INSTITUTO FEDERAL
Santa Catarina

# Projeto 03

Verificação do protocolo da etapa 1

**Curso:** Engenharia de Telecomunicações
**Disciplina:** PTC29008 – Projeto de Protocolos
**Professor:** Marcelo Maia Sobral

**Aluno**
Renan Rodolfo da Silva

09 de julho de 2019

# 1 Introdução

Este projeto tem por objetivo, verificar o comportamento e as propriedades de duas partes do protocolo desenvolvido na etapa1, sendo essas: o Enquadramento e o Gerenciamento de Sessão. Para tal finalidade, foi desenvolvido um modelo de verificação utilizando a linguagem PROMELA, no qual teve sua execução e verificação realizada pelo SPIN, uma ferramenta que possibilita executar esses modelos de forma a simulá-los ou verificá-los de forma exaustiva.

# 2 Verificação do protocolo da etapa 1

O objetivo desse projeto era fazer a verificação das seguintes propriedades:

1. Sessão: término de sessão eventualmente será concluído para ambas as partes
2. Sessão: duas entidades são capazes de estabelecerem uma sessão, independente de qual delas iniciar a sessão
3. Enquadramento: Perdas de sincronismo no enquadramento são recuperadas em algum momento futuro

## 2.1 Término de sessão

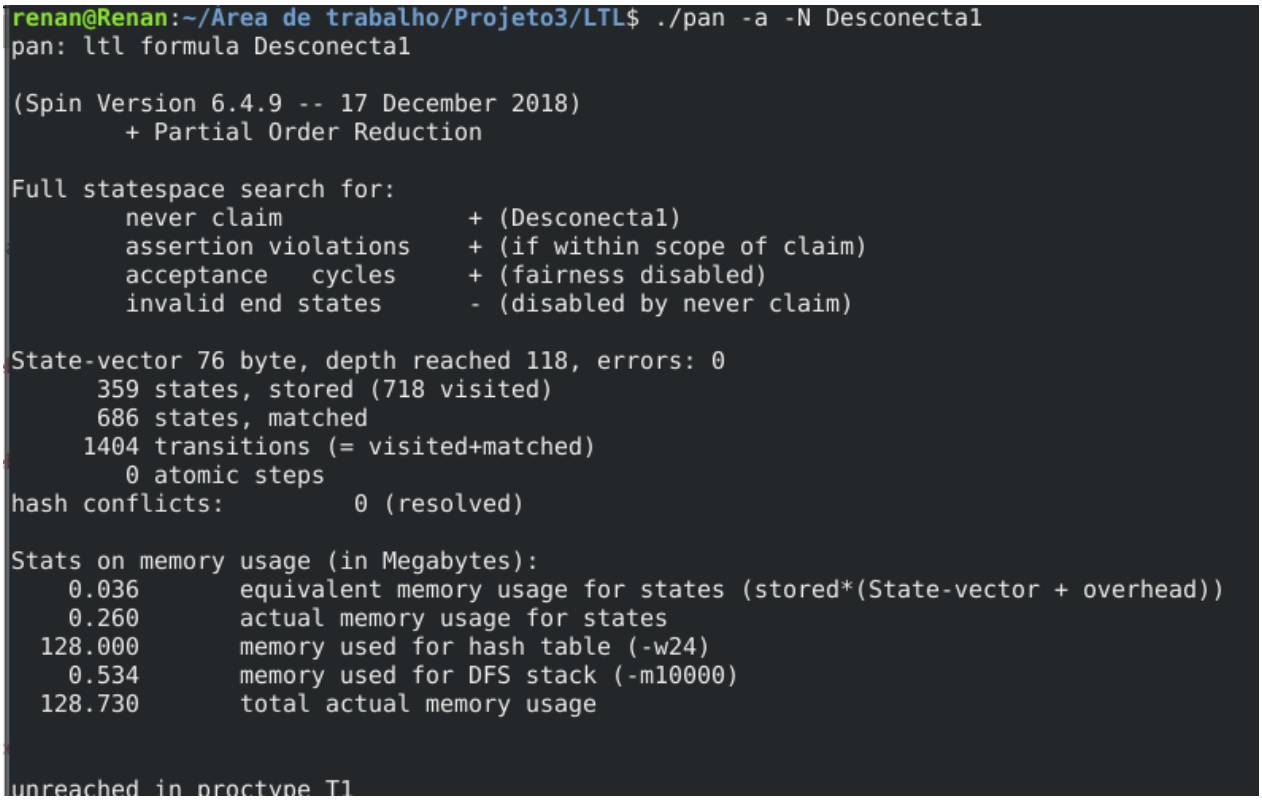


Figura 1: Ltl do término de sessão da fórmula Desconecta1

Para fazer a verificação do item 1 citado acima, foi utilizado duas fórmulas LTL. A primeira fórmula verificada para a desconexão foi:

Fórmula 1: **ltl Desconecta1 { <> (aux == 1) && <>(aux1==1)}** .

