

2. Transition Systems

José Proença

(slides mainly from Nelma Moreira)

Concurrent programming (CC3040) 2025/2026

CISTER – U.Porto, Porto, Portugal

<https://fm-dcc.github.io/cp2526>

Sistemas de Transição

- Um conjunto de estados
- Um conjunto de transições entre estados
- Um estado inicial
- Cada transição é etiquetada por uma ação que acciona a mudança de estado.

Estados

- Cor actual de um semáforo de trânsito



Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Transições

- Passagem de um semáforo de trânsito de vermelho para verde

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Transições

- Passagem de um semáforo de trânsito de vermelho para verde
- Execução de um comando num programa

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Transições

- Passagem de um semáforo de trânsito de vermelho para verde
- Execução de um comando num programa
- Aterragem de um avião

Estados

- Cor actual de um semáforo de trânsito
- Valor corrente das variáveis de um programa e do contador de programa
- A avião a voar
- Valor da conta bancária

Transições

- Passagem de um semáforo de trânsito de vermelho para verde
- Execução de um comando num programa
- Aterragem de um avião
- Depositar dinheiro numa conta bancária

Definição

Um sistema etiquetado de transições (LTS) sobre Act é um triplo $(S, \longrightarrow, s_0)$

- *S conjunto de estados*
- *$\longrightarrow \subseteq S \times Act \times S$ a relação de transição*
- *$s_0 \in S$ estado inicial*

Definição

Um sistema etiquetado de transições (LTS) sobre Act é um triplo $(S, \longrightarrow, s_0)$

- S conjunto de estados
- $\longrightarrow \subseteq S \times Act \times S$ a relação de transição
- $s_0 \in S$ estado inicial

Um Fósforo...



Definição

Um sistema etiquetado de transições (LTS) sobre Act é um triplo $(S, \longrightarrow, s_0)$

- S conjunto de estados
- $\longrightarrow \subseteq S \times Act \times S$ a relação de transição
- $s_0 \in S$ estado inicial

Um Fósforo...



- $S = \{UNUSED, BURNING, EXTINGUISHED\}$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$
- $Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), C \subseteq S$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$
- $Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), C \subseteq S$
- $Post(C, A) = \bigcup_{s \in C} \bigcup_{\alpha \in A} Post(s, \alpha), C \subseteq S, A \subseteq Act$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$
- $Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), C \subseteq S$
- $Post(C, A) = \bigcup_{s \in C} \bigcup_{\alpha \in A} Post(s, \alpha), C \subseteq S, A \subseteq Act$
- Ações que ocorrem no estado s

$$Act(s) = \{ \alpha \in Act \mid \exists s' : s \xrightarrow{\alpha} s' \}$$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$
- $Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), C \subseteq S$
- $Post(C, A) = \bigcup_{s \in C} \bigcup_{\alpha \in A} Post(s, \alpha), C \subseteq S, A \subseteq Act$
- Ações que ocorrem no estado s

$$Act(s) = \{ \alpha \in Act \mid \exists s' : s \xrightarrow{\alpha} s' \}$$

- Ações que podem ser observadas no estado s

$$Com(s) = \{ \alpha \in Com \mid \exists s' : s \xrightarrow{\alpha} s' \}$$

Em vez de $(s, \alpha, s') \in \longrightarrow$ escrevemos $s \xrightarrow{\alpha} s'$ e dizemos que s' é um **sucessor** de s , sendo $\alpha \in Act$ uma ação.

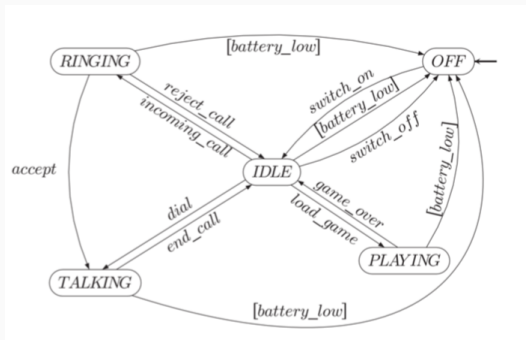
- $Post(s, \alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$
- $Post(s, A) = \bigcup_{\alpha \in A} Post(s, \alpha), A \subseteq Act$
- $Post(s) = Post(s, Act)$
- $Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), C \subseteq S$
- $Post(C, A) = \bigcup_{s \in C} \bigcup_{\alpha \in A} Post(s, \alpha), C \subseteq S, A \subseteq Act$
- Ações que ocorrem no estado s

