Software Confiável

Especificação e Verificação

Alvaro Heiji Miyazawa

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

28 de maio de 2007

Programa

Programa

Programa

Programa

Confiabilidade

Definição

Confiabilidade *é a probabilidade de um componente, ou sistema, funcionar corretamente durante um período de tempo sob um conjunto dado de condições operacionais [Storey, 1996].*

Definição

Um sistema ou componente ***funciona corretamente*** *quando opera como definido na sua especificação [Storey, 1996].*

Confiabilidade

Definição

Confiabilidade *é a probabilidade de um componente, ou sistema, funcionar corretamente durante um período de tempo sob um conjunto dado de condições operacionais [Storey, 1996].*

Definição

Um sistema ou componente ***funciona corretamente*** *quando opera como definido na sua especificação [Storey, 1996].*

Confiabilidade

Definição

Confiabilidade *é a probabilidade de um componente, ou sistema, funcionar corretamente durante um período de tempo sob um conjunto dado de condições operacionais [Storey, 1996].*

Definição

Um sistema ou componente ***funciona corretamente*** *quando opera como definido na sua especificação [Storey, 1996].*

Confiabilidade

Definição

Confiabilidade *é a probabilidade de um componente, ou sistema, funcionar corretamente durante um período de tempo sob um conjunto dado de condições operacionais [Storey, 1996].*

Definição

Um sistema ou componente ***funciona corretamente*** *quando opera como definido na sua especificação [Storey, 1996].*

Métodos Formais

Definição

Métodos Formais *referem-se a técnicas e ferramentas matematicamente rigorosas para a especificação, projeto e verificação de sistemas de software e hardware [NASA, 2001].*

Métodos Formais

Definição

Métodos Formais *referem-se a técnicas e ferramentas matematicamente rigorosas para a especificação, projeto e verificação de sistemas de software e hardware [NASA, 2001].*

Métodos Formais

Definição

Métodos Formais *referem-se a técnicas e ferramentas matematicamente rigorosas para a especificação, projeto e verificação de sistemas de software e hardware [NASA, 2001].*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*

*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*

*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*
*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*
*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*
*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Definição

*Uma **especificação** de um sistema é a descrição desse sistema em alguma linguagem.*
*Uma **especificação formal** é uma descrição em uma linguagem formal.*

Definição

*A **verificação** é o ato de demonstrar a correção ou incorreção de uma especificação em relação a outra.*

Métodos Formais

Objetivo

Melhorar a confiabilidade de um sistema.

Como

Verificando especificações do mesmo sistema em diferentes graus de refinamento.

Porém, métodos formais não garantem a confiabilidade, e devem ser usados em conjunto com outras técnicas.

Métodos Formais

Objetivo

Melhorar a confiabilidade de um sistema.

Como

Verificando especificações do mesmo sistema em diferentes graus de refinamento.

Porém, métodos formais não garantem a confiabilidade, e devem ser usados em conjunto com outras técnicas.

Métodos Formais

Objetivo

Melhorar a confiabilidade de um sistema.

Como

Verificando especificações do mesmo sistema em diferentes graus de refinamento.

Porém, métodos formais não garantem a confiabilidade, e devem ser usados em conjunto com outras técnicas.

Métodos Formais

Objetivo

Melhorar a confiabilidade de um sistema.

Como

Verificando especificações do mesmo sistema em diferentes graus de refinamento.

Porém, métodos formais não garantem a confiabilidade, e devem ser usados em conjunto com outras técnicas.

Outline

Notação Z

- Notação formal
- Teoria de Conjuntos
- Lógica Matemática
- Schemas

Notação Z

- Notação formal
- Teoria de Conjuntos
- Lógica Matemática
- Schemas

Notação Z

- Notação formal
- Teoria de Conjuntos
- Lógica Matemática
- Schemas

Notação Z

- Notação formal
- Teoria de Conjuntos
- Lógica Matemática
- Schemas

Schema

Estrutura que descreve variáveis sob restrições.

Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Schema

Estrutura que descreve variáveis sob restrições.
Operações sobre Schemas:

- Conjunção
- Disjunção
- Negação
- Quantificação
- Composição
- *Priming*

Exemplo

$[OBJETO]$

Exemplo

$[OBJETO]$

Pilha

$objetos : \mathrm{seq}\ OBJETO$

Exemplo

$[OBJETO]$

Pilha

$objetos : \mathrm{seq}\ OBJETO$

$\Delta Pilha$

$Pilha$

$Pilha'$

Exemplo

Insere

$\Delta Pilha$

$?o : OBJETO$

$objetos' = \langle ?o \rangle \frown objetos$

Exemplo

Insere

$\Delta Pilha$

$?o : OBJETO$

$objetos' = \langle ?o \rangle \frown objetos$

Remove

$\Delta Pilha$

$objetos \neq \emptyset$

$objetos' = tail\ objetos$

Exemplo

Topo

$$Pilha$$
$$o! : OBJETO$$

$$objetos \neq \emptyset$$
$$o! = head\ objetos$$

Outline

Triplas de Hoare

$$(\!| \phi |\!) P (\!| \psi |\!)$$

Se P é executado em um estado que satisfaz ϕ, então o estado seguinte satisfaz ψ.

Regras de Prova Parcial

$$\frac{(\!| \phi |\!)\, C_1 \,(\!| \eta |\!) \qquad (\!| \eta |\!)\, C_2 \,(\!| \psi |\!)}{(\!| \phi |\!)\, C_1 ;\ C_2 \,(\!| \psi |\!)}\ \textsc{Composição}$$

Regras de Prova Parcial

$$\frac{(\!| \phi |\!)\, C_1 \,(\!| \eta |\!) \qquad (\!| \eta |\!)\, C_2 \,(\!| \psi |\!)}{(\!| \phi |\!)\, C_1 ;\ C_2 \,(\!| \psi |\!)}\ \textsc{Composição}$$

$$\frac{}{(\!| \psi[E/x] |\!)\, x = E \,(\!| \psi |\!)}\ \textsc{Atribuição}$$

Regras de Prova Parcial

$$\frac{(\!| \phi \wedge B |\!)\, C_1 \,(\!| psi |\!) \qquad (\!| \phi \wedge \neg B |\!)\, C_2 \,(\!| \psi |\!)}{(\!| \phi |\!)\, if\ B\, \{C_1\}\ else\ \{C_2\} \,(\!| \psi |\!)}\ \text{IF}$$

Regras de Prova Parcial

$$\frac{(\!| \phi \wedge B |\!)\, C_1 \,(\!| \mathit{psi} |\!) \qquad (\!| \phi \wedge \neg B |\!)\, C_2 \,(\!| \psi |\!)}{(\!| \phi |\!)\, \mathit{if}\ B\, \{C_1\}\ \mathit{else}\ \{C_2\}\, (\!| \psi |\!)}\ \textsc{If}$$

$$\frac{(\!| \psi \wedge B |\!)\, C \,(\!| \psi |\!)}{(\!| \psi |\!)\, \mathit{while}\ B\, \{C\}\, (\!| \psi \wedge \neg B |\!)}\ \textsc{While-Parcial}$$

Regras de Prova Parcial

$$\frac{\vdash \phi' \Rightarrow \phi \qquad (\!| \phi |\!) C (\!| \psi |\!) \qquad \vdash \psi' \Rightarrow \psi}{(\!| \phi' |\!) C (\!| \psi' |\!)} \text{ IMPLICAÇÃO}$$

Exemplo

$(\!| \top |\!)$

```
y = 1
z = 0
while (z != x) {
 z = z + 1
 y = y * z
}
```

$(\!| y = x! |\!)$

Exemplo

$(\!| \eta \wedge B |\!)$

z = z + 1

z = z + 1

y = y * z

y = y * z

$(\!| \eta |\!)$

Exemplo

$$(\!| \; y = z! \wedge z \neq x \; |\!)$$

z = z + 1

z = z + 1
y = y * z

y = y * z
$(\!| \; y = z! \; |\!)$

Exemplo

$$(\!| \; y = z! \wedge z \neq x \; |\!)$$

```
z = z + 1
y = y * z
```

$$(\!| \; y \times z = z! \; |\!)$$

```
z = z + 1
y = y * z
```

$$(\!| \; y = z! \; |\!)$$

Exemplo

```
z = z + 1
y = y * z
```

$$(\!| \; y = z! \wedge z \neq x \; |\!)$$
$$(\!| \; y \times (z+1) = (z+1)! \; |\!)$$
```
z = z + 1
```
$$(\!| \; y \times z = z! \; |\!)$$
```
y = y * z
```
$$(\!| \; y = z! \; |\!)$$