Obteve o resultado que consta na Figura 1, onde a variável aux e aux1 são incrementas quando as entidades estiverem no estado Con e receberem um dc, 20(erro ARQ), ou quando estiverem nos estados Half1/Half2 prestes a irem para o estado Disc. Assume assim, que eventualmente aux e aux1 vão ser igual a 1. Representando assim, que em algum momento futuro as entidades vão ser desconectadas.

A segunda fórmula verificada e que tem como resultado dado pelo verificador na Figura 2:

Fórmula 2: **Desconecta2{ (((<>(T1@Con U T1@Half2)) && (<>(T2@Con U T2@Half1)))||((<>(T1@Con U T1@Half1)) && (<>(T2@Con U T2@Half2))))}**.

Como o ciclo só ocorre uma vez(um start), usa-se uma formula ltl em que eventualmente se a entidade1 estiver no estado Con e a entidade2 tomar a iniciativa de desconexão, a entidade1 irá para Half2 e a entidade2 passará de Con para Half1, que nesse caso, Half1 e Half2 sempre vão gerar desconexão. Vice versa para quando a entidade1 toma a iniciativa.

```
renan@Renan:~/Área de trabalho/Projeto3/LTL$ ./pan -a -N Desconecta2
pan: ltl formula Desconecta2

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (Desconecta2)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 76 byte, depth reached 110, errors: 0
      617 states, stored (1234 visited)
     1150 states, matched
     2384 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.061       equivalent memory usage for states (stored*(State-vector + overhead))
    0.260       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

Figura 2: Ltl do término de sessão da fórmula Desconecta2

## 2.2 Estabelecimento de sessão

Para fazer a verificação do item 2 citado no tópico 2, também foram utilizadas duas fórmulas LTL, sendo a primeira fórmula utilizada para a verificação:

Fórmula 3: **ltl Estabelece1 { <>(Conectou == 2)}**

Onde o resultado do verificador pode ser observado na Figura 3

```
renan@Renan:~/Área de trabalho/Projeto3/LTL$ ./pan -a -N Estabelece1
pan: ltl formula Estabelece1

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (Estabelece1)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 76 byte, depth reached 68, errors: 0
      107 states, stored (214 visited)
      196 states, matched
      410 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.011       equivalent memory usage for states (stored*(State-vector + overhead))
    0.260       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

Figura 3: LTL estabelecimento de Sessão da fórmula Estabelece1

Nessa fórmula, a variável Conectou é incrementa assim que alguma entidade chegue no estado Con, se as duas entidades chegarem no Con, entende-se que eventualmente foi estabelecida uma conexão, onde o lado que inicia é independente, pois ambos os processos estão tomando a iniciativa.

A segunda fórmula teve a mesma lógica utilizada no Desconecta2, em que diz que eventualmente se a entidade1 estiver no estado Hand1 e a entidade2 tomar a iniciativa de conexão, a entidade1 irá para Con e a entidade2 passará de Hand2 para Con, que nesse caso, Hand1 e Hand2 sempre vão gerar Conexão,já que só vão receber start e cc/cr/ca. Vice versa para quando a entidade1 toma a iniciativa. Fórula utililizada é:

Fórmula 4: **ltl Estabelece2 { (((<>(T1@Hand1 U T1@Con)) && (<>(T2@Hand2 U T2@Con))) || ((<>(T1@Hand2 U T1@Con)) && (<>(T2@Hand1 U T2@Con))))}**

```
renan@Renan:~/Área de trabalho/Projeto3/LTL$ ./pan -a -N Estabelece2
pan: ltl formula Estabelece2

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (Estabelece2)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 76 byte, depth reached 70, errors: 0
      177 states, stored (354 visited)
      326 states, matched
      680 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.018       equivalent memory usage for states (stored*(State-vector + overhead))
    0.260       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

Figura 4: LTL estabelecimento de Sessão da fórmula Estabelece2

## 2.3 Conclusão sobre a parte de Sessão

Uma falha que foi possível observar no projeto da etapa 1, é que as entidades após serem desconectas permanecem sempre desconectadas, pois só ocorre um start no inicio do processo. Isso pode ser verificado na Figura 5, onde após ser desconectado, o processo é encerrado. Na imagem só está demostrando o processo da entidade 1, mas na entidade 2 ocorre da mesma forma.

Pode se afirmar que término de sessão eventualmente será concluído para ambas as partes e que as entidades são capazes de estabelecerem uma sessão, independente de qual delas iniciar a sessão e que estão livres de Deadlock

