# Introdução

Da primeira parte do trabalho:

*“A telefonia foi o primeiro grande passo para a popularização das telecomunicações, trazendo ao público em geral a capacidade de se comunicar a distâncias gigantescas. Além dos avanços tecnológicos em transmissão, modulação e criptografia de sinais que a telefonia proporcionou, as linhas telefônicas são um exemplo bastante ubíquo de máquina de estados finitos. Um exemplo exaustivamente testado durante mais de uma centena de anos de uso.*

*A proposta do trabalho é modelar a máquina de estados de uma linha telefônica simples: para efeitos de modelagem, considerou-se uma linha fixa trivial, que somente aceita e origina ligações. Uma vez modelada a linha em si, o objetivo final do trabalho é modelar uma central telefônica capaz de gerenciar duas ou mais dessas linhas telefônicas e garantir que chamadas possam ser completadas entre essas N linhas assinantes de serviço.*

*Muito embora a modelagem de cada linha em si seja trivial, o problema torna-se incrementalmente mais atrativo com o aumento do número de linhas gerenciadas, já que a quantidade de estados da central cresce exponencialmente em relação a combinação dos estados das linhas gerenciadas.”*

Nesta segunda parte do trabalho a ideia é adicionar uma camada de complexidade desafiando a ferramenta de prova e a consistência do modelo.

# Metodologia

Na primeira parte do trabalho foi modelada uma central telefônica simples com duas linhas que podem executar ligações entre si. As transições de estados de cada linha são totalmente dependentes das transições da segunda, de forma que a complexidade adicional dessa comunicação tornava uma aparente simplória máquina de estados num caso complexo de prova.

Nesta segunda parte decidiu-se adotar o agravante de um meio compartilhado com acesso limitado. A abstração desejada é a de um tronco de linhas telefônicas com capacidade limitada, que somente permita uma conexão entre duas diferentes linhas ao mesmo tempo. As linhas conflitantes em cada um dos lados da conexão precisariam, portanto, competir pelo acesso a esse meio.

A fim de tornar esse problema interessante, o número de linhas telefônicas precisa ser dobrado. Na nova abstração, duas centrais distintas estão conectadas por um tronco, cada uma possuindo duas linhas telefônicas. Uma linha precisa acessar o meio comum – o tronco – a fim de conectar-se a uma linha remota, mas não localmente.

## Modelagem

Este é diagrama da máquina de estados da linha telefônica modelada no trabalho passado:



Do último trabalho:

*“Na figura estão modelados os estados da linha e nomeadas suas transições. Foram omitidos todos os loops para efeitos de legibilidade, porém é suficiente dizer que todos os estados, com exceção do estado de erro, podem se manter em loop enquanto as condições para a transição não se satisfizerem. Em especial, num mundo ideal, diversos destes loops possuem timeouts, que não foram implementados a priori.”*

## Implementação

Decidiu-se por modificar a implementação original, adotando as capacidades de reuso de código da linguagem SMV. No primeiro trabalho, por desconhecimento dessas capacidades, houve muita repetição de código, já que cada uma das linhas telefônicas precisou ter todos os seus estados explicitados. Como possuímos o dobro de linhas dessa vez, além do meio comum, a quantidade de estados é significativamente maior e explicitar todas as transições seria inviável.

A nova implementação utiliza o módulo principal “main” como controlador do ambiente e instancia quatro linhas telefônicas representadas pelo módulo “linha”. Dessa forma o código pôde ser simplificado a um mínimo razoável de transições de estados:

|  |
| --- |
| MODULE linha(tronco, ID)  VAR  estado: { WAIT, CALL, RING, TALK, DIAL, ERR };    ASSIGN  init(estado) := WAIT;  next(estado) := case  estado = WAIT & tronco = ID: { WAIT, CALL, RING, DIAL };  estado = WAIT & tronco != NONE & tronco != ID: WAIT ;  estado = CALL & tronco = ID: { WAIT, CALL };  estado = CALL & tronco != ID: ERR;  estado = RING & tronco = ID: { WAIT, RING, TALK };  estado = RING & tronco != ID: ERR;  estado = TALK & tronco = ID: { WAIT, TALK };  estado = TALK & tronco != ID: ERR;  estado = DIAL & tronco = ID: { WAIT, CALL };  estado = DIAL & tronco != ID: ERR;  estado = ERR: ERR;  TRUE: ERR;  esac; |

Na nova implementação, cada linha recebe como parâmetro uma variável que representa o tronco à qual está conectada, e um identificador único. A variável de tronco funciona como abstração da porta à qual a linha está conectada na central telefônica, enquanto o identificador, tratado aqui como ID, abstrai a ideia do número telefônico em si, que é efetivamente o identificador único de uma linha telefônica real.

