

Oficina: Introdução à Especificação Formal de Sistemas com TLA+

Jefferson O. Andrade

1. Introdução

1.1. Objetivos do Mini-curso

- Introduzir PlusCal.
- Ensinar fundamentos da linguagem TLA+.
- Aplicações práticas.

1.2. O que é TLA+?

TLA+ é uma linguagem para escrever e verificar “especificações”, ou designs de sistemas.

- Com a especificação, você pode então testar a especificação diretamente para bugs, mesmo antes de ter escrito qualquer código.
- Verificação de propriedades como segurança e vivacidade.

1.3. Quem Criou TLA+?

Leslie Lamport, que também é a pessoa por trás da tolerância a falhas bizantina, Paxos e LaTeX.

Curiosidade: LaTeX é a abreviação de “Lamport’s TeX”!

2. Começando: Configuração do TLA+ Toolbox

2.1. Por que usar o TLA+ Toolbox?

- Ferramenta gráfica para especificação formal.
- Auxilia na escrita e verificação de modelos TLA+.

2.2. Baixando o TLA+ Toolbox

- Acesse o site oficial: <https://lamport.azurewebsites.net/tla/toolbox.html>
 - Ou acesse o repositório no Github: <https://github.com/tlaplus/tlaplus/releases>
 - Ou acesse o Flathub: <https://flathub.org/apps/org.lamport.tla.toolbox>
- Selecione a versão compatível com seu sistema operacional (Windows, macOS ou Linux).

2.3. Instalando o TLA+ Toolbox

Instalação no Windows

- **Baixe e Extraia o Arquivo:** Faça o download do arquivo zip e extraia-o em uma pasta conveniente.
- **Executando o Toolbox:** Abra a subpasta toolbox e execute o arquivo toolbox.exe. Para facilitar o acesso, crie um atalho ou "Pin to Start/taskbar".
- **Aviso de Segurança:** O sistema pode avisar que o programa não é seguro. Procure a opção "Executar de qualquer forma".

Instalação no Linux

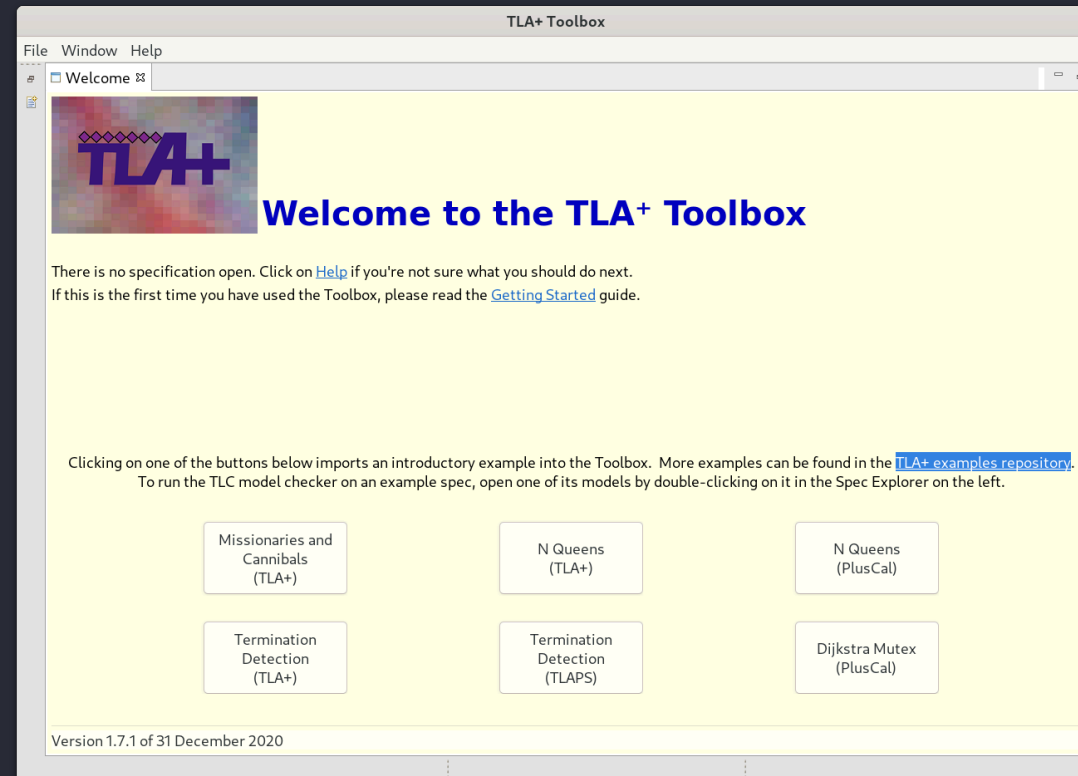
- **Extração do Arquivo:** Extraia o conteúdo do arquivo zip em um diretório conveniente.
- **Executando o Toolbox:** Navegue até a subpasta toolbox e execute o arquivo toolbox.

Instalação no MacOS

- **Download e Extração:** Baixe e descompacte o arquivo, resultando no TLA+ Toolbox.app. Mova-o para a pasta de Aplicativos.
- **Executando o Toolbox:** Dê um clique com o botão direito no app, selecione "Abrir" e confirme a execução.
- **Alternativa com Homebrew:** Se você usa Homebrew, pode instalar com o comando `brew install tla-plus-toolbox`.

