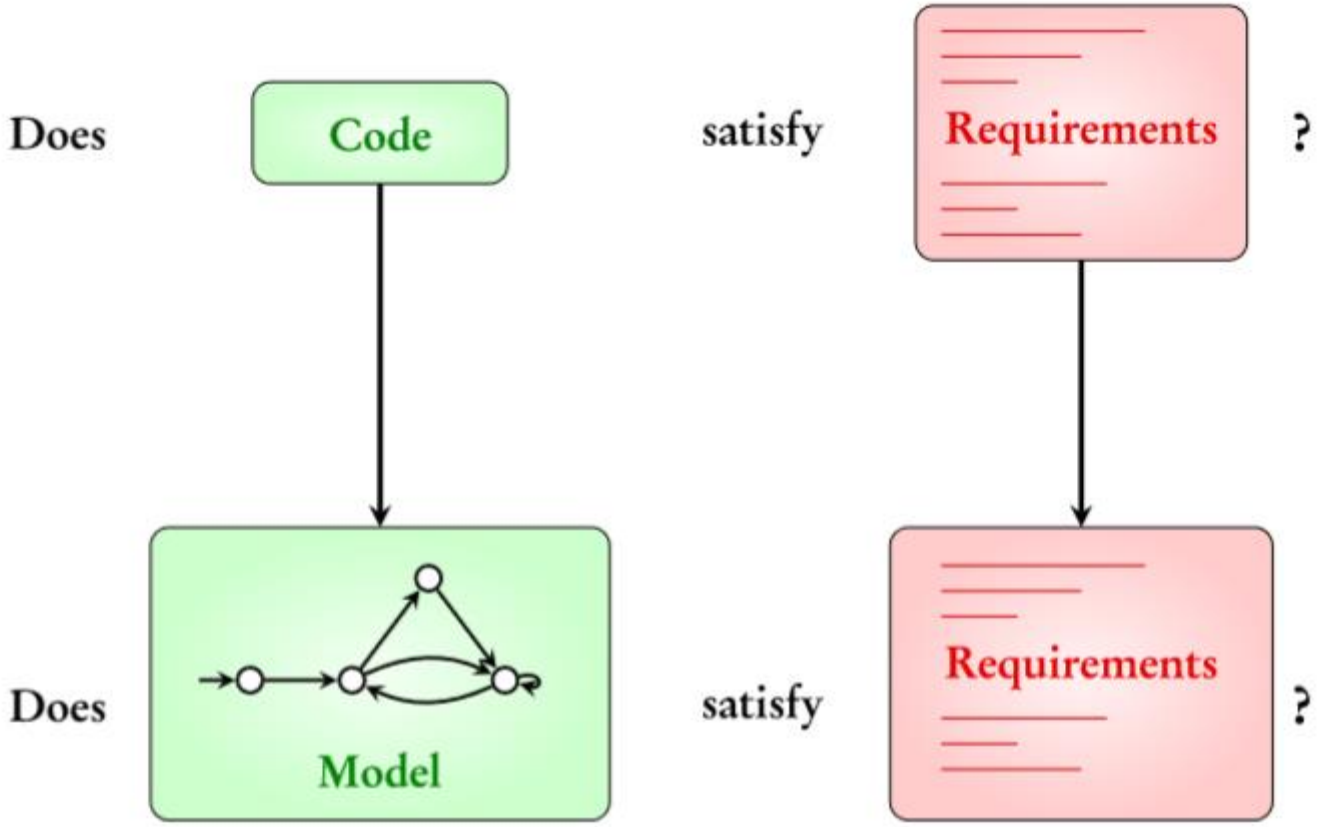
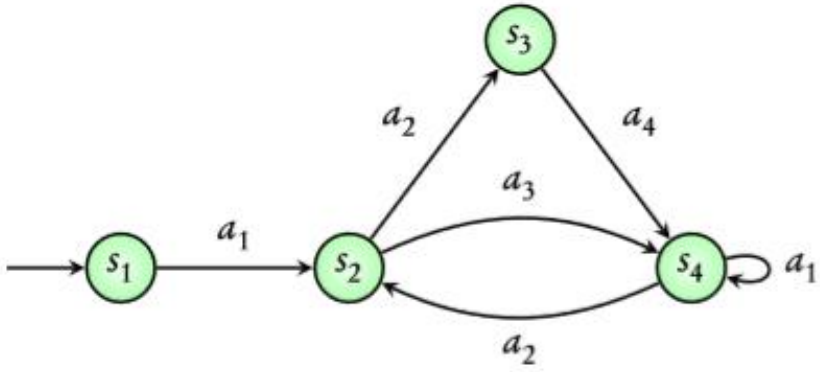
Model Checking



