Especificação e verificação de sistemas de tempo real

Introdução ao UPPAAL

Simão Melo de Sousa, Joel Carvalho

RELEASE - Reliable and Secure Computation Group,
Computer Science Department
University of Beira Interior, Portugal

5 de Maio de 2009

Outline

Objectivo desta apresentação

Introduzir

- Modelação por autómato de sistemas concorrentes e de tempo real
- Especificação do comportamento dinâmico na sua dimensão temporal: sequência e duração
- Verificação da concordância entre modelo e comportamento esperado pela técnica do Model Checking
- Ilustração e aplicação destes conceitos no UPPAAL

O que deixamos fora:

- Exploração detalhada dos conceitos e da algoritmia que fundamentam a verificação por modelos
- Arte e perícia do model-checking, nomeadamente: como enfrentar os seus limites, como tirar proveito das especificidades de cada tipo de verificação por modelos
- Utilização avançada do UPPAAL
- Arte e perícia na verficação por modelos em ferramentas alternativas como o SMV, o SPIN, o Kronos, etc...

Contexto

Model Checking de sistemas de tempo real

- os modelos operacionais subjacentes: sistemas de transição (autómatos de estados finitos, de Büchi, estruturas de kripke, autómatos temporizados, etc...)
- as lógicas subjacentes: as lógicas modais, temporais, dinâmicas, de duração etc...
- os processos de verificação: Assumindo a existência de
  - um sistema de transição que especifica o sistema por implementar
  - uma lógica particular e
  - uma fórmula desta lógica que estabelece o que se pretende que o sistema verifique,

  os processos de verificação são constatações baseadas em percursos adequados do sistema de transição (model checking directo) ou de uma representação compacta (model checking simbólico)

Ênfase

num acetato...model checking de autómatos temporizados

- Autómatos de Büchi = Autómatos de estados finitos estendidos para poderem aceitar entradas infinitas.
- Noção de aceitação = a execução potencialmente infinita passa infinitamente (*infinitely often*) por um estado final.
- Autómatos temporizados: redes de autómatos de Büchi estendidos com a noção de relógios locais.
- Pode-se assim, com este formalismo, discursar sobre a sequência das operações durante as execuções possíveis (concorrência e distribuição) mas também sobre os prazos esperados para cada uma delas (tempo real).
- Detalhes, mais para a frente...

Bibliografia

- Principles of Model Checking, Christel Baier, Joost-Pieter Katoen. The MIT Press (May 31, 2008),ISBN: 978-0262026499.
- Systems and Software Verification: Model-Checking Techniques and Tools, B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. Springer Verlag, 2001, ISBN 3540415238.
- Model Checking, Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled, MIT Press, 1999, ISBN 0-262-03270-8.
- Timed Automata: Semantics, Algorithms and Tools, Johan Bengtsson and Wang Yi. In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.
- A Tutorial on Uppaal, Gerd Behrmann, Alexandre David, and Kim G. Larsen. In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS 3185.

Table of Contents

Uma definição formal e abrangente

Assume-se a existência de um conjunto $Prop = \{P_1, \ldots\}$ de proposições (propriedades) elementares.

Definition (Autómato)

Um autómato é o $5-tuplo$ $\mathcal{A} = (Q, E, T, Q_0, I)$ onde:

- Q é um conjunto de estados
- E é um conjunto de etiquetas (label) de transições
- $T \subseteq Q \times E \times Q$ é o conjunto das transições
- $Q_0 \subseteq Q$ é o conjunto de estados iniciais. Iremos aqui considerar (sem perda de generalidade) somente o caso em que há um só estado inicial. q_0 é o estado inicial do autómato
- $I : Q \rightarrow \mathcal{P}_F(Prop)$ é uma aplicação que associa a cada estado q de Q o conjunto finito de propriedades que este verifica

Caso se pretenda que o autómato produza *output* então considera-se uma sexta componente $\mathcal{O}$ que é uma aplicação que atribui a cada estado o *output* que este produz (fala-se neste caso de autómatos de Moore)

Exemplo

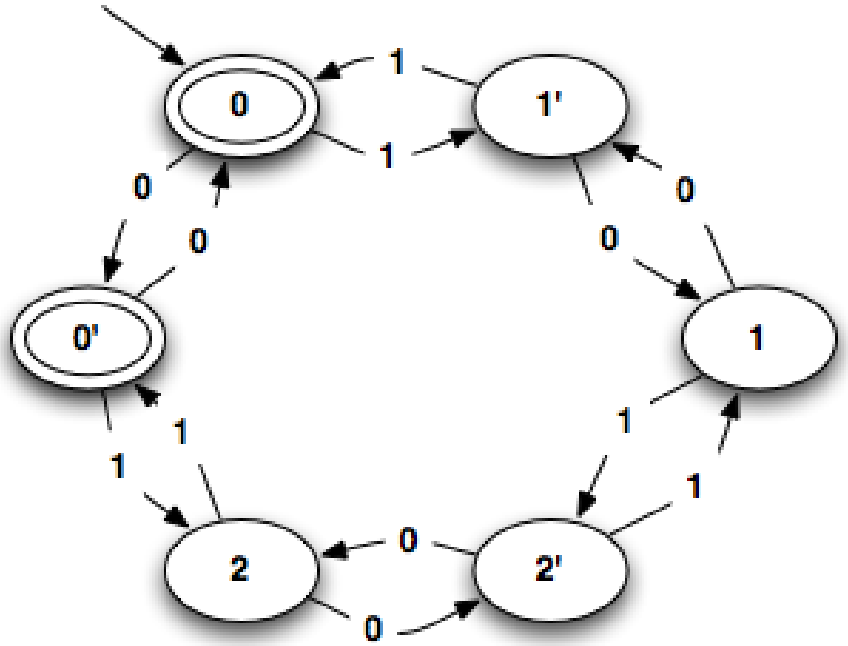


Figura: Múltiplos de 3

Outras definições de base

Definition

- Comecemos por assumir a existência de um autómato $\mathcal{A}$.
- Um *caminho* em $\mathcal{A}$ é uma sequência (eventualmente infinita) σ de transições (q, e, r) encadeadas (i.e. para toda a transição (q, e, r) da sequência, a transição seguinte é da forma (r, e', s),).
- O *comprimento* de um caminho σ, notação $|\sigma|$ é o número de transições que este contém. $|\sigma| \in \mathbb{N} \cup \{\omega\}$ (o caminho pode ser infinito).
- Para um caminho σ, um índice $i < |\sigma|$, $\sigma(i)$ refere-se ao estado q da i-ésima transição (q, e, r) de σ.

Por comodidade utilizaremos esta notação:

$$0 \xrightarrow{1} 1' \xrightarrow{0} 1 \xrightarrow{1} 2' \xrightarrow{1} 1 \xrightarrow{0} 1' \xrightarrow{1} 0 \xrightarrow{0} 0' \xrightarrow{0} 0 \quad \text{(reconhecimento de 45)}$$

no lugar de

$$(0, 1, 1')(1', 0, 1)(1, 1, 2')(2', 1, 1)(1, 0, 1')(1', 1, 0)(0, 0, 0')(0', 0, 0)$$

Outras definições de base

Definition

- Uma *execução parcial* de $\mathcal{A}$ é um caminho começando pelo estado inicial q_0.
- Uma *execução completa* é uma caminho parcial *máximo*. Por máximo entendemos que o caminho não pode ser mais estendido. Este caso acontece quando o caminho considerado é infinito ou acabou num estado a partir do qual não há transições possíveis. Este ultimo é designado de *execução com bloqueio*.
- *Árvores de execução*: representação arborescente de todas as execuções possíveis a partir dum determinado estado (em geral q_0).
- Um estado é designado de *acessível* (ou atingível) se aparece na árvore de execução com raíz q_0. Ou seja, existe pelo menos uma execução que passe por ele.

Algumas extensões aos autómatos de base

Variáveis, atribuições e transições com guardas

- Por comodidade, podemos juntar variáveis de estados na modelação por autómatos.
- Neste caso as transições são aumentadas com a capacidade de testar as variáveis (guardas) e, caso o teste permita a selecção da transição, de alterar o valor das variáveis.
- Assim, tal autómato passa a considerar um conjunto de variáveis (tipificadas), e uma transição passa a ser um 5-tuplo (q, g, l, a, r) onde:
  - q é o estado de partida e r o estado de chegada
  - a guarda g que é uma propriedade sobre as variáveis do autómato
  - a etiqueta l
  - o conjunto $a = \{x_1 := e_1 \ldots\}$ das actualizações de variáveis (atribuições) pode realizar

Exemplo

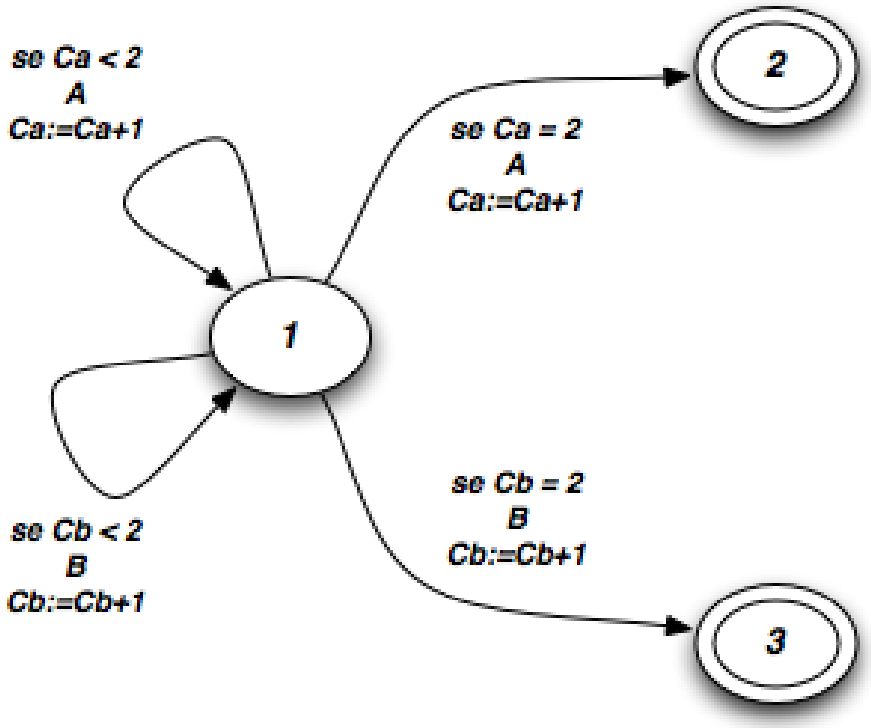


Figura: Um autómato com variável

Exemplo

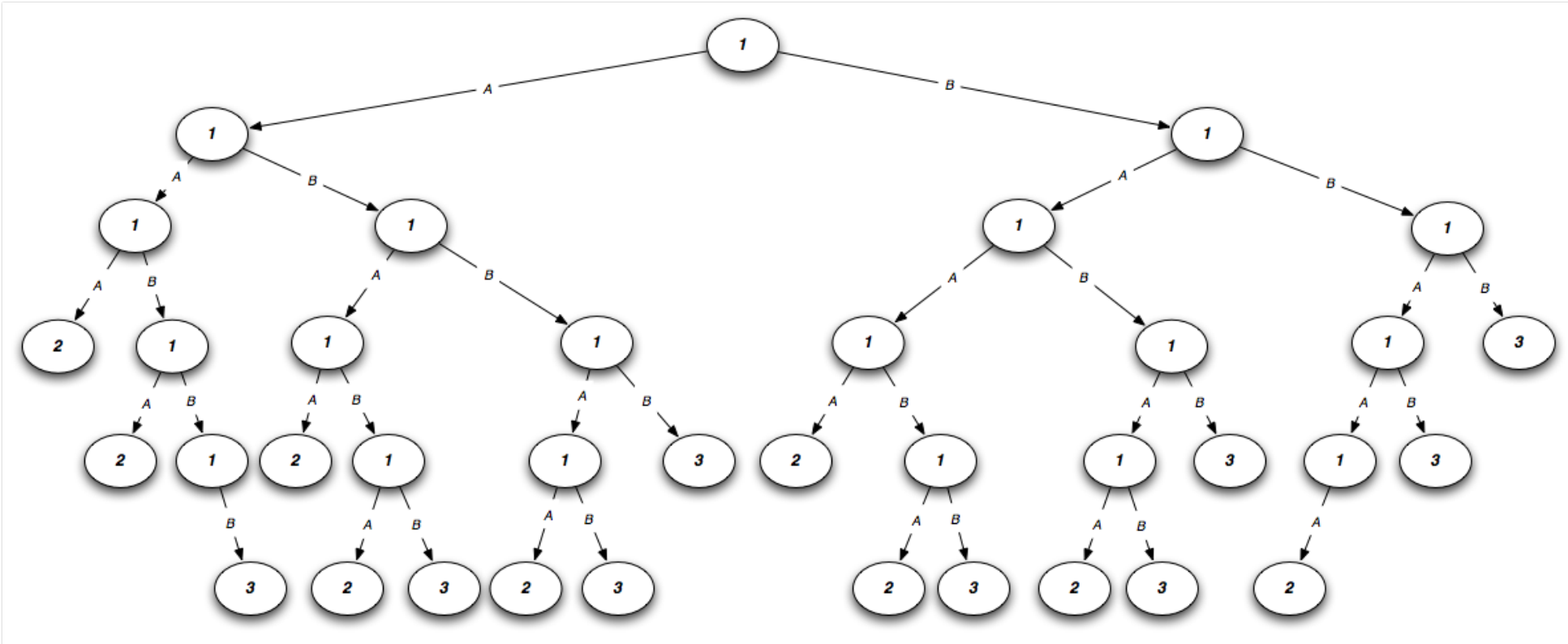


Figura: Arvore de execução do autómato anterior

Algumas extensões aos autómatos de base

Variáveis, atribuições e transições com guardas

- uma transição (q, g, l, a, r) pode ser seleccionada quando o estado q é activo, quando a guarda g é verificada e quando a entrada do autómato começa pela etiqueta l da transição. Quando seleccionada, a actualização a é accionada e a actividade passa do estado q para o estado r .
- As noções de caminho e de execução são estendidas naturalmente.

Autómato desdobrado ou sistema de transição

Extensões vs. clássico

- Os Model-Checkers (e os algoritmos subjacentes) não trabalham directamente com autómatos extendidos. Se estes forem suportados, então é preciso passar para uma representação clássica.
- A tradução resulta no *autómato desdobrado*, ou *sistema de transição associado* ao autómato A
- É possível a partir dum autómato A com variáveis e sob certas condições obter um autómato finito sem variáveis equivalente.
- Exemplos de condições: conjunto dos valores possíveis para as variáveis é finito. O conjunto dos valores é infinito, mas existe uma partição finita pertinente (pares/impares, positivo/negativo...), etc...
- Sem essas condições, os autómatos resultantes são infinitos.
- Os estados do autómato desdobrado são designados de *estados globais*. Este contém a informação do estado original de quem provêm (designado de estado ou informação de *controlo*) e o valor das variáveis numa configuração particular que este estado representa. De facto o estado de controlo permite determinar no autómato desdobrado que transições são relevantes ou não.