$$Act(s) = \{ \alpha \in Act \mid \exists s' : s \xrightarrow{\alpha} s' \}$$

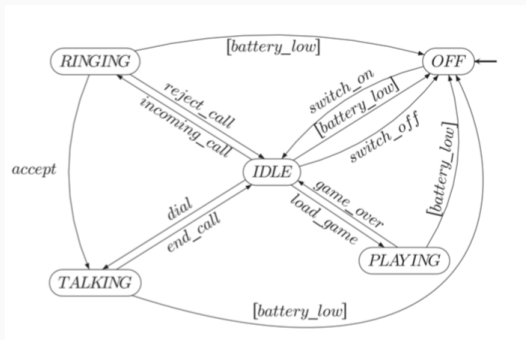
- Ações que podem ser observadas no estado s

$$Com(s) = \{ \alpha \in Com \mid \exists s' : s \xrightarrow{\alpha} s' \}$$

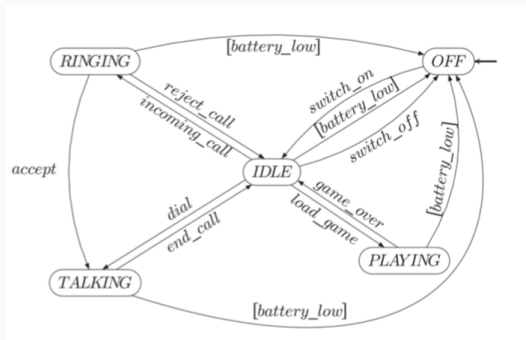
- Análogo para $Int(s)$



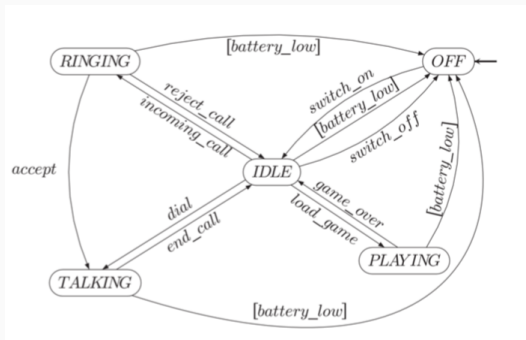
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.



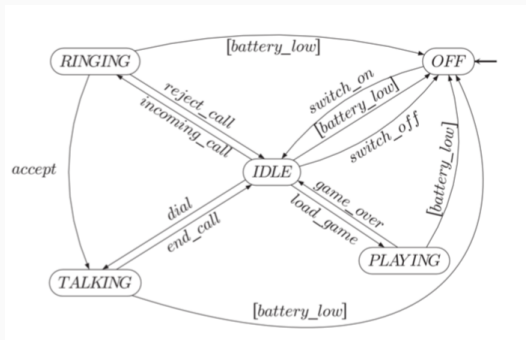
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) =$



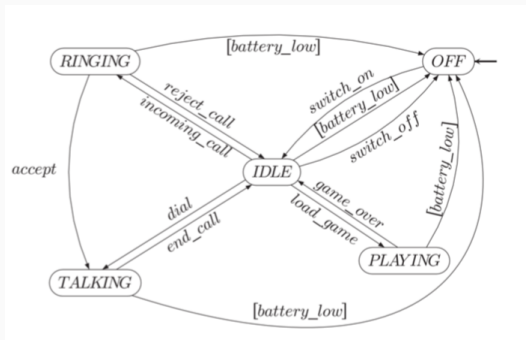
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) =$



