



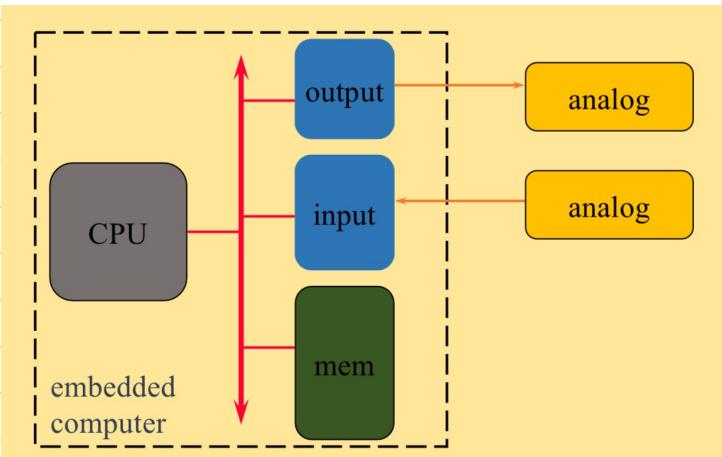
Introdução

Sistema Embutido: sistema de computação em miniatura desenvolvido para dispositivos de baixo consumo e alto desempenho, que deve realizar um número fixo de tarefas predefinidas com prazos — sistema especializado para uma aplicação particular e que contém um controlador de micro-computador.

Sistema Embutido: dispositivo eletrônico com capacidade computacional, mas cujo propósito principal não é computação e no qual os utilizadores não conseguem instalar o seu próprio software aplicacional — sistema em mídia que combina HW, SW e FW para uma aplicação/cansa específica

Sistema Embutido: sistema projetado para uma tarefa específica (não um computador multi-propósito) com restrições de memória, potência, tamanho e custo, cujos componentes necessários são locais — sistema digital que faz interface com hardware, na fronteira entre os mundos físico e digital

Sistema Computacional Embutido: qualquer dispositivo que inclui um computador programável mas não é, por si só, um computador de propósito-geral, tirando proveito das características aplicacionais para otimizar o design!



- Características**
- 1. Computacional
 - 2. Integro com Processos Físicos — Sensores & Atuadores
 - 3. Reativo
 - 4. Heterogéneo — Hardware & Software
 - 5. Em Rede — Partilhado & Adaptativo

Micro-Controlador: inclui dispositivos de I/O e memória

Processador de Sinal Digital: micro-processador otimizado para PSD

- Características**
- 1. funcionalidade sofisticada — algoritmos/interfaces sofisticados
 - 2. operação em tempo real — finalizar operações a tempo
 - 3. baixo custo de fabrico } — requisitos não-funcionais
 - 4. baixa potência }
 - 5. projetada para prazos apertados por equipas pequenas

"Hard Real Time": falhas prazo causa falha

"Soft Real Time": falha prazo resulta em pior desempenho

Sistema "Multi-Rate": deve lidar com operações em plenórios amplamente variáveis

→ Micro-Processadores não muito eficientes porque podem usar a mesma lógica para desempenhar muitas funções diferentes, simplificando o "design" de famílias de produtos

Circuitos Integrados:

- 1. Application Specific Integrated Circuit (ASIC) — específico para aplicações
- 2. Field-Programmable Gate Arrays (FPGA) — configurável após construção
- 3. Application Specific Instruction-set Processors (ASIP) — específico
- 4. Central Processing Unit (CPU) — propósito geral

Sistema Ciber-Físico = Computacional + Físico

Requisitos

"Top-Down": da descrição mais abstrata para a mais detalhada

"Bottom-Up": dos componentes pequenos para o sistema grande

Requisitos: descrição em linguagem formal do que o utilizador quer e espera obter

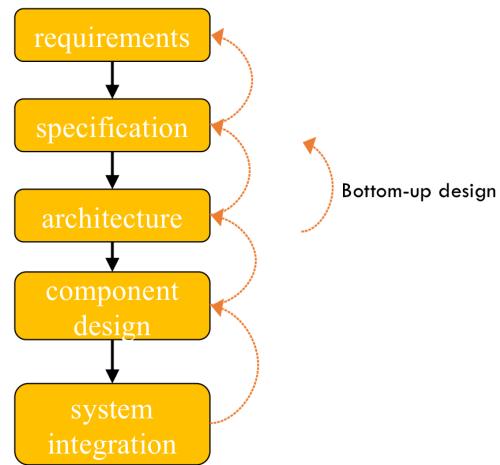
Requisito Funcional

Requisito Não-Funcional

Forma:

1. nome
2. propósito
3. inputs
4. outputs
5. funções
6. desempenho
7. custo de fabrico
8. potência
9. tamanho/peso

Top-down design



Especificação: descrição mais precisa do sistema que, além implicar uma arquitetura particular, fornece informação sobre o processo de design de arquitetura, podendo incluir elementos funcionais e não-funcionais, bem como ser executável ou em forma matemática para provas

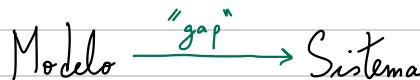
Design de Arquitetura: que componentes principais satisfazem a especificação?
HW + SW FUNCIONAL OU NÃO

Modelação, Design e Análise

Modelo: descrição de um sistema que permite extrair informações/conhecimento sobre as suas propriedades

Ex: esquema (BD); sistema de software (componentes + API)

Os modelos tomam tipicamente a forma de formalismos matemáticos e/ou objetos — o modelo é uma abstração matemática do sistema



Fidelidade do Modelo: distância entre o modelo e o sistema

Design: pegar num modelo do sistema e projetar/implementar uma instância do modelo — instância concreta (hardware + software)

Escolher: sensores; atuadores; processadores; memória; VO; software

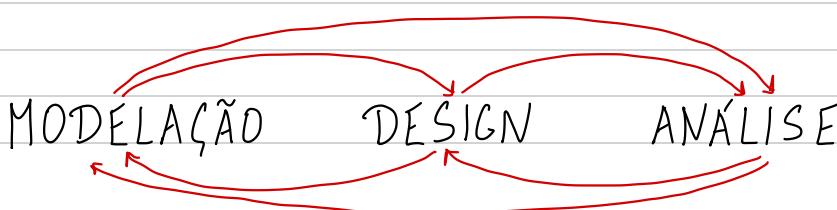
↓ PONDERAR

- concorrência
- tempo-real

Análise: usar ferramentas para verificar que o sistema se comporta como especificado pelo modelo

Ferramentas:

- análise invariantes
- lógica temporal
- equivalência e afinção
- verificação do modelo
- análise quantitativa
- segurança e privacidade



Design Baseado no Modelo

Modelação: processo de ganhar um conhecimento mais profundo de um sistema através de imitação — os modelos especificam o que um sistema faz

Design: criação estruturada de artefactos — especificam como um sistema faz o que faz

Análise: processo de ganhar um conhecimento mais profundo de um sistema através de distinção — especifica porque é que um sistema faz o que faz (ou falha em fazer o que o modelo diz que devia fazer)

Modelo: artefacto que imita o sistema, processo ou artefacto de interesse

Modelo Matemático: conjunto de definições e fórmulas/objetos matemáticos

Princípio de Koopczyk: muitas propriedades (preditivas) afirmadas sobre sistemas (determinismo, tempo, fiabilidade, segurança) de facto não são propriedades de um sistema implementado, mas sim propriedades do modelo do sistema — as propriedades das realizações do sistema são inferidas a partir das do modelo
FIDELIDADE

Design Baseado no Modelo:

1. Criar um modelo matemático de todas as partes do sistema embutido
2. Construir a implementação a partir do modelo

Objetivo: automatizar esta construção, como um compilador

TÉCNICAS DE MODELAÇÃO

1. Fenômenos Físicos — Equações Diferenciais Ordinárias
2. Sistemas de Controlo por Feedback — Modelação Tempo-Domínio
3. Comportamento Modular — Autômatos Finitos/Híbricos
4. Sensores e Atuadores — Calibração e Ruído
5. Software — Concorrência e Tempo Real
6. Redes — Latência, Ráios de Erros, Perda de Pacotes