Exemplo

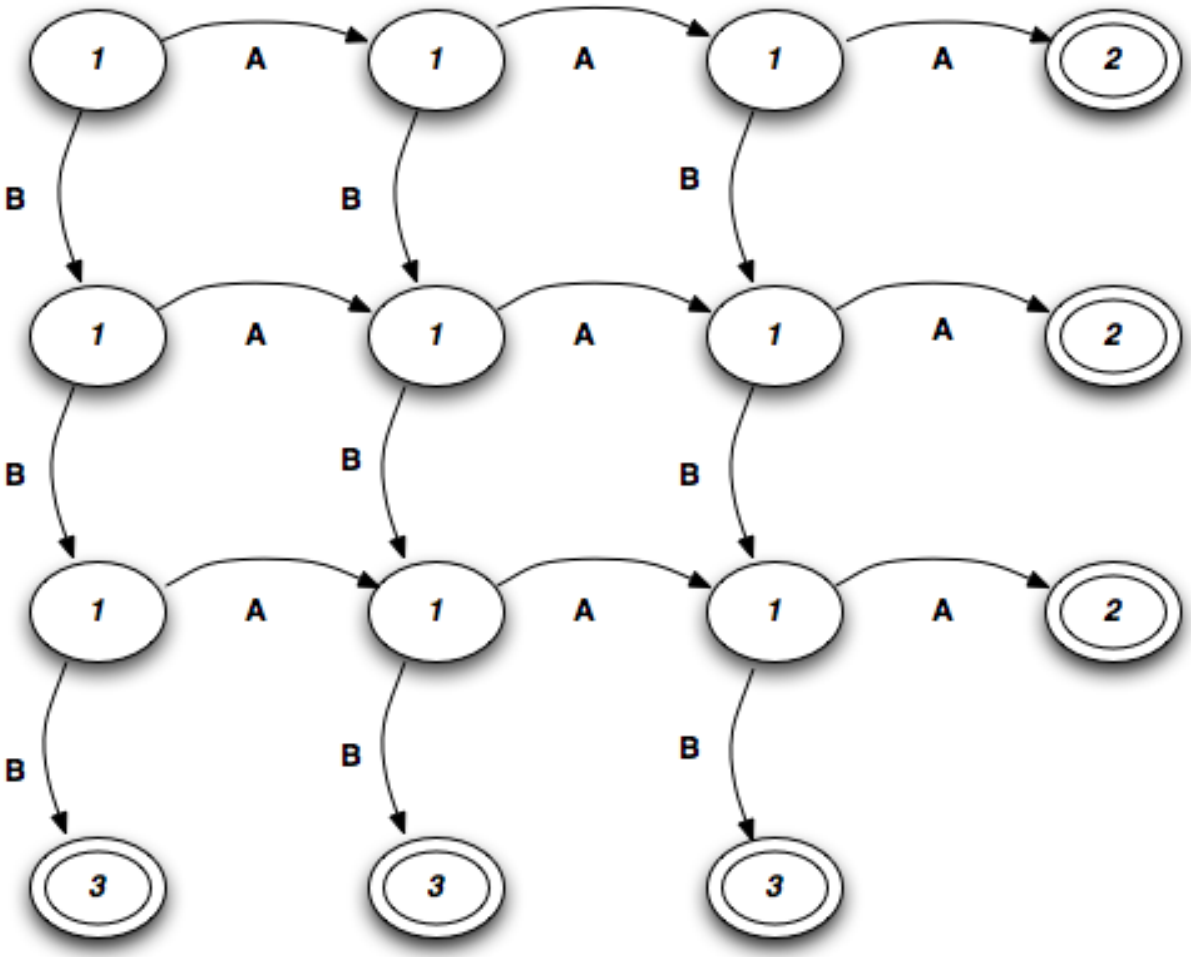


Figura: O autómato desdobrado correspondente

Redes de autómatos e produto sincronizado

Sincronização e Redes de autómatos

Na arte da modelação por autómatos, a estratégia "divide-and-conquer" corresponde à modelação dos sub-sistemas por autómatos adequados. A modelação do sistema em si (a coordenação) consiste em *sincronizar* entre si os diferentes subsistemas. Fala-se assim de *produto sincronizado* de autómatos.

Redes de autómatos e produto sincronizado

Definition (Produto sincronizado)

Consideremos uma família de n autómatos $A_i = (Q_i, E_i, q_{0,i}, l_i)$ (com $i = 1, \ldots, n$). Consideremos também a etiqueta "_" que corresponde a acção "nada por fazer".

- Define-se por *conjunto de sincronizações sync*, um conjunto tal que $sync \subseteq \Pi_{1 \leq i < n}(E_i \cup \{\_\})$.
- Define-se por produto sincronizado da família de autómatos A_i (Notação: $A_1||A_2||\cdots||A_n$), o autómato $A = (Q, E, T, q_0, I)$ tal que:
  - $Q = Q_i \times \cdots \times Q_n$;
  - $E = \Pi_{1 \leq i < n}(E_i \cup \{\_\})$;
  - $T = \left\{ \begin{array}{l} ((q_1, \ldots, q_n), (e_1, \ldots, e_n), (q'_1, \ldots, q'_n)) \mid \\ \quad (e_1, \ldots, e_n) \in Sync \wedge \\ \quad \forall i,\ (e_i =' \_' \wedge q'_i = q_i) \vee (e_i \neq' \_' \wedge (q_i, e, q'_i) \in T_i) \end{array} \right\}$
  - $q_0 = (q_{0,1}, \cdots, q_{0,n})$;
  - $I((q_1, \cdots, q_n)) = \bigcup_{1 \leq i < n} l_i(q_i)$

Redes de autómatos e produto sincronizado

Definition (Produto sincronizado (cont.))

Quando $sync = \Pi_{1 \leq i < n}(E_i \cup \{\_\})$, obtemos o produto cartesiano dos autómatos considerados (não há restrições sobre o funcionamento e a cooperação)

Vantagens e desvantagens

- É possível (e cómodo) representar de forma modular e clara sistemas complexos.
- A "arte" é determinar que subsistemas dão origem a autómatos e como os sincronizar
- Infelizmente, o produto destes autómatos tem um tamanho exponencial comparativamente ao tamanho dos autómatos de que é composto: fenómeno conhecido por *explosão do espaço de estados*.

Exemplo

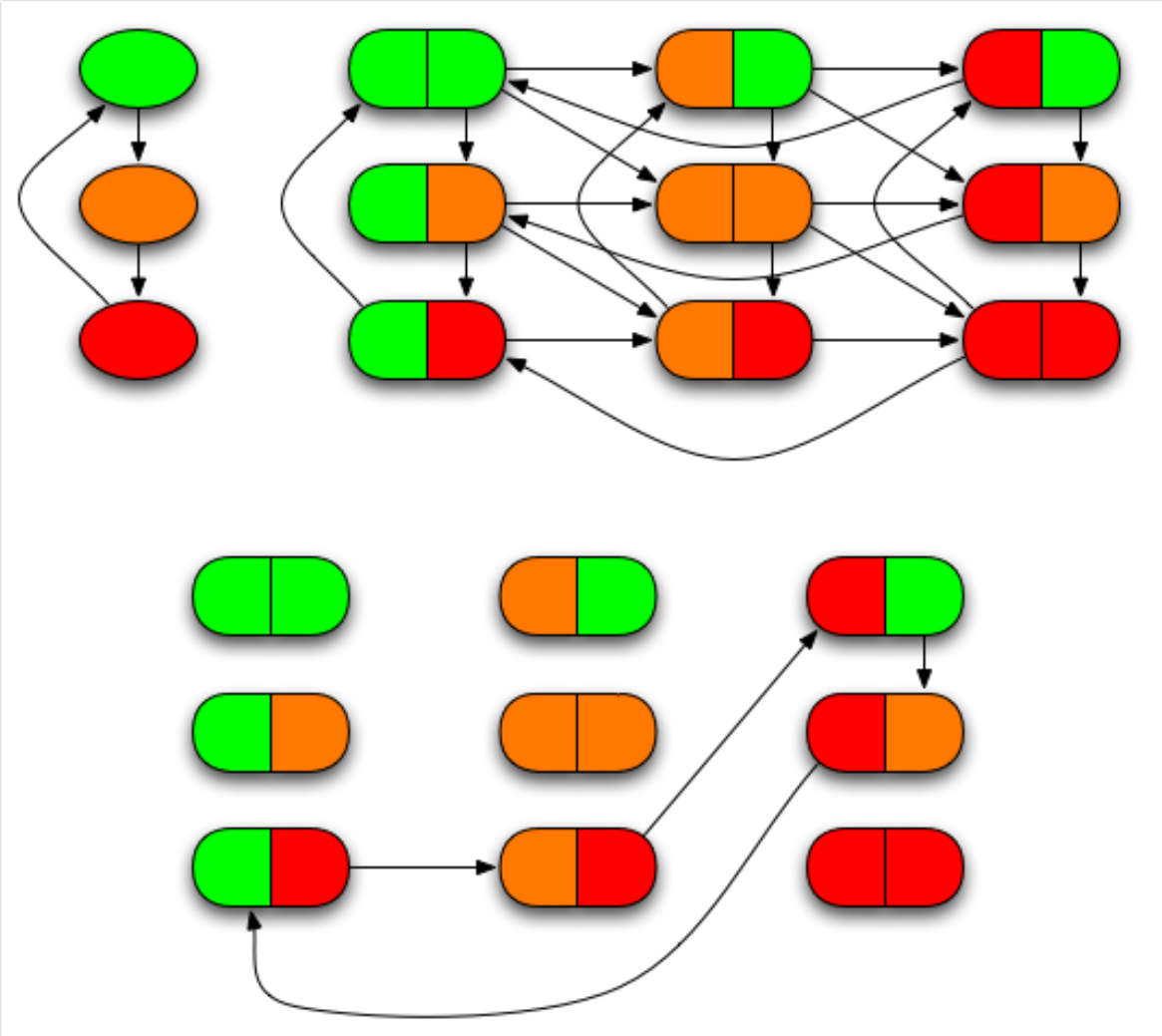


Figura: Produto com e sem sincronização de dois semáforos

Conjunto dos estados acessíveis - Grafo de acessibilidade

Acessibilidade

- *Estado inacessível*: estado pelo qual nunca poderá passar um caminho oriundo do estado inicial.
- O produto (sincronizado) de autómatos gera autómatos com estados que não são acessíveis. O número destes estados depende da sincronização definida mas é, em geral, importante.
- Um autómato que consiste na remoção explícita dos estados não acessíveis é designada de *grafo de acessibilidade*.
- Além da obvia comodidade em lidar com autómatos menores, a noção de acessibilidade permite expressar algumas propriedades interessantes.
- Por exemplo: modelação da partilha de um recurso por n utilizadores.
  - A gestão de um utilizador = um autómato.
  - O sistema de partilha = produto sincronizado.
  - Nenhum estado representando a utilização em simultânea do recurso por mais do que um utilizador é acessível.
- Assim, uma função importante de um Model-Checker é ser capaz de calcular eficientemente o grafo de acessibilidade de um produto sincronizado.

Exemplo

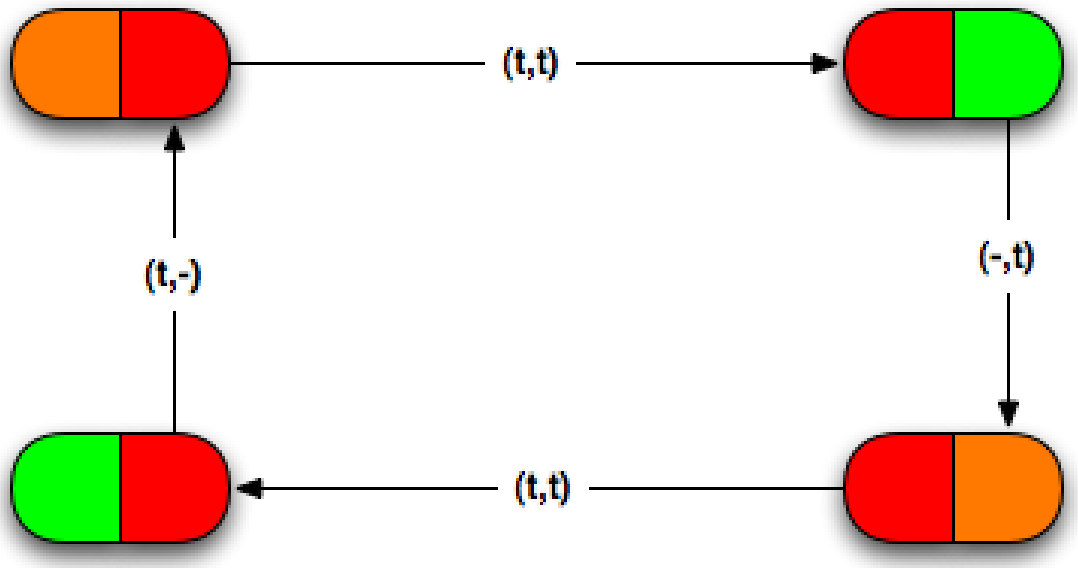


Figura: Grafo de acessibilidade dos dois semáforos

modalidades de sincronização

Sincronização por mensagens

- Sincronização = envio/recepção de mensagens.
- Os autómatos por sincronizar contemplam etiquetas como m! (envio da mensagem m), ?m (recepção da mensagem m).
- A sincronização só contempla transições onde qualquer emissão de mensagem é acompanhada pela recepção da própria.

