Especificação, Modelação e Projecto de Sistemas Embutidos

Linguagens de especificação: StateCharts

Paulo Pedreiras, Luís Almeida

{pbrp,lda}@ua.pt

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

V1.0 Setembro/2008

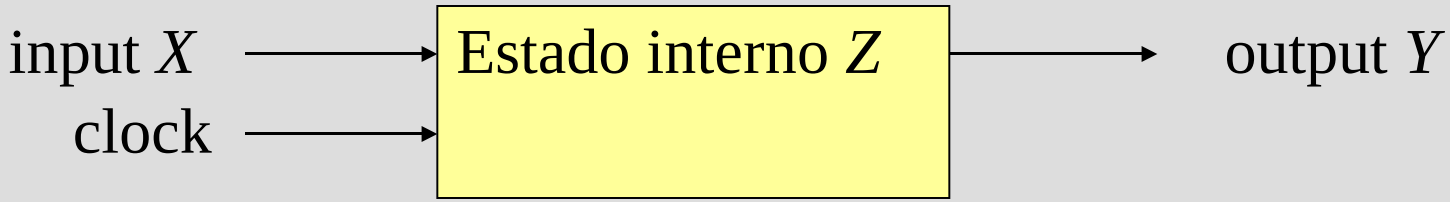
StateCharts

StateCharts: características gerais

- Modelo baseado em *Communicating Finite State Machines*
- Apropriado para modelar **sistemas reactivos** e **complexos**
  - Suporte a composição **hierárquica**
- Exemplo (proeminente) de modelo de computação baseado em ***Shared Memory***
- Apropriado para "**sistemas locais**" (i.e., centralizados, não distribuídos)

StateCharts: recapitulação da noção de automata (clássico)

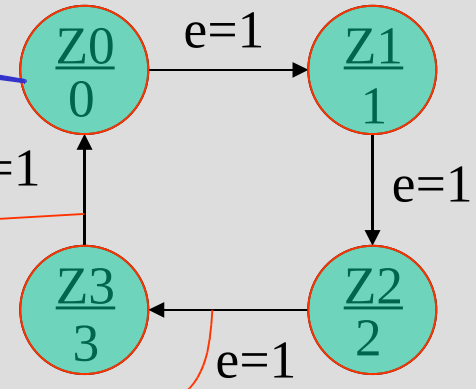
Automata (clássico):



- Estado seguinte (Z^+) calculado por uma função δ
- Output calculado por uma função λ

Moore- + Mealy automata = finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda(Z)$; $Z^+ = \delta(X, Z)$
- Mealy-automata
  $Y = \lambda(X, Z)$; $Z^+ = \delta(X, Z)$



StateCharts

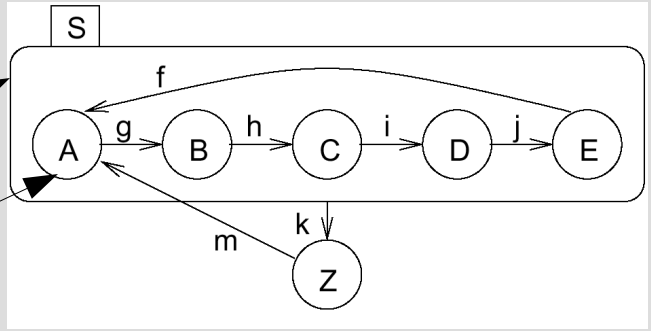
- **Porquê StateCharts?**
  - Automata clássicos **não** são **capazes** de modelar **sistemas reactivos complexos**
    - Gráficos tendem a ficar **complicados** e **ininteligíveis** para os seres humanos
  - Introduzidos por D. Harel
    - Harel, D. “StateCharts: A visual formalism for complex systems”. Science of Computer Programming, pp. 231-274.
  - StateChart = a única **combinação não usada** de *„flow“ ou „state“ com „diagram“ ou „chart“*

StateCharts: composição hierarquica

–Os StateCharts descrevem FSMs extendidas

- A extensão chave é a **hierarquia**

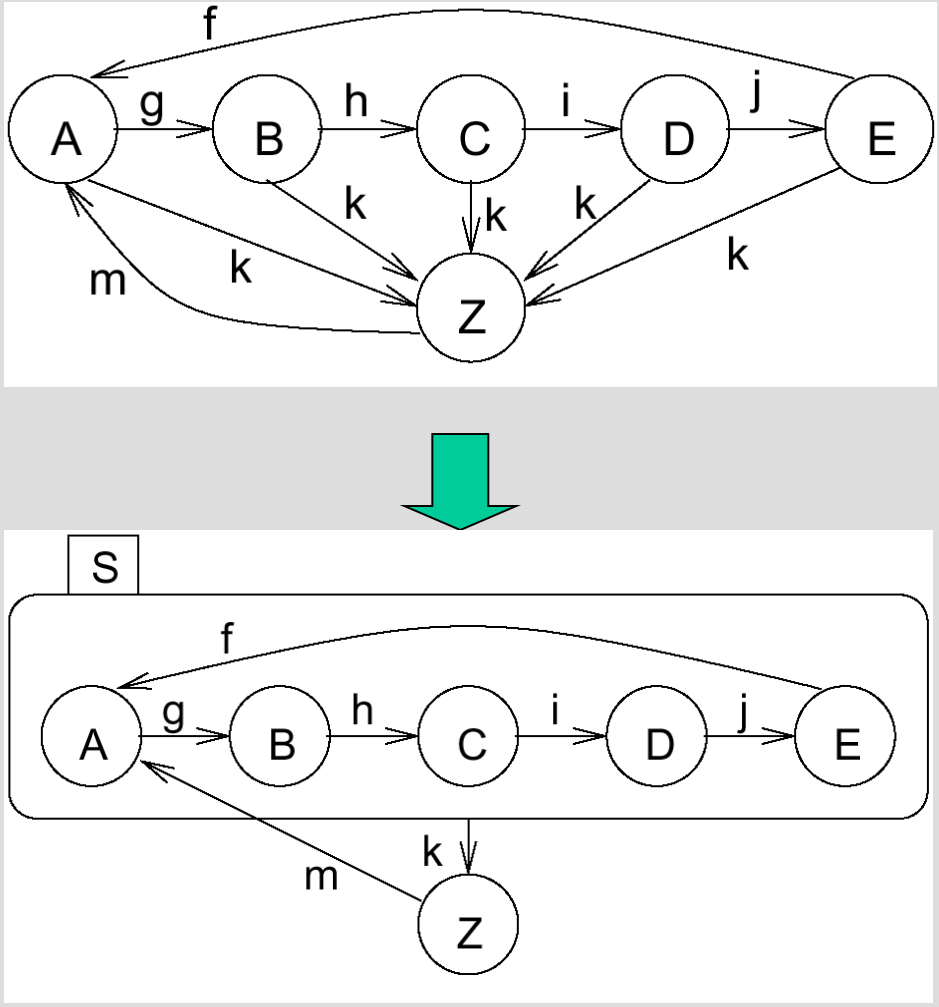
–A hierarquia é introduzida por meio de **super-estados**