Exemplo

```
z = z + 1
y = y * z
```

$$(\!| \; y = z! \wedge z \neq x \; |\!)$$
$$(\!| \; y \times (z+1) = (z+1)! \; |\!)$$
```
z = z + 1
```
$$(\!| \; y \times z = z! \; |\!)$$
```
y = y * z
```
$$(\!| \; y = z! \; |\!)$$

$$y = z! \wedge z \neq x \Rightarrow$$
$$y \times (z+1) = (z+1)!$$

Exemplo

```
z = z + 1
y = y * z
```

$(\!| y = z! \wedge z \neq x |\!)$

$(\!| y \times (z+1) = (z+1)! |\!)$

```
z = z + 1
```

$(\!| y \times z = z! |\!)$

```
y = y * z
```

$(\!| y = z! |\!)$

$y = z! \wedge z \neq x \Rightarrow$
$y \times (z+1) = (z+1)!$ ✓

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| \eta |\!)$

while (z != x) { ... }

$(\!| \eta \wedge \neg B |\!)$

$(\!| y = x! |\!)$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

y = 1

$(\!| y = 0! |\!)$

z = 0

$(\!| y = z! |\!)$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

$(\!| 1 = 1 |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

Exemplo

$(| \top |)$

y = 1

z = 0

$(| y = z! |)$

while (z != x) { ... }

$(| y = z! \wedge \neg (z \neq x) |)$

$(| y = x! |)$

$(| \top |)$

$(| 1 = 1 |)$

y = 1

$(| y = 1 |)$

z = 0

$(| y = z! |)$

$\top \Rightarrow 1 = 1$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

$(\!| 1 = 1 |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

$\top \Rightarrow 1 = 1 \checkmark$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

$(\!| 1 = 1 |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

$\top \Rightarrow 1 = 1 \checkmark$

$y = z! \wedge \neg (z \neq x) \Rightarrow y = x!$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

$(\!| 1 = 1 |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

$\top \Rightarrow 1 = 1 \checkmark$

$y = z! \wedge z = x \Rightarrow y = x!$

Exemplo

$(\!| \top |\!)$

y = 1

z = 0

$(\!| y = z! |\!)$

while (z != x) { ... }

$(\!| y = z! \wedge \neg (z \neq x) |\!)$

$(\!| y = x! |\!)$

$(\!| \top |\!)$

$(\!| 1 = 1 |\!)$

y = 1

$(\!| y = 1 |\!)$

z = 0

$(\!| y = z! |\!)$

$\top \Rightarrow 1 = 1 \checkmark$

$y = z! \wedge z = x \Rightarrow y = x! \checkmark$

Outline

CCS

Calculus of Communicating Systems

Sistema Complexos:

- Agentes (componentes)
- Ações

CCS

Calculus of Communicating Systems
Sistema Complexos:

- Agentes (componentes)
- Ações

CCS

Calculus of Communicating Systems
Sistema Complexos:

- Agentes (componentes)
  - Ações
    - Interação
    - Independente

CCS

Calculus of Communicating Systems
Sistema Complexos:

- Agentes (componentes)
  - Ações
    - Interação
    - Independente

CCS

Calculus of Communicating Systems
Sistema Complexos:

- Agentes (componentes)
  - Ações
    - Interação
    - Independente

CCS

Calculus of Communicating Systems
Sistema Complexos:

- Agentes (componentes)
  - Ações
    - Interação
    - Independente

CCS

- Prefixação - $\alpha.P$ (α é uma ação)
- Soma - $\sum_{i \in I} E_i$
- Composição - $E_1 \mid E_2$
- Restriçao - $E \backslash L$ (L é um conjunto de nomes)
- *R*elabelling - E[f] (f é uma função de mudança de nome)