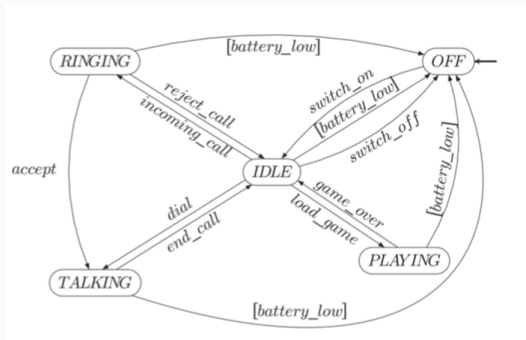
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) = \{TALKING, IDLE, OFF\}$
- $Post(Post(RINGING, Act), Int) =$



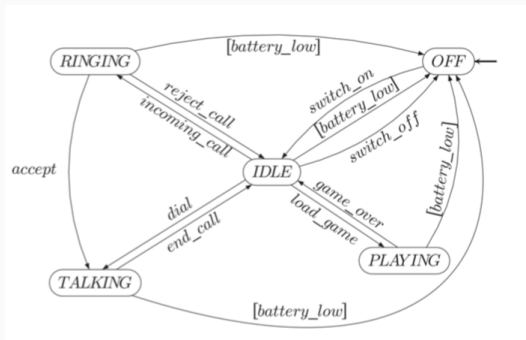
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) = \{TALKING, IDLE, OFF\}$
- $Post(Post(RINGING, Act), Int) =$



- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) = \{TALKING, IDLE, OFF\}$
- $Post(Post(RINGING, Act), Int) = \{OFF\}$
- $Com(RINGING) =$



- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) = \{TALKING, IDLE, OFF\}$
- $Post(Post(RINGING, Act), Int) = \{OFF\}$
- $Com(RINGING) =$



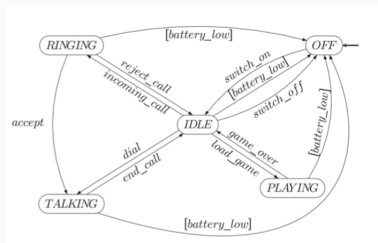
- Descreve o sistema de transições $(S, \longrightarrow, s_0)$ e $Act = Com \cup Int$.
- Alguns exemplos: $Post(RINGING, Act) = \{TALKING, IDLE, OFF\}$
- $Post(Post(RINGING, Act), Int) = \{OFF\}$
- $Com(RINGING) = \{accept, reject_call\}$

- Seja $TS = (S, \longrightarrow, s_0)$ um LTS.
- Um estado $s' \in S$ é **atingível** de s se $s' = s$ ou
$$\exists n \geq 0 \text{ e estados } s_1, s_2, \dots, s_n \text{ tal que } s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots \xrightarrow{\alpha_n} s_n \text{ e } s_n = s'.$$
- neste caso diz-se que existe um **caminho** de tamanho n entre s e s'
- um caminho é **acíclico** se $s_i \neq s_j$ para todo $i \neq j$; caso contrário diz-se cíclico.
- $Reach(s)$ conjunto de estados atingíveis de s
- $Reach(TS) = Reach(s_0)$

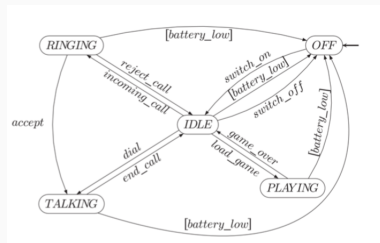
Sendo $\longrightarrow^* \subseteq S \times Act^* \times S$ o fecho reflexivo e transitivo de $\longrightarrow$, então $s \xrightarrow{\omega}^* s'$ se e só se $s' \in Reach(s)$, onde $\omega = \alpha_1 \cdots \alpha_n$ para alguns $\alpha_i \in Act$.