EX: Modelação de Movimento Físico

→ 6 graus de liberdade

Posição: x, y, z

Orientação: ω, β, γ

$$\begin{array}{l} x: \mathbb{R} \rightarrow \mathbb{R} \\ y: \mathbb{R} \rightarrow \mathbb{R} \\ z: \mathbb{R} \rightarrow \mathbb{R} \end{array} \quad \left. \begin{array}{l} x: \mathbb{R} \rightarrow \mathbb{R}^3 \\ - t \in \mathbb{R}, \quad x(t) \in \mathbb{R}^3 \end{array} \right\}$$

$$\dot{x}: \mathbb{R} \rightarrow \mathbb{R}^3, \quad \dot{x}(t) = \frac{d}{dt} x(t)$$

$$\ddot{x}: \mathbb{R} \rightarrow \mathbb{R}^3, \quad \ddot{x}(t) = \frac{d^2}{dt^2} x(t)$$

$$F: \mathbb{R} \rightarrow \mathbb{R}^3, \quad F(t) = \frac{d}{dt} (M \dot{x}(t)) = M \ddot{x}(t) \quad \text{SEGUNDA LEI DE NEWTON}$$

$$\begin{aligned} x(t) &= x(0) + \int_0^t \dot{x}(\tau) d\tau \\ &= x(0) + \int_0^t \left[\dot{x}(0) + \int_0^\tau \ddot{x}(\omega) d\omega \right] d\tau \\ &= x(0) + t \dot{x}(0) + \frac{1}{2} M \int_0^t \int_0^\tau F(\omega) d\omega dt \end{aligned}$$

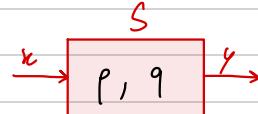
$$\begin{aligned} \theta(t) &= \begin{bmatrix} \theta_x(t) \\ \theta_y(t) \\ \theta_z(t) \end{bmatrix} \\ \cdot I(t) &= \frac{d}{dt} (I(t) \dot{\theta}(t)) \Leftrightarrow \begin{bmatrix} I_x(t) \\ I_y(t) \\ I_z(t) \end{bmatrix} = \frac{d}{dt} \begin{pmatrix} I_{xx}(t) & I_{xy}(t) & I_{xz}(t) \\ I_{yx}(t) & I_{yy}(t) & I_{yz}(t) \\ I_{zx}(t) & I_{zy}(t) & I_{zz}(t) \end{pmatrix} \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix} \end{aligned}$$

$$\rightarrow I_y(t) = I_{yy} \dot{\theta}_y(t)$$

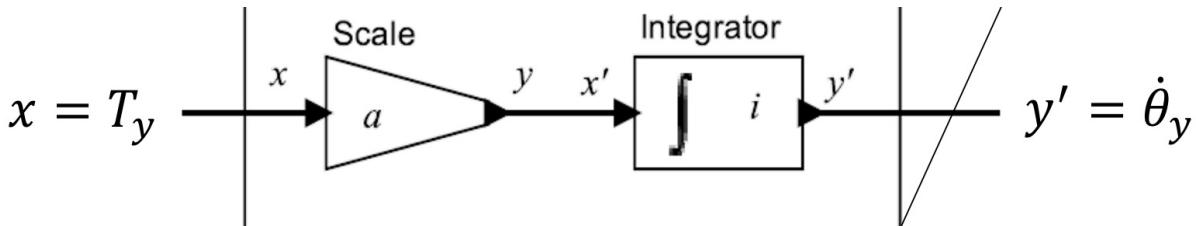
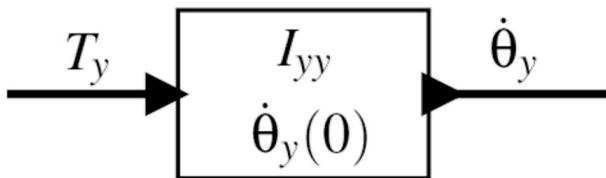
$$\rightarrow \dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t I_y(\tau) d\tau$$

Sistema: função que aceita um sinal de input e retorna um sinal de output

- O domínio e o alcance da função do sistema são conjuntos de sinais, que não são eles próprios funções
- Os parâmetros podem afetar a definição da função S

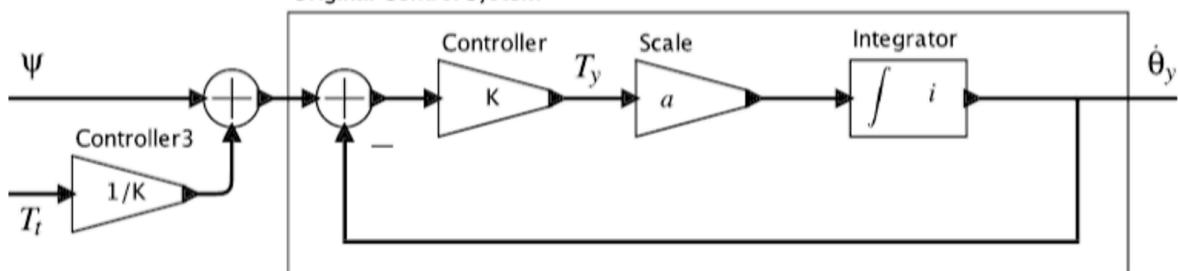


Helicopter



$$\begin{aligned} \dot{\theta}_y(t) &= \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t \dot{\theta}_y(\tau) d\tau \\ &\stackrel{\psi(\tau)=0}{=} \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t (\psi(\tau) - \dot{\theta}_y(\tau)) d\tau \\ &\stackrel{\dot{\theta}_y(t)=0}{=} \dot{\theta}_y(0) - \frac{1}{I_{yy}} \int_0^t \dot{\theta}_y(\tau) d\tau = \dot{\theta}_y(0) e^{-kt/I_{yy}} u(t) \end{aligned}$$

Original Control System



PROPRIÉDADES DO SISTEMA

Estabilidade "Bounded-Input Bounded-Output" (BIBO): se um input é sempre limitado, então o output também permanece sempre limitado

Input limitado: $\exists A \in \mathbb{R}, \forall t \in \mathbb{R}, |x(t)| < A$

Output limitado: $\exists B \in \mathbb{R}, \forall t \in \mathbb{R}, |y(t)| < B$

Restrição no Tempo: $x|_{t < T}$ é um input que só tem valores em tempos $t < T$

Causalidade: o output atual só pode depender dos inputs atual e passados

\uparrow
SE $x_1|_{t < T} = x_2|_{t < T}$ ENTAO um sistema S para o qual x_1 e x_2 são inputs válidos é causal se e só se $S(x_1)|_{t < T} = S(x_2)|_{t < T}$

Causalidade Estrita: o output atual só pode depender dos inputs passados

SE $x_1|_{t < T} = x_2|_{t < T}$ ENTAO um sistema S para o qual x_1 e x_2 são inputs válidos é estritamente causal se e só se $S(x_1)|_{t < T} = S(x_2)|_{t < T}$

Sem Memória: o output atual só depende do input atual

$\hookrightarrow \exists f: A \rightarrow B, (S(x))(t) = f(x(t))$ — o output em t só depende do input em t

Linearidade: $S(ax_1 + bx_2) = aS(x_1) + bS(x_2)$ — Hiper-posição

Aditividade: $S(x_1 + x_2) = S(x_1) + S(x_2)$

Homogeneidade: $S(ax) = aS(x)$

Imparânciâ Temporal: o output do sistema a atual numa versão atrasada do input é igual à versão atrasada do output do sistema a atual no sistema original — seja D_T um sistema que atrasa um sinal tal que $D_T(x(t)) = x(t - T)$, então um sistema S é temporalmente invariante se e só se

$$S(D_T(x)) = D_T(S(x))$$

Modelação do Comportamento

Sistema Contínuo: o comportamento varia continuamente no tempo
Ex: helicóptero

Sistema Discreto: o comportamento varia descontinuamente no tempo - salta de estado para estado de acordo com estímulos externos
Ex: contar o número de carros que entram e saem de um arque

Sinal Puro: presente ou ausente - nenhum outro valor possível - $\{R \rightarrow \{0, 1\}\}$
Sinal Numérico: o valor do sinal tem um significado para cada input