Exemplo

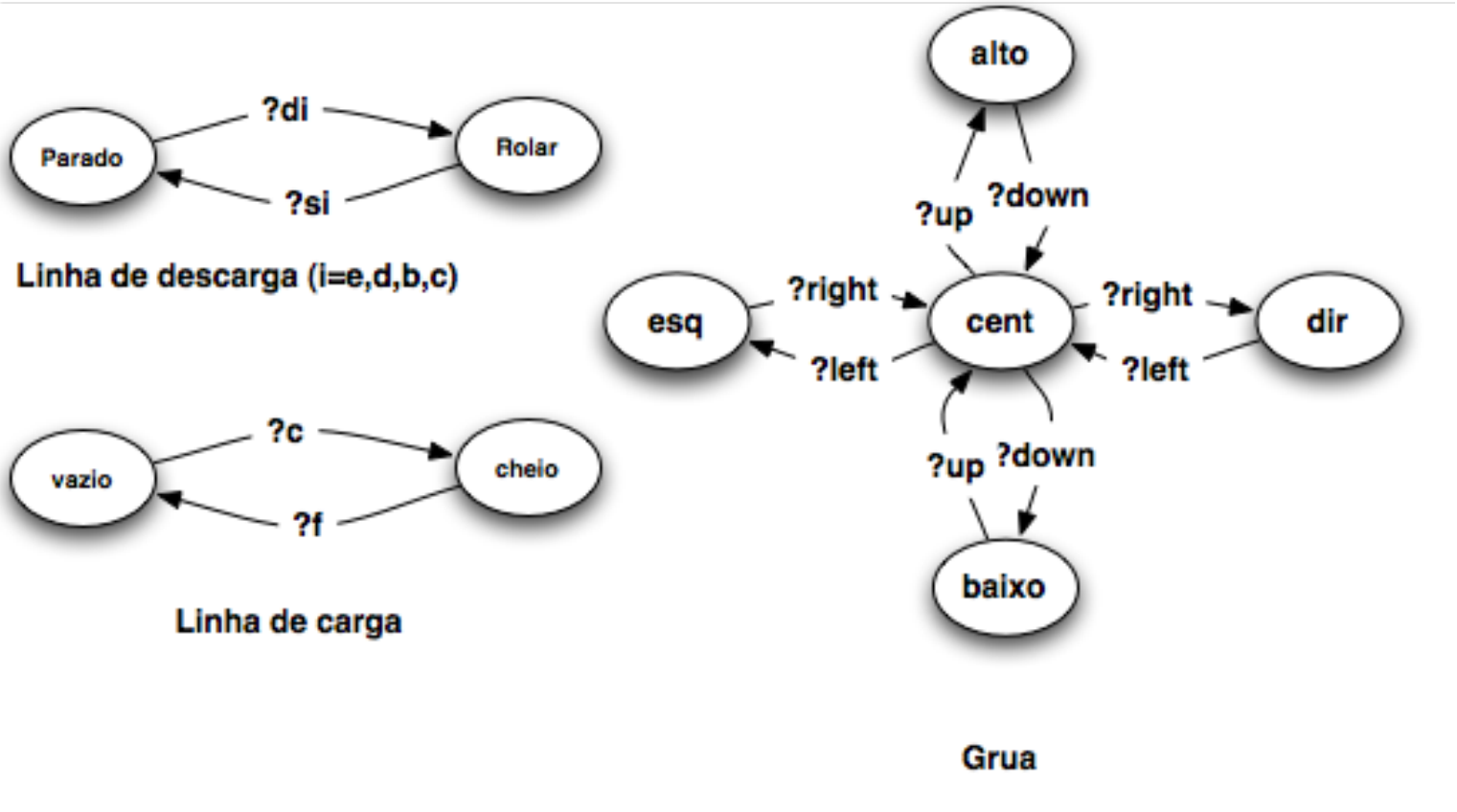


Figura: linha de montagem

Exemplo

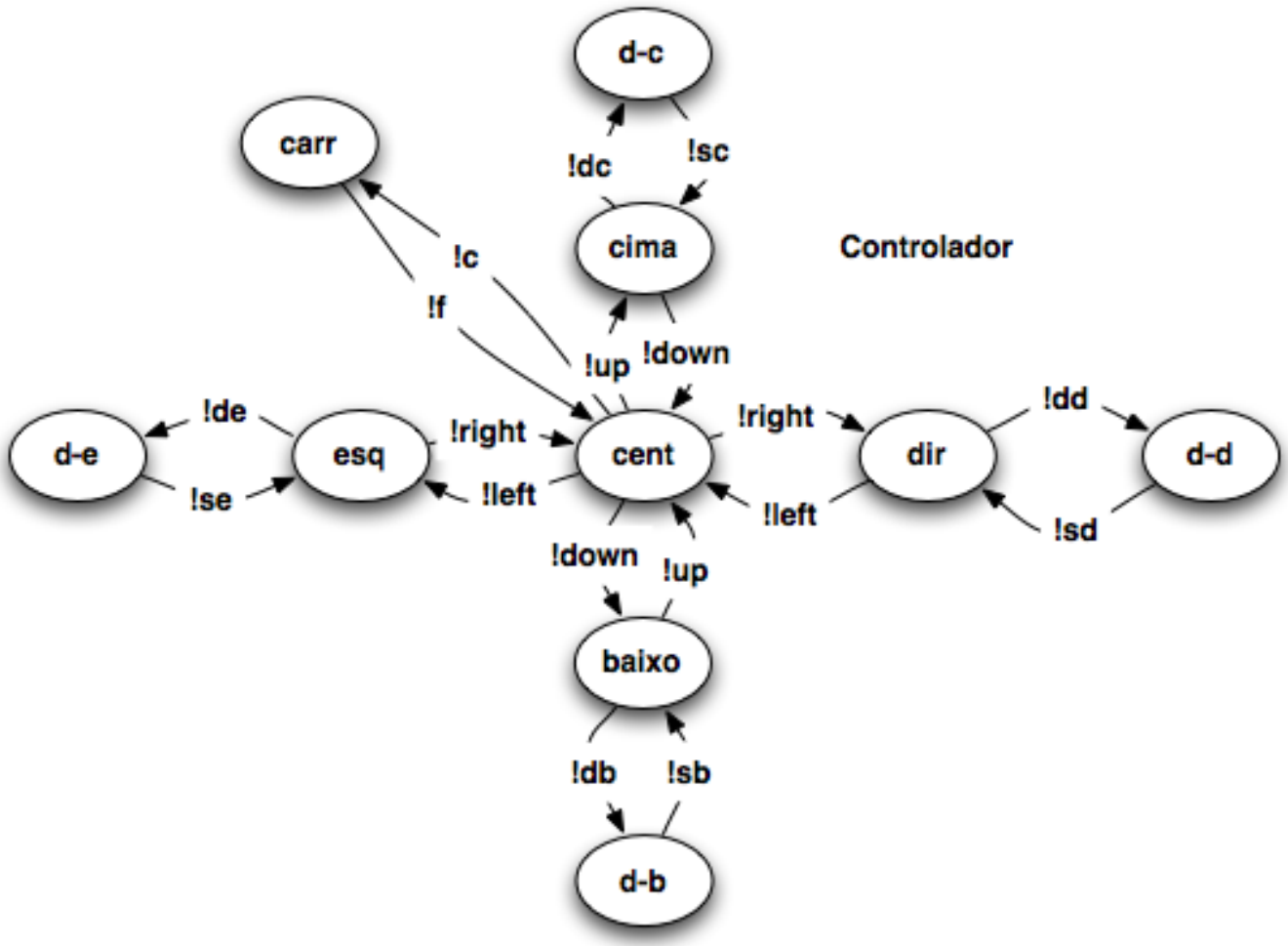


Figura: linha de montagem - cont.

modalidades de sincronização

Sincronização por variáveis partilhadas

- Embora seja possível evitar o uso das variáveis, é prático poder beneficiar do seu uso explícito.
- Assim, em termos práticos, podemos querer sincronizar autómatos com base na gestão partilhada de variáveis.
- Reencontramos no contexto dos autómatos a problemática da exclusão mútua e das suas soluções (algoritmo de Peterson, semáforos, etc...),

Exemplo

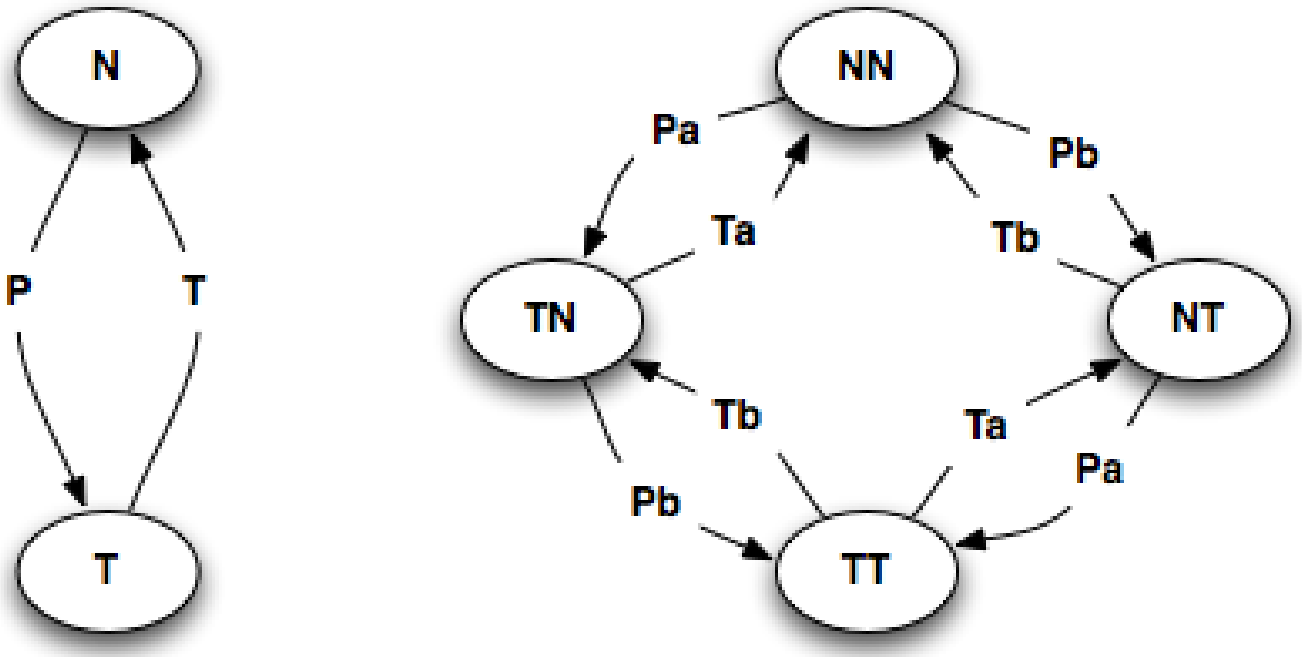


Figura: Uma impressora partilhada por dois utilizadores: versão simplista

Exemplo

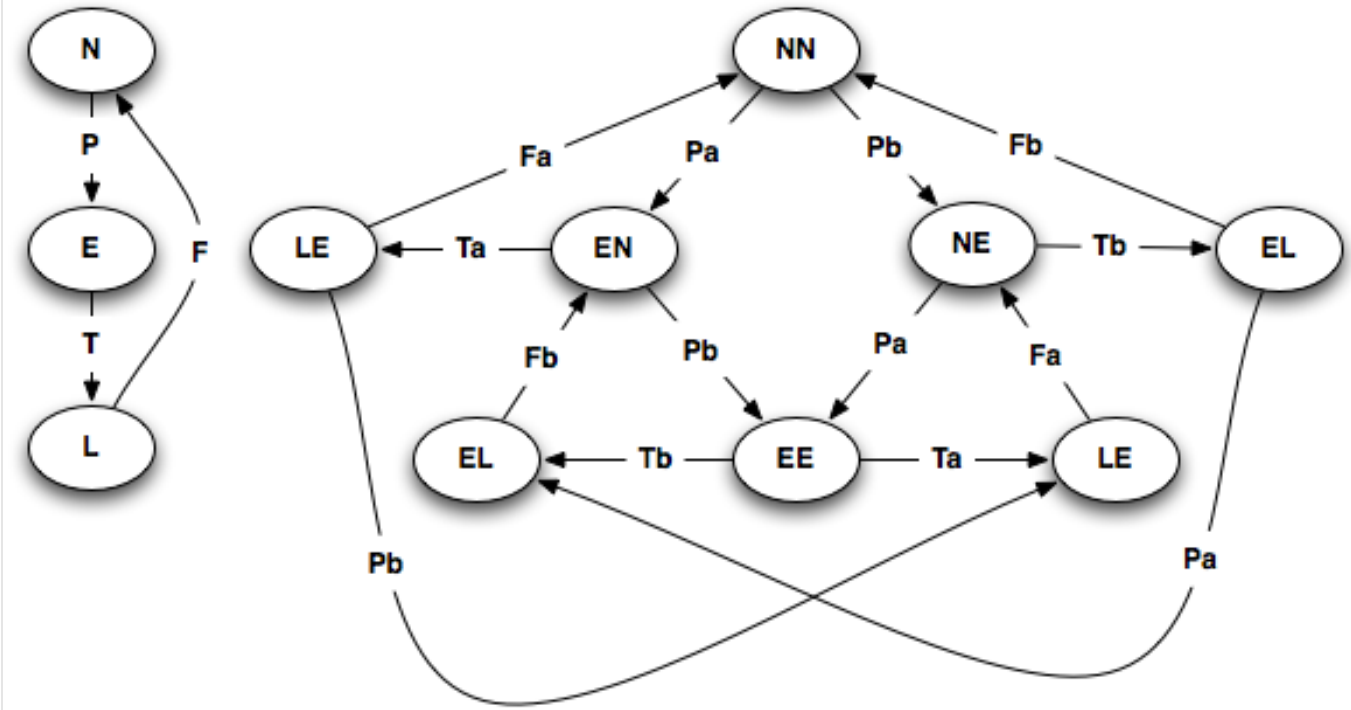


Figura: Uma impressora partilhada por dois utilizadores: versão "unfair"

Exemplo

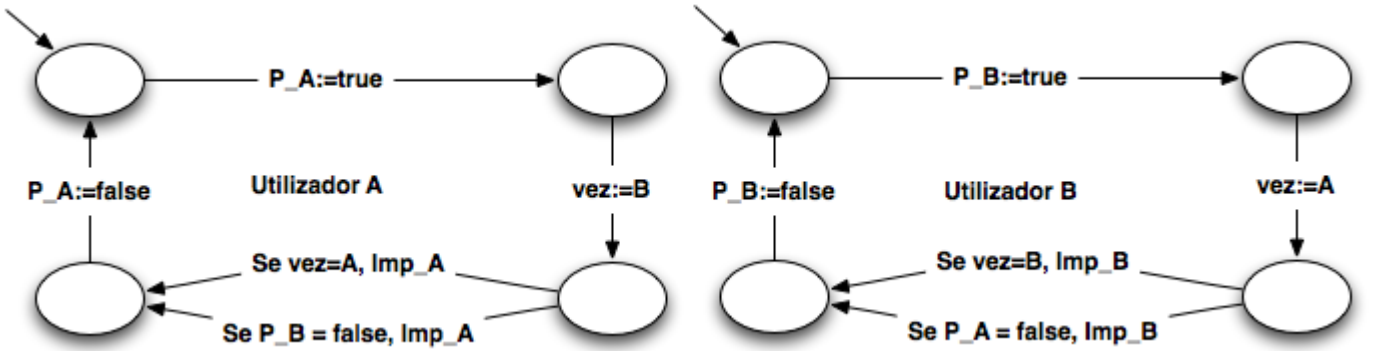


Figura: Uma impressora partilhada por dois utilizadores: versão com exclusão mútua

Table of Contents

CTL^* in a nutshell

Sintaxe (formato BNF) da lógica CTL^*

$\phi, \psi ::=$

$P_1 \mid P_2 \mid \cdots$	(proposições atómicas)
$\neg\phi \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \implies \psi) \mid \cdots$	(conectivas lógicas classicas)
$X\ \phi \mid F\ \phi \mid G\ \phi \mid (\phi\ U\ \psi) \mid$	(conectivas temporais de estado)
$E\ \phi \mid A\ \phi$	(conectivas temporais de caminho)

CTL^* in a nutshell

Elementos de semântica da CTL^*

Consideremos o conjunto das execuções de um autómato.
Este pode-se representar como um conjunto de sequências ou como uma árvore.

- Tempo do ponto visto linear:
  - $X\ \phi$: Na execução considerada, o estado imediatamente a seguir (neXt) verifica ϕ.
  - $F\ \phi$: Na execução considerada, um estado a seguir (um estado no Futuro) verifica ϕ
  - $G\ \phi$: Na execução considerada, todos os estado seguintes verificam ϕ
  - $\phi\ U\ \psi$: Na execução considerada, Até (Until) chegar num estado em que *psi* se verifica, verifica-se *phi*.