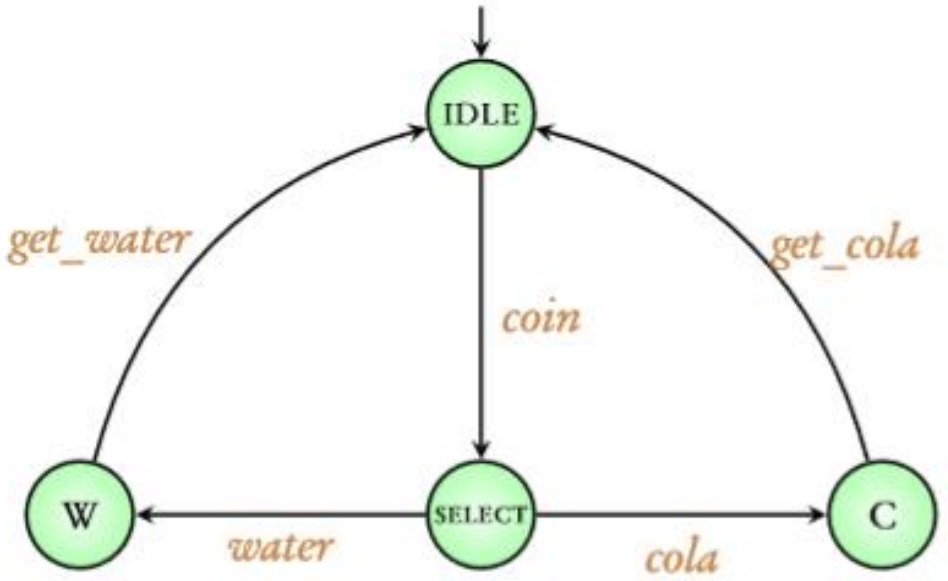
O Modelo

Sistema de Transições

- Estados
- Ações
- Transições
- Estado Inicial



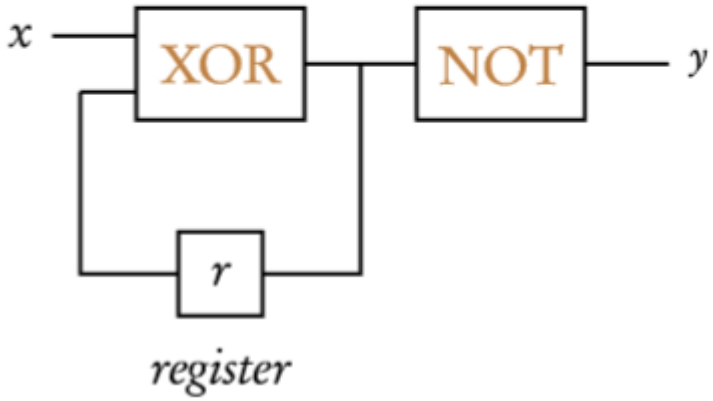
Modelando uma Vending Machine

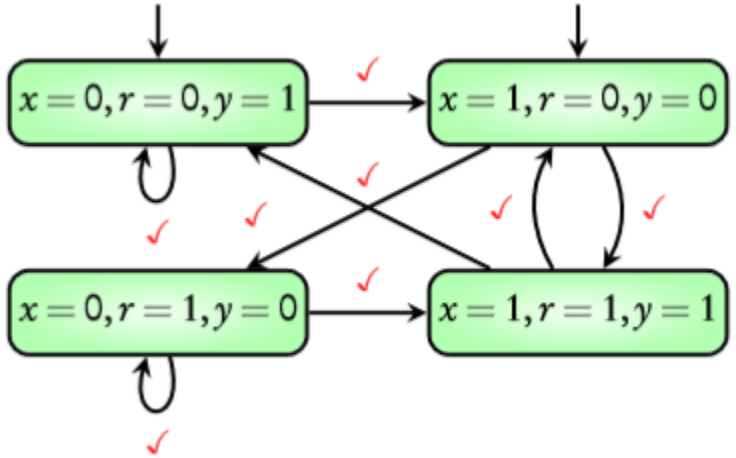


Execução: Idle → Select → W → Idle → Select → W → Idle → Select → C → Idle ...

Modelando um Circuito

$$y = \text{NOT}\,(\,\text{XOR}\,(x, r)\,)$$
$$r_{next} = \text{XOR}\,(x, r)$$

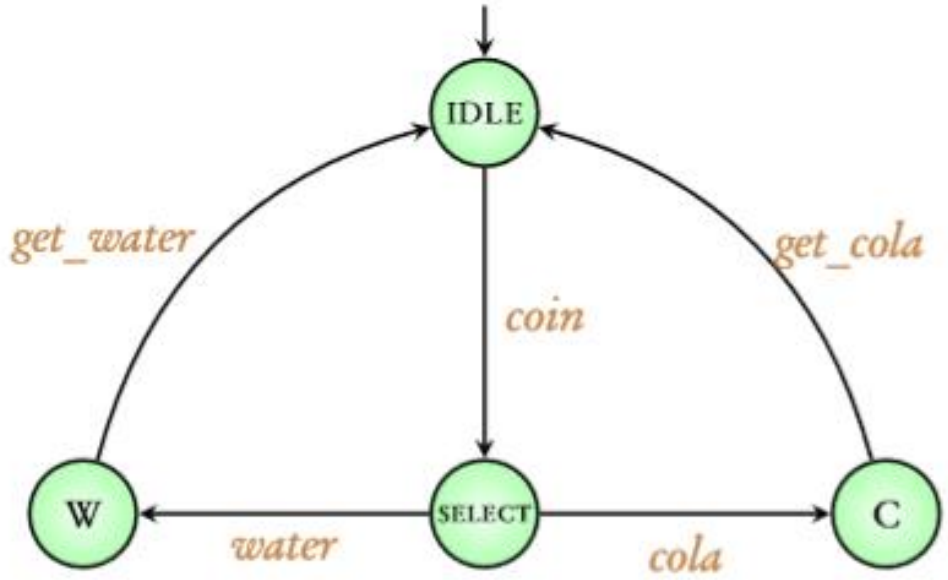




x	1	1	0	1	1	1	0	0	1	0
r	0	1	0	0	1	0	1	1	1	0
y	0	1	1	0	1	0	0	0	1	1

Programas com dependência de dados

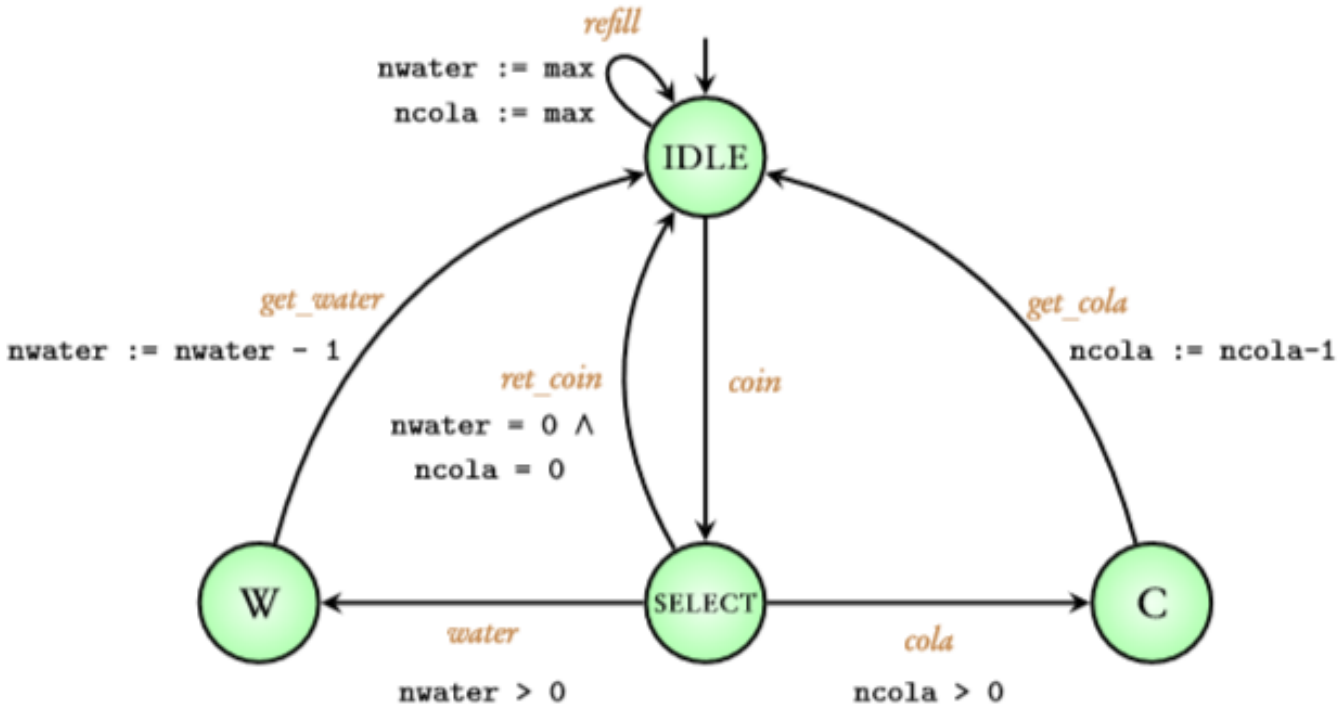
- Variáveis + Transições condicionais + Atribuições
- Refinando o modelo da Vending Machine