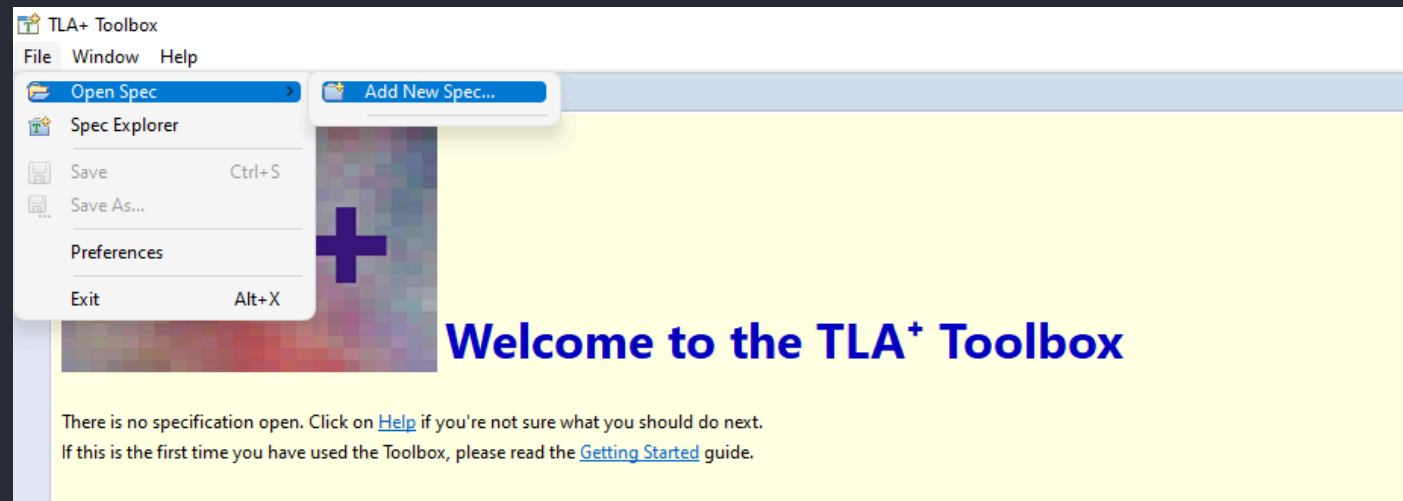
2.4. Executando o TLA+ Toolbox

Se o TLA+ Toolbox foi instalado corretamente você verá a tela abaixo:

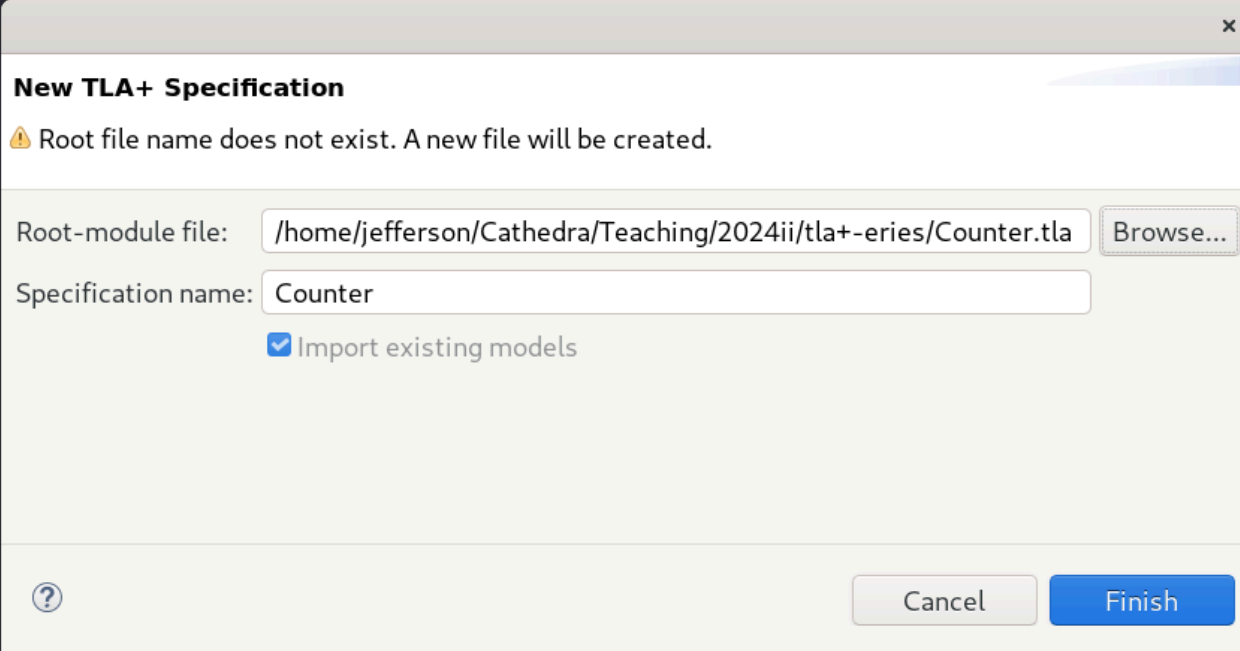


2.5. Criando um Novo Projeto

- Abra o Toolbox e crie um novo projeto.
- Selecione **File > Open Spec > Add New Spec.**



- Indique o nome do arquivo com a especificação (**Counter.tla**)
- Indique o nome do módulo (**Counter**)



A screenshot of a 'New TLA+ Specification' dialog box. The title bar is light gray with a close button (X) in the top right corner. The main area has a white background. At the top, the title 'New TLA+ Specification' is in bold. Below it, a warning icon (yellow triangle with an exclamation mark) is followed by the text 'Root file name does not exist. A new file will be created.' The dialog contains two text input fields: 'Root-module file:' with the path '/home/jefferson/Cathedra/Teaching/2024ii/tla+-eries/Counter.tla' and a 'Browse...' button to its right; and 'Specification name:' with the text 'Counter'. Below the 'Specification name' field is a checked checkbox labeled 'Import existing models'. At the bottom left is a help icon (question mark in a circle). At the bottom right are two buttons: 'Cancel' and 'Finish'.