De posse dessas duas variáveis é possível criar uma implementação agnóstica dos estados externos, melhorando a modelagem obtida. Como comparação, a implementação anterior assumia que cada linha telefônica conhecia diretamente o estado das demais, motivo pelo qual tornava-se inviável modelar uma central com mais que duas ou três linhas telefônicas.

O diagrama a seguir mostra como estão conectadas as linhas ao tronco compartilhado:

Como mencionado, as linhas à esquerda do tronco competem pelo acesso a esse meio entre si ao tentarem se conectar às linhas à direita, e vice versa.

O tronco é implementado como duas variáveis distintas que representam as duas pontas do meio compartilhado. A variável tronco\_A pode receber os valores A0 e A1 representando assim qual das linhas está usando esse tronco no momento. Já o tronco\_B pode receber, analogamente, os valores B0 e B1.

A fim de controlar o acesso das linhas a esse tronco, foi necessário implementar uma instrumentação de bloqueio simples, no caso representada por dois *mutexes* lock\_A e lock\_B. Para obter acesso ao tronco, uma linha precisa, primeiro, travar o lock daquele tronco, e somente em seguida será provido o acesso a ele.

Tal sistema de travamento é implementado através da limitação das transições de estado dos troncos baseando-se no estado prévio dos seus locks. Um tronco só pode receber uma linha caso seu lock esteja previamente travado por essa mesma linha, e uma vez que esse lock seja liberado, o tronco deve ser imediatamente liberado da mesma forma.

A implementação do sistema de transição dos troncos e suas locks segue abaixo:

|  |
| --- |
| MODULE main  VAR  tronco\_A: { NONE, A0, A1, ERR };  tronco\_B: { NONE, B0, B1, ERR };  lock\_A: { NONE, A0, A1, ERR };  lock\_B: { NONE, B0, B1, ERR };  linha\_A0: linha(tronco\_A, A0);  linha\_A1: linha(tronco\_A, A1);  linha\_B0: linha(tronco\_B, B0);  linha\_B1: linha(tronco\_B, B1);  ASSIGN  init(lock\_A) := NONE;  next(lock\_A) := case  lock\_A = NONE & tronco\_A = NONE: { A0, A1 };  lock\_A = A0 & tronco\_A = NONE: A0;  lock\_A = A1 & tronco\_A = NONE: A1;  lock\_A = A0 & tronco\_A = A0 : A0;  lock\_A = A1 & tronco\_A = A1 : A1;  TRUE: ERR;  esac;  init(lock\_B) := NONE;  next(lock\_B) := case  lock\_B = NONE & tronco\_B = NONE: { B0, B1 };  lock\_B = B0 & tronco\_B = NONE: B0;  lock\_B = B1 & tronco\_B = NONE: B1;  lock\_B = B0 & tronco\_B = B0 : B0;  lock\_B = B1 & tronco\_B = B1 : B1;  TRUE: ERR;  esac;  init(tronco\_A) := NONE;  next(tronco\_A) := case  lock\_A = NONE: NONE;  lock\_A = A0 & tronco\_A = NONE: A0;  lock\_A = A0 & tronco\_A = A0: { A0, NONE};  lock\_A = A1 & tronco\_A = NONE: A1;  lock\_A = A1 & tronco\_A = A1: { A1, NONE};  TRUE: ERR;  esac;  init(tronco\_B) := NONE;  next(tronco\_B) := case  lock\_B = NONE: NONE;  lock\_B = B0 & tronco\_B = NONE: B0;  lock\_B = B0 & tronco\_B = B0: { B0, NONE };  lock\_B = B1 & tronco\_B = NONE: B1;  lock\_B = B1 & tronco\_B = B1: { B1, NONE };  TRUE: ERR;  esac; |

Como pode-se ver, a quantidade de código nesta versão é substancialmente menor do que na versão anterior, ainda que o poder de expressão do modelo seja significativamente maior.

