. github.com/will62794/mongo-repl-tla-models

3. www.cockroachlabs.com/blog/parallel-commits

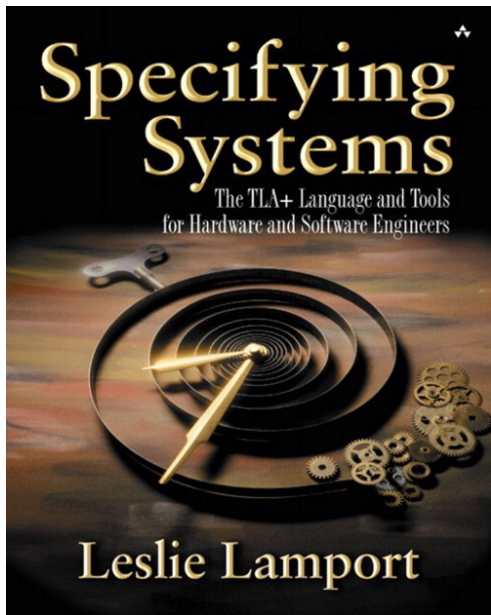
4. github.com/cockroachdb/cockroach/tree/master/docs/tla-plus

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!)

Introduction to TLA+

TLA+ Video Course — Lecture 1
Leslie Lamport

INTRODUCTION TO TLA+

▶ 0:00 / 20:18



← skip →

Speed

Contents

How to View

Lecture 1

Web page: 20 March 2017

Video: 17 March 2017

Contents

Prologue

What is TLA+?

Abstraction

What Engineers Say

What Can TLA+ Check?

The Basic Abstraction

State Machines

A Tiny Example

Epilogue

TLA+ Video Course

by Leslie Lamport

lamport.azurewebsites.net/video/videos.html

3.5 часа TLA+, обязательно
посмотреть

(Лампорт 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!



ITOOLABS 

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