New TLA+ Specification

⚠ Root file name does not exist. A new file will be created.

Root-module file:

Specification name:

☒ Import existing models

2.6. Estrutura Básica do Projeto

- Se tudo correu bem, você deve ver alguma coisa assim:

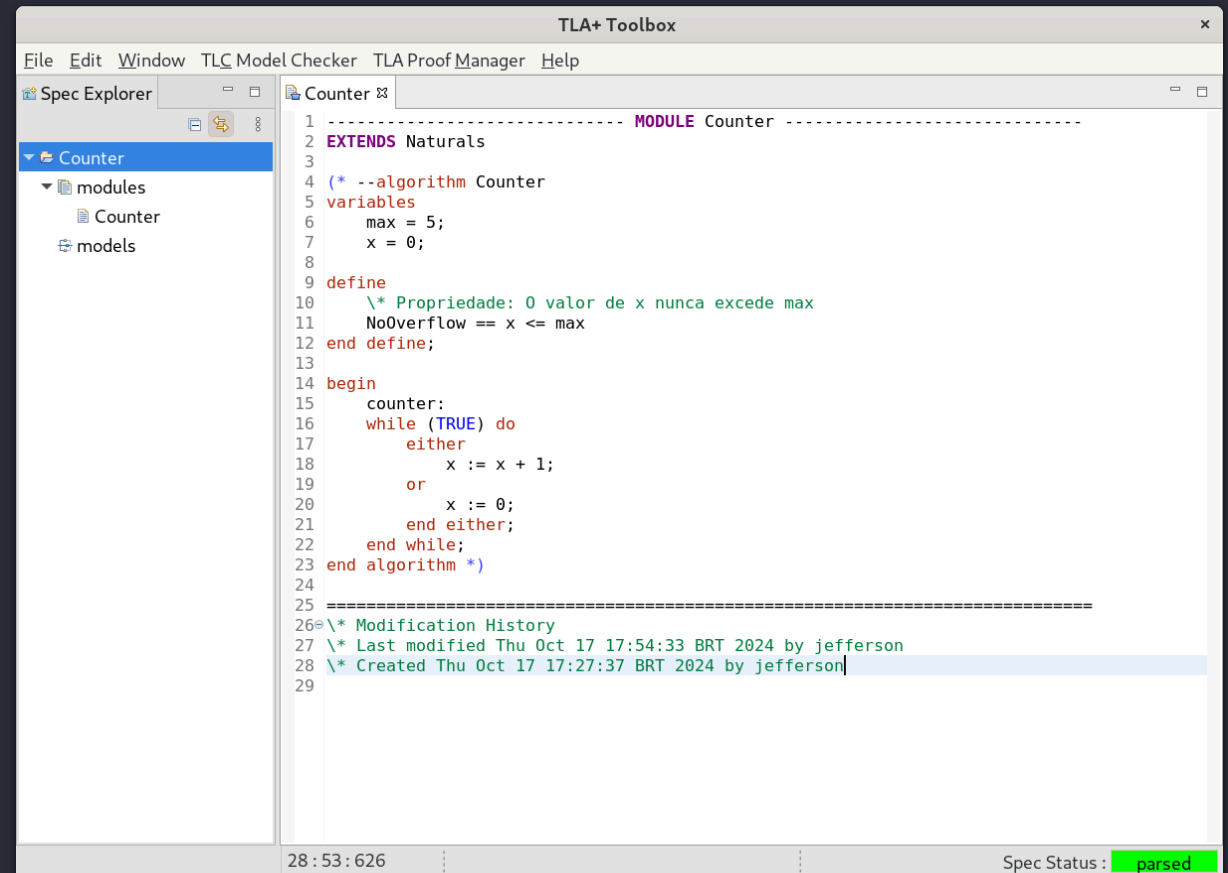
```

----- MODULE counter -----
\* Modification History
\* Created Thu Oct 06 14:33:50 BRT 2024 by jefferson
```

- Por razões históricas, **MODULE \$name** precisa estar cercado de ao menos 4 hífen (-----).
- O módulo precisa terminar com ao menos 4 sinais de igual ('====').

2.7. Primeira Especificação

Vamos substituir o modelo com o conteúdo do módulo **Counter**.