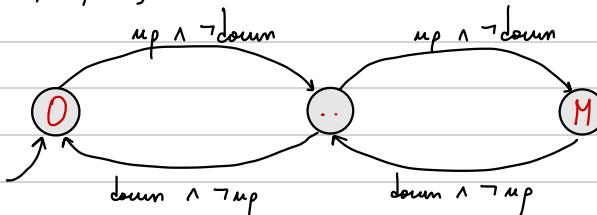
Ex: $\underbrace{\{R \rightarrow \{\text{ausente, presente}\}}}_{\text{up}} \rightarrow \underbrace{\{R \rightarrow \{\text{ausente, presente}\}}}_{\text{down}} \rightarrow \underbrace{\{R \rightarrow \{\text{ausente}\}}}_{\text{count}} \cup \text{IN}$

$\forall t \in \mathbb{R}$: $up(t) \neq \text{ausente} \wedge down(t) \neq \text{ausente}$, o contador reage, produzindo um valor de output e alterando o seu estado interno

Inputs = $(\{up, down\} \rightarrow \{\text{ausente, presente}\})$

Outputs = $(\{count\} \rightarrow \{\text{ausente}\}) \cup \text{IN}$

States = $\{0, 1, 2, \dots, M\}$

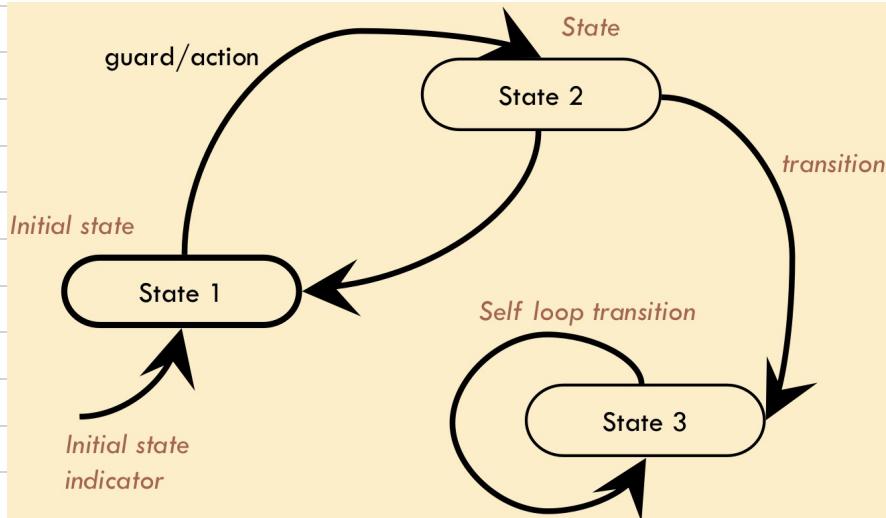


Update = States \times Inputs \rightarrow States \times Outputs

initial State = 0

Máquina de Estados Finita: (States, Inputs, Outputs, Update, initialState)

$\text{update}(\Delta, i) \left\{ \begin{array}{l} ((\Delta+1, \Delta+1), \Delta < M \wedge i(\text{up}) = \text{presente} \wedge i(\text{down}) = \text{ausente}) \\ ((\Delta-1, \Delta-1), \Delta > 0 \wedge i(\text{up}) = \text{ausente} \wedge i(\text{down}) = \text{presente}) \\ ((\Delta, \text{ausente}), \text{caso contrario}) \end{array} \right.$



<i>true</i>	Transition is always enabled
p_1	Transition is enabled if p_1 is present
$\neg p_1$	Transition is enabled if p_1 is absent
$p_1 \wedge p_2$	Transition is enabled if both p_1 and p_2 are present
$p_1 \vee p_2$	Transition is enabled if either p_1 or p_2 are present
$p_1 \wedge \neg p_2$	Transition is enabled if p_1 is present and p_2 is absent

p_3	Transition is enabled if p_3 is present (not absent)
$p_3 = 1$	Transition is enabled if p_3 is present and has value 1
$p_3 = 1 \wedge p_1$	Transition is enabled if p_3 has value 1 and p_1 is present
$p_3 > 5$	Transition is enabled if p_3 is present and has value greater than 5

"Hysteresis": solução para evitar "chattering"

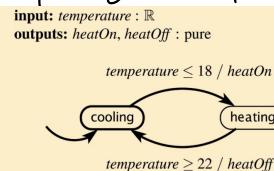
Modelo Guiado por Eventos: as transições ocorrem quando o ambiente lança uma reação

Modelo Guiado pelo Tempo: as transições ocorrem de acordo com um relógio externo

Uma transição predefinida é ativada se nenhuma transição não predefinida está ativa e não tem nenhuma guarda ou a guarda avalia para verdadeiro — Nós armazenadas se têm guardas ou produzem outputs

```

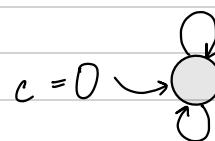
States = {cooling, heating}
Inputs = (temperature → ℝ)
Outputs = ({heatOn, HeatOff} → {absent, present})
Update: States × Input → States × Outputs
initialState = cooling
    
```



$$update(s, i) = \begin{cases} (\text{heating}, \text{present}, \text{absent}) & \text{if } s = \text{cooling} \\ (\text{cooling}, \text{absent}, \text{present}) & \text{if } s = \text{heating} \\ (s, \text{absent}, \text{absent}) & \text{otherwise} \end{cases}$$

MÁQUINA DE ESTADOS ESTENDIDA

Variável = $\{c : \{0, \dots, M\}\}$
Inputs = $\{\text{up}, \text{down}\}$
Outputs = $\text{count} : \{0, \dots, M\}$



$$\text{up} \wedge \neg \text{down} \wedge c < M \quad / \quad c + 1$$

$$c = c + 1$$

$$\text{down} \wedge \neg \text{up} \wedge c > 0 \quad / \quad c - 1$$

$$c = c - 1$$

estados | < nós × valores variáveis

Comportamento: sequência de pares (não gaguejantes)

Rasto: registo de inputs, estados e outputs num comportamento

Árvore de Computação: representação gráfica de todos os rastos possíveis

Máquinas de Estados Finitos não adequadas para análise formal

Ex:

up	=	(A, P, A, A, P, \dots)	$0 \xrightarrow{A, P/A} 0 \xrightarrow{P, A/1} 1 \xrightarrow{A, P/0} 0$
$down$	=	(P, A, P, P, A, \dots)	
$count$	=	$(A, 1, 0, A, 1, \dots)$	

Transição Gaguejante: transição predefinida (possivelmente implícita) que é ativada quando todos os inputs estão ausentes, não altera o estado e produz outputs ausentes

Receptividade: para quaisquer valores de input, alguma transição é ativada para algum estado ≥ 1

Determinismo: em cada estado, para quaisquer valores de input, exatamente um transição (possivelmente implícita) é ativada $= 1$

Máquina de Estados Finito Não-Determinista: a função de atualização é substituída por uma relação de atualização que mapeia uma combinação (estado, input) num conjunto de possíveis pares (próximo estado, output)

possible Updates: States \times Inputs \rightarrow $2^{\text{States} \times \text{Outputs}}$

1. Modelar aspectos desconhecidos do ambiente ou sistema
2. Ocultar detalhes de uma especificação do sistema

→ Este modelo permite composição de máquinas de estado

Composição de Máquinas de Estado

A principal desvantagem das máquinas de estados finitas é que o número de estados pode ser muito grande

Solução: construir sistemas grandes pela composição de sistemas mais pequenos

Sintaxe: regras das notações,

Semântica: significado das notações

→ Utiliza-se o modelo de atores para máquinas de estados finitas!
(Estados, Inputs, Outputs, Atualização, Estado Inicial)

- Composição
- 1. Concorrente \oplus
 - a. Síncrona - reação simultânea
 - b. Assíncrona - reação independente
 - 2. Sequencial \otimes
 - 3. Hierárquica

A composição é modular no sentido em que a própria composição se torna num componente que pode ser posteriormente composto como se fosse atómico

Uma composição síncrona de duas FSM também é uma FSM

Uma propriedade é **composicional** se o facto de ser verdade para dois componentes implicar que também seja verdade para a sua composição

$$C = (S_A \times S_B, I_A \times I_B, O_A \times O_B, u_C, (i_A, i_B))$$

$$u_C((s_A, i_A), (s_B, i_B)) = ((s'_A, s'_B), (o_A, o_B))$$

Composição Concorrente Síncrona: A e B reagem simultaneamente
Composição Concorrente Assíncrona: A e B reagem independentemente

(1) Uma reação de C é uma reação de A ou B - "interleaving"
(2) Uma reação de C é uma reação de A ou B ou ambos

- A escolha entre A ou B é não-determinista
- A e B podem falhar eventos de input

(3) Uma reação de C é uma reação de ou A ou B, em que o ambiente (tempojador) escolhe A ou B

(4) Uma reação de C é uma reação de A ou B ou ambos, em que a escolha é feita pelo ambiente

• A constuição (1) ou (2) não é determinista - o determinismo não é uma propriedade componencial para composição assíncrona

→ Podem existir variáveis partilhadas!