Não-determinismo

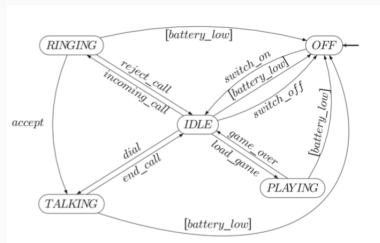
- Permite descrever o comportamento de sistemas reais



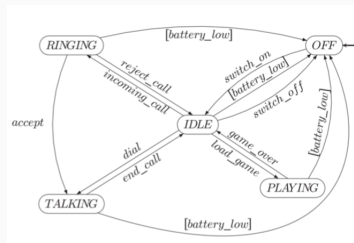
- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas



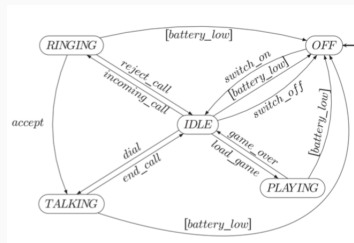
- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas
 - não é necessária uma descrição completa



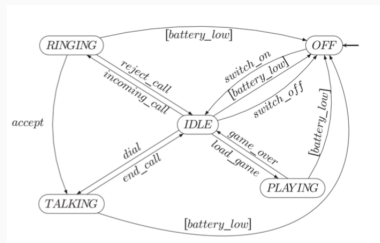
- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas
 - não é necessária uma descrição completa
 - os sistemas podem ser muito complexos



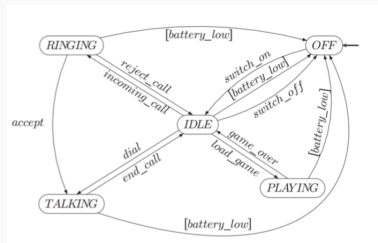
- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas
 - não é necessária uma descrição completa
 - os sistemas podem ser muito complexos
 - alguns parâmetros são desconhecidos



- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas
 - não é necessária uma descrição completa
 - os sistemas podem ser muito complexos
 - alguns parâmetros são desconhecidos
- existem várias maneiras de interagir



- Permite descrever o comportamento de sistemas reais
- Abstração de detalhes dos sistemas
 - não é necessária uma descrição completa
 - os sistemas podem ser muito complexos
 - alguns parâmetros são desconhecidos
- existem várias maneiras de interagir
- a noção é mais geral que em Linguagens Formais



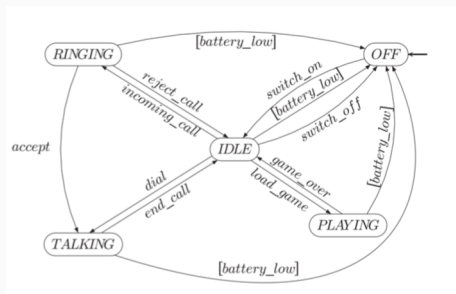
- Seja $TS = (S, \longrightarrow, s_0)$ um LTS.

- Seja $TS = (S, \longrightarrow, s_0)$ um LTS.
- TS é **determinístico** se e só se para todo $s \in S$

- Seja $TS = (S, \longrightarrow, s_0)$ um LTS.
- TS é **determinístico** se e só se para todo $s \in S$
- $|Post(s)| \leq 1$ e $|Act(s)| \leq 1$

- Seja $TS = (S, \longrightarrow, s_0)$ um LTS.
- TS é **determinístico** se e só se para todo $s \in S$
- $|Post(s)| \leq 1$ e $|Act(s)| \leq 1$
- senão é **não-determinístico**

Um sistema é não determinístico se tem um estado com duas ou mais transições mas não sabemos qual irá acontecer.

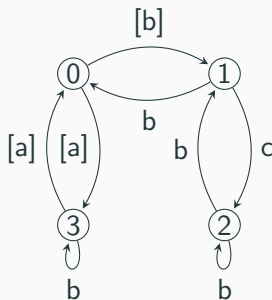


Um estado s é

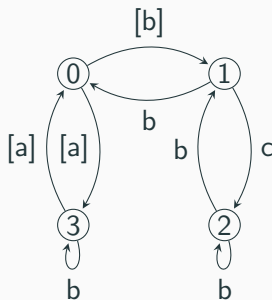
- não-determinístico **externo** sse $|Com(s)| > 1$

Um estado s é

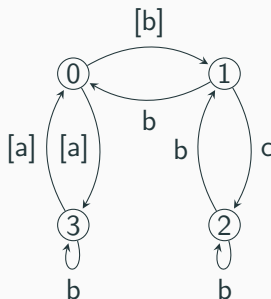
- não-determinístico **externo** sse $|Com(s)| > 1$
- não-determinístico **interno** sse $|Post(s, Int)| > 1$ ou $|Post(s, a)| > 1$ para algum $a \in Com(s)$



- 1 é não-determinístico externo



- 1 é não-determinístico externo
- 0 e 2 são não-determinísticos internos.