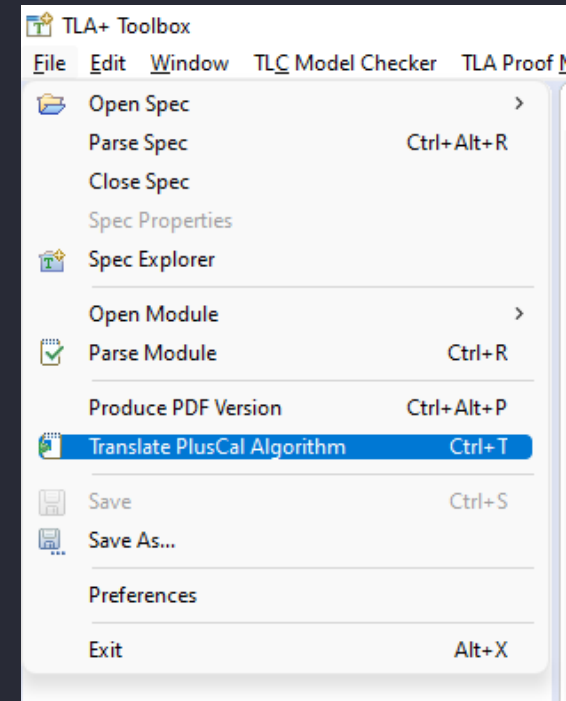
The screenshot shows the TLA+ Toolbox interface. On the left, the 'Spec Explorer' pane displays a tree structure with 'Counter' selected under 'modules'. The main editor shows the 'Counter' module specification, which includes extending 'Naturals', defining variables 'max' and 'x', and a 'begin' block with a 'counter' section containing a 'while' loop. The status bar at the bottom indicates 'Spec Status: parsed'.

```
1 ----- MODULE Counter -----
2 EXTENDS Naturals
3
4 (* --algorithm Counter
5 variables
6   max = 5;
7   x = 0;
8
9 define
10   \* Propriedade: 0 valor de x nunca excede max
11   NoOverflow == x <= max
12 end define;
13
14 begin
15   counter:
16     while (TRUE) do
17       either
18         x := x + 1;
19       or
20         x := 0;
21       end either;
22     end while;
23 end algorithm *)
24
25 =====
26 \* Modification History
27 \* Last modified Thu Oct 17 17:54:33 BRT 2024 by jefferson
28 \* Created Thu Oct 17 17:27:37 BRT 2024 by jefferson
29
```

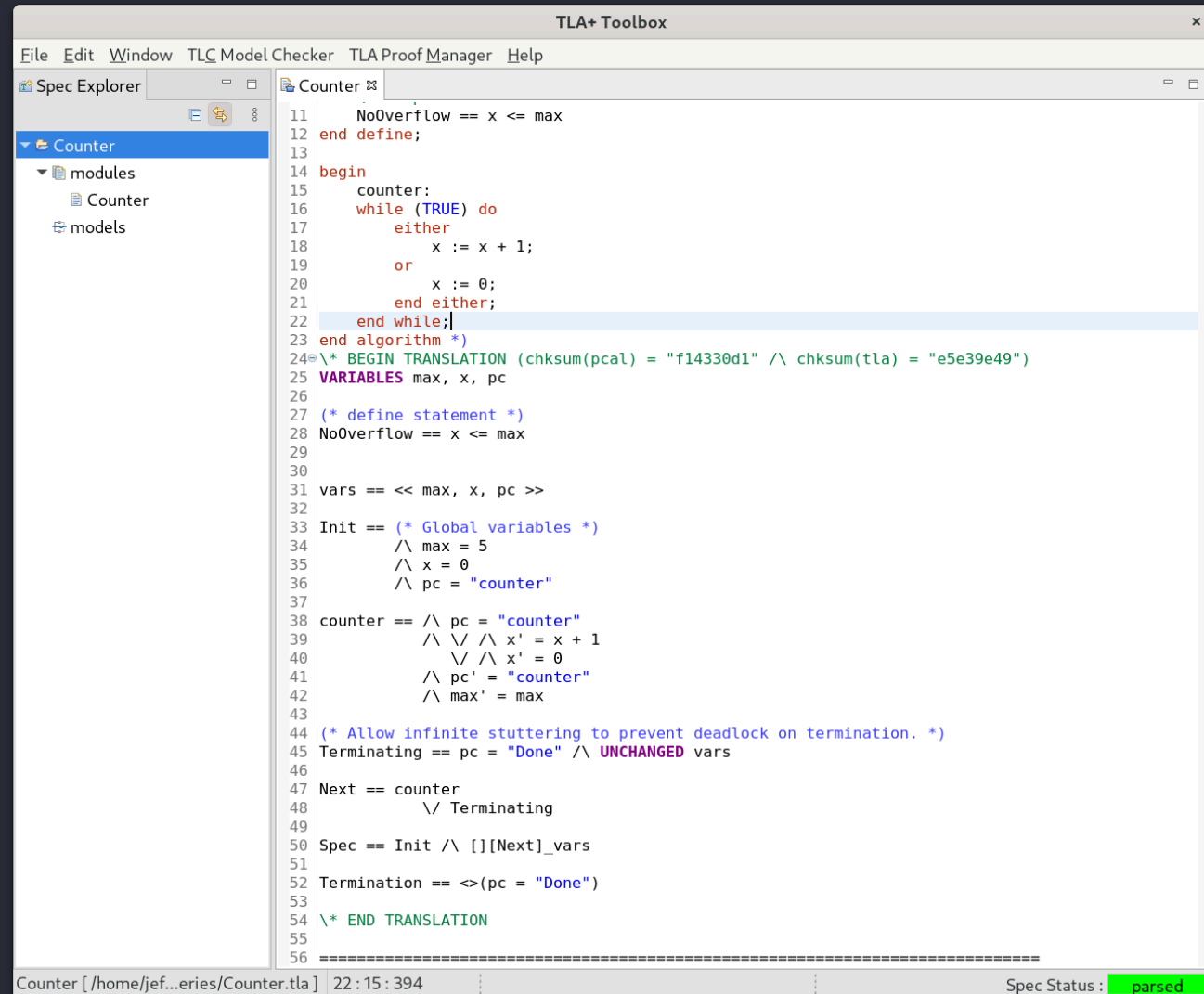
28 : 53 : 626 Spec Status : **parsed**

2.8. Traduzindo a Especificação

- Neste momento, não vamos entrar em detalhes de TLA+.
- Entretanto para que a especificação possa ser verificada, ela precisa ser traduzida para TLA+.
- Faça a tradução selecionando **File > Translate PlusCal Algorithm**.