## Extração de Propriedades

As seguintes propriedades foram extraídas do modelo gerado:

|  |
| --- |
| --  -- Primeiro garantimos que os locks e troncos nunca entrarao em estado de erro  --  SPEC AG !(tronco\_A = ERR)  SPEC AG !(tronco\_B = ERR)  SPEC AG !(lock\_A = ERR)  SPEC AG !(lock\_B = ERR)  --  -- Linhas tambem nunca podem entrar em estado de erro  --  SPEC AG !(linha\_A0.estado = ERR)  SPEC AG !(linha\_A1.estado = ERR)  SPEC AG !(linha\_B0.estado = ERR)  SPEC AG !(linha\_B1.estado = ERR)  --  -- Se uma linha detém o tronco, deve também deter a lock  --  SPEC AG !(tronco\_A = A0 & lock\_A != A0)  SPEC AG !(tronco\_A = A1 & lock\_A != A1)  SPEC AG !(tronco\_A = B0 & lock\_A != B0)  SPEC AG !(tronco\_A = B1 & lock\_A != B1)  --  -- Toda linha deve ser capaz de se conectar ao tronco  --  SPEC EG ((linha\_A0.estado in { CALL, TALK, DIAL, RING}) & tronco\_A = A0 & lock\_A = A0)  SPEC EG ((linha\_A1.estado in { CALL, TALK, DIAL, RING}) & tronco\_A = A1 & lock\_A = A1)  SPEC EG ((linha\_B0.estado in { CALL, TALK, DIAL, RING}) & tronco\_B = B0 & lock\_B = B0)  SPEC EG ((linha\_B1.estado in { CALL, TALK, DIAL, RING}) & tronco\_B = B1 & lock\_B = B1)  --  -- Se uma linha está conectada, deve possuir o tronco e o lock  --  SPEC AG !((linha\_A0.estado in {CALL, TALK, DIAL, RING}) & (tronco\_A != A0 | lock\_A != A0))  SPEC AG !((linha\_A1.estado in {CALL, TALK, DIAL, RING}) & (tronco\_A != A1 | lock\_A != A1))  SPEC AG !((linha\_B0.estado in {CALL, TALK, DIAL, RING}) & (tronco\_B != B0 | lock\_B != B0))  SPEC AG !((linha\_B1.estado in {CALL, TALK, DIAL, RING}) & (tronco\_B != B1 | lock\_B != B1))  --  -- Todas as linhas devem ser capazes de comunicar-se entre si  --  SPEC EG (linha\_A0.estado = TALK & linha\_B0 = TALK)  SPEC EG (linha\_A0.estado = TALK & linha\_B1 = TALK)  SPEC EG (linha\_A1.estado = TALK & linha\_B0 = TALK)  SPEC EG (linha\_A1.estado = TALK & linha\_B1 = TALK) |

# Problemas Encontrados

Dessa vez, diferentemente do estágio inicial, não foi possível, em tempo hábil, completar-se um modelo conciso e funcional da proposta apresentada. Algumas das propriedades mais simples, como as que garantiam que as linhas jamais entrariam em estado de erro, falharam. Após diversas tentativas de se obter o resultado desejado, infelizmente, o prazo para a entrega da atividade foi alcançado e sua completude foi, portanto, prejudicada.