```
unreached in proctype T1
        Session.pml:25, state 10, "Ent!cc"
        Session.pml:26, state 11, "printf('Disc - Recebeu cr e Enviou cc \n')"
        Session.pml:37, state 20, "Ent!ca"
        Session.pml:38, state 21, "printf('Hand1 - Recebeu Start e Enviou ca \n')"
        Session.pml:45, state 28, "printf('Hand1 - Recebu ca \n')"
        Session.pml:54, state 36, "Ent!cr"
        Session.pml:55, state 37, "cont = (cont+1)"
        Session.pml:57, state 39, "cont = 0"
        Session.pml:53, state 41, "((cont<=2))"
        Session.pml:53, state 41, "else"
        Session.pml:64, state 46, "printf('Hand2 1 \n')"
        Session.pml:67, state 48, "printf('Hand2 - Recebeu ca \n')"
        Session.pml:70, state 51, "Ent!dr"
        Session.pml:71, state 52, "printf('Hand2 - Recebeu dr e Enviou dr \n')"
        Session.pml:88, state 66, "Ent!dr"
        Session.pml:89, state 67, "printf('Con - Recebeu dr e Enviou dr  \n')"
        Session.pml:92, state 70, "printf('Con - Recebeu dc  \n')"
        Session.pml:93, state 71, "aux = (aux+1)"
        Session.pml:103, state 81, "Ent!dr"
        Session.pml:104, state 82, "printf('Con - Recebeu close e Enviou dr  \n')"
        Session.pml:105, state 83, "aux = (aux+1)"
        Session.pml:130, state 106, "Ent!dr"
        Session.pml:131, state 107, "printf('Half1 Timeout - Enviando Dr  \n')"
        Session.pml:136, state 111, "printf('Half2 1 \n')"
        Session.pml:139, state 113, "Ent!dr"
        Session.pml:140, state 114, "printf('Half2 - Recebeu dr e Enviar  \n')"
        Session.pml:143, state 117, "printf('Half2 - Recebeu dc  \n')"
        Session.pml:144, state 118, "aux = (aux+1)"
        Session.pml:147, state 121, "printf('Half2 Timeout \n')"
        Session.pml:148, state 122, "aux = (aux+1)"
        Session.pml:138, state 124, "Ent2?dr"
        Session.pml:138, state 124, "Ent2?dc"
        Session.pml:138, state 124, "(timeout)"
        Session.pml:160, state 132, "printf('Timeout Check\n')"
        Session.pml:161, state 133, "aux = (aux+1)"
        Session.pml:165, state 138, "-end-"
        (33 of 138 states)
```

Figura 5: Partes não alcançadas da LTL de desconexão

## 2.4 Perdas de sincronismo do Enquadramento

Por fim, para verificar a propriedade em que perdas de sincronismo no enquadramento são recuperadas em algum momento futuro, foi utilizada a seguinte fórmula ltl:

**ltl Enquadramento  []<>(Quadro == true)**

Onde a variável Quadro recebe o valor true somente quando o estado estiver na recepcão e receber uma flag delimitadora de quadro. Ou seja, a fórmula diz que indefinidamente frequente o processo recebe um quadro completo. Sendo assim, pode se concluir que ele se recupera caso tenha alguma perda de sincronismo. O resultado obtido pelo verificador pode ser verificado na Figura 6, no caso está livre de deadlocks.

```
renan@Renan:~/Área de trabalho/Projeto3/LTL$ ./pan -a -N Enquadramento
warning: only one claim defined, -N ignored

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (Enquadramento)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 48 byte, depth reached 269, errors: 0
      479 states, stored (683 visited)
      503 states, matched
     1186 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.035       equivalent memory usage for states (stored*(State-vector + overhead))
    0.283       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

Figura 6: LTL do sincronimo do Enquadramento

### 2.5 Conclusão sobre sincronismo do Enquadramento

Com a verificação obtida pelo Spin, pode se afirmar que o Enquadramento desenvolvido no projeto 1 pode se recuperar de eventuais perda de sincronismo.