CTL^* in a nutshell

Elementos de semântica da CTL^*

Consideremos o conjunto das execuções de um autómato.
Este pode-se representar como um conjunto de sequências ou como uma árvore.

- Tempo visto como uma árvore:
  - $A\ \phi$: qualquer execução que parte do estado corrente satisfaz a fórmula ϕ
  - $E\ \phi$: existe uma execução que parte do estado corrente que satisfaz a fórmula ϕ

CTL^* in a nutshell

modelo da CTL^* = estrutura de Kripke

Uma estrutura de Kripke é um tuplo (S, i, R, L) onde

- S é um conjunto finito de estados;
- $i \in S$ é o estado inicial;
- $R \subseteq S \times S$ é uma relação de transição. No caso presente procura-se relações totais (que verificam, $\forall s \in S, \exists s' \in S, (s, s') \in R$) para garantir a ausencia de deadlock.
- $L : S \rightarrow \mathcal{P}(P)$ é uma função que etiqueta cada estado com o conjunto de fórmulas atómicas válidas nesse estado.

CTL^* in a nutshell

Relação com os autómatos

- Assim, uma estrutura de Kripke não é mais do que um caso particular de autómato (onde se ignora as etiquetas).
- As estruturas vão permitir definir formalmente o que entendemos por *ser verdade em* CTL^*
- No que nos diz respeito, as considerações de semântica permitem-nos estabelecer as regras para afirmar quando um autómato respeita ou não uma fórmula (propriedade) CTL^* (i.e. autómato = modelo da fórmula)
- Model checking = processo algorítmico que permite estabelecer esta afirmação (ou não)

CTL^* in a nutshell

Satisfação e semântica

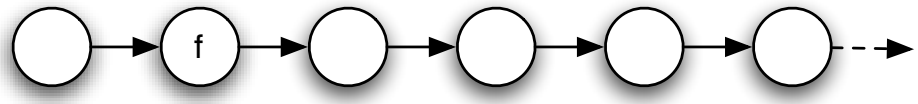
$\mathcal{A}$: Autómato/estrutura de Kripke. Dado σ uma execução, $i \in \mathbb{N} \cup \{\omega\}$ um momento na execução e $\sigma(i)$, o estado correspondente (o i-ésimo elemento da execução)
Notação: $\mathcal{A}, \sigma, i \models \phi$, ou $\sigma, i \models \phi$. No tempo i da execução σ do autómato $\mathcal{A}$, ϕ é válida.

$$
\begin{array}{lll}
\sigma, i \models P & sse & P \in I(\sigma(i)) \\
\sigma, i \models \neg\phi & sse & \sigma, i \not\models \phi \\
\sigma, i \models \phi \wedge \psi & sse & (\sigma, i \models \phi) \wedge (\sigma, i \models \psi) \\
 & & \\
\sigma, i \models X\ \phi & sse & i < |\sigma| \ \wedge \ \sigma, i+1 \models \phi \\
\sigma, i \models F\ \phi & sse & \exists j \geq i,\ j < |\sigma| \ \wedge \ \sigma, j \models \phi \\
\sigma, i \models G\ \phi & sse & \forall j \geq i,\ j < |\sigma| \implies \sigma, j \models \phi \\
 & & \\
\sigma, i \models \phi\ U\ \psi & sse & \exists j \geq i,\ j < |\sigma| \ \wedge \ \sigma, j \models \psi \\
 & & \wedge \ \forall k, i \leq k < j \implies \sigma, k \models \phi \\
 & & \\
\sigma, i \models E\ \phi & sse & \exists \sigma', \sigma(0) \ldots \sigma(i) = \sigma(0)' \ldots \sigma(i)' \wedge \sigma', i \models \phi \\
\sigma, i \models A\ \phi & sse & \forall \sigma', \sigma(0) \ldots \sigma(i) = \sigma(0)' \ldots \sigma(i)' \wedge \sigma', i \models \phi
\end{array}
$$

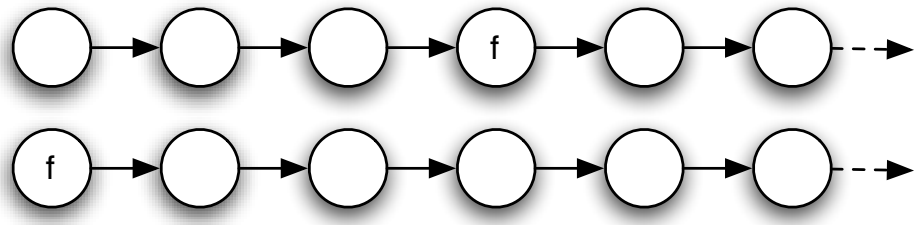
$\mathcal{A} \models \phi$ sse para toda a execução σ de $\mathcal{A}$ verifica-se $\sigma, 0 \models \phi$

Semântica

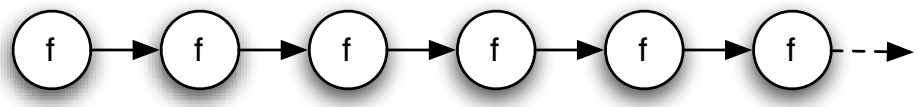
$$\sigma, i \models X\ f \quad sse \quad i < |\sigma| \ \wedge \ \sigma, i+1 \models f$$



$$\sigma, i \models F\ f \quad sse\ \exists j \geq i,\ j < |\sigma| \ \wedge \ \sigma, j \models f$$

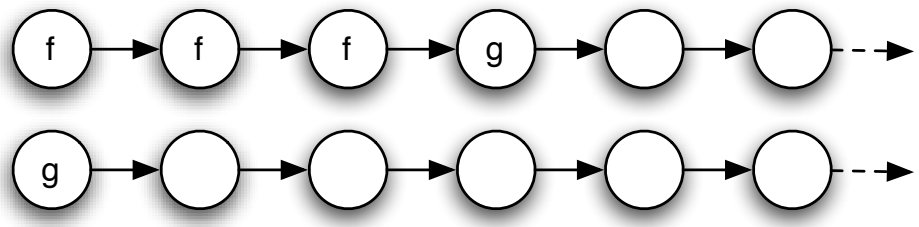


$$\sigma, i \models G\ f \quad sse\ \forall j \geq i,\ j < |\sigma| \implies \sigma, j \models f$$



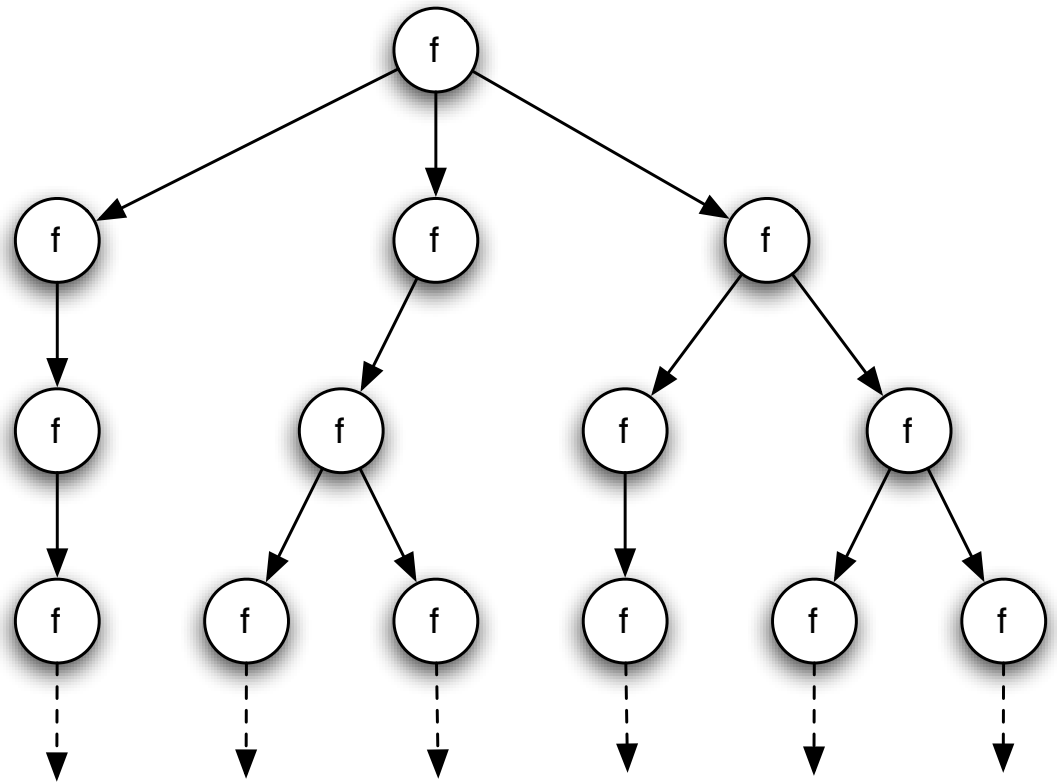
Semântica

$$M, \pi \models f \ U \ g \quad sse \quad \exists i \geq 0 \cdot M, \pi^i \models g \text{ e } \forall 0 \leq j < i \cdot M, \pi^j \models f$$



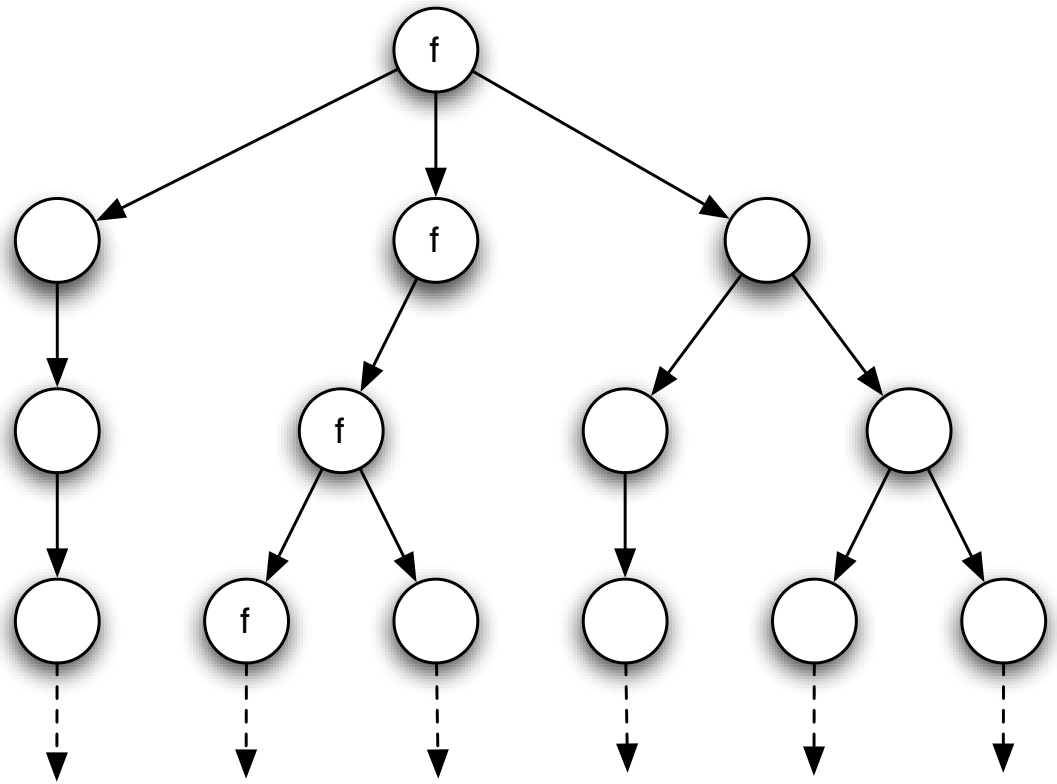
Combinações Típicas

$AG\ f$



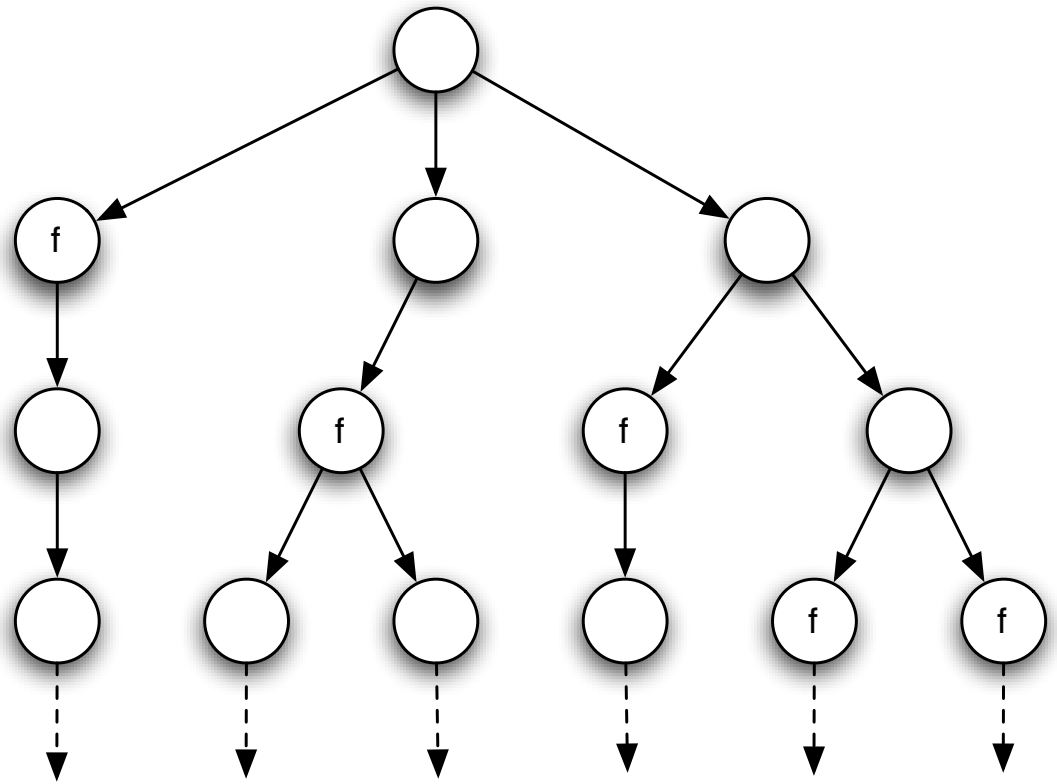
Combinações Típicas

$EG\ f$



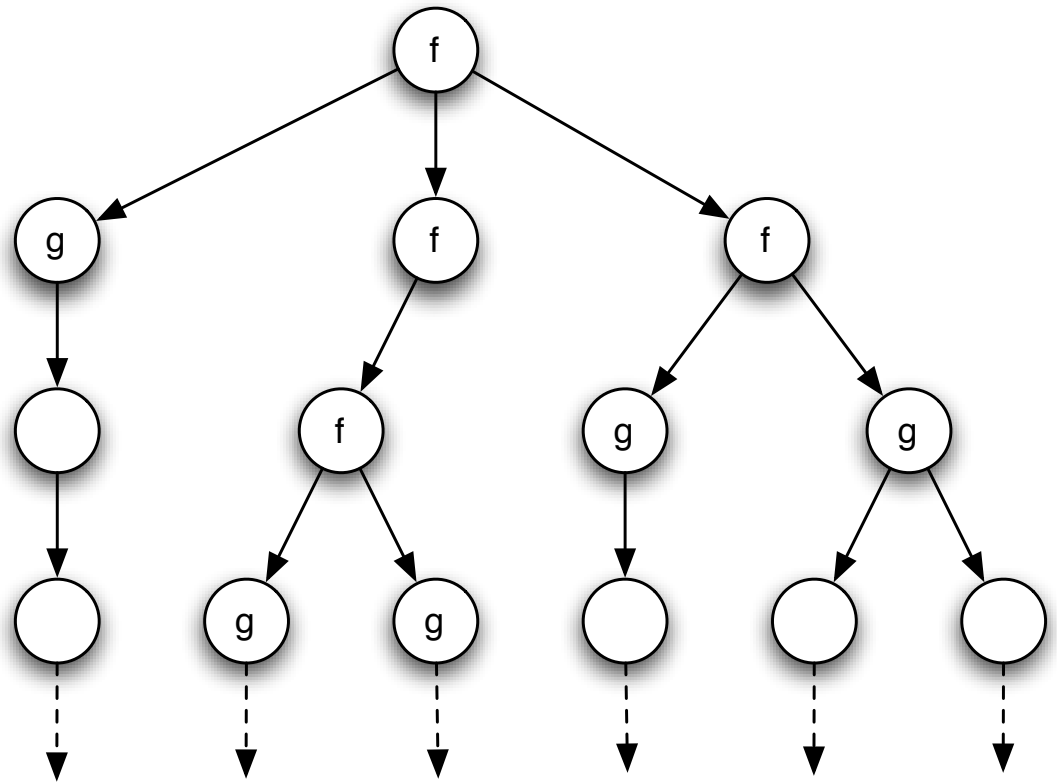
Combinações Típicas

$AF\ f$



Combinações Típicas

$$A(f\ U\ g)$$



A arte de especificar

formulário

- $F\ \phi \equiv true\ U\ \phi$
- $G\ \phi \equiv \neg F\ \neg\phi$
- $A\ \phi \equiv \neg E\ \neg\phi$

abreviaturas

- $\overset{\infty}{F}\ \phi = GF\ \phi$. *Infinitely often*: Em qualquer momento é possivel garantir que ϕ se venha a verificar.
- $\overset{\infty}{G}\ \phi = FG\ \phi$. *Always in the future*: A partir dum estado/momento, ϕ será continuamente verificada.