Uma vez que tenha feito a tradução você deve ver:



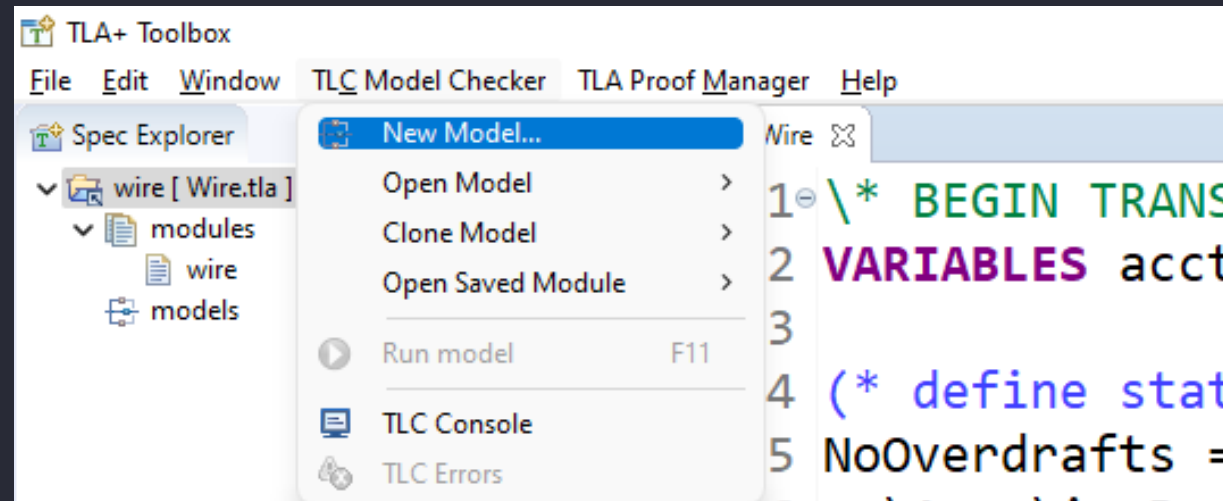
The screenshot shows the TLA+ Toolbox application window. The title bar is "TLA+ Toolbox". The menu bar includes "File", "Edit", "Window", "TLC Model Checker", "TLA Proof Manager", and "Help". The left pane is the "Spec Explorer" showing a tree structure with "Counter" selected under "modules". The main pane displays the Counter.tla specification code, which includes a while loop for a counter, variable declarations, and termination conditions. The status bar at the bottom shows the file path, time, and "Spec Status: parsed".

```
11 NoOverflow == x <= max
12 end define;
13
14 begin
15   counter:
16     while (TRUE) do
17       either
18         x := x + 1;
19       or
20         x := 0;
21       end either;
22     end while;
23 end algorithm *)
24 \* BEGIN TRANSLATION (chksum(pcal) = "f14330d1" /\ chksum(tla) = "e5e39e49")
25 VARIABLES max, x, pc
26
27 (* define statement *)
28 NoOverflow == x <= max
29
30
31 vars == << max, x, pc >>
32
33 Init == (* Global variables *)
34   /\ max = 5
35   /\ x = 0
36   /\ pc = "counter"
37
38 counter == /\ pc = "counter"
39            /\ \/ /\ x' = x + 1
40              \/ /\ x' = 0
41            /\ pc' = "counter"
42            /\ max' = max
43
44 (* Allow infinite stuttering to prevent deadlock on termination. *)
45 Terminating == pc = "Done" /\ UNCHANGED vars
46
47 Next == counter
48       \/ Terminating
49
50 Spec == Init /\ [][Next]_vars
51
52 Termination == <>(pc = "Done")
53
54 \* END TRANSLATION
55
56 =====
```

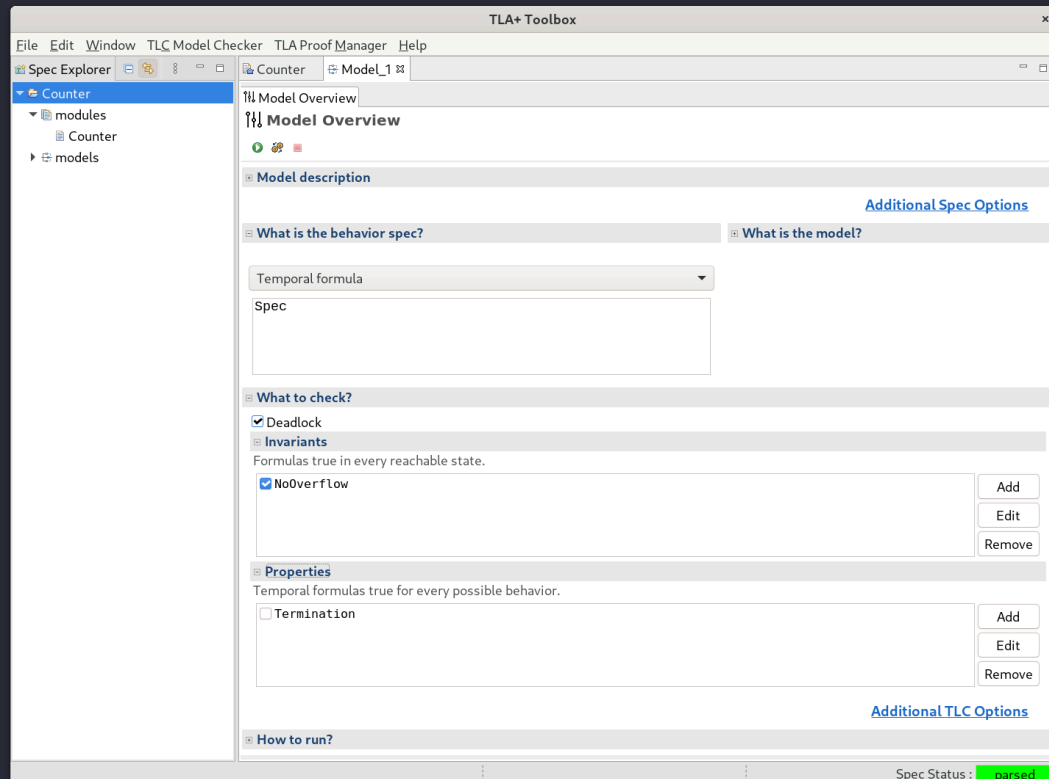
Counter [/home/jef...eries/Counter.tla] 22:15:394 Spec Status: **parsed**