–Algumas definições:

- Estados que são **compostos** por outros estados são denominados **super-estados**
- **Estados incluídos** em outros estados são denominados **sub-estados**
- Um estado que **não é composto** por outros estados é denominado **estado básico**
- Para cada **estado básico** *s*, os super-estados que o **contêm** denominam-se **estados ancestrais** (*ancestor states*)
- O estado em que em que uma FSM se **encontra** num dado instante denomina-se por **estado activo**

Super-estados (X)OR



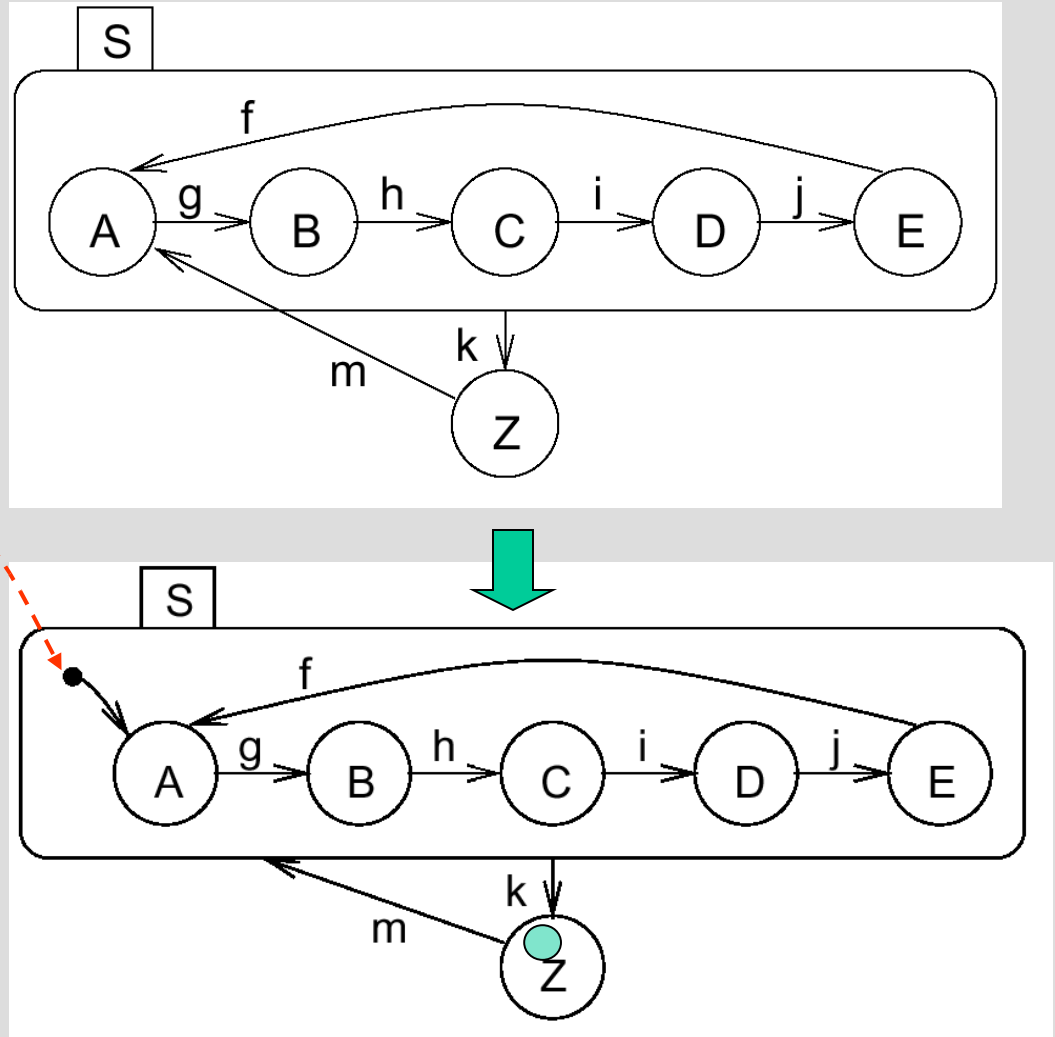
Super-estados (X)Or

• Num super-estado S de tipo Or a FSM estará **num e num único sub-estado** de S sempre que S for o estado **activo**

Estado por defeito (*default state*)

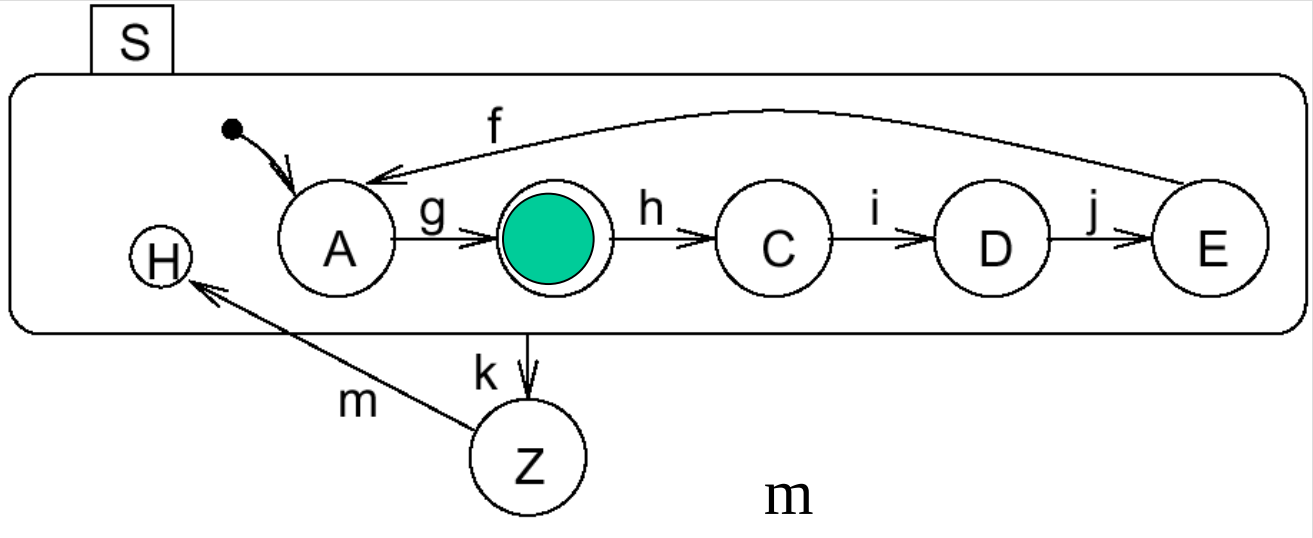
A representação hierárquica pressupõe a capacidade de "esconder" a estrutura interna dos estados