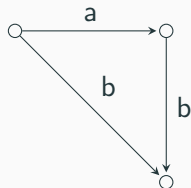


- 1 é não-determinístico externo
- 0 e 2 são não-determinísticos internos.
- 3 é não-determinístico mas nem interno nem externo.

Tipo de Sistemas de Transição

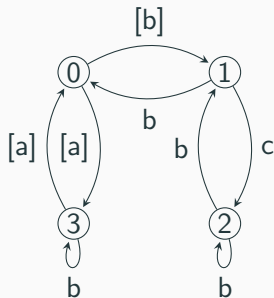
Seja $TS = (S, \longrightarrow, s_0)$ com ações em Act

- **Finito** Se o grafo é acíclico e S e Act finitos



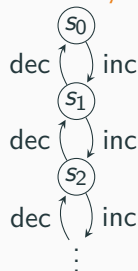
Seja $TS = (S, \longrightarrow, s_0)$ com ações em Act

- **Finito** Se o grafo é acíclico e S e Act finitos
- **Finito por estados** Se S e Act são conjuntos finitos



Seja $TS = (S, \longrightarrow, s_0)$ com ações em Act

- **Finito** Se o grafo é acíclico e S e Act finitos
- **Finito por estados** Se S e Act são conjuntos finitos
- **Ramificação-limitada** $(\exists k \geq 0)(\forall s \in S)(|Post(s)| \leq k)$



s_n guarda o valor n que pode ser incrementado ou decrementado de 1. Tem grau de ramificação 2. Não é finito por estados

Seja $TS = (S, \longrightarrow, s_0)$ com ações em Act

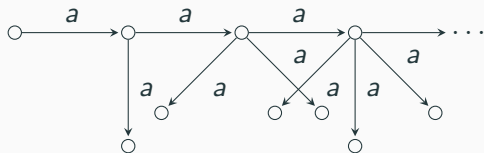
- **Finito** Se o grafo é acíclico e S e Act finitos
- **Finito por estados** Se S e Act são conjuntos finitos
- **Ramificação-limitada** $(\exists k \geq 0)(\forall s \in S)(|Post(s)| \leq k)$
- **Finitamente Ramificado** $(\forall s \in S)(|Post(s)| \leq \infty)$. Caso contrário é infinitamente ramificado



Não é finitamente ramificado

Seja $TS = (S, \longrightarrow, s_0)$ com ações em Act

- **Finito** Se o grafo é acíclico e S e Act finitos
- **Finito por estados** Se S e Act são conjuntos finitos
- **Ramificação-limitada** $(\exists k \geq 0)(\forall s \in S)(|Post(s)| \leq k)$
- **Finitamente Ramificado** $(\forall s \in S)(|Post(s)| \leq \infty)$. Caso contrário é infinitamente ramificado



É finitamente ramificado mas **não é** de ramificação-limitada

- Uma questão fulcral é a de decidir quando dois LTSs são equivalentes.
- Em princípio serão quando um observador não os consegue distinguir.
- Mas como definir isso? Podemos obrigar a que
 - os LTS são isomorfos como grafos (Isomorfismo)
 - ou os LTS tenham os mesmos caminhos (Equivalência por traços (linguagens))
 - ou ...
 - iremos deixar isto para mais tarde

Modelação de Processos Concorrentes

- Cada processo é representado por um sistema de transições (LTS)

- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)

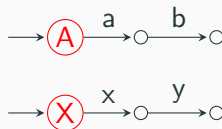
- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)
- O que acontece se executam concorrentemente?