# Apêndice **II**: Código

|  |
| --- |
| --  -- Linha e agora uma maquina de estados que compreende o modelo  -- de ligacao telefonica  --  MODULE linha(tronco, ID)  VAR  estado: { WAIT, CALL, RING, TALK, DIAL, ERR };    ASSIGN  init(estado) := WAIT;  --  -- As mudancas de estado foram simplificadas, o objetivo agora  -- e descobrir se a linha fica presa em estado de erro  --  next(estado) := case  estado = WAIT & tronco = ID: { WAIT, CALL, RING, DIAL };  estado = WAIT & tronco != NONE & tronco != ID: WAIT ;  estado = CALL & tronco = ID: { WAIT, CALL };  estado = CALL & tronco != ID: ERR;  estado = RING & tronco = ID: { WAIT, RING, TALK };  estado = RING & tronco != ID: ERR;  estado = TALK & tronco = ID: { WAIT, TALK };  estado = TALK & tronco != ID: ERR;  estado = DIAL & tronco = ID: { WAIT, CALL };  estado = DIAL & tronco != ID: ERR;  estado = ERR: ERR;  TRUE: ERR;  esac;  --  -- Main agora eh um Tronco: um meio compartilhado por uma ou mais  -- centrais telefonicas  --  MODULE main  VAR  tronco\_A: { NONE, A0, A1, ERR };  tronco\_B: { NONE, B0, B1, ERR };  lock\_A: { NONE, A0, A1, ERR };  lock\_B: { NONE, B0, B1, ERR };  linha\_A0: linha(tronco\_A, A0);  linha\_A1: linha(tronco\_A, A1);  linha\_B0: linha(tronco\_B, B0);  linha\_B1: linha(tronco\_B, B1);  ASSIGN  init(lock\_A) := NONE;  next(lock\_A) := case  lock\_A = NONE & tronco\_A = NONE: { A0, A1 };  lock\_A = A0 & tronco\_A = NONE: A0;  lock\_A = A1 & tronco\_A = NONE: A1;  lock\_A = A0 & tronco\_A = A0 : A0;  lock\_A = A1 & tronco\_A = A1 : A1;  TRUE: ERR;  esac;  init(lock\_B) := NONE;  next(lock\_B) := case  lock\_B = NONE & tronco\_B = NONE: { B0, B1 };  lock\_B = B0 & tronco\_B = NONE: B0;  lock\_B = B1 & tronco\_B = NONE: B1;  lock\_B = B0 & tronco\_B = B0 : B0;  lock\_B = B1 & tronco\_B = B1 : B1;  TRUE: ERR;  esac;  init(tronco\_A) := NONE;  next(tronco\_A) := case  lock\_A = NONE: NONE;  lock\_A = A0 & tronco\_A = NONE: A0;  lock\_A = A0 & tronco\_A = A0: { A0, NONE};  lock\_A = A1 & tronco\_A = NONE: A1;  lock\_A = A1 & tronco\_A = A1: { A1, NONE};  TRUE: ERR;  esac;  init(tronco\_B) := NONE;  next(tronco\_B) := case  lock\_B = NONE: NONE;  lock\_B = B0 & tronco\_B = NONE: B0;  lock\_B = B0 & tronco\_B = B0: { B0, NONE };  lock\_B = B1 & tronco\_B = NONE: B1;  lock\_B = B1 & tronco\_B = B1: { B1, NONE };  TRUE: ERR;  esac;  --  -- Primeiro garantimos que os locks e troncos nunca entrarao em estado de erro  --  SPEC AG !(tronco\_A = ERR)  SPEC AG !(tronco\_B = ERR)  SPEC AG !(lock\_A = ERR)  SPEC AG !(lock\_B = ERR)  --  -- Linhas tambem nunca podem entrar em estado de erro  --  SPEC AG !(linha\_A0.estado = ERR)  SPEC AG !(linha\_A1.estado = ERR)  SPEC AG !(linha\_B0.estado = ERR)  SPEC AG !(linha\_B1.estado = ERR)  --  -- Se uma linha detém o tronco, deve também deter a lock  --  SPEC AG !(tronco\_A = A0 & lock\_A != A0)  SPEC AG !(tronco\_A = A1 & lock\_A != A1)  SPEC AG !(tronco\_A = B0 & lock\_A != B0)  SPEC AG !(tronco\_A = B1 & lock\_A != B1)  --  -- Toda linha deve ser capaz de se conectar ao tronco  --  SPEC EG ((linha\_A0.estado in { CALL, TALK, DIAL, RING}) & tronco\_A = A0 & lock\_A = A0)  SPEC EG ((linha\_A1.estado in { CALL, TALK, DIAL, RING}) & tronco\_A = A1 & lock\_A = A1)  SPEC EG ((linha\_B0.estado in { CALL, TALK, DIAL, RING}) & tronco\_B = B0 & lock\_B = B0)  SPEC EG ((linha\_B1.estado in { CALL, TALK, DIAL, RING}) & tronco\_B = B1 & lock\_B = B1)  --  -- Se uma linha está conectada, deve possuir o tronco e o lock  --  SPEC AG !((linha\_A0.estado in {CALL, TALK, DIAL, RING}) & (tronco\_A != A0 | lock\_A != A0))  SPEC AG !((linha\_A1.estado in {CALL, TALK, DIAL, RING}) & (tronco\_A != A1 | lock\_A != A1))  SPEC AG !((linha\_B0.estado in {CALL, TALK, DIAL, RING}) & (tronco\_B != B0 | lock\_B != B0))  SPEC AG !((linha\_B1.estado in {CALL, TALK, DIAL, RING}) & (tronco\_B != B1 | lock\_B != B1))  --  -- Todas as linhas devem ser capazes de comunicar-se entre si  --  SPEC EG (linha\_A0.estado = TALK & linha\_B0 = TALK)  SPEC EG (linha\_A0.estado = TALK & linha\_B1 = TALK)  SPEC EG (linha\_A1.estado = TALK & linha\_B0 = TALK)  SPEC EG (linha\_A1.estado = TALK & linha\_B1 = TALK) |