- **Estado por defeito:** (*default state*)
  Representa o sub-estado activado sempre que um super-estado é activado



O *default state* não é um estado per se!!!

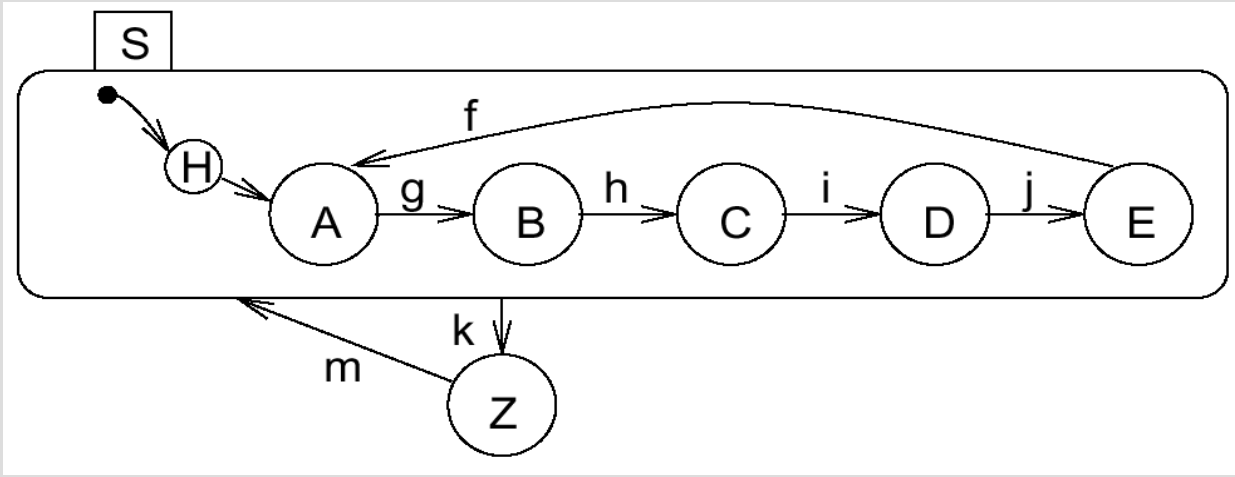
O mecanismo de história



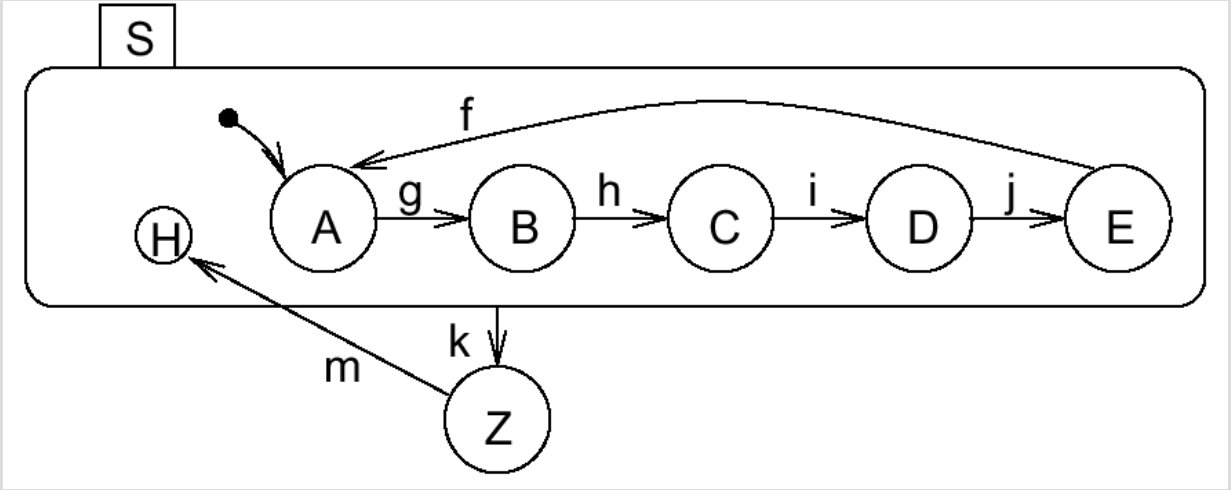
(**Comportamento distinto do slide anterior**)

- Para o **input *m*** S **entra** no sub-estado que **estava activo** imediatamente antes de ter **saído** de S pela **última vez**
  - pode ser qualquer dos estados {A, B, C, D, E}.
- Quando S se torna **activo** pela **primeira vez** aplica-se o mecanismo de *default state*
- Os mecanismo de **historia** e ***default state*** podem ser **compostos** hierarquicamente