CCS

- Prefixação - $\alpha.P$ (α é uma ação)
- Soma - $\sum_{i \in I} E_i$
- Composição - $E_1 \mid E_2$
- Restriçao - $E \backslash L$ (L é um conjunto de nomes)
- *R*elabelling - E[f] (f é uma função de mudança de nome)

CCS

- Prefixação - $\alpha.P$ (α é uma ação)
- Soma - $\sum_{i \in I} E_i$
- Composição - $E_1 \mid E_2$
- Restriçao - $E \backslash L$ (L é um conjunto de nomes)
- *R*elabelling - E[f] (f é uma função de mudança de nome)

CCS

- Prefixação - $\alpha.P$ (α é uma ação)
- Soma - $\sum_{i \in I} E_i$
- Composição - $E_1 \mid E_2$
- Restriçao - $E \backslash L$ (L é um conjunto de nomes)
- *R*elabelling - E[f] (f é uma função de mudança de nome)

CCS

- Prefixação - $\alpha.P$ (α é uma ação)
- Soma - $\sum_{i \in I} E_i$
- Composição - $E_1 \mid E_2$
- Restriçao - $E \backslash L$ (L é um conjunto de nomes)
- *R*elabelling - E[f] (f é uma função de mudança de nome)

CCS

Semântica

A semântica é dada termos de transições. Por exemplo:

$$\text{AÇÃO}\ \frac{}{\alpha.E \xrightarrow{\alpha} E}$$

CCS

Semântica

A semântica é dada termos de transições. Por exemplo:

$$\text{Ação}\ \frac{}{\alpha.E \xrightarrow{\alpha} E}$$

$$\text{Soma}_j\ \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j}\ (j \in I)$$

CCS

Semântica

A semântica é dada termos de transições. Por exemplo:

$$\text{Ação}\ \frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \textit{Com}_1\ \frac{E \xrightarrow{\alpha} E'}{E \mid F \xrightarrow{\alpha} E' \mid F}$$

$$\textit{Soma}_j\ \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j}\ (j \in I)$$

CCS

Semântica

A semântica é dada termos de transições. Por exemplo:

$$\text{AÇÃO}\ \frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \textit{Com}_1\ \frac{E \xrightarrow{\alpha} E'}{E \mid F \xrightarrow{\alpha} E' \mid F}$$

$$\textit{Soma}_j\ \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j}\ (j \in I) \qquad \textit{Com}_{geral}\ \frac{E \xrightarrow{l} E' \qquad F \xrightarrow{\bar{l}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$$

Exemplo: Problema do canal não confiável

Problema

Dado um canal de comunicação não confiável (pode perder ou duplicar informações, porém não corrompê-las), definir um protocolo que torne esse canal confiável.

Exemplo: Problema do canal não confiável

Problema

Dado um canal de comunicação não confiável (pode perder ou duplicar informações, porém não corrompê-las), definir um protocolo que torne esse canal confiável.

Solução

Protocolo AB (Alternating-Bit)

Protocolo AB

O protocolo apresenta quatro entidade:

- Remetente
- Destinatário
- Canal da Mensagem
- Canal de Confirmação

Protocolo AB

O protocolo apresenta quatro entidade:

- Remetente
- Destinatário
- Canal da Mensagem
- Canal de Confirmação

Protocolo AB

O protocolo apresenta quatro entidade:

- Remetente
- Destinatário
- Canal da Mensagem
- Canal de Confirmação

Protocolo AB

O protocolo apresenta quatro entidade:

- Remetente
- Destinatário
- Canal da Mensagem
- Canal de Confirmação

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgora, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgota, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgota, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgota, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgora, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgora, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgora, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB

O protocolo funciona da seguinte maneira:

- O Remetente, após aceitar uma mensagem, envia a mensagem com um bit b, aciona um temporizador e se:
  - o tempo se esgota, ele reenvia a mensagem com o bit b
  - recebe uma confirmação b através do canal C, ele está pronto para aceitar uma nova mensagem com bit b-1
  - recebe uma confirmação b-1 através do canal C, ele a ignora
- O Destinatário, após entregar a mensagem uma mensagem com bit b, ele confirma com o bit b através de C, aciona um temporizador e se:
  - o tempo se esgora, ele reconfirma com o bit b através de C
  - recebe uma nova mensagem com bit b-1, ele está pronto para entregar uma nova mensagem
  - recebe uma mensagem com bit b, ele a ignora