Composição Sequencial: type(01) \subseteq type(02)

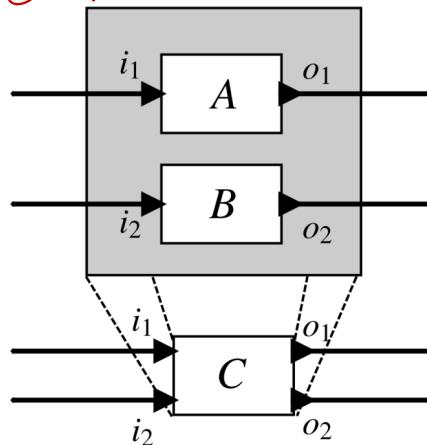
$C \left\{ \begin{array}{l} 1. A \text{ reage com } \text{input} \xrightarrow{\quad} \text{output} \\ 2. B \text{ reage com } \text{input} \xleftarrow{\quad} \text{output} \end{array} \right\}$ SIMULTÂNEO ($\Delta t = 0$) $\left\{ \begin{array}{l} A = 0 \Rightarrow C = 0 \\ A = 1 \Rightarrow C = 1 \end{array} \right.$

$$C = (S_A \times S_B, I_A, O_B, f, (i_A, i_B)) \quad a = 0 \wedge b = 0 \rightarrow c = 0$$

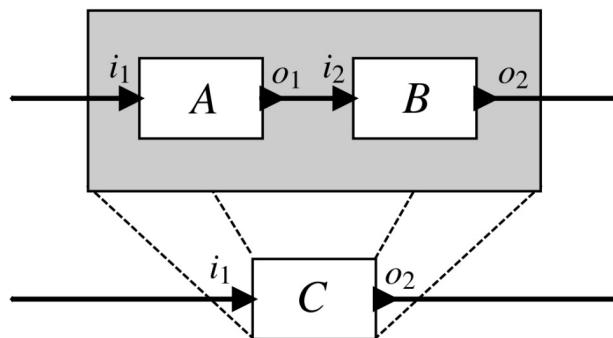
Composição Hierárquica: refinamento de estados - comportamento abstrato num ^{nó refinado}

Transições Preemptivas: guardas avaliadas antes de o refinamento reagir
Transições Históricas: o refinamento reinicia sempre no estado em que estava

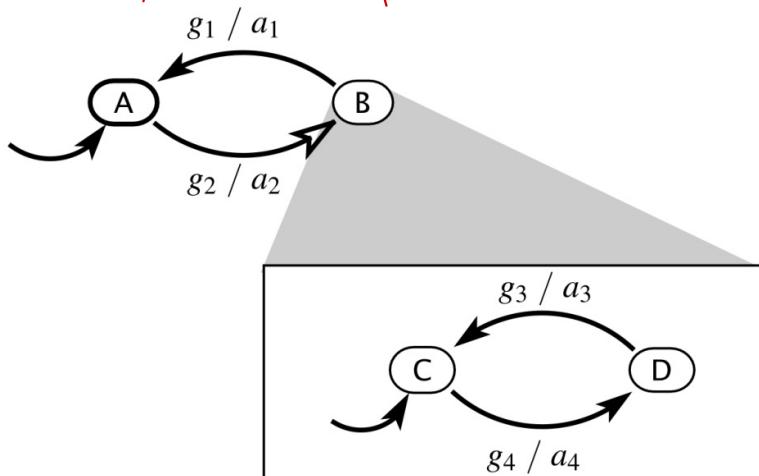
CONCORRENTE



SEQUENCIAL



HIERÁRQUICA



Modelos de Computação Concorrente

Sistema Concorrente: partes conceptualmente diferentes operam simultaneamente - não há ordem particular das operações **INTERAÇÕES**

Modelo de Computação { 1. O que constitui um componente? **ATOR COM PORTOS**
{ 2. Quais são os mecanismos de concorrência?
{ 3. Quais são os mecanismos de comunicação?

Ação de Execução: define como o ator reage a inputs para produzir outputs e alterar o estado

Modelo de Feedback: interconexões fixas de atores que especificam caminhos de comunicação, que toma a forma de um rinal, que consiste em um ou mais eventos de comunicação

Sinal Discreto: $s: R \rightarrow V_s \cup \{\text{ausente}\}$
Tipo

Evento de Comunicação: valor não ausente do sinal

Sinal de Relógio: sinal puro só presente em múltiplos de um valor real P

$$s(t) = \begin{cases} \text{presente, se } t \text{ é múltiplo de } P \\ \text{ausente, caso contrário} \end{cases}$$

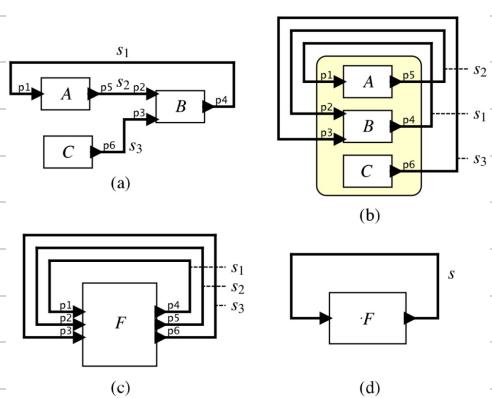
Ex:

$$\Delta_1 = B(s_2, s_3)$$

$$\Delta_2 = A(\Delta_1)$$

$$\Delta_3 = C(\emptyset)$$

$$\Delta = F(\Delta), \Delta = (\Delta_1, \Delta_2, \Delta_3)$$



Função de Ponto Fijo: $f: X \rightarrow X$, $\exists x \in X: f(x) = x$

$s = f(s)$ afirma que a semântica de um determinado ator da rede é um ponto ^{fíco}

Modelo Síncrono-Reativo: sistema discreto cujos níveis estão sempre presentes例外 (por vezes) em tiques de um relógio global - a execução é uma sequência de reações globais que ocorrem em tempos discretos e, em cada evento, a reação de todos os atores é simultânea e instantânea

Função de Disparo: $F_n: V_1 \cup \{\text{ausente}\} \rightarrow V_2 \cup \{\text{ausente}\}$

Seja F uma máquina de estados e $s = f(s)$

Dado o seu estado atual, no mesmo tique do relógio global, deve encontrar-se o valor de s tal que seja um input válido e um output válido da máquina de estados

Seja $s(n)$ o valor do nível s na mesma reação

O objetivo é determinar, em cada tique do relógio global, o valor de $s(n)$ que satisfaz a equação de ponto fixo

! Qualquer conexão de um ponto de output para um ponto de input significa que o valor no ponto de input é o mesmo que o valor no ponto de output sempre

$f_{ire_X}: \{\text{presente, ausente}\} \rightarrow \{\text{presente, ausente}\}$ $\rightarrow s(n) = f_{ire_X}(s(n))$

(1) Não há ponto fixo

(2) Há mais do que um ponto fixo

? SISTEMA BEM/MAL FORMADO

Como construir o ponto fixo? Para cada estado alcançável K

1. começar com $s(n)$ desconhecido
2. determinar tanto quanto possível sobre $f_{i \in K}(s(n))$
3. repetir 2 até todos os valores de $s(n)$ serem conhecidos ou não dar mais
4. se permanecerem valores desconhecidos, então rejeitar o modelo