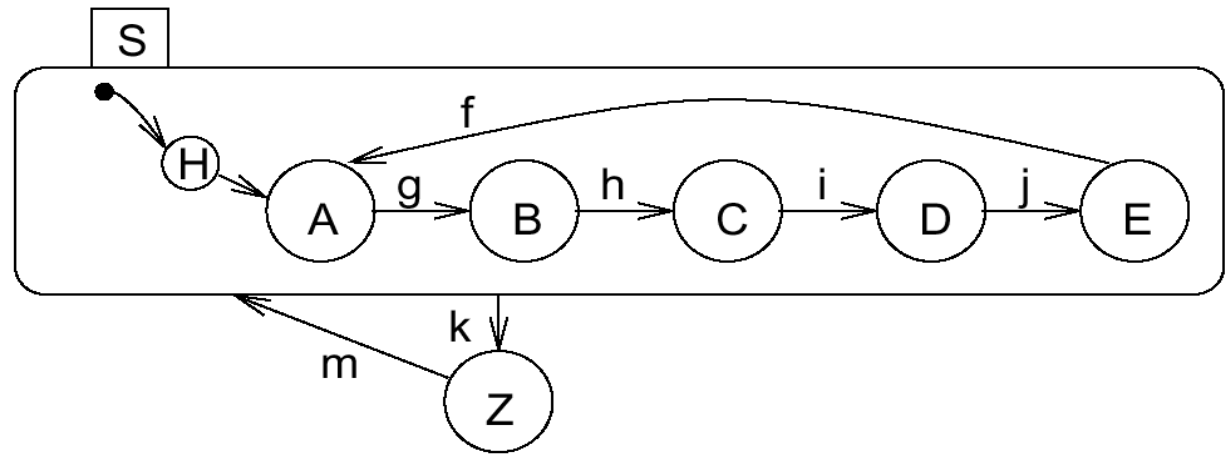
Combinando os mecanismos de história e *default state*



mesmo significado!



Relevância dos mecanismos de história e *default state*

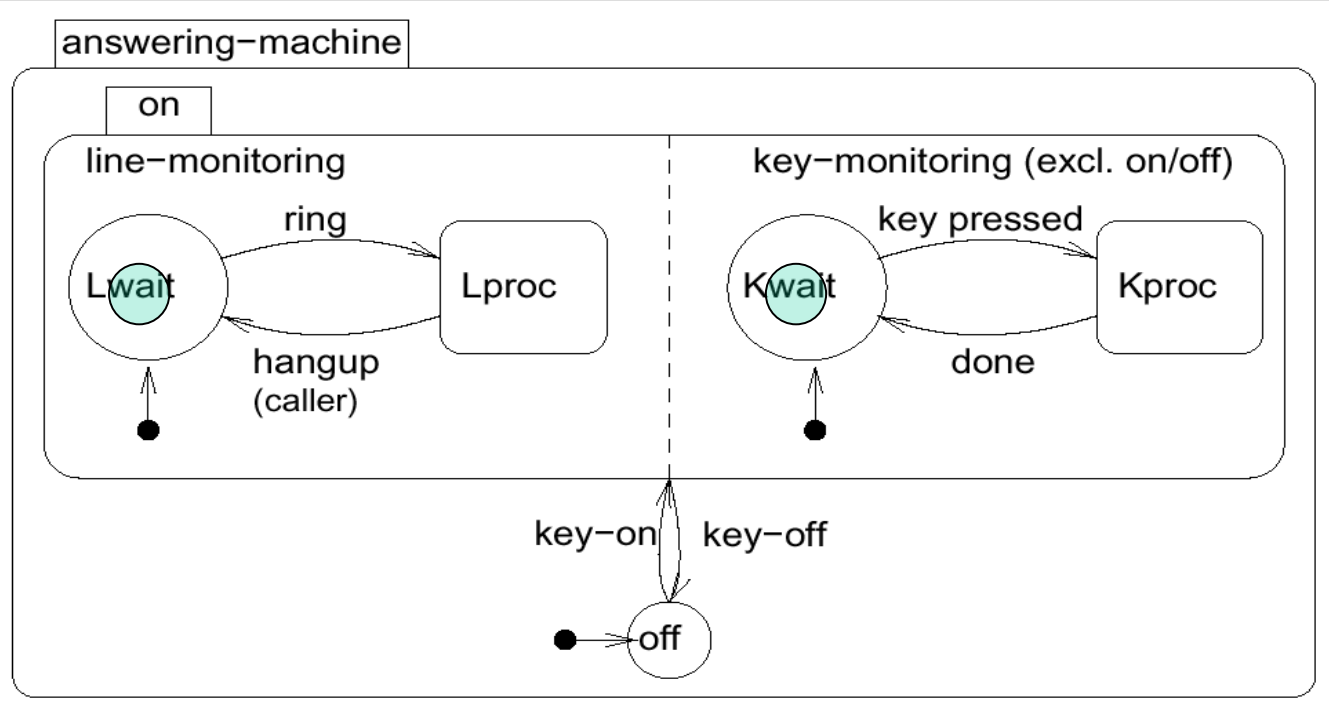


- Estrutura interna do super-estado **não** necessita de **conhecida** no **exterior** (porquê?)
- Modo conveniente para representar **excepções** (porquê?)
- Modo conveniente para representar **chamadas a funções** (porquê?)

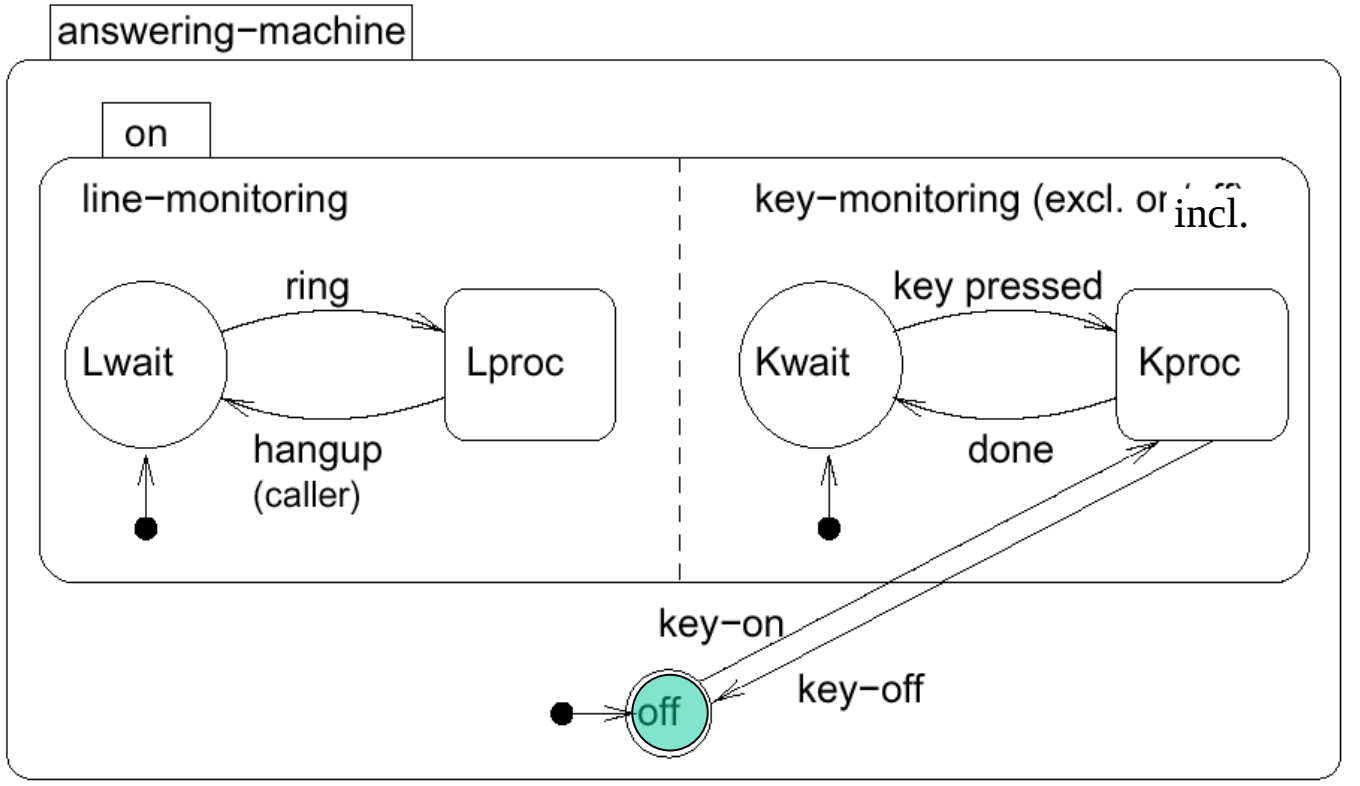
Concorrência

As técnica de especificação devem ser capazes de **descrever** convenientemente situações de **concorrência**.

- **Definição:** Um **super-estado** S é do tipo **AND** se a FSM está em **todos** os seus **sub-estados imediatos**
- Exemplo:



Entrando e saindo de super-estados AND



Os estados *Line-monitoring* e *key-monitoring* são **ambos activados** e **desactivados** sempre que o interruptor de serviço é operado

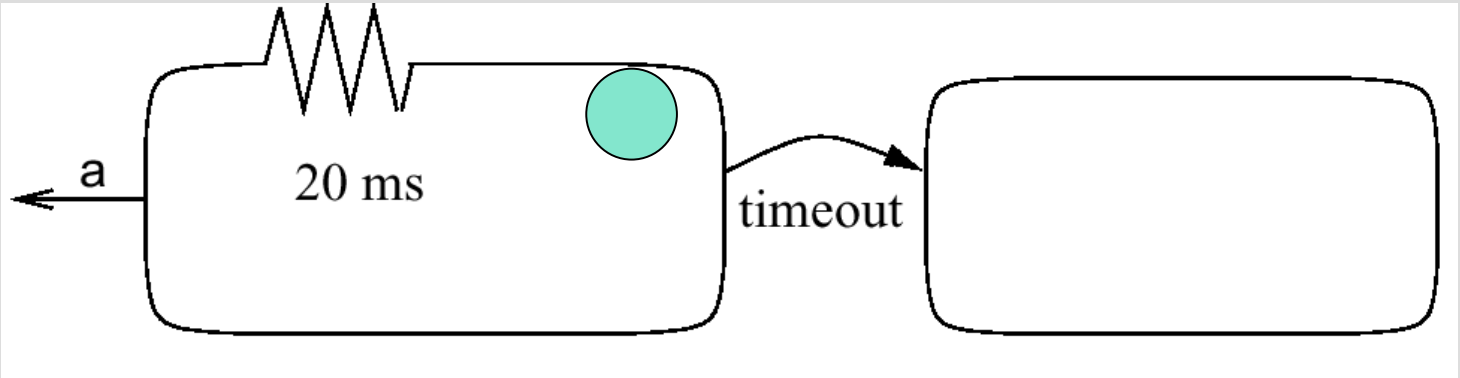
Tipos de estados

Recapitulando, em StateCharts, os estados são:

- **Estados básicos** (*basic states*), ou
- **Super-estados AND** (*AND-super-states*), ou
- **Super-estados OR** (*OR-super-states*).