O poder das combinações

- $EF\ \phi$: existe uma execução que torna ϕ válida no futuro
- $AF\ \phi$: todas as execuções permitem validar ϕ no futuro
- $AGF\ \phi$ (ou $A\ \overset{\infty}{F}\ \phi$): a propriedade ϕ é verificada infinitamente frequentemente em todas as execuções
- $EG\ \phi$: existe uma execução em que ϕ é sempre verificada
- $AG\ \phi$: em todas as execução, ϕ é sempre verificada
- $AGEF\ \phi$: qualquer que seja a execução, é possivel a qualquer momento encontrar uma execução que valide ϕ no futuro: ϕ é sempre acessível.

PLTL e *CTL*

PLTL vs. *CTL*

- $PLTL = CTL^*$ sem as conectivas A e E. As fórmulas expressáveis são *fórmulas de caminho*.
- $CTL = CTL^*$ onde é exigido que as conectivas temporais de estado X, F e U estejam sempre no âmbito (*scope*) imediato de uma conectiva de caminho: EX, AX, EF, AF, $E\_U\_$, $A\_U\_$. São *fórmulas de estado*: a validade da formula por explorar depende exclusivamente do estado actual (e da exploração de todos os caminhos que saem dele).

PLTL e CTL

Qual escolher?

PLTL:

- Não consegue discursar sobre execuções alternativas que podem ocorrer nos momentos que constituem uma execução examinada σ. Por exemplo não pode expressar propriedades como " em qualquer momento, é possível validar a propriedade ϕ (ϕ é sempre acessível)"
- Excepto o ponto anterior, a lógica tem, em prática, uma expressividade apreciável.
- O Model checking é algoritmicamente pesado e complexo. Mas pode ser utilizado parcialmente mas de forma satisfatória "on-the-fly" (construção incremental e parcial do grafo de acessibilidade, etc..)

CTL:

- Não permite combinar e encadear os operadores de caminho entre eles. Não pode expressar propriedades ricas sobre os caminhos. O que se pode revelar uma limitação na prática.
- Por exemplo a existência dum caminho verificando ϕ infinitamente ($E \overset{\infty}{F} \phi$)
- O Model checking é algoritmicamente mais fácil e eficiente
- É possível utilizar extensões ao CTL que colmatam algumas das lacunas de expressividade (e.g. Fair CTL).

Classes de propriedades temporais

Quando pretendemos especificar e verificar um sistema é preciso saber listar e classificar as propriedades desejadas

As classes

- Acessibilidade (reachability): certa situação (desejada) pode ser atingida (é acessível).
- Segurança (safety): sob certas condições, certa situação (indesejável) nunca será atingida.
- Evolução/Produtividade (liveness): sob certas condições, certa situação acabará por acontecer.
- Equidade (fairness): sob certas condições, algo acontecerá infinitamente frequentemente (ou não)
- Ausência de bloqueio: é sempre possível ter um estado seguinte.

Classes de propriedades temporais

Porquê?

- Metodologia de especificação: é importante saber colocar-se (e responder a) questões como "quais são as propriedades de evolução do sistema? ", "quais são as propriedades de acessibilidade? "
- Economia da verificação: As propriedades de acessibilidade e de segurança são em geral mais importantes. É preciso assim saber bem determiná-las e dedicar-lhes mais atenção. Acontece que são também mais fáceis de verificar do que as outras.
- Metodologia da verificação: as lógicas, mas também os métodos de verificação, só se aplicam a determinados tipos de propriedades. As propriedades podem guiar a escolha do método de verificação (por exemplo a lógica ou a própria ferramenta).
- Metodologia da modelação: as estratégias de modelação e mesmo de transformação (dos autómatos) podem, e devem, tomar conta das propriedades visadas.

Exemplos

Acessibilidade

- Podemos ter ϕ (e.g. $\phi =$ secção cítica): $EF\ \phi$
- Não podemos ter ϕ: $\neg EF\ \phi$ ou $AG\ \neg\phi$
- Não se pode atingir o estado *crash*: $\neg EF\ crash$
- Pode-se entrar em secção crítica sem passar por ϕ: $E\ \neg\phi Usec\_crit$
- Pode-se sempre voltar ao estado inicial: $AG\ (EF\ init)$
- É possível voltar ao estado inicial: $EF\ init$

Exemplos

Segurança

- A segurança é a classe dual da acessibilidade. Pretende-se enunciar que nada de indesejável vai acontecer.
- Expressa-se essencialmente a custa de AG em CTL ou G em PLTL.
- A situação ϕ é impossível: $AG\ \neg\phi$ (CTL) ou $G\ \neg\phi$ (PLTL)
- Nunca dois sistemas estarão em secção crítica em simultâneo: $AG\ \neg(sec\_crit_1 \wedge sec\_crit_2)$ (CTL) ou $G\ \neg(sec\_crit_1 \wedge sec\_crit_2)$ (PLTL)
- Nunca haverá ϕ (e.g. *memory overflow*) : $AG\ \neg memory\_overvflow$ (CTL) ou $G\ \neg memory\_overvflow$ (PLTL)
- Enquanto não se colocar a chave, o carro não arrancará: $A\ \neg arranca\ W\ chave$ (onde $\phi W \psi \equiv (\phi U \psi) \vee G\phi$)
- $A\ \neg arranca\ W\ chave$ não é igual a $A\ \neg arranca\ U\ chave$. Esta última exige que a chave seja usada um dia.

Exemplos

Segurança

- Em algumas extenções sintácticas da CTL^*, é possivel expressar propriedades de segurança de forma cómoda usando operadores temporais referentes ao passado (e.g. X^{-1} para no "estado anterior" ou F^{-1}, num estado anterior) .
  $AG\ (arranca \implies F^{-1} chave)$
- Verificar que uma propriedade de segurança não é verificada exige uma exploração finita das execuções possíveis.
- Se não é satisfeita então existe uma execução *finita* que conduz o estado inicial para uma situação não desejada. Este caminho é um contra-exemplo.
- As ferramentas de Model-checking suportam indirectamente as formulas com referência ao passado via a técnica das variáveis de histórico (que arquivam as ocorrências de eventos passados de interesse). Nesta situação transforma-se uma propriedade de segurança numa propriedade de acessibilidade (ver bibliografia)

Exemplos

Ausência de bloqueio

- De forma geral: O sistema tem sempre forma de progredir, nunca se encontra numa situação em que não pode avançar.
- Genericamente: $AG\ EX\ true$.
- Quando o model-checker não suporta este tipo de formulação, é possível encontrar uma fórmula que expressa a ausência de bloqueio num determinado modelo/automato.

Exemplos

Evolução

- F é o operador que melhor descreve as propriedades de evolução (liveness)
- Qualquer pedido será satisfeito: $AG(pedido \implies AF\ tratado)$ (CTL) ou $G(pedido \implies F\ tratado)$ (PLTL)
- Se o elevador é chamado, este acabará por aparecer (semelhante ao caso anterior)
- $A\ \phi U \psi$ e $E\ \phi U \psi$ são propriedades de evolução sobre ψ

Exemplos

Equidade

- Pode ser considerada como uma propriedade de evolução repetida (e até mesmo de acessibilidade repetida).
- A cancela será levantada um numero de vezes infinita: $A \overset{\infty}{F}\ cancela\_levantada$.
- Se o pedido de entrada em secção crítica é feito infinitamente então o acesso é concedido infinitamente: $A(\overset{\infty}{F}\ pedido\_sec\_crit \implies \overset{\infty}{F}\ entrada\_sec\_crit)$, ou $A(\overset{\infty}{F}\ entrada\_sec\_crit \vee \overset{\infty}{G}\ \neg pedido\_sec\_crit)$
- CTL puro não consegue expressar propriedades de equidade. No entanto existem extensões simples ao CTL que permite tal expressão (*Fair-CTL* ou $ECTL^{+}$)

Mais Propriedades

Ver em:

http://patterns.projects.cis.ksu.edu

Table of Contents

Introdução

Sequência no tempo vs. duração

- Os autómatos clássicos permitem modelar sistemas e eventos, e raciocinar sobre a ordem no tempo em que esses ocorrem.
- Mas não conseguem capturar fenómenos como: do conjunto de eventos possíveis "estes" são os mais prováveis
- ou ainda: este evento seguirá e durará até 5 segundos.
- Vamos a seguir considerar uma extensão dos autómatos que tenta cobrir os aspectos *quantitativos* sobre o tempo (e.g. o exemplo do ponto anterior).

Composição dos autómatos temporizados

Composição e princípios de base

- Um autómato finito clássico, com a descrição dos estados de controlo do sistema por modelar
- Relógios que são variáveis/valores que permitem quantificar o tempo que decorre. A quantificação ou restricções temporais são colocadas nas transições.
- As transição são instantâneas. O tempo decorre nos estados.
- Para poder discursar sobre a duração de um evento, é preciso considerar dois estados: o estado que modela o inicio do evento e outro para o seu fim.
- vantagem: esta extensão não desnatura a modelação por autómatos.

Um exemplo simples

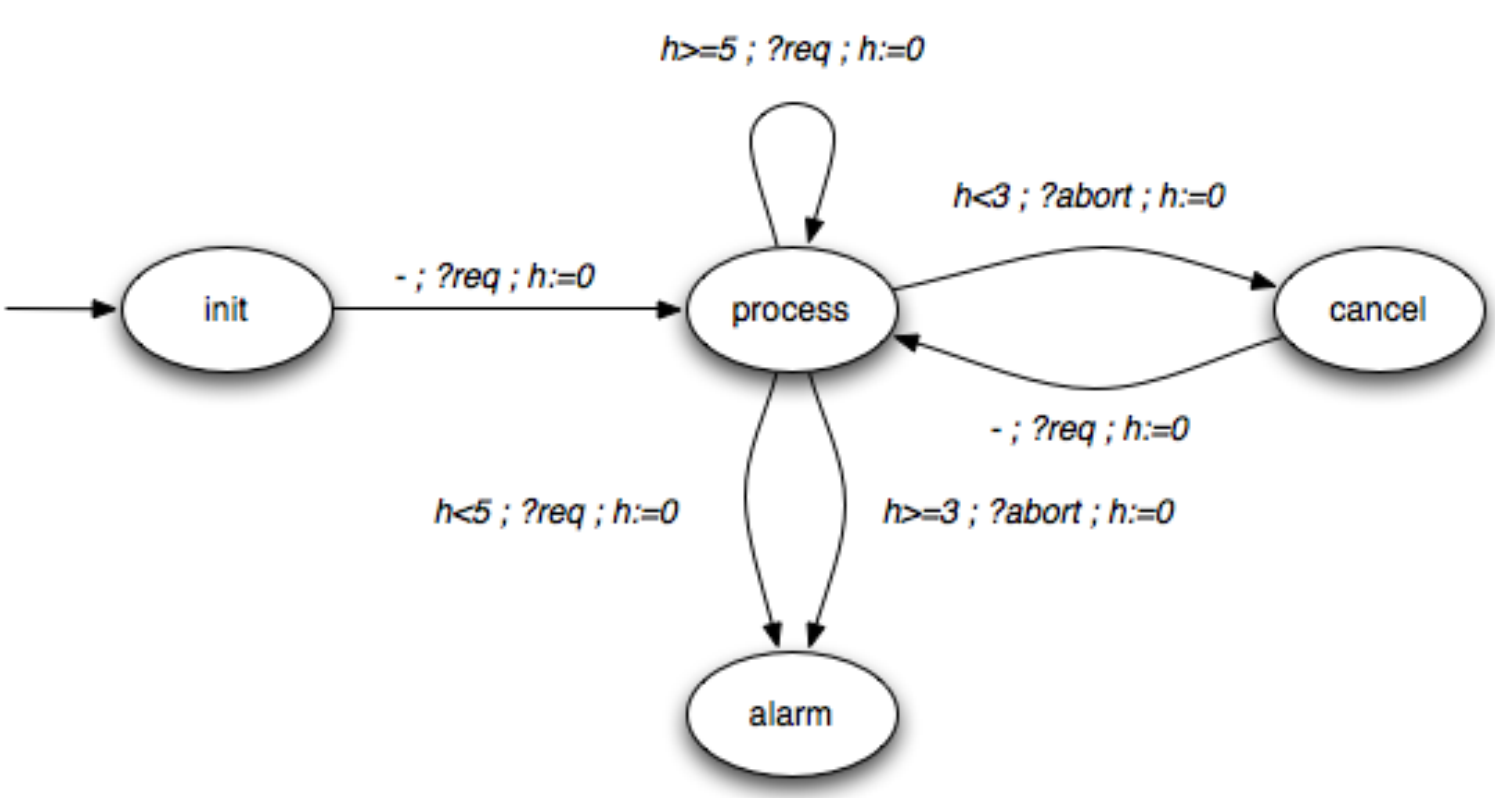


Figura: Exemplo básico

Relógios e transições

- Os relógios são variáveis reais de $\mathbb{R}^+$ com valor inicial 0.
- Progridem todas à mesma velocidade.
- As transições são assim constituídas pelas seguintes componentes:
  - uma guarda
  - uma etiqueta ou acção
  - acções de reinicialização de relógios
- Assim, devido às reinicializações possíveis, os relógios medem prazos e não o tempo.
- O sistema modelado funciona como se possuísse um relógio global interno que permite a sincronização com os relógios utilizados.