→ pode rejeitar modelos que têm um único ponto fixo

Máquina de Estados Constitutiva: o procedimento funciona em todos os alcancáveis estados

Análise Tem-Pode: o que tem de ser verdade sobre os outputs e o que pode ser verdade sobre os outputs

Modelo de Fluxo de Dados: as reações podem ocorrer simultaneamente ou não
↳ quando as reações dependemumas das outras, a dependência deve-se ao fluxo de dados e não à sincronia dos eventos — se uma reação de um ator A requer dados produzidos por uma reação de um ator B, então a reação de A deve ocorrer depois da reação de B

1. Fluxo de Dados Estático
2. Fluxo de Dados Dinâmico
3. Fluxo de Dados Estruturado
4. Redes de Processamento

Tóken: mensagem/mensagem que fornece comunicação entre os atores — $s: N \rightarrow V_A$

Função do Ator: mapeia sequências inteiros de I para sequências inteiros de O

Função de Disparo: mapeia uma porção finita de sequências de I para O

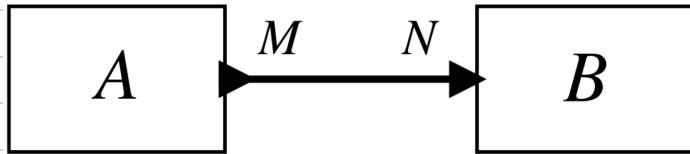
Regra de Disparo: especifica o número de tókens necessários em cada porto de input para disparar o ator

Atores de Ataharó: $D(x_1, x_2, \dots) = D(d, x_1, x_2, \dots)$ $d_1(s) = (d)$ $d_2(x_1, \dots) = (x_1)$

Como o disparo dos atores é assíncrono, um token enviado de um ator para outro deve ser "buffered", isto é, guardado até que o ator de destino esteja pronto para o consumir, sendo depois descartado. ↴ Como limitar memória?

Execução Ilimitada: modelo que executa para sempre ? INDECIDÍVEL
"Deadlock": ciclo dirigido com tokens insuficientes para disparar.

Fluxo de Dados Estático: para cada ator, qualquer disparo consome um número fixo de tokens de input em cada porto de input e produz um número fixo de tokens de output em cada porto de output - não há relógio global

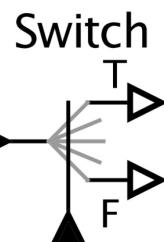
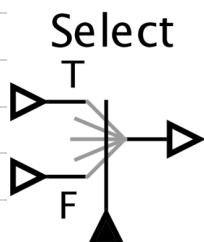


A produz M tokens ao disparar
 B consome N tokens ao disparar
 $q_A M = q_B N$

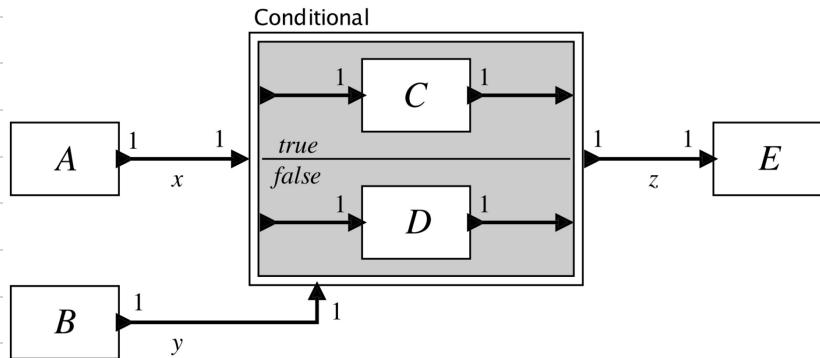
Fluxo de Dados Dinâmico: cada ator tem múltiplas regras de disparo e não estão restritas a produzir o mesmo número de tokens de output em cada disparo

Seleção: requer um token (T/F) no porto de input inferior, com o qual ativa a próxima regra de disparo, nem produzir output

Troca: requer um token em ambos os portos de input e o token da esquerda é enviado para T ou F conforme o valor do token inferior



Fluxo de Dados Estruturado: um ator de alto-nível é um ator que tem um ou mais modelos como parâmetros — Condicional é parametrizado por dois sub-modelos, um contendo o ator C e outro contendo o ator D, que, quando dispara, consome um token de cada porto de input e produz um token no porto de output — o output depende do valor de input



Regras de Processamento: cada ator executa concorrentemente no seu próprio processo e todos os atores executam simultaneamente — um ator é definido por um programa (tipicamente não-terminante) que lê tokens de dados a partir de portos de input de forma bloqueante e escreve tokens de dados para portos de output

- As leituras são bloqueantes!

- As escritas não são bloqueantes — sucede e retorna imediatamente

Modelo Temporizado: introduz mecanismos para divisão periodicamente computações distribuídas de acordo com um relógio distribuído que mede a passagem do tempo para formar uma arquitetura ativada/guidada pelo tempo que pode tolerar alguns tipos de falhas não disruptivas

- Há um relógio global que coordena a computação

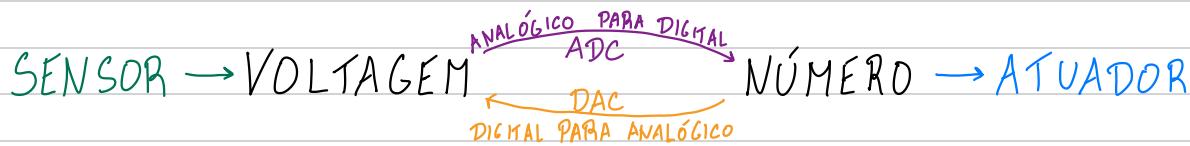
- As computações levam tempo, não são simultâneas/intantâneas

- Não há interações entre as computações entre tiques

Sensores & Atuadores

Sensor: dispositivo que mede uma quantidade física

Atuador: dispositivo que altera uma quantidade física



Sensor Digital: com ADC - precisão limitada pelos bits da representação

Atuador Digital: com DAC - precisão limitada pelos bits da representação

Sensores/Atuadores podem ter microprocessadores e interfaces de rede

Não-Linearidade: grau com que se desvia de uma medição proporcional

Alcance: intervalo da variável física para o qual é linear

Ruído: quantidade de variação aleatória introduzida pelo processo de medição

Quantização: introdução de erros pela conversão de sinal analógico para digital

Alcance Dinâmico: quanto bem se reproduz um sinal analógico no seu alcance

O comportamento dos sensores pode ser modelado por uma função afim
MAS $f(x(t)) = ax(t) + b$

Os valores de a e b têm de ser calibrados e o alcance é limitado!

$$f(x(t)) = \begin{cases} ax(t) + b, & \text{se } L \leq x(t) \leq H \\ aH + b, & \text{se } x(t) > H \\ aL + b, & \text{se } x(t) < L \end{cases}$$

$$D = H - L \quad D_{dB} = 20 \log_{10}(D)$$
$$\rho = (H - L) / 2^n$$

Precisão: menor diferença absoluta entre dois valores de uma quantidade física para a qual as leituras do sensor são distinguíveis

• Numa sensor digital ideal, duas quantidades físicas que diferem em precisão P devem ser representadas por quantidades digitais que diferem em 1 bit!

Rácio Sinal-Ruído: qualidade da medição do sensor $SNR_{dB} = 20 \log_{10}(X/N)$

Amostragem: a quantidade física é amostrada em pontos particulares no tempo para criar um sinal discreto

Amostragem Uniforme: intervalo de tempo fixo T entre amostras Rácio: $1/T$

• Há muitas funções distintas que amostradas retornam o mesmo sinal!

Teorema de Amostragem Nyquist-Shannon: para um sinal em que a variação esperada mais rápida ocorre com frequência $R/2$, então amostrar o sinal com rádio de pelo menos R vai resultar em amostras que representam unicamente o sinal

EXEMPLOS

Acelerômetro: mede aceleração - pode ser implementado como circuitos integrados

GPS: sistema de navegação sofisticado baseado em satélite que usa triangulação - os satélites GPS transmitem o tempo atual e a sua localização

Giroscópio: dispositivo usado para medir ou manter orientação e velocidade angular - não afetado pela gravidade

Microfone: mede alterações na pressão do som - indução; capacância; cristal;...