2.9. Execução do Model Checker (TLC)

- Para realmente verificar a especificação com TLC, temos que criar um novo modelo para verificar.
- Faça isso em **TLC Model Checker > New Model**.



- Informe o nome do modelo (e.g., **Model_1**) e você verá a tela abaixo:



- “What is the behavior spec?” deve ser “Temporal Formula” e “Spec”. Se não for, certifique-se de ter apenas um conjunto de **====** na especificação, e o TLA+ traduzido está acima dele, então defina manualmente os dois campos.
- Clique na caixa “Invariants” para abri-la.
- Clique em “Add” e então insira o texto **NoOverflow**.
- Execute o modelo, ou pressione F11.

Ao executar isso, você verá um erro aparecer no lado direito:

The screenshot displays the TLA+ Toolbox interface. The main window is titled 'TLA+ Toolbox'. The left sidebar shows a project tree with 'Counter' selected. The main pane is divided into several sections:

- Model Checking Results**: Shows the start and end times of the model check. A red banner indicates '1 Error'.
- Statistics**: A table showing state space progress.
- Evaluate Constant Expression**: A section for evaluating expressions.
- User Output**: A section for user-generated output.
- Progress Output**: A section for progress output.

The **Statistics** table is as follows:

Time	Diam	States F	Distinct S	Queue S	Module	Action	Location	States Found	Di
00:00:01	7	12	7	0	Counter	Terminating	line 45, col 1 to line 45, col 11	0	
00:00:01	0	1	1	1	Counter	Init	line 33, col 1 to line 33, col 4	2	
					Counter	counter	line 38, col 1 to line 38, col 7	23	6

The **Error-Trace Exploration** pane shows the error trace for the 'Terminating' error. It lists the states and actions that led to the error:

- <Initial predicate State (num = 1)**: max = 5, pc = "counter", x = 0.
- <counter line 38 State (num = 2)**: max = 5.

The **Error-Trace** pane also includes instructions for navigating the error trace.

Este é um rastreamento de erro, mostrando o conjunto exato de etapas que levam a uma invariante sendo violada.

3. PlusCal: Introdução

3.1. O que é PlusCal?

- Linguagem de pseudocódigo que facilita a escrita de algoritmos.
- Tradução automática para TLA+.

3.2. Duas Sintaxes

O PlusCal tem duas sintaxes separadas, a **sintaxe-p**, prolixa, e a **sintaxe-c**, mais compacta.

Definição completa da sintaxe-p em <https://lamport.azurewebsites.net/tla/p-manual.pdf> .

Sitaxe-p:

```
while x > 0 do
  if y > 0 then
    y := y-1;
    x := x-1
  else
    x := x-2
  end if
end while;
print y;
```

Sitaxe-c:

```
while (x > 0) {
  if (y > 0) {
    y := y-1;
    x := x-1
  }
  else x := x-2
};
print y;
```

3.3. Resumo da Sintaxe p

Estrutura básica

- `--algorithm NomeDoAlgoritmo begin ... end algorithm`: Define um algoritmo com um nome específico.
- `variable x = 1, y = 2;`: Declara variáveis e seus valores iniciais.
- `begin ... end`: Delimita blocos de código do algoritmo.

Comandos

- `x := expressão;`: Atribui o valor da expressão à variável `x`.
- `if condição then ... [elsif condição then ...] else ... end if`: Executa blocos de código diferentes dependendo da condição.
- `while condição do ... end while`: Repete um bloco de código enquanto a condição for verdadeira.
- `either ... or ... end either`: Escolhe não deterministicamente um dos blocos de código para executar.
- `with x \in S do ... end with`: Escolhe não deterministicamente um valor do conjunto `S` e atribui a `x`.
- `print expressão;`: Imprime o valor da expressão.
- `assert expressão;`: Verifica se a expressão é verdadeira; caso contrário, relata um erro.
- `skip;`: Não faz nada.
- `goto Rótulo;`: Salta para o rótulo especificado.
- `Rótulo:`: Define um rótulo para um comando.
- `await expressão;`: Espera valor da expressão booleana ser `TRUE`.

Processos

- Um algoritmo multiprocesso contém um ou mais processos.
- Um processo começa de uma das duas maneiras:
 - `process ProcName ∈ IdSet`: Define um conjunto de processos.
 - `process P = Id`: Define um processo individual chamado P.
- `variable x = 1;`: Declara uma variável local ao processo.

Macros

- Uma macro é como um procedimento, exceto que uma chamada de macro é expandida no momento da tradução.
- `macro M(arg1, arg2, ...) ... end macro`: Declara a macro `M`.

```
macro M(s, i)
begin
  await s ≥ i;
  s := s - i;
end macro;
```

- O corpo da macro não pode conter rótulos, nenhuma instrução `while`, `call`, `return` ou `goto`.
- `M(val1, val2, ...)`: Chama a macro `M` (não precisa de `call`).

```
M(sem, y+17);
```

Expande para:

```
await sem ≥ (y + 17);
sem := sem - (y + 17);
```

Definições

- A instrução PlusCal **define** permite que você escreva definições TLA+ de operadores que dependem das variáveis globais do algoritmo.
- Por exemplo:

```
--algorithm Test
variables x \in 1..N; y;
define zy = y*(x+y)
      zx(a) = x*(y-a)
end define;
```


Comunicação entre processos:

- **Canais**: Permitem a comunicação entre processos através de mensagens.
- `c!expressão`: Envia o valor da expressão pelo canal `c`.
- `c?variável`: Recebe um valor do canal `c` e atribui à variável.