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

$$Accept(b) \stackrel{def}{=} accept.Send(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

$$Accept(b) \stackrel{def}{=} accept.Send(b)$$

$$Reply(b) \stackrel{def}{=} \overline{reply_b}.\overline{time}.Replying(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

$$Accept(b) \stackrel{def}{=} accept.Send(b)$$

$$Reply(b) \stackrel{def}{=} \overline{reply_b}.\overline{time}.Replying(b)$$

$$Replying(b) \stackrel{def}{=} timeout.Reply(b) + trans_{b-1}.timeout.Deliver(b) + trans_b.Replying(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

$$Accept(b) \stackrel{def}{=} accept.Send(b)$$

$$Reply(b) \stackrel{def}{=} \overline{reply_b}.\overline{time}.Replying(b)$$

$$Replying(b) \stackrel{def}{=} timeout.Reply(b) + trans_{b-1}.timeout.Deliver(b) + trans_b.Replying(b)$$

$$Deliver(b) \stackrel{def}{=} \overline{deliver}.Reply(b)$$

Protocolo AB em CCS

$$Send(b) \stackrel{def}{=} \overline{send_b}.\overline{time}.Sending(b)$$

$$Sending(b) \stackrel{def}{=} timeout.Send(b) + ack_b.timeout.Accept(b-1) + ack_{b-1}.Sending(b)$$

$$Accept(b) \stackrel{def}{=} accept.Send(b)$$

$$Reply(b) \stackrel{def}{=} \overline{reply_b}.\overline{time}.Replying(b)$$

$$Replying(b) \stackrel{def}{=} timeout.Reply(b) + trans_{b-1}.timeout.Deliver(b) + trans_b.Replying(b)$$