LED: guiadas diretamente pelos pinos digitais de I/O

Motor: aplica uma força angular - Modulação Pulso-Largura

MicroProcessadores

Não existe arquitetura dominante em sistemas embutidos!

Arquitetura do Conjunto de Instruções (ISA): definição de instruções que o processador pode executar e certas restrições estruturais

Processador/Chip: pedaço de silício vendido por um vendedor de semicondutores

Uma ISA é uma abstração partilhada por muitas realizações!

MicroControlador: computador pequeno com um único circuito integrado que consiste em Unidade de Processamento Central (CPU) relativamente simples com dispositivos periféricos como memórias, I/O e temporizadores

Application Specific Integrated Circuit (ASIC): circuito integrado produzido para um uso específico - alto desempenho, mas inflexível

Field Programmable Gate Array (FPGA): circuito integrado análogo a uma "breadboard" - flexível e barato, mas não tem bom desempenho

Digital Signal Processor (DSP): processador projetado especificamente para importar aplicações de processamento de sinal numericamente intensivas

DESEMPENHO

1. Buffer Circular: caso se saiba o número de valores envolvidos na operação
2. Paralelismo ao Nível das Instruções
3. Máquinas "Multicore" Heterogêneas: combinam uma variedade de tipos de ARQUITETURAS MULTICORE processadores num único chip
4. Aritmética de Ponto Fixo: imaginar um ponto binário que separa bits em vez de números

M emória

O sistema de memória tem mais impacto no desempenho que o processamento!

↓ PORQUE

1. variedade de tecnologias
2. hierarquia de memória
3. espaço de endereçamento

Memória Flash: memória não-volátil com tempos de leitura razoavelmente rápidos (não tanto como RAM), mas escrita lenta e limitada

Memória NOR: tempos de escrita e leitura mais longos, mas acessível como RAM

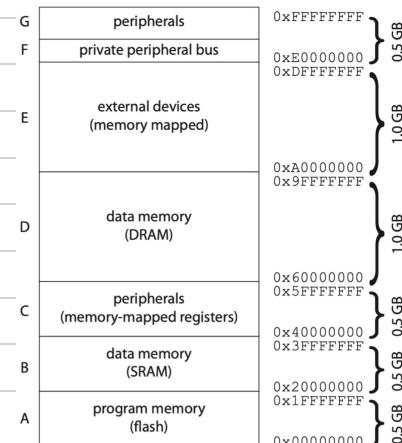
Memória NAND: tempos de eliminação rápidos, mas acessível por blocos

Memória Virtual: o sistema operativo e/ou o hardware traduzem os endereços convertendo endereços lógicos em localizações físicas TLB

Mapa de Memória: mapeia os endereços em hardware - N bits $\rightarrow 2^N$ localizações BYTES

Arquitetura Harvard: espacos de endereços separados para programa e dados

Arquitetura von Neuman: armazena programa e dados na mesma memória



Input / Output

- Os mecanismos de I/O não expositos como pinos do chip

Princípios de I/O de Ponto Geral: permitem ao software ler ou escrever níveis de voltagem representando zero ou um

Lógica "Active High": $V = V_{fornecida} \rightarrow 1$ $V = 0 \rightarrow 0$
Lógica "Active Low": $V = V_{fornecida} \rightarrow 0$ $V = 0 \rightarrow 1$

- O software consegue controlar dispositivos físicos externos, com cuidado

Input: software pode reagir a níveis de voltagem fornecidos extensamente

Output: software pode escrever num registo para ver a voltagem (alta ou baixa)

- Os pinos de input "Schmitt-triggered" têm histerese — menos vulneráveis a ruído
- Os vários dispositivos podem partilhar uma única conexão elétrica — cuidado!

Circuito Coletor Aberto: escrever 1 → ativa transistores → $V = 0$

As interfaces do coletor aberto podem estar conectadas a um "pull-up resistor", que traz a voltagem da linha para V_{DD} quando todos os transistores estão desligados — se um transistor se ligar, então $V = 0$

As interfaces em série são mais comuns do que paralelas por simplicidade!

Interrupção: mecanismo para pausar a execução do que o processador está a fazer e executar uma sequência pré-definida de código — rotina que lida

- Hardware:** hardware externo altera o nível de voltagem na linha de interrupção
- Software:** o programa executa uma instrução especial (armadilha)
- Exceção:** hardware interno detecta uma falha — programa não recupera

Agendamento

Agendador: decide o que fazer a seguir quando um processador fica disponível

Sistema em Tempo Real: coleção de tarefas com (1) restrições de ordem impostas pela precedência entre as tarefas e (2) restrições temporais (prazos, ...)

Um agendador "multicore" tem de decidir que tarefa e onde a executar

ATRIBUIÇÃO DO CORE

- Decisões
- 1. Atribuição: que núcleo deve executar a tarefa
 - 2. Ordenação: em que ordem cada núcleo executa as suas tarefas
 - 3. Temporização: o tempo em que cada tarefa executa

Agendador Totalmente Estático: decide tudo em tempo de compilação — especificação precisa nem máfices nem "locks", mas difícil de concretizar pela volatilidade

Agendador de Ordem Estática: atribuição e ordenação em tempo de compilação, mas temporizações em tempo de execução — considera precedências e "locks"

Agendador de Atribuição Estática: atribuição em tempo de compilação e ordenação e temporizações em tempo de execução — o que executar a seguir?

Agendador Totalmente Dinâmico: decide tudo em tempo de execução

Agendador Preemptivo: pode tomar uma decisão de agendamento durante a execução de uma tarefa, interrompendo-a para começar outra tarefa — preempção

Agendador Não-Preemptivo: nunca interrompe uma tarefa, deixa sempre terminar

Modelo de Tarefa: conjunto de assumções sobre as tarefas

- O conjunto de tarefas pode ser finito
- As tarefas podem ser conhecidas previamente ou chegar durante a execução
- As tarefas podem executar uma vez, esporadicamente ou periodicamente
- As tarefas podem ter restrições de precedência - $t_i < t_x$
- As tarefas podem ter pré-condições

Tempo de Libertação / Chegada: tempo mais cedo em que a tarefa é permitida - r_i

Tempo de Início: tempo em que a execução começa - $s_i \geq r_i$

Tempo de Fim: tempo em que a tarefa completa a execução - $f_i \geq s_i \geq r_i$

Tempo de Resposta: tempo entre libertação e conclusão - $\theta_i = f_i - r_i$

Tempo de Execução: tempo que a tarefa lembra a executar

Prazo: tempo em que uma tarefa deve estar completa

Prazo Hard: restrição física imposta pela aplicação - falhar é ruim

Prazo Soft: decisão de design não restrito - falhar não é ruim

Agendador Baseado em Prioridade: assume que é atribuída uma prioridade numérica a cada tarefa e escolhe sempre a tarefa com prioridade mais alta

Prioridade Fixa: permanece constante durante todos os execuções da tarefa

Prioridade Dinâmico: pode mudar durante a execução

Agendador Preemptivo Baseado em Prioridades: suporta a chegada de tarefas a qualquer altura e executa sempre a tarefa com maior prioridade

Agendador Não-Preemptivo Baseado em Prioridade: usa as prioridades para determinar a próxima tarefa a executar, mas nunca interrompe a tarefa em execução

Agenda Viável: todas as execuções cumprem os prazos - $f_i \leq d_i$

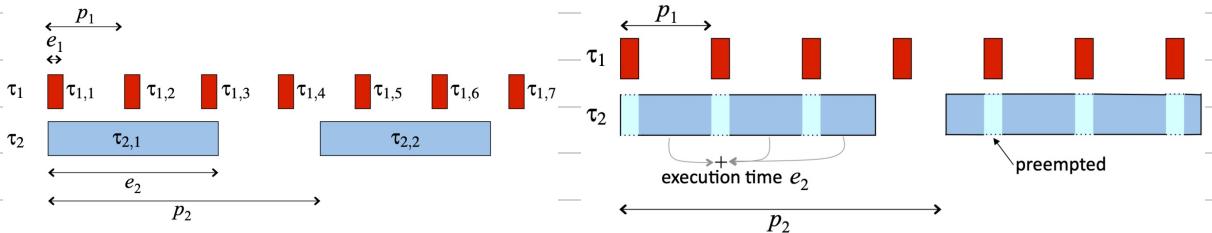
Agendador Ótimo: retorna uma agenda viável para todas as tarefas