Exemplo

```
—— MODULE ExemploP ——  
(* --algorithm Exemplo  
variable x = 1, y = 2;  
  
begin  
  if x < y then  
    x := x + 1;  
  else  
    y := y - 1;  
  end if;  
  print x, y;  
end algorithm *)  
=====
```

3.4. Exemplo: Algoritmo de Exclusão Mútua

```
MODULE FastMutex
EXTENDS Naturals, TLC
CONSTANT N

Procs = 1..N

(* --algorithm FastMutex
variables x , y = 0 , b = [i ∈ 1..N 7 ↦ FALSE];
process Proc \in Procs;
variable j;
begin
  ncs: while TRUE do
    skip ; \* The noncritical section.
    start: b[self] := TRUE;
    l1: x := self;
    l2: if y ≠ 0 then
      l3: b[self] := FALSE;
      l4: await y = 0;
      goto start;
    end if;
    l5: y := self;
    l6: if x ≠ self then
      l7: b[self] := FALSE;
      j := 1;
      l8: while j ≤ N do await ~b[j] ;
        j := j+1
      end while ;
      l9: if y ≠ self then
        l10: await y = 0 ;
        goto start ;
      end if ;
    end if;
    cs: skip ; \* The critical section.
    l11: y := 0 ;
    l12: b[self] := FALSE ;
  end while ;
end process
end algorithm *)
```

4. Exemplo: Integridade de Transferências Bancárias

4.1. Introdução

- Problema: Transferências Bancárias Concorrentes
 - Alice e Bob têm contas no Bankgroup, cada uma com um saldo inicial.
 - O Bankgroup quer implementar uma nova funcionalidade de "transferência bancária" que permita que os usuários transfiram dinheiro entre si.
- Requisitos da Transferência:
 - A transferência deve ocorrer entre pessoas diferentes e deve transferir pelo menos 1 dólar.
 - Se bem-sucedida, o valor é debitado da conta do remetente e creditado na conta do destinatário.
 - Se falhar, os saldos permanecem inalterados.
 - Nenhuma transferência pode resultar em saldo negativo.

4.2. Requisitos de Concorrência

- Escalabilidade e Concorrência
 - O sistema deve suportar múltiplas transferências simultâneas.
 - As transferências podem ter diferentes tempos de execução, ou seja, uma transferência iniciada primeiro pode ser concluída depois de outra.
- Objetivo do Algoritmo:
 - Garantir que todas as propriedades e restrições sejam satisfeitas, independentemente da ordem de início e término das transferências.

4.3. Objetivo

- **Garantir transferências seguras e consistentes entre contas**, mesmo em situações de concorrência, preservando a integridade dos saldos e a ordem das operações.
- **Manter a consistência do sistema**, independentemente do número de transferências simultâneas.

4.4. Propriedades

1. **Exclusão Mútua:** Nenhuma transferência deve permitir que o saldo de uma conta seja negativo.
2. **Conservação de Valor:** O total de dinheiro no sistema deve permanecer constante (exceto em falhas, onde nenhuma mudança ocorre).
3. **Isolamento de Transações:** O resultado de transferências concorrentes deve ser o mesmo como se elas ocorressem em sequência, independentemente da ordem de finalização.
4. **Validade da Transferência:** Cada transferência deve ocorrer entre duas contas diferentes e transferir pelo menos 1 dólar.

4.5. *Boilerplate*

```
————— MODULE wire —————  
EXTENDS Integers  
(*--algorithm wire  
begin  
    skip;  
end algorithm;*)  
=====
```

4.6. Especificando — Variáveis

4.6. Especificando — Variáveis

- Base

```
EXTENDS Integers
(* --algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5];
begin
  skip;
end algorithm; *)
```

4.6. Especificando — Variáveis

- Base

```
EXTENDS Integers
(* --algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5];
begin
  skip;
end algorithm; *)
```

- Variáveis para transferência única

```
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],
  sender = "alice",
  receiver = "bob",
  amount = 3;
```

4.7. Especificando — Invariante

```
EXTENDS Integers
(* --algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],
  sender = "alice",
  receiver = "bob",
  amount = 3;

define
  NoOverdrafts =  $\forall p \in \text{people}: \text{acc}[p] \geq 0$ 
end define;

begin
  skip;
end algorithm; *)
```

4.8. Implementando — Transferência Única

```
EXTENDS Integers
(*--algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],
  sender = "alice",
  receiver = "bob",
  amount = 3;

define
  NoOverdrafts =  $\forall p \in \text{people}: \text{acc}[p] \geq 0$ 
end define;

begin
  Withdraw:
    acc[sender] := acc[sender] - amount;
  Deposit:
    acc[receiver] := acc[receiver] + amount;
end algorithm;*)
```

4.9. Traduzir e verificar — Transferência Única

- Traduzir
 - `File > Translate PlusCal Algorithm`
- Verificar
 - Criar modelo
 - `TLC Model Checker > New Model`
 - Model Overview
 - What is the behavior spec?: Temporal Formula / Spec
 - Invariants: `NoOverdrafts`
- Tudo OK. 4 estados.