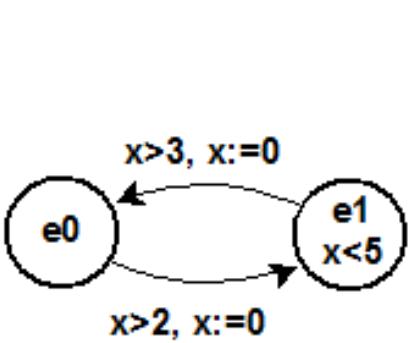
Configuração e execução

- configuração = estado de controlo activo + valor de cada relógio. Seja μ uma aplicação que associa a cada relógio o seu valor. esta é designada de *valoração*. Assim, configuração = (q, μ).
- O sistema muda de configuração das duas formas seguintes:
  - decorre um prazo $d \in \mathbb{R}^+$. Todos os relógios são incrementados de acordo. *Transição de prazo*
  - o autómato é activado. Uma transição do autómato é executada. Os relógios por reinicializar são reinicializados a zero. O valor dos outros relógios são mantidos. *Transição discreta* ou *transição de accção*.
- $(init, 0) \rightarrow (init, 8.8) \overset{?req}{\rightarrow} (process, 0) \rightarrow (process, 9.4) \overset{?req}{\rightarrow} (process, 0) \rightarrow (process, 3.4) \overset{?req}{\rightarrow} (alarm, 0)$

Execuções e Trajectórias

- Uma execução de um autómato temporizado, também designada de *trajectória*, pode ser vista como uma aplicação ρ dos reais positivos para as configurações.
- $\rho(0)$ representa assim a configuração no instante global 0. $\rho(t)$ representa a configuração no instante global t.
- No exemplo anterior, $\rho(9.9) = (process, 1.1)$
- O ponto de vista das trajectórias é de um tempo global externo ao sistema enquanto as valorações μ dão uma perspectiva local e interna ao sistema.

Um exemplo simples



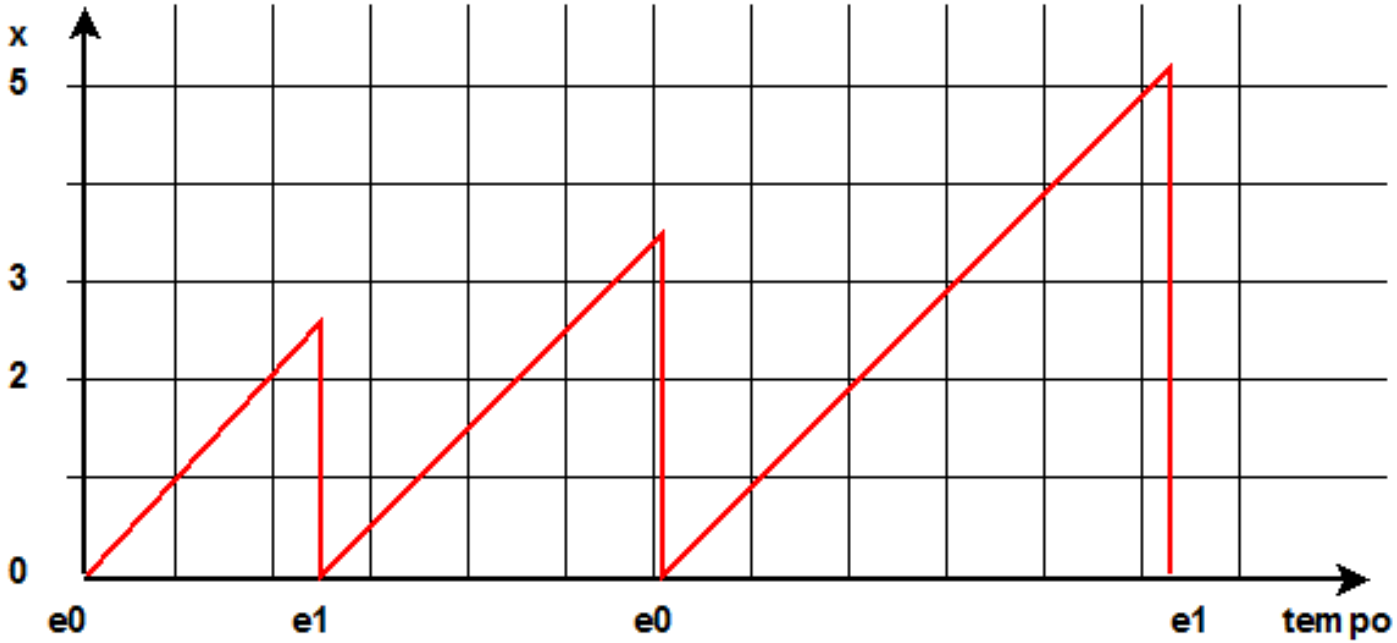


Figura:

Redes de autómatos temporizados e sincronização

Execução duma rede de autómatos temporizados (RAT)

- Uma rede de autómatos temporizados = sistema completo = composição adequada (sincronizada) de autómatos temporizados (visto como sub-sistemas)
- Uma execução de uma RAT = sequência (eventualmente infinita) de configurações onde, configuração = registo do estado de controlo de cada um dos autómatos e valores de cada um dos relógios.
- Duas formas de sequencializar configurações:
  - Deixar correr um prazo d. Todos os relógios são incrementados de d.
  - Escolher uma acção aplicável. Só o controlo dos estados ligados a esta acção poderá mudar. Conforme especificado pela acção, certos relógios poderão ser reinicializados. Os outros permanecem intactos.
- Assim todos os autómatos são executados em paralelo e à mesma velocidade.
- Processo de sincronização: Realizado, como no caso simples, pelas transições.
- Partilha de relógios entre vários autómatos. É possível, mas é preciso ter atenção que a reinicialização poderá afectar os autómatos que partilham o recurso.
- Notação: configuração = (q, μ) onde q é um vector de estado (a lista dos estados activos, um por em cada autómato da rede) e μ uma valoração dos relógios.

Variantes e extensão do modelo de base

Invariantes

- Invariantes de estado: condições (sobre os relógios) impostas ao nível dos estados.
- Invariante: representa a noção de *progresso/necessidade* (à diferença das guardas nas transições = *possibilidade*)
- Semântica: um invariante P num estado q significa que o controlo só pode estar em q quando P é verificada.
- uma configuração é *autorizada* se os invariantes dos seus estados de controlo são todos verificados.
- O tempo só pode decorrer numa determinada configuração enquanto esta permanecer autorizada. Quando esta deixa de o ser, uma transição elegível deve ser accionada.
- *livelock* (deadlock temporizado): nenhuma configuração autorizada é atingível.

Variantes e extensão do modelo de base

Urgência

- Por razões de conveniência, podemos querer que certas acções sejam tomadas sem que o tempo decorra (i.e. de forma urgente) .
- Transição/canal urgente:
  - Não há espera se uma transição urgente é elegível.
  - Neste caso, não se pode colocar guardas sobre relógios nas transições urgentes.
- É também possível associar a noção de urgência a estados.
- Vantagem: Os ganhos são mais em termos de dimensão do autómato resultante (menos relógios necessários para a modelação) do que em termos de expressividade.
- Mecanismo semelhante: estados com compromissos (*committed locations*) para especificar sequenciais de acções atómicas.
- mais detalhe, mais adiante....

Variantes e extensão do modelo de base

Sistemas híbridos

Generalização dos autómatos temporizados: os relógios são substituídos por variáveis dinâmicas que podem evoluir com ritmos diferentes ou respeitar leis de evolução diferentes da do tempo (altura, temperatura, velocidade, ou até tempo).

Lógica Temporal Temporizada

Sintaxe (formato BNF) da lógica *TCTL*

$$
\begin{array}{lll}
\phi, \psi ::= & P_1 \mid P_2 \mid \cdots & \text{(proposições atómicas)} \\
& \neg\phi \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \implies \psi) \mid \cdots & \text{(conectivas lógicas classicas)} \\
& EF_{\sharp k}\ \phi \mid EG_{\sharp k}\ \phi \mid E(\phi\ U_{\sharp k}\ \psi) \mid & \text{(conectivas temporais)} \\
& AF_{\sharp k}\ \phi \mid AG_{\sharp k}\ \phi \mid A(\phi\ U_{\sharp k}\ \psi) \mid &
\end{array}
$$

onde $\sharp \in \{<, \leq, =, >, \geq\}$ e $k \in \mathbb{Q}$

- Como o tempo é contínuo, não faz sentido falar da configuração seguinte. Daí não considerarmos a conectiva X.
- $AG = AG_{\geq 0}$, etc.
- Não apresentamos aqui a semântica da TCTL. Mas informalmente: uma configuração verifica, por exemplo, $E\phi U_{\leq 3}\psi$ se existe uma trajectória iniciada nesta configuração e um tempo $t \leq 3$ tal que $\rho(t) \models \psi$ e $\forall t' < t,\ \rho(t') \models \phi$

Exemplos de propriedades

- Se um problema ocorre, o alarme é accionado imediatamente e por um tempo mínimo de 5 unidade de tempo: $AG(pb \implies AG_{\leq 5} alarme)$
- O que será $AG(\neg longe \implies AF_{\leq 7} longe)$?
- Bounded liveness = segurança:
  - O programa termina em menos de 10 unidades de tempo: $AF_{\leq 10}(end)$
  - Qualquer pedido é satisfeito em menos de 5 unidades de tempo: $AG(req \implies AF_{\leq 5} satisfied)$
- Pontualidade: Existe uma execução que permite satisfazer um pedido em exactamente 11 unidades de tempo. $EG(req \implies AF_{=11} satisfied)$
- Periodicidade:
  - $AG(AF_{=15} event)$ ou $AG(event \implies \neg event U_{=15} event)$
  - Intervalo mínimo de periodicidade: $AG(event \implies \neg event U_{\geq 150} event)$
  - Intervalo de periodicidade: $AG(event \implies ((\neg event U_{\geq 150} event) \wedge (\neg event U_{\leq 350} event))$

Autómato de teste e afins

Dada uma propriedade ϕ por verificar sobre uma RAT D, é possível transformar a verificação de ϕ num problema de acessibilidade. Para isso, duas técnicas:

- Autómato de teste: desenhar um autómato T_ϕ que interage com D de tal forma que atinge um estado q caso a propriedade ϕ seja violada. Afirmar que D respeita ϕ é verificar que q não é acessível. (ver: Model Checking via Reachability Testing for Timed Automata)
- Alteração do modelo de tal forma que a propriedade possa ser transformada numa propriedade de acessibilidade (juntar relógios, variáveis, guardas, etc., de acordo com ϕ). (ver: Formal Design and Analysis of a Gear controller).

Algoritmia e Model-Checking

Problemática e Solução

- A semântica informalmente apresentada sugere um sistema de transição subjacente infinito por natureza.
- A verificação em tais sistemas só é possível via uma discretização adequada do sistema (a luz da teoria da interpretação abstracta): autómato das regiões e das zonas.
- Feita esta discretização, a algoritmia clássica do Model-cheking de tipo CTL pode ser aplicada.
- Região: Partição (área) temporal (cada relógio é uma dimensão por considerar) na qual o comportamento do sistema é globalmente a mesma. Estas regiões são os estados do autómato das regiões.
- Zonas: conjunto de regiões contíguas que se comportam de forma homogénea.

Algoritmia e Model-Checking

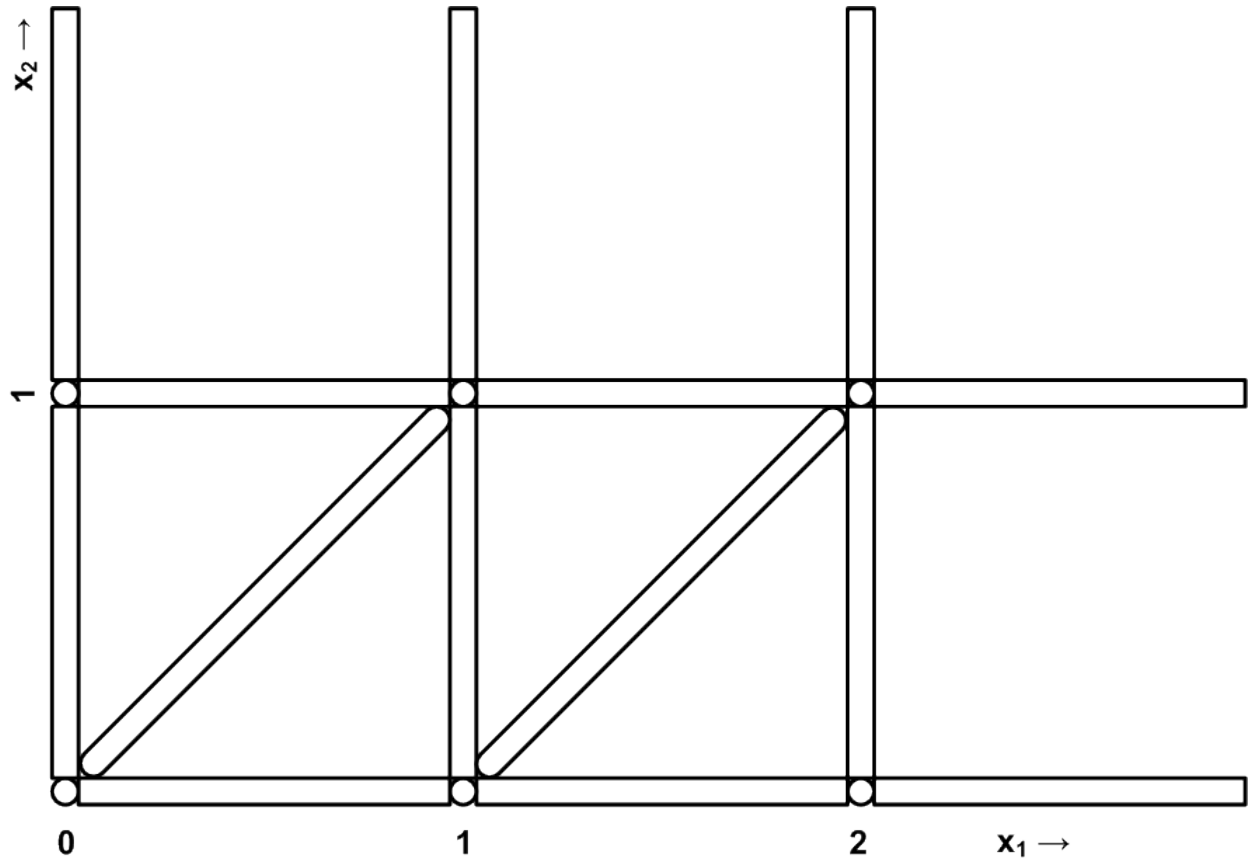


Figura: 28 regiões para dois relógios x_1 e x_2 e restrições sobre os valores $\{0, 1, 2\}$ para x_1 e $\{0, 1\}$ para x_2