Utilização: percentagem de tempo que o processador passa a executar tarefas

Latência Máxima: $L_{\max} = \max(f_i - d_i), t_i \in T$

Tempo Total: $M = \max(f_i) - \min(r_i), t_i \in T$

Agendamento de Rácio Monotônico: dá maior prioridade a tarefas com menores períodos — o prazo da k -ésima execução de t_i é $r_{i,1} + k p_i$ **PREENCITIVO**



O temporizador deve ser definido para o máximo divisor comum dos períodos

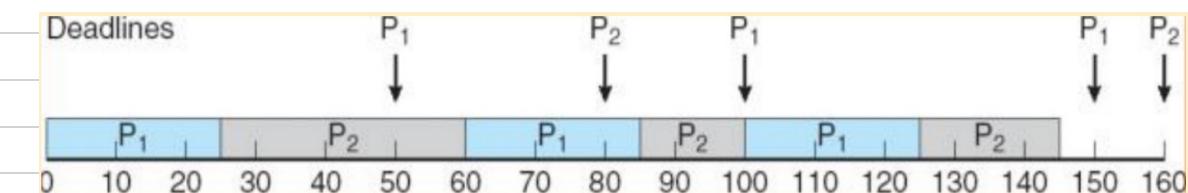
Utilização: $\mu = \sum_{i=1}^n \frac{r_i}{p_i}$

Se $\mu > 1$, então não é viável!

Se $\mu \leq n(2^{1/n} - 1)$, então é viável!

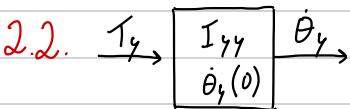
Prazo Mais Cedo Primeiro: executa as tarefas na mesma ordem que os seus prazos — a tarefa com o prazo mais cedo executa primeiro **DINÂMICO**

Teorema: dado um conjunto finito de tarefas não repetitivas $T = \{t_1, \dots, t_n\}$ com prazos associados d_1, \dots, d_n e sem restrições de precedência, então **Early Due Date** é ótimo no sentido em que minimiza a latência máxima, mas não suporta a chegada de tarefas nem execução periódica ou repetida



EXEMPLOS

2.1. $\ddot{\theta}_y(t) = T_y(t)/I_{yy} \Rightarrow \dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$



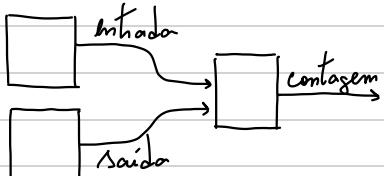
2.3.
$$\begin{cases} y_1(t) = a x_1(t) \\ y_2(t) = x_2 + \int_0^t x_2(\tau) d\tau \end{cases}$$

2.4. INSTÁVEL

2.5. $\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau = \dot{\theta}_y(0) + \frac{K}{I_{yy}} \int_0^t (\psi(\tau) - \dot{\theta}_y(\tau)) d\tau$

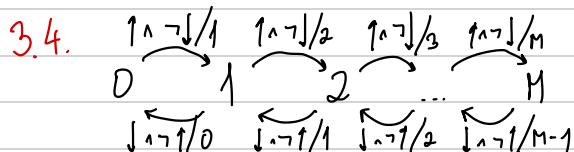
$\rightarrow e(t) = \psi(t) - \dot{\theta}_y(t)$
 $\rightarrow T_y(t) = K e(t)$

3.1.



3.2. $P = \{up, down\}$ $V_{up} = V_{down} = \{present\}$

3.3. $\emptyset = \{count\}$ $V_{count} = \mathbb{Z}$





HISTERESE

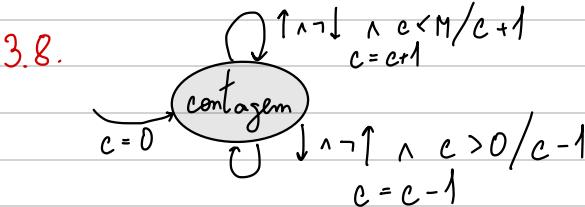
3.7. Estados = {0, 1, ..., M}

Inputs = {up, down} → {presente, ausente}

Outputs = {count} → {0, 1, ..., M, ausente}

Estado Inicial = 0

$$\text{atualização}(\lambda, i) = \begin{cases} (\lambda + 1, \lambda + 1), & \lambda < M \wedge i(\text{up}) = \text{presente} \wedge i(\text{down}) = \text{ausente} \\ (\lambda - 1, \lambda - 1), & \lambda > 0 \wedge i(\text{up}) = \text{ausente} \wedge i(\text{down}) = \text{presente} \\ (\lambda, \text{ausente}), & \text{else} \end{cases}$$



3.10. |ESTADOS| = n_p^m n=1 m=1 p=M+1 |ESTADOS| = M+1

3.11. ↑ ↗ ↓ → FSM

3.13. pedestre é final paro

3.15. Estados = {none, walking, crossing}

Inputs = {rig L, rig Y, rig R} → {presente, ausente}

Outputs = {pedestre} → {presente, ausente}

Estado Inicial = {crossing}

$$\text{possible Updaters}(\lambda, i) = \begin{cases} \{(none, ausente)\}, & \lambda = \text{crossing} \wedge i(\text{rig L}) = \text{presente} \\ \{(none, ausente), (waiting, presente)\}, & \lambda = \text{zone} \\ \{(crossing, ausente)\}, & \lambda = \text{waiting} \wedge i(\text{rig R}) = \text{presente} \\ \{(\lambda, ausente)\} & \text{else} \end{cases}$$

3.16.

$$S_{\text{up}} = \{P, A, P, A, P\}$$

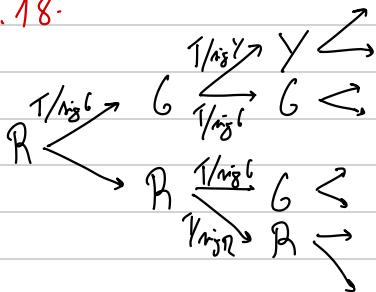
$$S_{\text{down}} = \{P, A, A, P, A\}$$

$$S_{\text{count}} = \{A, A, 1, 0, 1\}$$

3.17.

$$0 \xrightarrow{\uparrow \downarrow /} 0 \xrightarrow{/} 0 \xrightarrow{\uparrow / 1} 1 \xrightarrow{\downarrow / 0} 0 \xrightarrow{\uparrow / 1} 0$$

3.18.



EXAME

1

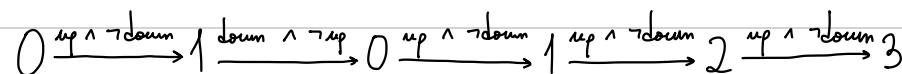
- a) ADC: Analog to Digital Converter - converte sinal analógico para digital
- b) FIR: Finite Impulse Response - filtro digital para processamento de sinais digitais
- c) DSP: Digital Signal Processor - processa sinal digital com operações matemáticas
- d) JTAG: Joint Test Action Group - padrão de interface de teste e debugging
- e) PWM: Pulse Width Modulation - técnica para modulação de sinal

2. $a = \frac{1}{I_{yy}} \quad i = \hat{\theta}_y(0)$

3. $\text{update}(\lambda, i) = \begin{cases} (\lambda+1, \lambda+1), & \text{se } \lambda < M \wedge i(\text{up}) = \text{PRESENTE} \wedge i(\text{down}) = \text{AUSENTE} \\ (\lambda-1, \lambda-1), & \text{se } \lambda > 0 \wedge i(\text{up}) = \text{AUSENTE} \wedge i(\text{down}) = \text{PRESENTE} \\ (\lambda, \text{AUSENTE}), & \text{caso contrário} \end{cases}$

4.

$$\begin{aligned} \text{up} &= (\text{true}, \text{false}, \text{true}, \text{true}, \text{true}, \dots) \\ \text{down} &= (\text{false}, \text{true}, \text{false}, \text{false}, \text{false}, \dots) \\ \text{output} &= (1, 0, 1, 2, 3, \dots) \end{aligned}$$



5.

$$\text{update}(\lambda, i) = \begin{cases} ((\lambda_1, \lambda_3), a), & \text{se } \lambda = (\lambda_2, \lambda_4) \\ ((\lambda_2, \lambda_4), b), & \text{se } \lambda = (\lambda_1, \lambda_3) \end{cases}$$

Os estados (λ_1, λ_4) e (λ_2, λ_3) não alcançáveis.