## 3 Comparação entre código original do Projeto1 e modelo Promela

O modelo Promela em que foi verificado a perda de sincronismo foi baseado em uma máquina de estado da parte de Enquadramento do protocolo de comunicação de dados. O método em questão foi definido como **def handle_fsm(self, octeto)**, onde fica recebendo bytes pela serial e verifica se recebe um quadro completo ou não, em que no modelo Promela voi simulado utilizando dois estados em que o primeiro sempre envia um quadro completo, sendo a transmissão feita byte a byte, e o segundo envia um quadro faltando a flag delimitadora de fim de quadro. Um modelo resumido na linguagem original(Python), onde pode ser feita a comparação do código com o modelo em promela que se encontra em anexo ao projeto 3, pode ser verificado na Figura 7

```
def handle_fsm(self, octeto):
    while True:
                    if (self.state == self.ocioso):
                        if (self.bit == b'~' and self.n == 0):
                            self.buffer = []
                            self.state = self.recp
                            self.enable_timeout()
                            return
                        else:
                            self.state = self.ocioso
                    elif (self.state == self.recp):
                        if (self.bit == b'~' and self.n > 0):
                            self.quadro = True
                            self.state = self.ocioso
                        elif (self.bit == b'~' and self.n == 0):
                            self.state = self.recp
                        elif (self.bit == b'}'):
                            self.state = self.Escape
                        elif (self.bit != b"~" and self.bit != b"}"):
                                self.state = self.ocioso
                        else:
                            self.state = self.ocioso
                    elif (self.state == self.Escape):
                        if (self.bit == b'~'):
                            self.buffer = []
                            print("Zerou o buffer", self.buffer)
                            self.state = self.ocioso
                            self.fim = 1
                            self.disable_timeout()
                        elif (self.bit == b'}'):
                            self.bit = self.fd.read(1)
                            aux_Dec = ord(self.bit)
                            aux_Stuff = aux_Dec ^ 32
                            self.buffer.append(aux_Stuff)
                            self.n = self.n + 1
                            self.state = self.recp
    return
```

Figura 7: Enquadramento feito em python, versão resumida

Em relação as verificações relacionadas a Sessão, teve como base a parte de Gerenciamento de Sessão do protocolo de comunicação de dados, em que o método transcrito para o modelo Promela foi o def Recebe_arq(self,quadro), no qual quando um usuário inicia a aplicação, o protocolo envia uma mensagem de start para essa máquina que por sua vez tenta estabelecer uma sessão. Como a máquina em questão possui um código extenso, também foi realizada a supressão de informações para deixar o código mais resumido. A Figura 8 e Figura 9 podem ser usadas para fazer a comparação com o modelo Promela anexo ao Projeto 3.

```
def Recebe_arq(self,quadro):
        if(self.state==self.Disc):
            if(quadro[0]==self.start):
                self.state = self.Hand1
                self.quadro_controle.append(self.cr)
            elif(quadro[0]==self.cr):
                self.state = self.Hand2
                self.quadro_controle.append(self.cc)

        elif(self.state==self.Hand1):
            if(quadro[0]==self.start):
                self.state = self.Con
                self.quadro_controle.append(self.ca)
            elif(quadro[0]==self.cc):
                self.state = self.Con
                self.quadro_controle.append(self.ca)
            elif(quadro[0]==self.ca):
                self.state = self.Con
                return
            elif(quadro[0]==self.cr):
                self.state = self.Hand1
                self.quadro_controle.append(self.cc)
```

Figura 8: Gerenciamento de Sessão feito em python, versão resumida

```
        elif(self.state==self.Hand2):
            if(quadro[0]==self.ca):
                self.state = self.Con
            elif(quadro[0]==self.dr):
                self.state = self.Half2
                self.quadro_controle.append(self.dr)
            elif(quadro[0]==self.dr):
                self.state=self.Half2
                self.quadro_controle.append(self.dr)

        elif(self.state==self.Con):
            self.enable_timeout()
            if(quadro[0]==self.dr):
                self.state = self.Half2
                self.quadro_controle.append(self.dr)
            elif(quadro[0]==self.dc):
                self.state = self.Disc
            elif(quadro[0]==self.Kr):
                self.state = self.Con
                self.quadro_controle.append(self.kc)
            elif(quadro[0]==self.ca):
                self.On_Off_TOP.Ativa_Camada_TOP()
                self.On_OFF_Tun.Ativa_Camada_Tun()
            elif(quadro[0]==20):
                self.state = self.Half1
                self.quadro_controle.append(self.dr)
```

Figura 9: Gerenciamento de Sessão feito em python, versão resumida

## 4 Conclusão

Com a utilização da linguagem Promela e o seu verificador Spin, pode-se modelar um sistema e verificar se seu funcionamento corresponde ao que de fato deveria fazer, pois ele testa todas as possibilidades possíveis exaustivamente. Dessa forma, é uma boa prática a se usar antes de desenvolver algum protocolo, pois podemos verificar se seu comportamento vai ser o correto.