Grafos do programa

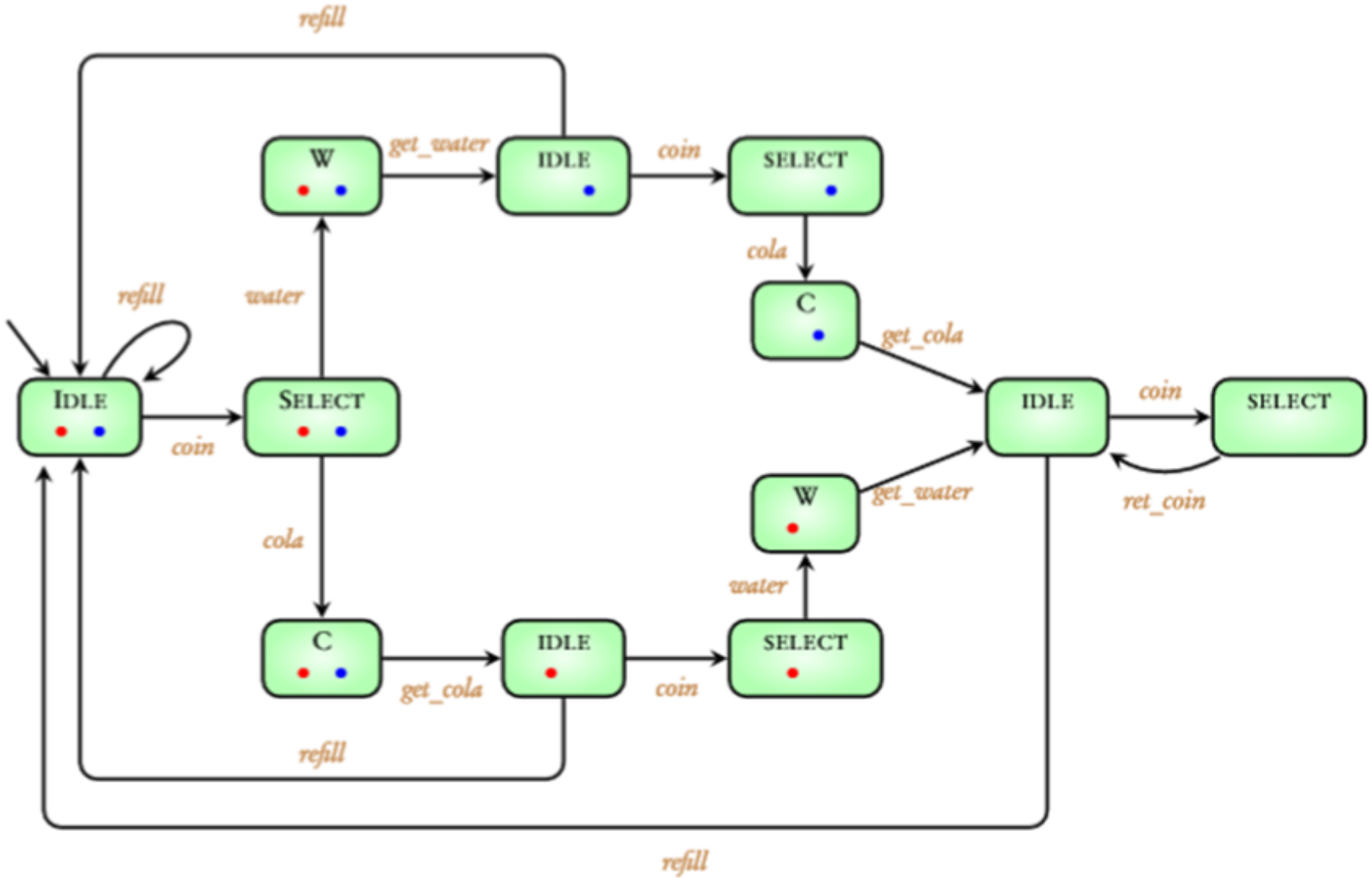
Variables:

`nwater, ncola, max`



Sistema de transição com condições iniciais

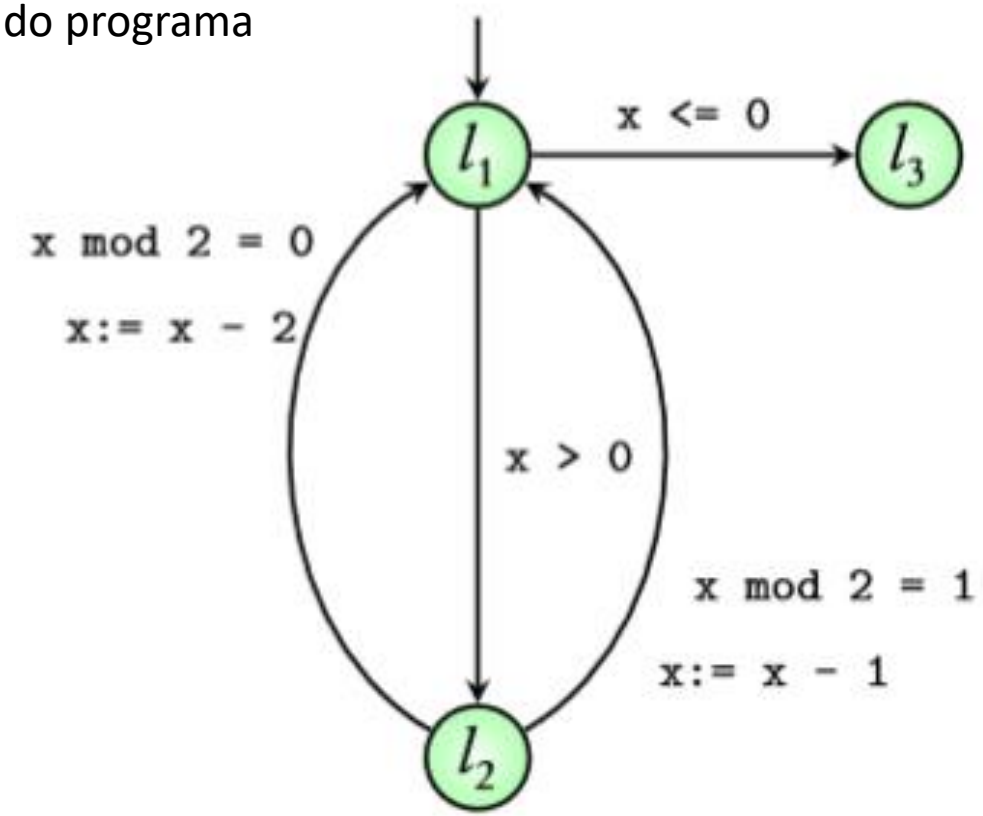
$\texttt{max} = 1$



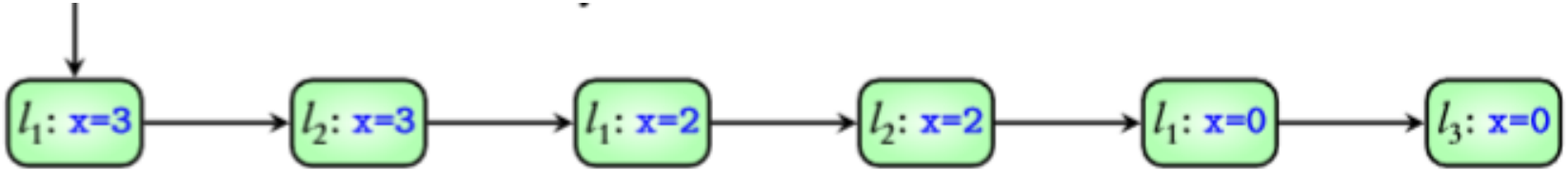
- Outro exemplo

```
l1: while (x > 0)
        if (x mod 2 = 0)
l2:             x := x - 2
        else    x:= x - 1
        ...
l3:
```

Grafo do programa



Sistema de transição com condição inicial x=3



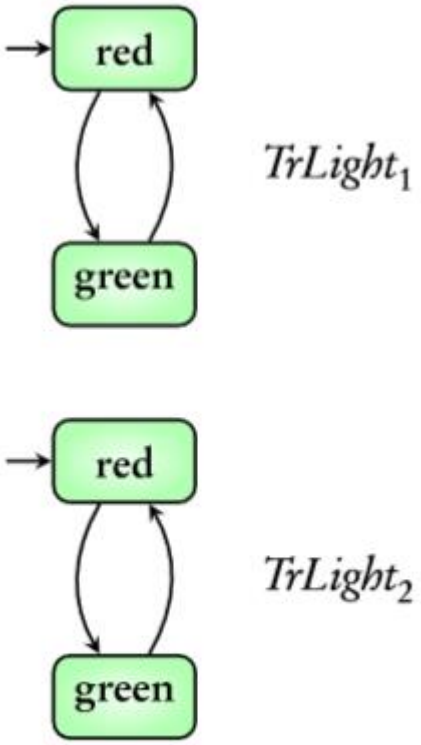
Modelando sistemas concorrentes

Como interligar os sistemas?