A composição é determinística porque em cada estado, para quaisquer valores de input, exatamente uma transição (possivelmente implícita) é ativada.

6. Dizer que um estado tem um refinamento significa que o comportamento do estado pode ser modelado de forma mais detalhada, mas é abstrato para a máquina de estados. A composição em causa é hierárquica. Este tipo de composição simplifica o design e o desenvolvimento de sistemas embutidos ao providenciar, por um lado, a abstração suficiente para alto-nível e, por outro lado, o detalhe necessário para baixo nível, de forma simultânea.

7. No estado s1, existem dois pontos fixos: $s(n) = \text{ausente}$ e $s(n) = \text{presente}$.
 No estado s2, existe um ponto fixo: $s(n) = \text{presente}$.
 Um ponto fixo ocorre quando o input e o output coincidem ($x=y$).

8. A frequência mínima de amostragem para reconstrução dainal deve ser 2ω , pelo Teorema de Amostragem, que afirma que, para um sinal ω que a variação esperada mais rápida ocorre com frequência $R/2$, então amostrar o sinal com rácio de pelo menos R vai resultar em amostras que representam unicamente o sinal.

9. O problema principal associado com o uso de "dead reckoning" é com acelerômetros xx-yy-zz para calcular a posição de um dispositivo é o "drift", isto é, o erro inerente acumulado ao estimar posições subsequentes a partir da posição inicial. Um componente de hardware que pode calcular a posição e orientação de forma precisa usando, em parte acelerômetros, é uma Unidade de Medição Inercial (IMU), que também usa um giroscópio.

$$10. \mu = \frac{1}{4} + \frac{2}{6} + \frac{1}{8} = \frac{3}{8} + \frac{1}{3} = \frac{17}{24} \approx 0,71$$

$$\mu_{\text{LIMRE}} = n(2^{\frac{1}{m}} - 1) = 3(2^{\frac{1}{2}} - 1) \approx 0,78 \quad 0,71 < 0,78$$

(t=0) A, B, B, C, (t=4) A, X, (t=6) B, B, (t=8) A, C, X, X (t=12)

Sistema Embutido: dispositivo computacional de propósito específico

ASIC: Application Specific Integrated Circuit

FPGA: Field-Programmable Gate Array

ASIP: Application Specific Instruction-Set Process



Princípio de Kepet: as propriedades não são do sistema, mas sim do modelo

Estabilidade BIBO: se o input é limitado, então o output é limitado

Causalidade: o output atual só depende do input atual e passados

Causalidade Estrita: o output atual só depende dos inputs passados

Sem Memória: o output atual só depende do input atual

$$\text{update}(s, i) = \begin{cases} (s+1, s+1), & \text{se } s < M \wedge i(\text{up}) = \text{presente} \wedge i(\text{down}) = \text{ausente} \\ (s-1, s-1), & \text{se } s > 0 \wedge i(\text{up}) = \text{ausente} \wedge i(\text{down}) = \text{presente} \\ (\lambda, \text{ausente}), & \text{caso contrário} \end{cases}$$

$$\text{update}(s, i) = \begin{cases} (\text{aquecimento}, \text{presente}, \text{ausente}), & \text{se } s = \text{arrefecimento} \wedge i(\text{temperatura}) \leq 18 \\ (\text{arrefecimento}, \text{ausente}, \text{presente}), & \text{se } s = \text{aquecimento} \wedge i(\text{temperatura}) \geq 22 \\ (\lambda, \text{ausente}, \text{ausente}), & \text{caso contrário} \end{cases}$$

$$\text{possibleUpdates}(s, i) = \begin{cases} \{(\text{none}, \text{ausente})\}, & \text{se } s = \text{aquecimento} \wedge i(\text{rig C}) = \text{presente} \\ \{(\text{none}, \text{ausente}), (\text{espera}, \text{presente})\}, & \text{se } s = \text{none} \\ \{(\text{cryanamento}, \text{ausente})\}, & \text{se } s = \text{espera} \wedge i(\text{rig R}) = \text{presente} \\ \{(\lambda, \text{ausente})\}, & \text{caso contrário} \end{cases}$$

Ponto Fixo: $s(n) = f(s(n)) - \text{INPUT} = \text{OUTPUT} - \text{bem-formado} \hookrightarrow \text{nº 1}$

Fluxo de Dados: $q_A M = q_B N - A \xrightarrow{M} B$

$$f(x(t)) = \begin{cases} a \cdot x(t) + b, & \text{se } L \leq x(t) \leq H \\ aH + b, & \text{se } x(t) > H \\ aL + b, & \text{se } x(t) < L \end{cases}$$

$$D = \frac{H-L}{P}$$

$$P = \frac{H-L}{2^n}$$

$$\text{SNR}_{dB} = 20 \log \left(\frac{X}{N} \right)$$

Teorema de Amostragem: amostrar com $2f$

Rate Monotonic Scheduling: menor período primo

Earliest Deadline First: menor prazo primo

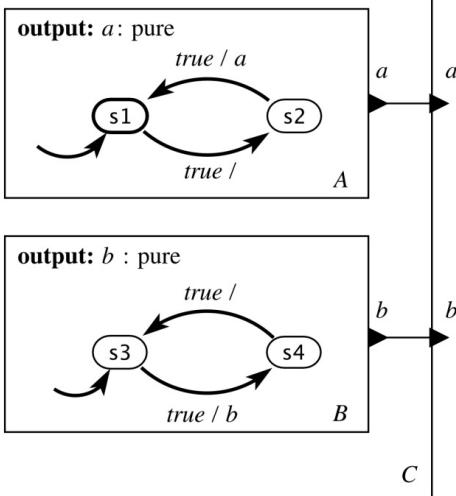
$$\mu = \sum_{i=1}^n \frac{L_i}{P_i}$$

$$\mu \leq n(2^{1/m} - 1)$$

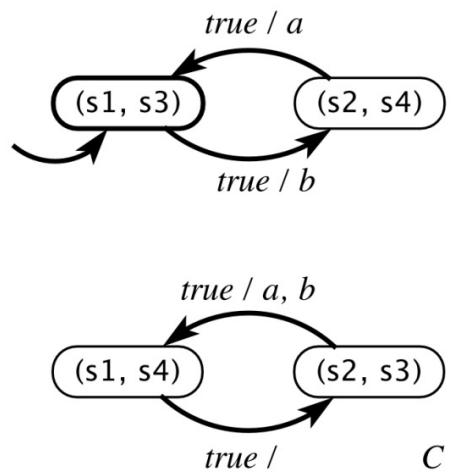
COMPOSIÇÃO

SÍNCRONA

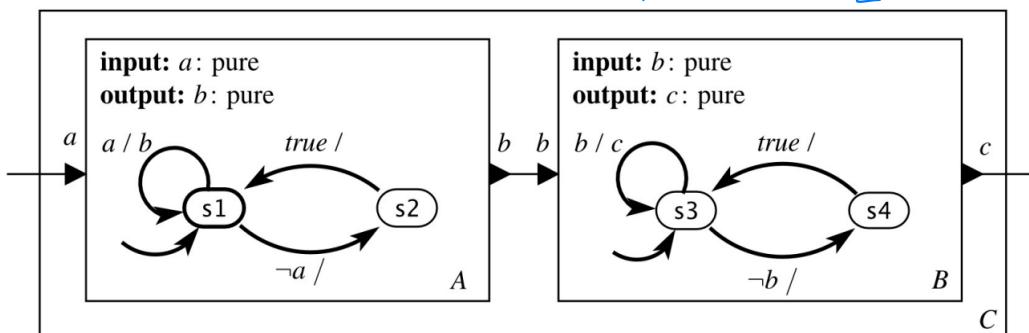
outputs: a, b : pure



outputs: a, b : pure



COMPOSIÇÃO SÉQUENCIAL



input: a : pure
output: c : pure