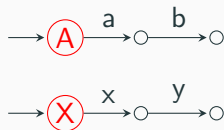
- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)
- O que acontece se executam concorrentemente?
- Assumimos que:

- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)
- O que acontece se executam concorrentemente?
- Assumimos que:
 - O tempo só é considerado de forma relativa (um processo p ocorre antes do processo q)

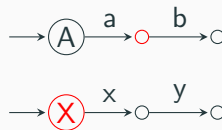
- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)
- O que acontece se executam concorrentemente?
- Assumimos que:
 - O tempo só é considerado de forma relativa (um processo p ocorre antes do processo q)
 - As ações são atômicas e instantâneas

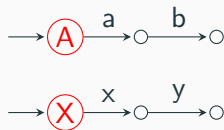
- Cada processo é representado por um sistema de transições (LTS)
- Neste caso, o tempo avança quando se muda de estado por uma transição (execução de um programa sequencial)
- O que acontece se executam concorrentemente?
- Assumimos que:
 - O tempo só é considerado de forma relativa (um processo p ocorre antes do processo q)
 - As ações são atômicas e instantâneas
 - Os processos concorrentes executam independentemente excepto se explicitamente comunicarem (coordenação)



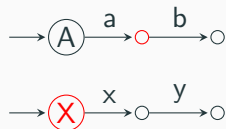


se A executa a

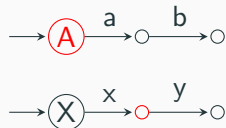


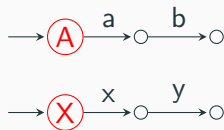


se A executa a

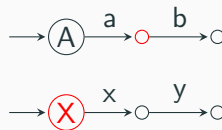


se X executa x

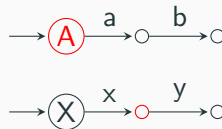




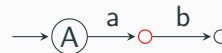
se A executa a



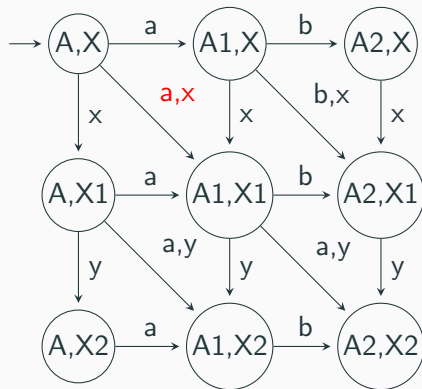
se X executa x



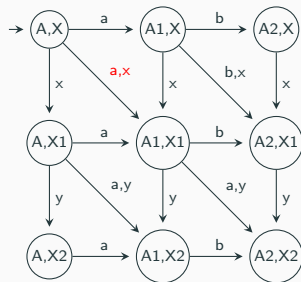
se A e X executam x e a



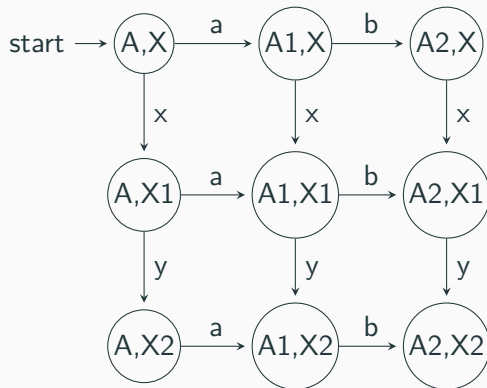
A execução de dois processos é um novo processo (LTS)!



- As diagonais têm conjuntos de ações (!)
- Mas como observar que duas ações independentes ocorrem ao mesmo tempo?
- O seu efeito corresponde a ser primeiro uma e depois outra não interessando a ordem
- Sendo não determinístico, podemos ignorar essas transições.
- Não são observáveis



Os estados atingíveis são os mesmos. As ações podem ocorrer por qualquer ordem (não determinismo).



Dados dois processos independentes a concorrência é a **intercalagem** (*interleaving/shuffle*) não-determinística de todas as ações dos dois processos.

O Diamante de intercalagem é o seguinte:

