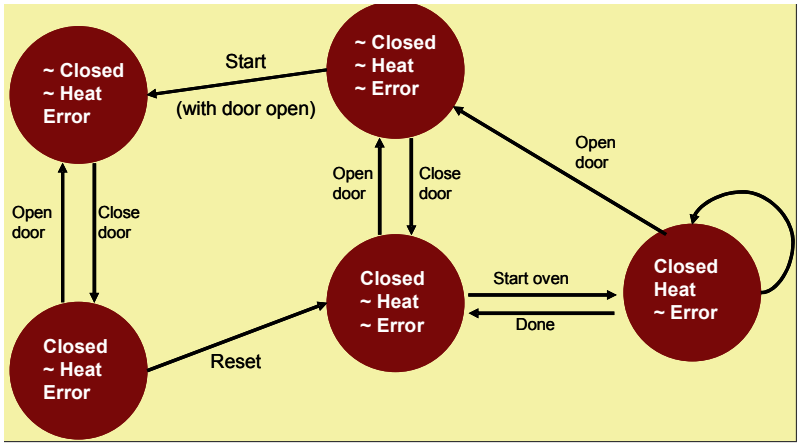
Aula 7

Um micro-ondas

Suponhamos um seguinte desenho de um controlador de um micro-ondas

- o aquecimento tem de estar *off* se a porta está aberta
- Se se pressionar o botão de *Start* com a porta aberta, entra em modo de erro
- Deixa o modo de erro se se pressionar no *Reset*

Um micro-ondas



Um micro-ondas

```
MODULE main
  VAR
    start: boolean;
    reset: boolean;
    closed : boolean;
    error : boolean;
     heat : boolean;

  ASSIGN
    init(error) := FALSE;
    init(heat) := FALSE;

    next(error) :=
        case
```

```
            start &  ! closed : TRUE;
            closed & reset  : FALSE;
            TRUE              : error;
        esac;

    next(heat) :=
        case
            start & closed  : TRUE;
            ! closed         : FALSE;
            TRUE              : heat;
        esac;

    SPEC AG (!closed -> AX (!heat))
    SPEC EF (heat)
```

Um elevador (simplificado)

Supor um modelo de elevador com uma cabina que pode andar por 4 andares mas seguindo sempre o padrão: 0,1,2,3,2,1,0,1,2,3...

```
MODULE main
VAR
 cabin: 0..3;
 dir: {up,down};
 request: array 0 .. 3 of boolean;

ASSIGN
 init(cabin):=0;
 init(dir):= up;
 init(request[0]):= FALSE;
 init(request[1]):= FALSE;
 init(request[2]):= FALSE;
 init(request[3]):= FALSE;

 next(cabin):= case
  dir = up & cabin <3 : cabin + 1;
  dir = down & cabin >0 : cabin - 1;
  TRUE: cabin;
  esac;

 next(dir):= case
   dir = up & cabin= 2: down;
   dir = down & cabin= 1: up;
   TRUE: dir;
  esac;
```

Um elevador (simplificado)

As chamadas podem ser feitas de qualquer andar, mas não se a cabina estiver nesse andar. Caso contrário, a chamada só desaparece se a cabina passar pelo referido andar.

```
next(request[0]):= case
    next(cabin)= 0 : FALSE;
    request[0]: TRUE;
    TRUE: {TRUE,FALSE};
    esac;

next(request[1]):= case
   next(cabin)= 1 : FALSE;
   request[1]: TRUE;
   TRUE: {TRUE,FALSE};
   esac;
.
.
.
```

Um elevador (simplificado)

Especificações a verificar:

- O elevador não entra em deadlock (i.e pode sempre passar a um estado seguinte)
- Todas as chamadas são satisfeitas
- Não é verdade que todas as chamadas são satisfeitas simultaneamente
- Se a cabina está subir não pode estar no andar 3
- Se a cabina está no primeiro andar e a subir, então irá a seguir para o segundo

LTL e CTL

O CTL não é estritamente mais expressivo que o LTL. Por exemplo

$$\mathrm{F}p \rightarrow \mathrm{F}q$$

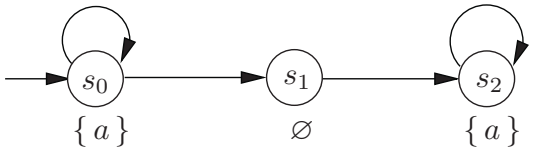
não se pode exprimir em CTL... O seu significado é

Todos os caminhos em que p é se verifica, também se verifica q.

Vê o que significa $\mathrm{AF}p \rightarrow \mathrm{AF}q$, ou $\mathrm{AG}(p \rightarrow \mathrm{AF}q)$.

LTL E CTL

FGφ não é AFAGφ

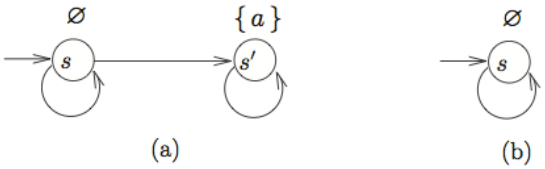


$M, s_0 \models \text{FG}a$ mas $M, s_0 \not\models \text{AFAG}a$

LTL e CTL

Mas AGEFa não se pode exprimir em LTL:

> A partir de qualquer estado é possível atingir um estado em que a é verdade.



Não existe nenhuma formula φ em LTL equivalente. Porque se houvesse $M_a, s \models$ AGEFa e $M_b, s \not\models$ AGEFa, mas ambos satisfazem φ.

Analogamente, tem-se que FX$a \equiv$ XF$a \equiv$ AXAFa mas não a AF AXa.

CTL*

CTL*

CTL onde não é obrigatório que um operador LTL {X, G,F,U} seja antecedido por um operador A ou E.

Exemplos:

- A[(pUr) $\vee$ (qUr)],
- E(GFϕ)
- A[Xp $\vee$ XXp]

O CTL* é estritamente mais expressivo que o LTL e o CTL, é computacionalmente muito menos eficiente ...

Sintaxe do CTL*

Fórmulas de Estado

São avaliadas num estado.

$$\phi \quad ::= \quad \top \mid p \mid (\neg\phi) \mid (\phi \ \wedge \ \phi) \mid (\mathrm{A}[\alpha]) \mid (\mathrm{E}[\alpha])$$

Fórmulas de Caminho

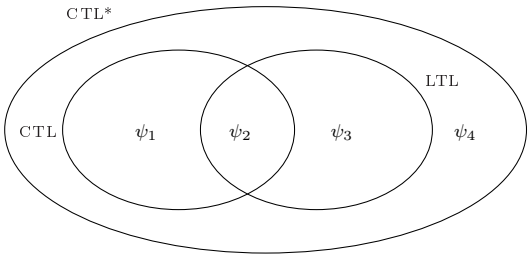
São avaliadas num caminho.

$$\alpha \quad ::= \quad \phi \mid (\neg\alpha) \mid (\alpha \ \wedge \ \alpha) \mid (\alpha \mathrm{U} \alpha) \mid (\mathrm{G}\alpha) \mid (\mathrm{F}\alpha) \mid (\mathrm{X}\alpha)$$

LTL, CTL e CTL*

Uma fórmula α LTL corresponde a A[α] do CTL*. O CTL é o fragmento de CTL* em que

$$\alpha \quad ::= \quad (\alpha \mathrm{U} \alpha) \mid (\mathrm{G}\alpha) \mid (\mathrm{F}\alpha) \mid (\mathrm{X}\alpha)$$



$$\begin{aligned} \psi_1 &= \mathrm{AGEF}p \\ \psi_2 &= \mathrm{AG}(p \rightarrow AFq) \\ \psi_3 &= \mathrm{A}[\mathrm{GF}p \rightarrow Fq] \\ \psi_4 &= \mathrm{E}[\mathrm{GF}p] \end{aligned}$$

LTL versus CTL

Aspect	*Linear time*	*Branching time*
"behavior" in a state s	path-based: $trace(s)$	state-based: computation tree of s
temporal logic	LTL: path formulae φ $s \models \varphi$ iff $\forall \pi \in Paths(s).\, \pi \models \varphi$	CTL: state formulae existential path quantification $\exists\varphi$ universal path quantification: $\forall\varphi$
complexity of the model checking problems	PSPACE–complete $\mathcal{O}\left(\lvert TS\rvert \cdot \exp(\lvert\varphi\rvert)\right)$	*PTIME* $\mathcal{O}(\lvert TS\rvert \cdot \lvert\Phi\rvert)$
implementation-relation	trace inclusion and the like (proof is PSPACE-complete)	simulation and bisimulation (proof in polynomial time)
fairness	no special techniques needed	special techniques needed