$$Deliver(b) \stackrel{def}{=} \overline{deliver}.Reply(b)$$

$$Timer \stackrel{def}{=} time.\overline{timeout}.Timer$$

Definindo os Canais através das transições

$$s, t \in \{0,1\}^*, b \in \{0,1\}$$

Definindo os Canais através das transições

$$s, t \in \{0,1\}^*, b \in \{0,1\}$$

$$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$$

Definindo os Canais através das transições

$$s, t \in \{0,1\}^*, b \in \{0,1\}$$

$$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$$

$$Ack(s) \xrightarrow{reply_b} Ack(sb)$$

Definindo os Canais através das transições

$$s, t \in \{0,1\}^*, b \in \{0,1\}$$

$$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$$

$$Ack(s) \xrightarrow{reply_b} Ack(sb)$$

$$Ack(sbt) \xrightarrow{\tau} Ack(st)$$

Definindo os Canais através das transições

$$s, t \in \{0,1\}^*, b \in \{0,1\}$$

$$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$$

$$Ack(s) \xrightarrow{reply_b} Ack(sb)$$

$$Ack(sbt) \xrightarrow{\tau} Ack(st)$$

$$Ack(sbt) \xrightarrow{\tau} Ack(sbbt)$$

Definindo os Canais através das transições

$s, t \in \{0,1\}^*, b \in \{0,1\}$

$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$

$Ack(s) \xrightarrow{reply_b} Ack(sb)$

$Ack(sbt) \xrightarrow{\tau} Ack(st)$

$Ack(sbt) \xrightarrow{\tau} Ack(sbbt)$

$Trans(sb) \xrightarrow{\overline{trans_b}} Trans(s)$

Definindo os Canais através das transições

$s, t \in \{0,1\}^*, b \in \{0,1\}$

$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$

$Ack(s) \xrightarrow{reply_b} Ack(sb)$

$Ack(sbt) \xrightarrow{\tau} Ack(st)$

$Ack(sbt) \xrightarrow{\tau} Ack(sbbt)$

$Trans(sb) \xrightarrow{\overline{trans_b}} Trans(s)$

$Trans(s) \xrightarrow{send_b} Trans(bs)$

Definindo os Canais através das transições

$s, t \in \{0,1\}^*, b \in \{0,1\}$

$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$

$Ack(s) \xrightarrow{reply_b} Ack(sb)$

$Ack(sbt) \xrightarrow{\tau} Ack(st)$

$Ack(sbt) \xrightarrow{\tau} Ack(sbbt)$

$Trans(sb) \xrightarrow{\overline{trans_b}} Trans(s)$

$Trans(s) \xrightarrow{send_b} Trans(bs)$

$Trans(tbs) \xrightarrow{\tau} Trans(ts)$

Definindo os Canais através das transições

$s, t \in \{0,1\}^*, b \in \{0,1\}$

$Ack(bs) \xrightarrow{\overline{ack_b}} Ack(s)$

$Ack(s) \xrightarrow{reply_b} Ack(sb)$

$Ack(sbt) \xrightarrow{\tau} Ack(st)$

$Ack(sbt) \xrightarrow{\tau} Ack(sbbt)$

$Trans(sb) \xrightarrow{\overline{trans_b}} Trans(s)$

$Trans(s) \xrightarrow{send_b} Trans(bs)$

$Trans(tbs) \xrightarrow{\tau} Trans(ts)$

$Trans(tbs) \xrightarrow{\tau} Trans(tbbs)$

Simplificação

Compondo Send em paralelo com Timer, e Reply com outro Timer:

Simplificação

Compondo Send em paralelo com Timer, e Reply com outro Timer:

$$Envia(b) \stackrel{def}{=} (Send(b) \mid Timer) \backslash \{time, timeout\}$$

Simplificação

Compondo Send em paralelo com Timer, e Reply com outro Timer:

$$Envia(b) \stackrel{def}{=} (Send(b) \mid Timer) \backslash \{time, timeout\}$$
$$Responde(b) \stackrel{def}{=} (Reply(b) \mid Timer) \backslash \{time, timeout\}$$

Simplificação

Obtemos novas expressões:

$$
\begin{aligned}
Envia(b) &= \overline{send_b}.Enviando(b) \\
Enviando(b) &= \tau.Envia(b) + ack_b.Aceita(b-1) + \\
&\quad ack_{b-1}.Enviand(b) \\
Aceita(b) &= accept.Envia(b)
\end{aligned}
$$

Simplificação

Obtemos novas expressões:

$$
\begin{aligned}
Envia(b) &= \overline{send_b}.Enviando(b) \\
Enviando(b) &= \tau.Envia(b) + ack_b.Aceita(b-1) + \\
&\quad ack_{b-1}.Enviand(b) \\
Aceita(b) &= accept.Envia(b) \\
Responde(b) &= \overline{reply_b}.Respodendo(b) \\
Respondendo(b) &= \tau.Reply(b) + trans_{b-1}.Entrega(b-1) + \\
&\quad trans_b.Respondendo(b) \\
Entrega(b) &= \overline{deliver}.Responde(b)
\end{aligned}
$$

Definição de AB

$$AB \stackrel{def}{=} Aceita(b-1) \,||\, Trans(\epsilon) \,||\, Ack(\epsilon) \,||\, Reply(b)$$

Definição de AB

$$AB \stackrel{def}{=} Aceita(b-1) \,||\, Trans(\epsilon) \,||\, Ack(\epsilon) \,||\, Reply(b)$$

O comportamento desejado do protocolo é o de um buffer simples:

$$Buff \stackrel{def}{=} accept.Buff'$$
$$Buff' \stackrel{def}{=} \overline{deliver}.Buff$$

Verificação de AB

É desejável que se prove que o protocolo AB, como foi descrito, se comporta como um buffer (equivalência observacional):

$$AB \approx Buff$$

Para provar isso, basta encontrar uma bissimulação que contém o par (AB,Buff). Uma bissimulação é a maior equivalência observacional entre dois processos.

Outline

Verificação de Modelos

Objetivo

Verificar a validade, ou não validade, de uma propriedade em um dado modelo.