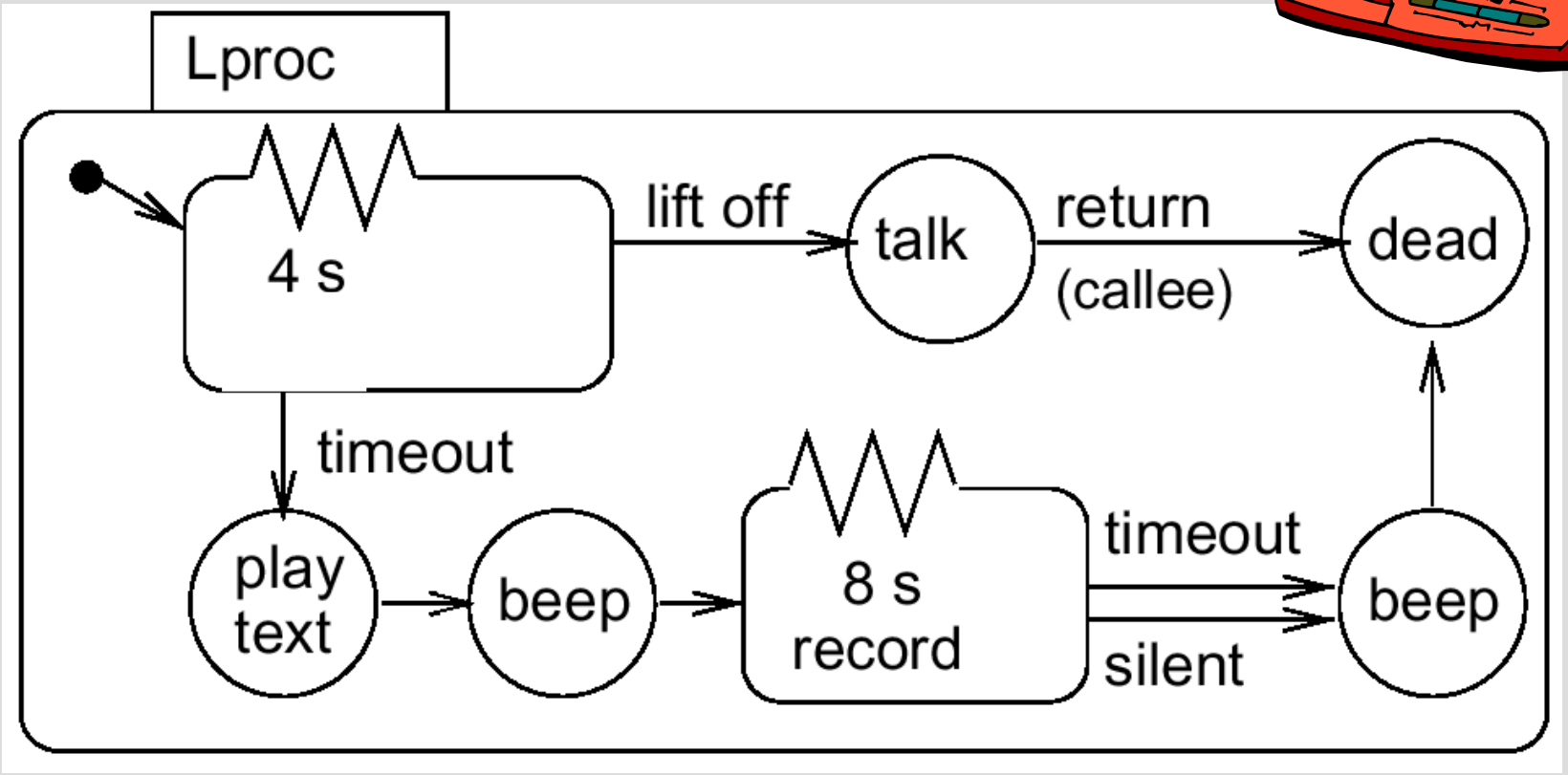
Timers

- Em muitos sistemas embutidos é necessário **modelar** a passagem do tempo
  - e.g. manter o sinal verde durante 1min num semáforo
- Em StateCharts são usadas transições especiais para representar o **expirar** de **intervalos de tempo**.



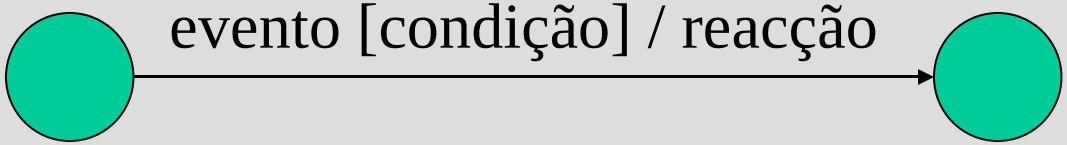
Se **20ms após entrar** no estado da esquerda o evento *a* não acontecer, ocorre um ***timeout*** que activa o estado à direita.

Exemplo: usando *timers* num atendedor de chamadas



- “play text” activado se chamada não atendida durante 4s
- “record” termina após 8s ou se não for detectada comunicação verbal

Forma geral do rótulo das transições



Evento:

- Especifica qual o **evento** que pode **causar** a transição
  - Um evento **existe apenas** até à **próxima execução** do modelo
- Pode ser gerado interna ou externamente

Condição:

- **Testa** o valor de **variáveis** ou o **estado** do sistema
  - Contráriamente aos eventos, as variáveis são permanentes, mantendo sempre o último valor que lhes fois atribuido

Reacção:

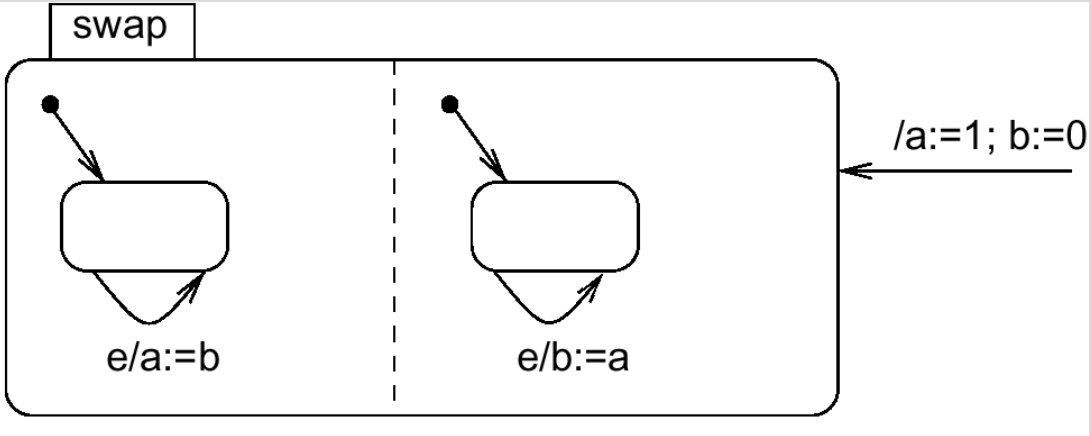
- Consiste na **geração** de **eventos** ou **atribuição** de valores a variáveis

Fases de simulação de StateCharts (StateMate Semantics)

As mudanças de estado são processadas em 3 fases:

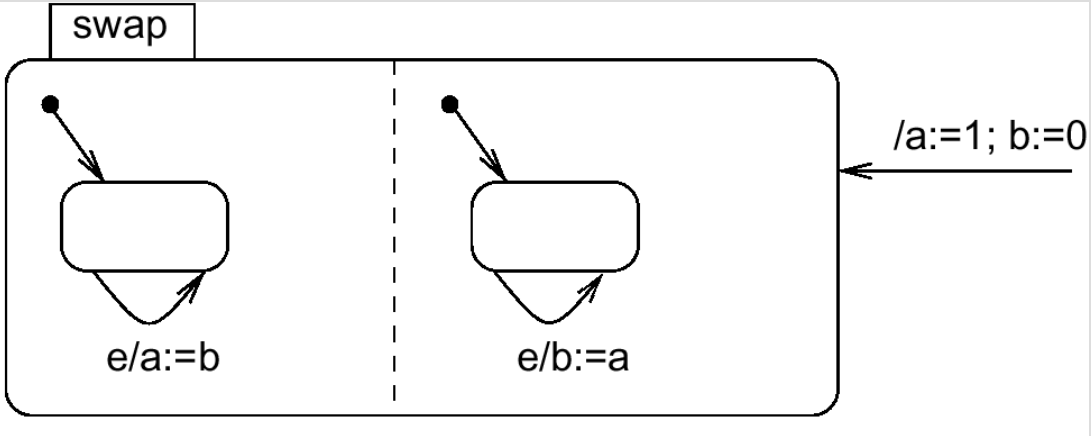
- **Fase 1** – Cálculo do efeito de alterações externas nas condições e eventos
  - Inclui a **avaliação** de **funções** que dependem de **eventos externos**
  - **Não** inclui nenhuma **mudança de estado**
- **Fase 2** – Cálculo do conjunto de **transições** que devem ser efectuadas no passo actual.
  - Os **valores** das **variáveis** são calculados mas são atribuídos a **variáveis temporárias**
- **Fase 3** – **Mudanças de estado** são efectivadas e as variáveis recebem os **novos valores**

Exemplo



- A **separação** entre as fases 2 e 3 garante **determinismo** e **reproducibilidade**.
- Exemplo:
  - na fase 2 as variáveis "*a*" e "*b*" são atribuídas a variáveis temporárias (sejam a' e b').
  - na fase 3 estas são atribuídas a "a" e "b" (b=a'; a=b').
    - O resultado é previsível e determinístico, não dependendo da ordem de execução!

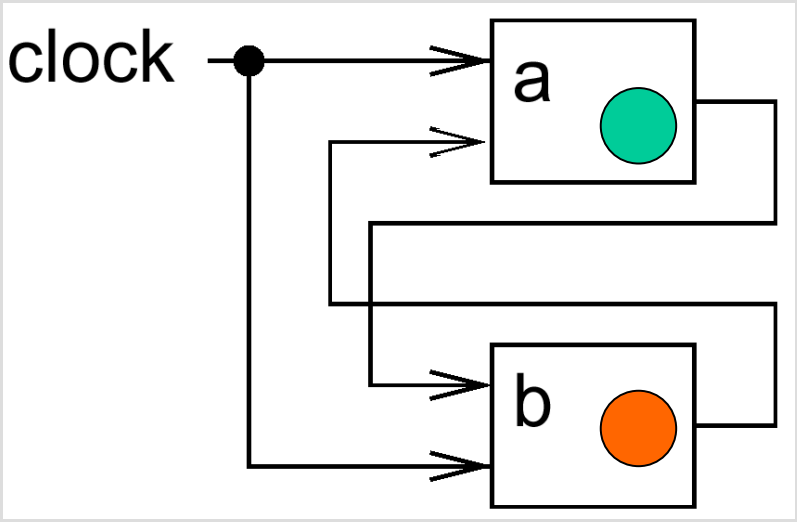
Exemplo (cont.)



Considere-se um ambiente de fase única.

- Qual o resultado se o estado da **esquerda** fosse executado em **primeiro lugar**?
- Qual o resultado se o estado da **direita** fosse executado em **primeiro lugar**?
- Comentários ...

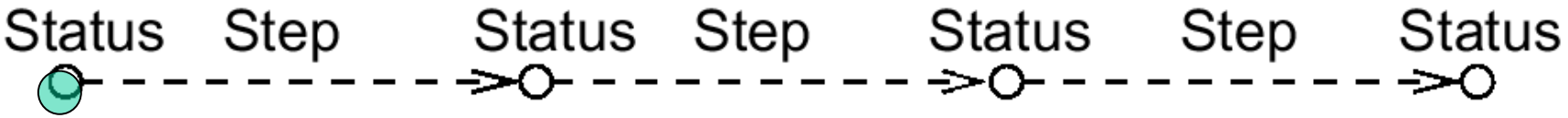
Modelo de hardware síncrono
(clocked hardware)



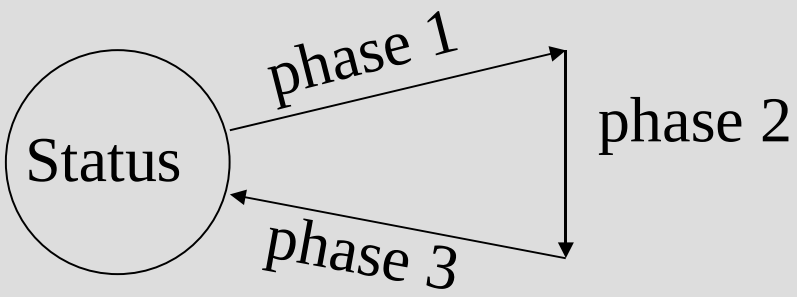
- Em **sistemas síncronos reais** os registos também seriam trocados
- Este esquema de **execução faseada** é encontrado em muitas outras linguagens, com especial incidência nas linguagens que pretendem **modelar hardware**

Sequência de execução

- A execução de um modelo StateMate consiste numa sequência de pares (*status*, *step*)



Status= valor de todas as vars + conjunto de eventos + tempo actual
Step = execução das três fases (semântica StateMate)



Algumas implementações de StateCharts não contemplam a execução em três fases, logo **não são deterministicas**!

Outras semânticas

- Diversas **outras linguagens** de especificação baseadas em máquinas de estados hierárquicas (UML, dave, …) **não contemplam** a execução em 3 fases.
- Estas correspondem mais a uma **abordagem** vista do ponto de vista do software, **sem relógios síncronos**
- O LabVIEW (*) permite activar ou não a execução em 3 fases

(*) Acrónimo de “Laboratory Virtual Instrument Engineering Workbench”. Linguagem de programação gráfica originária da National Instruments, muito utilizada em automação industrial e sistemas de supervisão

Mecanismo de Broadcast

- O valor das variáveis é **visível** para **todos** os **componentes** de um modelo StateChart !
- Os **novos valores** tornam-se **efectivos** na **fase 3** do passo corrente e são conhecidos por todos os componentes do sistema no passo seguinte