4.10. Novas Condições Iniciais

```
EXTENDS Integers
(*--algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],
  sender = "alice",
  receiver = "bob",
  amount \in 1..6;
define
  NoOverdrafts =  $\forall p \in \text{people}: \text{acc}[p] \geq 0$ 
end define;
begin
  Withdraw:
    acc[sender] := acc[sender] - amount;
  Deposit:
    acc[receiver] := acc[receiver] + amount;
end algorithm;*)
```


4.11. Traduzir e Verificar — Novas Condições Iniciais

- **Nem tudo OK!**
- Solução temporária: `amount \in 1..acc[sender]`

4.12. Múltiplos Processos

```
EXTENDS Integers
(*--algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5];

define
  NoOverdrafts =  $\forall p \in \text{people}: \text{acc}[p] \geq 0$ 
end define;

process Wire \in 1..2
variables
  sender = "alice",
  receiver = "bob",
  amount \in 1..acc[sender];
begin
  Withdraw:
    acc[sender] := acc[sender] - amount;
  Deposit:
    acc[receiver] := acc[receiver] + amount;
end process;
end algorithm;*)
```

4.13. Traduzir e Verificar — Múltiplos Processos

- **Nem tudo OK!**
- Múltiplas transferências em conflito!

4.14. Testar Saldo

EXTENDS Integers

(*--algorithm wire

variables

 people = {"alice", "bob"},
 acc = [p \in people \mapsto 5],

define

 NoOverdrafts = $\forall p \in \text{people}: \text{acc}[p] \geq 0$

end define;

process Wire \in 1..2

variables

 sender = "alice",
 receiver = "bob",
 amount \in 1..acc[sender];

begin

 CheckFunds:

 if amount \leq acc[sender] then

 Withdraw:

 acc[sender] := acc[sender] - amount;

 Deposit:

 acc[receiver] := acc[receiver] + amount;

 end if;

end process;

end algorithm;*)

4.15. Traduzir e Verificar — Testar Saldo

- **Este também falha!**
- Este erro é mais complicado...
 - Mesmo verificando que Alice tem dinheiro suficiente, ambos os processos podem passar na verificação antes de qualquer um deles sacar.
 - Detectamos uma **condição de corrida** em nosso código.

4.16. Transação Atômica

- E se colocássemos o teste e o saque no mesmo rótulo?

```
(*--algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],

define
  NoOverdrafts =  $\forall$  p \in people: acc[p]  $\geq$  0
end define;

process Wire \in 1..2
variables
  sender = "alice",
  receiver = "bob",
  amount \in 1..acc[sender];
begin
  CheckAndWithdraw:
    if amount  $\leq$  acc[sender] then
      acc[sender] := acc[sender] - amount;
      Deposit:
        acc[receiver] := acc[receiver] + amount;
    end if;
end process;
end algorithm;*)
```

- Parece que funciona (332 estados).

4.17. Propriedades Temporais

- *“Se a transferência falhar, a conta não será alterada.”*
- Ao contrário do **NoOverdrafts**, esta é uma **Propriedade Temporal**.
- Invariantes simples verificam se cada estado da especificação é válido.
- Propriedades temporais verificam se cada “tempo de vida” possível do algoritmo, do início ao fim, obedece a algo que relaciona diferentes estados na sequência entre si.
- Pense nisso como a diferença entre verificar se um banco de dados é “sempre consistente” versus “eventualmente consistente”.

4.18. Propriedades Temporais — Implementação

- Para o caso simples de duas pessoas, vamos verificar um requisito um pouco mais fraco, mas mais tratável:
 - “O valor final total das contas é o mesmo que o valor inicial total.”

```
EXTENDS Integers
(*--algorithm wire
variables
  people = {"alice", "bob"},
  acc = [p \in people  $\mapsto$  5],

define
  NoOverdrafts =  $\forall p \in \text{people}: \text{acc}[p] \geq 0$ 
  EventuallyConsistent =  $\Diamond[](\text{acc}["\text{alice}"] + \text{acc}["\text{bob}] = 10)$ 
end define;

process Wire \in 1..2
variables
  sender = "alice",
  receiver = "bob",
  amount \in 1..acc[sender];
begin
  CheckAndWithdraw:
    if amount  $\leq$  acc[sender] then
      acc[sender] := acc[sender] - amount;
    Deposit:
      acc[receiver] := acc[receiver] + amount;
    end if;
end process;
end algorithm;*)
```


5. Problema 1: Semáforo Simples

- **Objetivo:** Modelar um semáforo que alterna entre os estados "verde" e "vermelho".
- A propriedade verificada será que o semáforo nunca está ao mesmo tempo "verde" e "vermelho".

6. Problema 2: Contador Simples

- **Objetivo:** Implementar um contador que pode ser incrementado ou resetado.
- A propriedade que será verificada é se o contador nunca excede um determinado valor.

7. Problema 3: Produtor-Consumidor Simples

- **Objetivo:** Modelar um sistema simples de produtor e consumidor onde o produtor coloca itens em um buffer e o consumidor retira itens.
- A propriedade verificada será que o buffer nunca fica negativo.

8. Problema 4: Protocolo de Turno (Turn-Based)

- **Objetivo:** Modelar um protocolo simples de turnos entre dois processos.
- A propriedade verificada será que dois processos nunca podem ter o mesmo turno ao mesmo tempo.

9. Encerramento

9.1. Próximos Passos

- Recursos adicionais e materiais para estudo.
 - [The TLA+ Home Page](#)
 - [Learn TLA+](#)
 - Livro: [Specifying Systems](#)
 - [TLA+ Example Repository](#)