- *Modelo do sistema: autômato que representa um modelo de Kripke*
- *Especificação das propriedades: fórmulas de uma lógica temporal*

Verificação de Modelos

Objetivo

Verificar a validade, ou não validade, de uma propriedade em um dado modelo.

- *Modelo do sistema: autômato que representa um modelo de Kripke*
- *Especificação das propriedades: fórmulas de uma lógica temporal*

Verificação de Modelos

Objetivo

Verificar a validade, ou não validade, de uma propriedade em um dado modelo.

- *Modelo do sistema: autômato que representa um modelo de Kripke*
- *Especificação das propriedades: fórmulas de uma lógica temporal*

Computation Tree Logic

- Permite referir-se ao futuro
- *Branching-Time* - Futuro não determinado, múltiplos caminhos possíveis.

Computation Tree Logic

- Permite referir-se ao futuro
- *Branching-Time* - Futuro não determinado, múltiplos caminhos possíveis.

Computation Tree Logic

- Permite referir-se ao futuro
- *Branching-Time* - Futuro não determinado, múltiplos caminhos possíveis.

Conectivos

Lógica Proposicional Clássica
$AX\phi$, $EX\phi$, $A[\phi U \phi]$, $E[\phi U \phi]$, $AG\phi$, $EG\phi$, $AF\phi$, $EF\phi$

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- *$X\phi$ - ϕ vale no próximo estado*
- *$\phi U \psi$ - ϕ vale até que ψ valha*
- *$G\phi$ - ϕ vale em todos os estados futuros*
- *$F\phi$ - ϕ vale em algum estado futuro*

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- *$X\phi$ - ϕ vale no próximo estado*
- *$\phi U \psi$ - ϕ vale até que ψ valha*
- *$G\phi$ - ϕ vale em todos os estados futuros*
- *$F\phi$ - ϕ vale em algum estado futuro*

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- *Xϕ - ϕ vale no próximo estado*
- *ϕUψ - ϕ vale até que ψ valha*
- *Gϕ - ϕ vale em todos os estados futuros*
- *Fϕ - ϕ vale em algum estado futuro*

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- *Xϕ - ϕ vale no próximo estado*
- *ϕUψ - ϕ vale até que ψ valha*
- *Gϕ - ϕ vale em todos os estados futuros*
- *Fϕ - ϕ vale em algum estado futuro*

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- *Xϕ - ϕ vale no próximo estado*
- *ϕUψ - ϕ vale até que ψ valha*
- *Gϕ - ϕ vale em todos os estados futuros*
- *Fϕ - ϕ vale em algum estado futuro*

Computation Tree Logic

Semântica

- *A - quantificador universal sobre os caminhos do modelo*
- *E - quantificador existencial sobre os caminhos do modelo*
- $X\phi$ - ϕ *vale no próximo estado*
- $\phi U \psi$ - ϕ *vale até que* ψ *valha*
- $G\phi$ - ϕ *vale em todos os estados futuros*
- $F\phi$ - ϕ *vale em algum estado futuro*

NuSMV

- Desenvolvido por ITC-IRST e Carnegie Mellon University
- Baseado no SMV de McMillan
- Ferramenta para verificação de sistemas de estados finitos
- Especificação do sistema em uma linguagem própria
- Especificação das propriedades em lógica temporal CTL

NuSMV

- Desenvolvido por ITC-IRST e Carnegie Mellon University
- Baseado no SMV de McMillan
- Ferramenta para verificação de sistemas de estados finitos
- Especificação do sistema em uma linguagem própria
- Especificação das propriedades em lógica temporal CTL

NuSMV

- Desenvolvido por ITC-IRST e Carnegie Mellon University
- Baseado no SMV de McMillan
- Ferramenta para verificação de sistemas de estados finitos
- Especificação do sistema em uma linguagem própria
- Especificação das propriedades em lógica temporal CTL

NuSMV

- Desenvolvido por ITC-IRST e Carnegie Mellon University
- Baseado no SMV de McMillan
- Ferramenta para verificação de sistemas de estados finitos
- Especificação do sistema em uma linguagem própria
- Especificação das propriedades em lógica temporal CTL