Algoritmia e Model-Checking

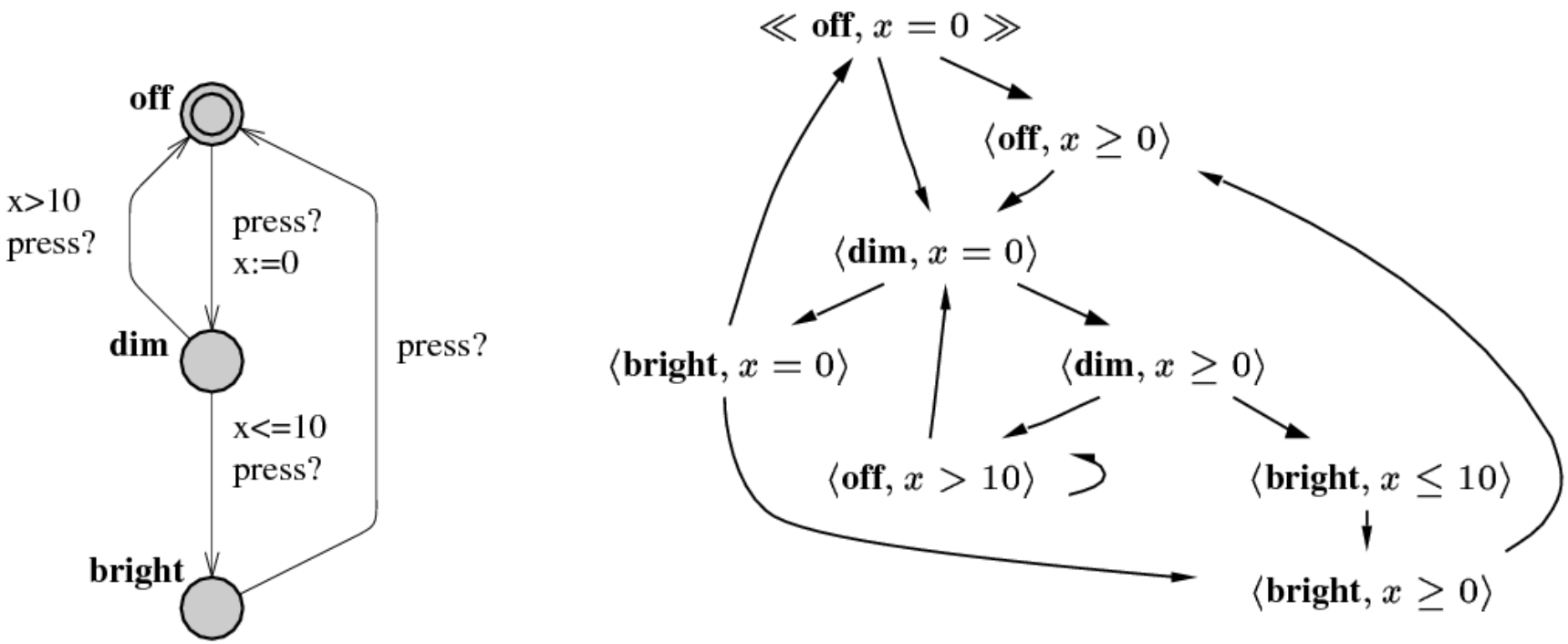


Figura: Um autómato temporizado e o seu autómato das zonas. (autómato de regiões $\approx$ 50 estados)

Conclusão

- No mundo da verificação de modelo, os autómatos temporizados formam um paradigma recente e em plena evolução.
- No entanto: já existam muitas aplicações bem sucedidas (ver sistemas como o hytech, kronos ou em www.uppaal.org)
- Área muito activa e palco de muita investigação.
- No que diz respeito à vertente "Theory, Algorithms and Tools":
  - melhores estruturas de dados de suporte para a representação e manipulação das zonas (e.g. DBM - Difference Bound Matrix)
  - melhores algoritmos de verificação
  - extensões ao modelo de base (e.g. autómatos híbridos, etc..)

  Numa frase: procura de melhores abstracções (no sentido da "interpretação abstracta")

Table of Contents

Apresentação curta

- Simulador e Model-checker *TCTL* para autómatos temporizados com variáveis sobre intervalo de inteiros, com suporte a matrizes.
- Desenvolvido principalmente pelas universidades de **Upp**sala e de **Aal**borg.
- Site: www.uppaal.org.
- Suporta de raíz a verificação de propriedades de: reachability, safety, (bounded) liveness, deadlock freeness.
- quando não directamente suportadas, as propriedades podem ser expressas e verificadas com base nas técnicas do autómato observador e enriquecimento do autómato (com relógios e varáveis).

Apresentação curta

- Um autómato/processo = instanciação de um *template*. Os templates podem ser parametrizados (com (vectores de) variáveis inteiras, constantes).
- Declarações globais, locais a um autómato, locais a uma transição.
- Constantes, variáveis em intervalo de inteiros, matrizes.
- Funções (de manipulação de variáveis), sintaxe *à la C*.
- Sincronizações:
  - Por canal (mensagem) binário.
  - Por canal múltiplo (um emissor, vários receptores).
  - Por variável.
- Transições com guardas, sincronizações por mensagens e actualizações de variáveis.
- Gestão do tempo e do controlo: transições urgentes, estados urgentes e *committed*. Invariantes de estados.

Guardas, Invariantes, Atribuições

Guardas

- Sem efeitos laterais, bem tipificado e resultado = booleano
- Só pode contemplar: (vectores de) variáveis de relógio, variáveis inteiras, constantes.
- Relógios (ou diferenças de relógios) só podem ser comparados a expressões inteiras.
- Guardas sobre relógios podem ser agrupadas em conjunções (as disjunções são autorizadas em condições inteiras)

Atribuições/actualizações

- Tem efeitos laterais e bem tipificado
- Só pode contemplar: (vectores de) variáveis de relógio, variáveis inteiras, constantes.
- os relógios só podem receber valores inteiros.

Invariantes

- São conjunções de condições da forma $x < e$ ou $x \leq e$ em que x : relógio e e : expressão inteira.

Sincronizações por canais

Canais binários

- declaração :

  `chan c1 , c2, c3[10];`
- Assumindo `chan a;`

  `a!` = emissão

  `a?` = recepção
- **Duas** transições em processos diferentes podem sincronizarem-se via canais se uma emite e outra recebe no mesmo canal.
- emissão sem recepção = bloqueio

Canais múltiplos

- declaração :

  `broadcast chan c1 , c2, c3[10];`
- Assumindo `broadcast chan a;`

  `a!` = emissão

  `a?` = recepção
- **Um conjunto de** transições em processos diferentes podem sincronizarem-se via canais se uma emite e as outras todas recebem no mesmo canal.
- As transição de recepção *devem* sincronizar se estão em situação de receber.
- não bloqueia

Urgência e compromisso

Canais urgentes

- Não há espera se a transição de sincronização pode ser escolhida.
- As guardas sobre relógios não são autorizadas (só sobre valores/variáveis inteiras).
- Relógios (ou diferenças de relógios) só podem ser comparados a expressões inteiras.
- Guardas sobre relógios podem ser agrupadas em conjunções (as disjunções são autorizadas em condições inteiras)
- `urgent chan a,b, c[10];`

estado urgente

- Não há espera. O tempo está "congelado".
- Utilização pode reduzir a necessidade de relógios.

Committed location

- Não há espera.
- Quando um estado *committed* tem o controlo, a próxima transição escolhida **deve** sair deste estado.
- Permite a definição do encadeamento de acções atómicas.
- Permite uma redução considerável do espaço de estado.

Fórmulas

- Restrições à *TCTL*: UPPAAL não permite o aninhamento de fórmulas de caminho.
- `[]` $= G$, `<>` $= F$
- *Reachability* $=$ `E<>`ϕ
- *Safety* $=$ `A[]`ϕ ou `E[]`ϕ
- *Liveness* $=$ `A<>`ϕ ou ϕ `-->` ψ ($\equiv AG(\phi \implies AF\psi)$)
- *Bounded liveness* ϕ `-->`$_{\leq t}$ ψ (em Uppaal, ϕ `-->` $(\psi \wedge c \leq t)$)

Fórmulas

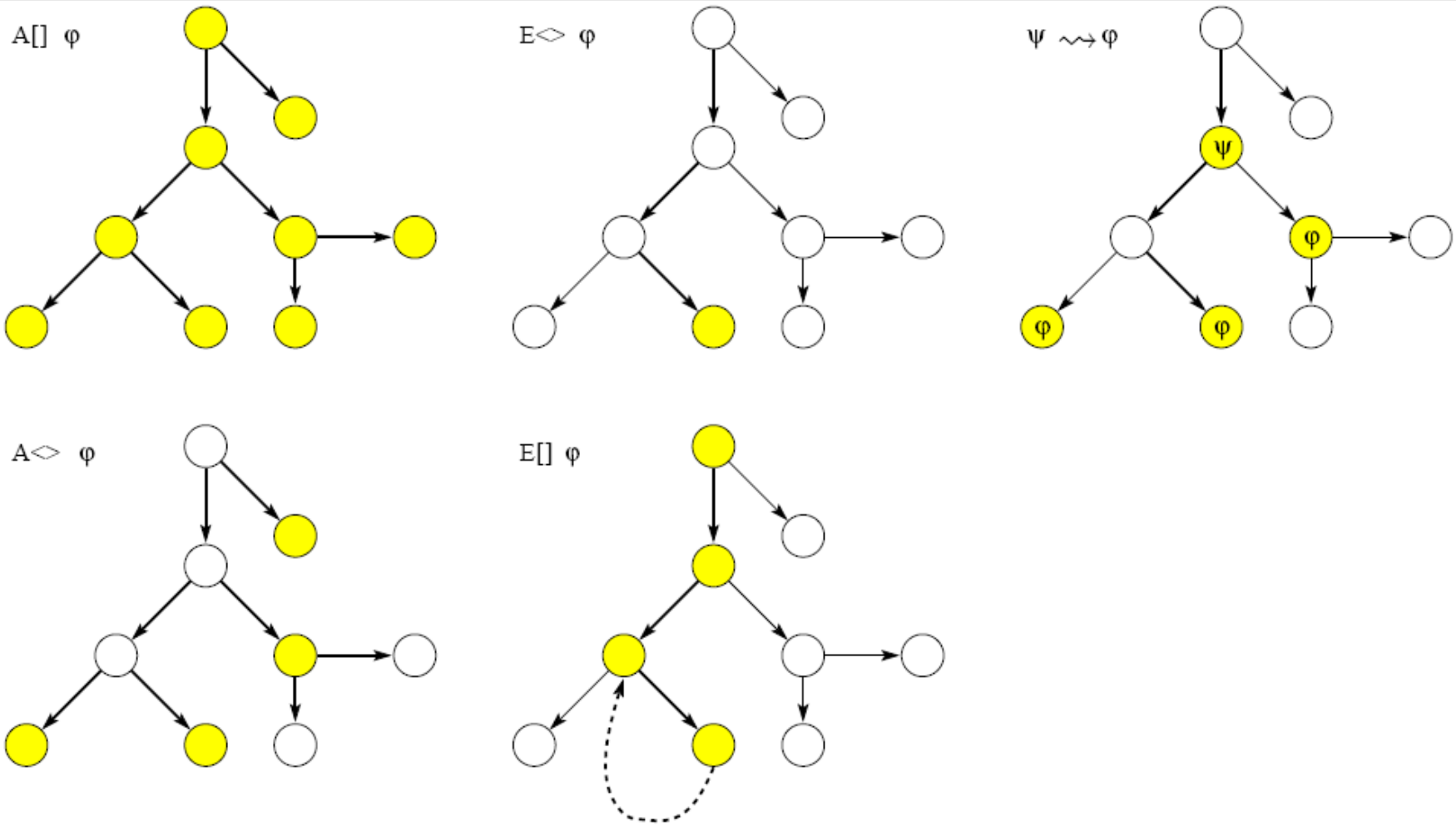


Figura: Fórmulas de caminho suportadas

Fórmulas

- As fórmulas lógicas ϕ e ψ devem ser bem tipificadas, não conter efeitos laterais e reduzirem-se a um valor booleano.
- Estas fórmulas só se podem referir a (vectores de) variáveis inteiras, constantes, relógios e estados.

Extensões

- Selecção: nas transições. Permite a escolha não determinística. `x : int [10,50]`
- Tipos: Definição de tipos, tipos estruturados.
- Definição de funções auxiliares "a la C"para as actualizações e testes.
- Quantificação universal e existencial: forall/exists (x:int [10,50]) ϕ.

Resultados

Examples and Case Studies
(retirado de
http://www.it.uu.se/research/group/darts/uppaal/examples.shtml)

Demonstração - Casos de estudos e exercícios

```
init)  boolean lockA = lockB = false;
       int turn = A; // or B
```

	Thread **A**	Thread **B**

pre)	x: lockA = true;
	y: turn = B;
	z: while(lockB
	w: && turn != A);
		x: lockB = true;
		y: turn = A;
		z: while(lockA
		w: && turn != B);

post)	x1: lockA = false;	x1: lockB = false;

Figura: Algoritmo de Peterson

```
init)  /* same as in Peterson's algorithm */
```

	Thread **A**	Thread **B**

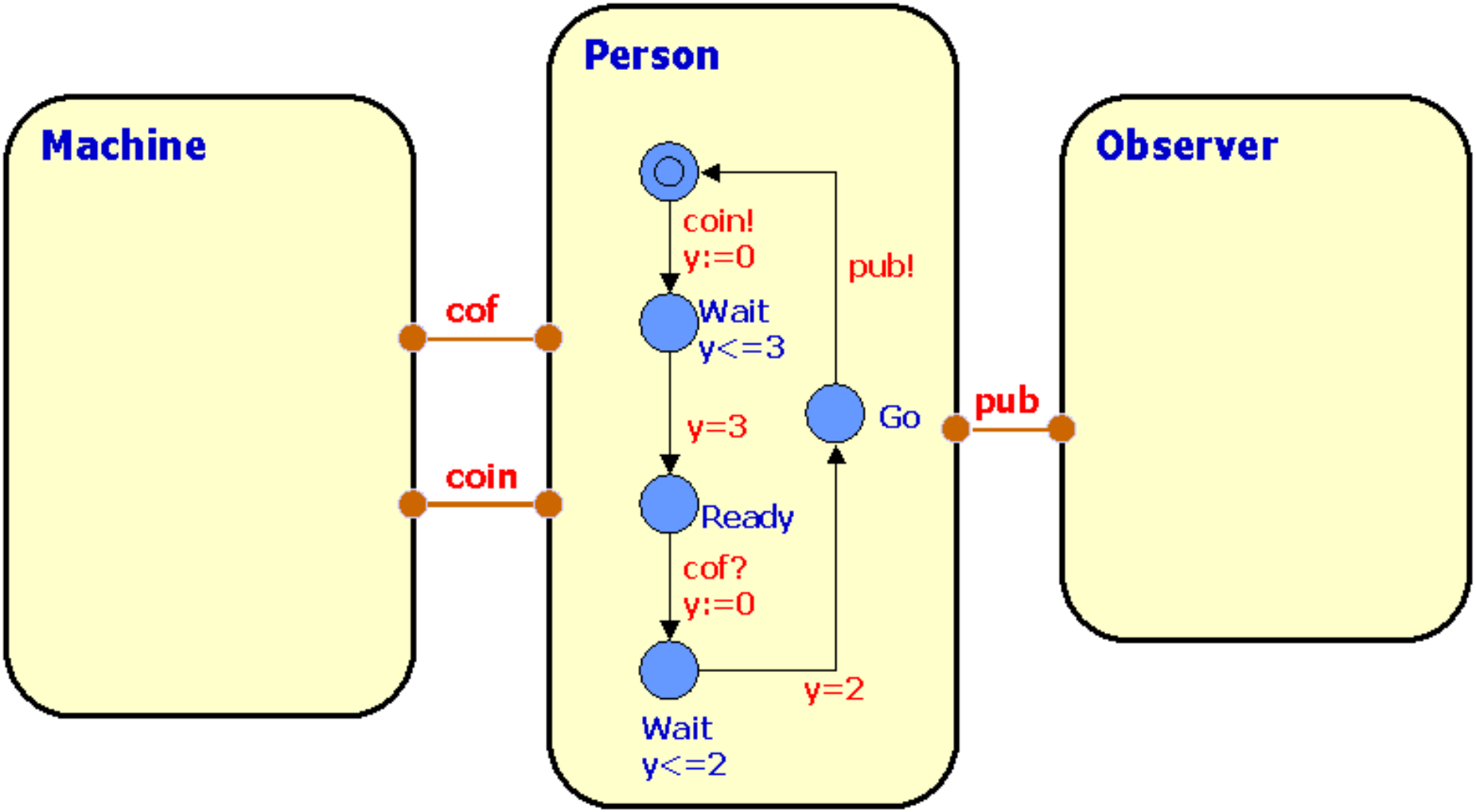
pre)	x: lockA = true;
	y: while(lockB) {
	z: if (turn != A) {
	u: lockA = false;
	v: while(turn != A);
	w: lockA = true;
	}
	}
		x: lockB = true;
		y: while(lockA) {
		z: if (turn != B) {
		u: lockB = false;
		v: while(turn != B);
		w: lockB = true;
		}
		}

post)	x1: lockA = false; y1: turn = B;	x1: lockB = false; y1: turn = A;