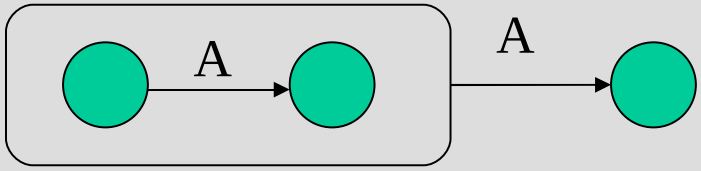
- Os StateCharts assumem implicitamente a existência de um mecanismo de **broadcast** do valor das variáveis
  - uso **implicito** de ***shared memory***
  - ***Implementações alternativas seriam muito ineficientes***
- Em consequência os StateCharts:
  - **são apropriados** para sistemas de controlo **locais** (☺)
  - **não são apropriados** para **aplicações distribuídas.** Efectuar o *update* das variáveis nos diversos nós pode requerer um tempo longo (☹)

Tempo de vida dos eventos

- Os eventos **não são** persistentes
- Os eventos apenas "**vivem**" até ao **passo seguinte** àquele em que são gerados
  - („one shot-events").
- Situação contrastante com as **variáveis**, que armazenam valores de uma forma **permanente**, sendo **alterados** apenas após uma **atribuição explicita**

StateCharts: deterministicos ou não?

- Entende-se por determinismo o facto de **qualquer simulador** (correcto) devolver **garantidamente** sempre o **mesmo resultado** quando um **mesmo modelo** (**arbitrário**) é sujeito ao **mesmo** conjunto de **inputs** (tb arbitrário)
  - Separação em pelo menos 2 fases é um requisito
  - Se semântica ≠ StateMate pode ser não determinístico
- Escolha entre transições conflituosas resolvida de forma arbitrária é uma outra potencial fonte de indeterminismo



As **ferramentas** tipicamente geram **alertas** quando detectam este tipo de situações de (potencial) não determinismo

- Em suma, obtém-se um comportamento **determinístico** para **semântica StateMate** se eventuais **transições conflituosas** forem **resolvidas** deterministicamente e não existirem outras fontes de indeterminismo.

Avaliação dos StateCharts (1)

Prós:

- A **hierarquia** permite uma composição (*nesting*) arbitrária de super-estados AND- e OR-
- **Semântica definida** (StateMate, artigo científico que se seguiu ao original).
- Grande número de **ferramentas** de simulação comerciais (StateMate, StateFlow, BetterState, ...)
- Existência de „back-ends“ que permitem efectuar a **“tradução” automática** de StateCharts para **código** C ou VHDL, permitindo implementações em **hardware** ou **software**

Avaliação dos StateCharts (2)

Contras:

- **Programas** C gerados são frequentemente **ineficientes**
- **Não** útil para sistemas **distribuídos**
- Não suporta descrição de características **não-funcionais**
- Não é **orientado a objectos**
- Não permite descrição de **hierarquia estrutural**

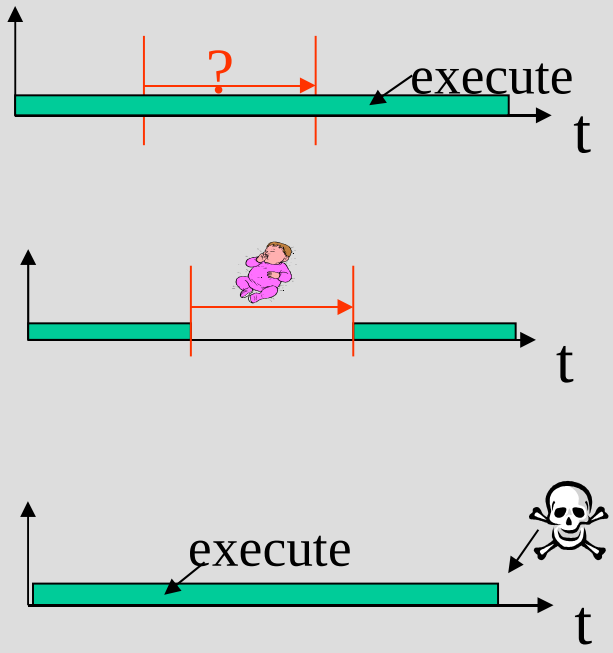
Extensão:

- "Module charts" para descrição de hierarquia estrutural

Avaliação dos StateCharts (3)

4 tipos de especificação temporal [Burns, 1990]:

- Medição de tempo decorrido
  - e.g. tempo entre activações de uma tarefa
- Tempo de atraso num processo
- Imposição de um timeout
  - Limitar tempo máximo que pode permanecer num certo estado
- Especificação de uma deadline



Os StateCharts **apenas contemplam**

- Especificação de **timeouts**
- Em algumas ferramentas é possível indicar *deadlines* num ficheiro de controlo separado

Avaliação dos StateCharts (4)

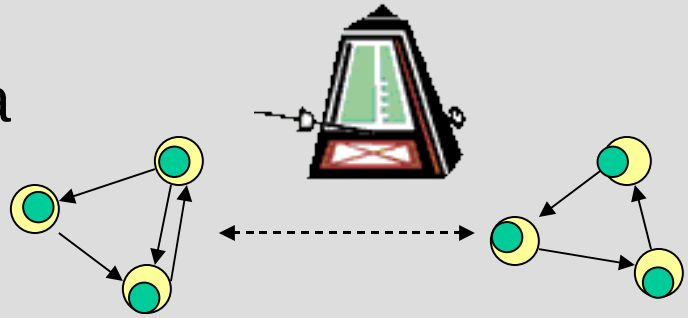
Concorrência:

- Em muitas linguagens a descrição de vários processos concorrentes **não é determinística**
  - A **ordem** de execução das tarefas **não é especificada**, podendo afectar o resultado
- As **linguagens síncronas** descrevem automata concorrentes
  - "*.. when automata are composed in parallel, a transition of the product is made of the "simultaneous" transitions of all of them*".

Avaliação dos StateCharts (4)

As linguagens síncronas assumem implicitamente a presença de um **relógio global**

- A cada ***tick*** todos os inputs são **avaliados**, os novos outputs e estados são **calculados** e só então as transições são **efectuadas**
- Requer um mecanismo de **broadcast** para todos os componentes do modelo
- Visão **idealística** da concorrência
- **Garante determinismo**



StateCharts com a semântica StateMate é uma linguagem síncrona

Sumário

StateCharts:

- Estados e composição de estados
- Estados AND e estados OR
- Eventos e transições
- Especificação temporal
- Mecanismo de broadcast
- Semântica
  - Modelo multi-fase e de fase única
- Avaliação geral do modelo