NuSMV

- Desenvolvido por ITC-IRST e Carnegie Mellon University
- Baseado no SMV de McMillan
- Ferramenta para verificação de sistemas de estados finitos
- Especificação do sistema em uma linguagem própria
- Especificação das propriedades em lógica temporal CTL

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

Vantagens

- *Interativo*
- *Análise de Invariantes*
- *Verificação de modelos com LTL*
- *Combina verificação baseada em BDD e SAT*

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

Vantagens

- *Interativo*
- *Análise de Invariantes*
- *Verificação de modelos com LTL*
- *Combina verificação baseada em BDD e SAT*

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

Vantagens

- *Interativo*
- *Análise de Invariantes*
- *Verificação de modelos com LTL*
- *Combina verificação baseada em BDD e SAT*

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

Vantagens

- *Interativo*
- *Análise de Invariantes*
- *Verificação de modelos com LTL*
- *Combina verificação baseada em BDD e SAT*

NuSMV

Linguagem

Descreve as transições de uma estrutura de Kripke finita através de expressões do cálculo proposicional.

Vantagens

- *Interativo*
- *Análise de Invariantes*
- *Verificação de modelos com LTL*
- *Combina verificação baseada em BDD e SAT*

Exemplo: Exclusão Mútua

```
MODULE main
  VAR
    semaforo : boolean;
    proc1    : process p(semaforo);
    proc2    : process p(semaforo);
  ASSIGN
    init(semaforo) := 0;
```

Exemplo: Exclusão Mútua

```
MODULE p(semaforo)
  VAR
    estado : {parado, entrando, critica, saindo};
  ASSIGN
    init(estado) := parado;
    next(estado) :=
      case
        estado = parado : {parado,entrando};
        estado = entrando & !semaforo : critica;
        estado = critica : {critica, saindo};
        estado = saindo                 : parado;
        1                               : estado;
      esac;
```

Exemplo: Exclusão Mútua

```
next(semaforo) :=
  case
    estado = entrando : 1;
    estado = saindo   : 0;
    1                 : semaforo;
  esac;
```

Exemplo: Exclusão Mútua

```
SPEC AG ! (proc1.estado = critica &
     proc2.estado = critica)
```

Exemplo: Exclusão Mútua

```
SPEC AG ! (proc1.estado = critica &
     proc2.estado = critica)
```

✓

Exemplo: Exclusão Mútua

```
SPEC AG ! (proc1.estado = critica &
     proc2.estado = critica)
```

✓

```
SPEC AG (proc1.estado = entrando ->
     AF proc1.estado = critica)
```

Exemplo: Exclusão Mútua

```
SPEC AG ! (proc1.estado = critica &
     proc2.estado = critica)
```

✓

```
SPEC AG (proc1.estado = entrando ->
     AF proc1.estado = critica)
```

⊥

Últimas Considerações

Algumas considerações sobre sistemas complexos:

- Não há um método que dê conta do desenvolvimento de um sistema complexo
- Métodos formais não são a solução de todos os problemas
- Métodos formais fazem parte do processo de desenvolvimento

Últimas Considerações

- Modularizar o Sistema
  - Parte Seqüencial
    - Pode-se usar Z para especificar a parte seqüencial do sistema
    - Pode-se usar Z para obter refinamentos da especificação
    - Pode-se usar lógica de Hoare para verificar código existente
    - Pode-se usar lógica de Hoare para verificar código recém-criado
  - Parte Concorrente
    - Pode-se usar CCS para especificar a parte concorrente do sistema
    - Pode-se usar CCS para verificar o comportamento do sistema
    - Pode-se usar Lógica Temporal para especificar propriedades do sistema
    - Pode-se usar Verificação de Modelos para checar propriedades

Referências


N. Storey
Safety-Critical Computer Systems.
Addison-Wesley, 1996.


NASA
Langley Formal Methods Site.


J. Woodcock e J. Davis
Using Z
Specification, Refinement, and Proof.
Prentice-Hall International Series in Computer Science, 1996.

Referências

R. Milner
Communication and Concurrency.
Prentice-Hall International Series in Computer Science, 1989.

M. R. A. Huth e M. D. Ryan
Logic in Computer Science
Modelling and reasoning about systems.
Cambdridge University Press, 2000.