Figura: Algoritmo de Dekker

Máquina de café

- O **utente** tenta repetidamente: inserir uma moeda, extrair o café servido e avisa o observador que obteve o que pretendia. Entre cada acção o utente espera durante um certo tempo antes de continuar.
- Depois de receber uma moeda, a **máquina** leva um determinado tempo para preparar o café. Se o café não for levantado pelo utente após um determinado prazo, a máquina deverá reagir e entrar em "time-out".
- O **observador** deverá emitir uma queixa se o intervalo de tempo entre dois avisos do utente for superior a 8 unidades de tempo.


Person
Machine
Observer
coin!
y:=0
pub!
Wait
y<=3
cof
y=3
Go
pub
coin
Ready
cof?
y:=0
y=2
Wait
y<=2

Máquina de café

- O sistema é infelizmente parcialmente descrito. Na descrição anterior existe a frase: *Se o café não for levantado pelo utente após um determinado prazo, a máquina deverá reagir e entrar em time-out.*
- Esta frase é um exemplo de *under-specification*. Por exemplo, o que deve fazer a máquina se o prazo for atingido? deitar fora o café? Neste caso iremos presenciar um bloqueio (porque se entraria em contradição com a sequência de acções realizadas pelo utente)
- A solução passa por entregar o café, mas só depois do *time-out*. Em termos práticos isto poderá significar colocar o café numa zona de *entrega* particular.

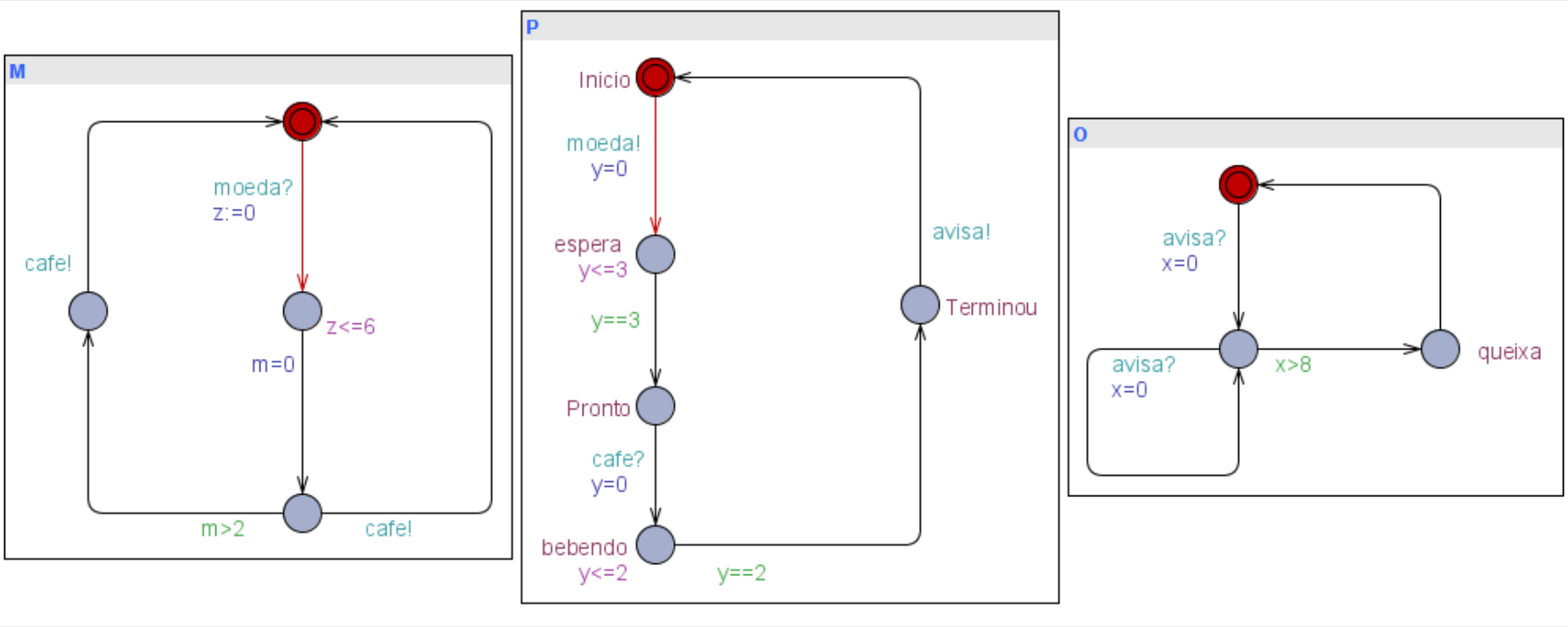
Máquina de café

Verificação de:

- A[] not deadlock
- E<> Observer.Complain

Que valores dos prazos invalidam ou validam a última propriedade?

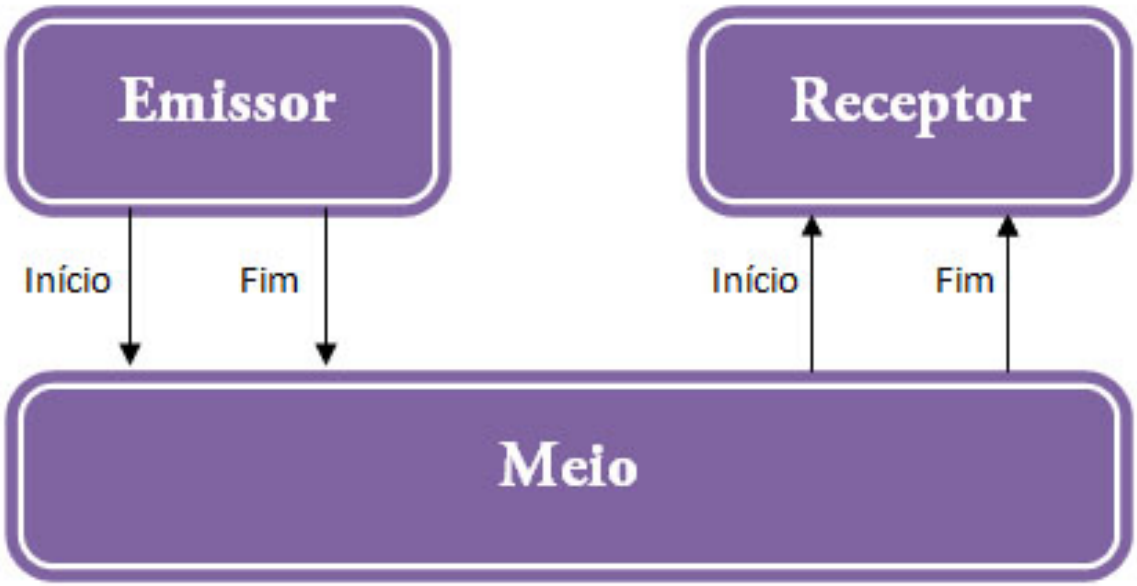
uma máquina de café temporizada



Num contexto de comunicação, simplista, é sugerida a implementação de um protocolo com três componentes interligadas: emissor, meio e receptor (ver figura a seguir).

- O emissor transmite uma mensagem de *tamanho* fixo. Esse tamanho corresponde ao tempo entre o início do envio e o fim do envio da mensagem.
- O meio corresponde à componente central responsável pela passagem da mensagem do emissor para o receptor. Este meio introduz um *atraso* fixo na comunicação que corresponde ao tempo entre o início do envio por parte do emissor e o início da recepção por parte do receptor ou o fim do envio por parte do emissor e o fim da recepção por parte do receptor.
- O receptor recepciona a mensagem vinda do meio de comunicação.

Figura: Esquema das componentes do protocolo.

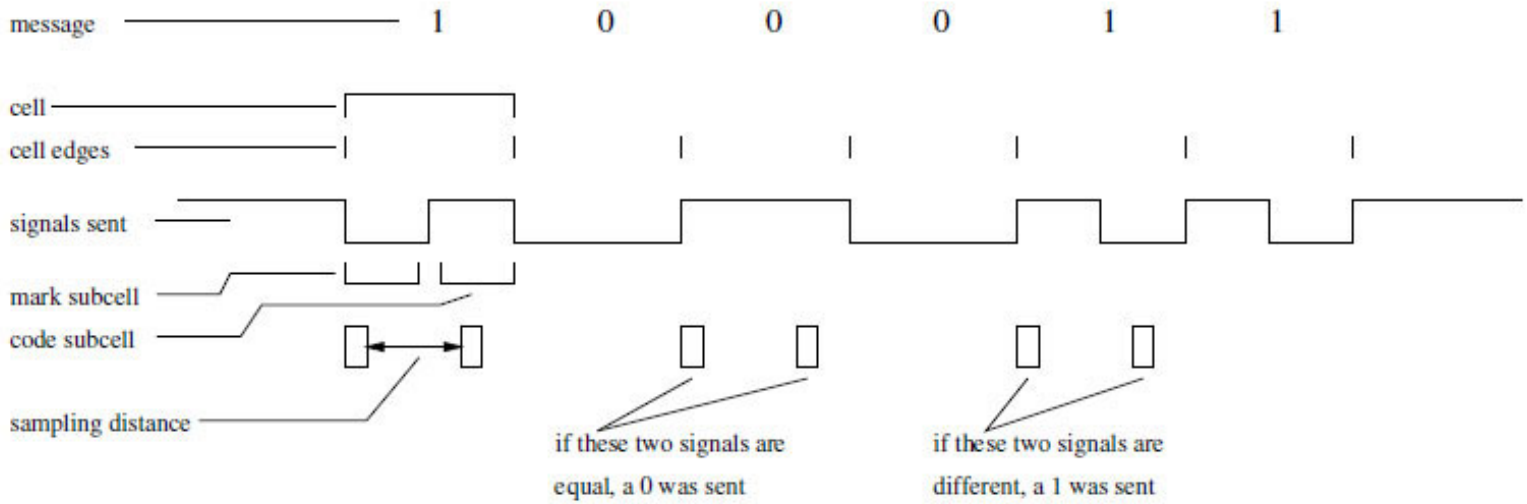


- Nesta versão parte-se do princípio que o tamanho é menor que o atraso ($tamanho < atraso$), i.e. o meio só transmite a mensagem para o receptor depois do envio (por parte do emissor) ter sido finalizado.
- É recomendada a utilização de constantes inteiras para o *tamanho* e para o *atraso*. O meio não deve, ter conhecimento do *tamanho* da mensagem, nem o emissor do *atraso*. O sistema modela-se com uma rede de autómatos sincronizados, utilizando o início e fim de comunicação do emissor e do receptor como sincronizações com o meio.
- Pretende-se ainda que o modelo seja verificado garantido a não existência de deadlock's e identificando qual o tempo decorrido do início do envio até ao fim da recepção da mensagem.

Fofocas...

Modele e analise o seguinte problema em UPPAAL.
Considere um número determinado de raparigas que gostam de partilhar segredos e coscuvilhices.
Cada uma das raparigas conhece um segredo particular (diferente dos segredos conhecidos pelas outras colegas).
Cada rapariga tem acesso a um telefone que pode usar para ligar a uma colega e partilhar os seus segredos.
De cada vez que duas raparigas falam, estas trocam todos os segredos que conheçam no momento da chamada. Não existe a possibilidade das chamadas em modo conferência.
A sua tarefa é então a seguinte:

- Modele o problema como uma rede de autómatos temporizados. Instancie o modelo para um conjunto de 4 amigas. Encontre o numero mínimo de chamadas para que todas conhecem todos os segredos.
- Refina o modelo por forma a que cada chamada acabe no fim de 60 segundos. Que tempo mínimo é necessário para a troca de todos os segredos.



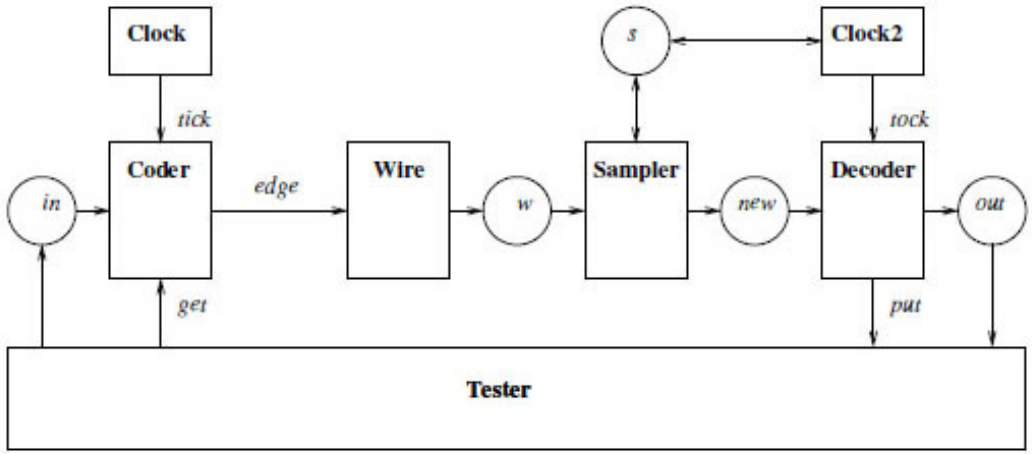


Figura: Biphase Mark Protocol