Operadores: Interleaving e Handshaking

- Independentes
- Variáveis compartilhadas
- Ações comuns

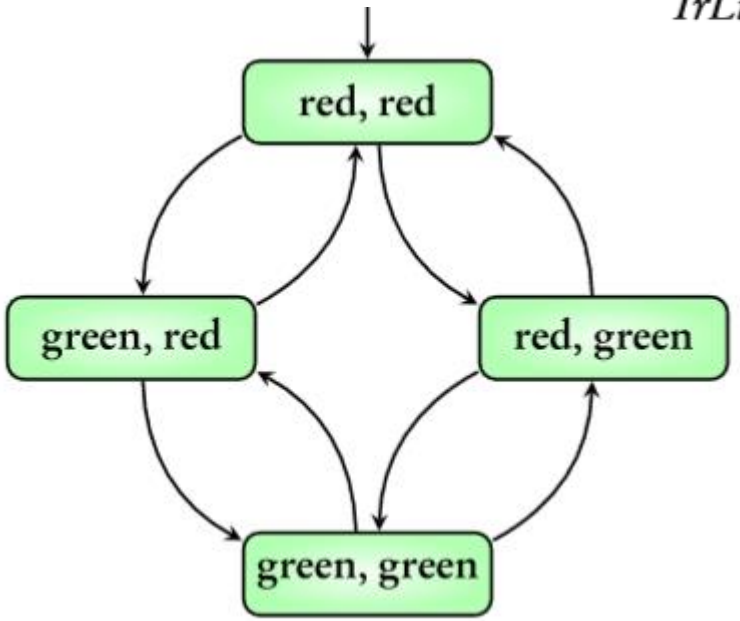
Independentes



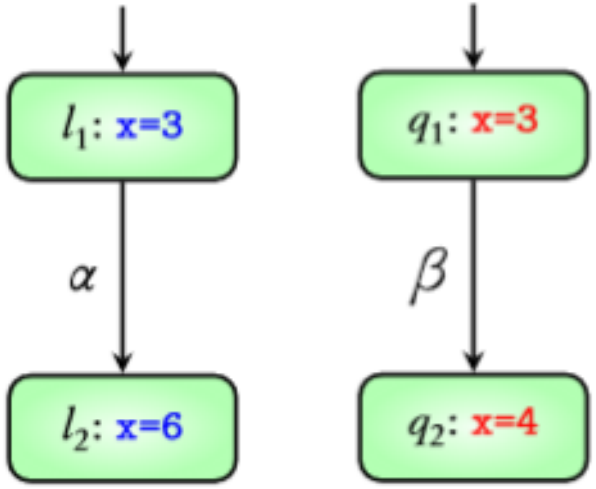
$TrLight_1$

$TrLight_2$

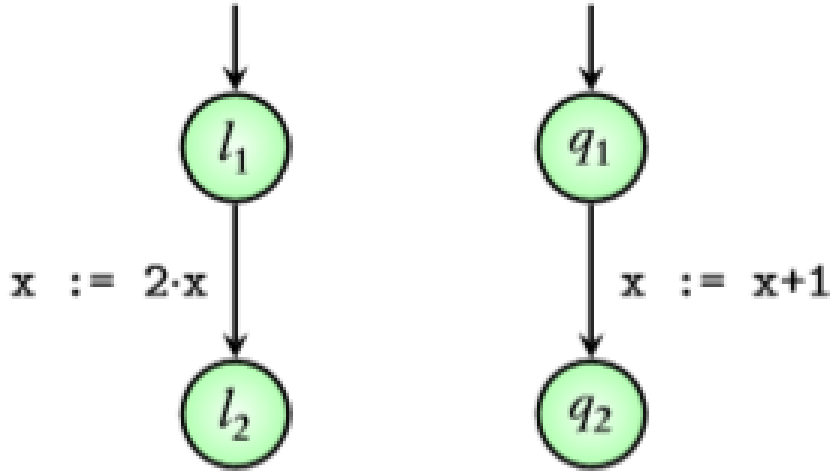
$TrLight_1 \;|||\; TrLight_2$



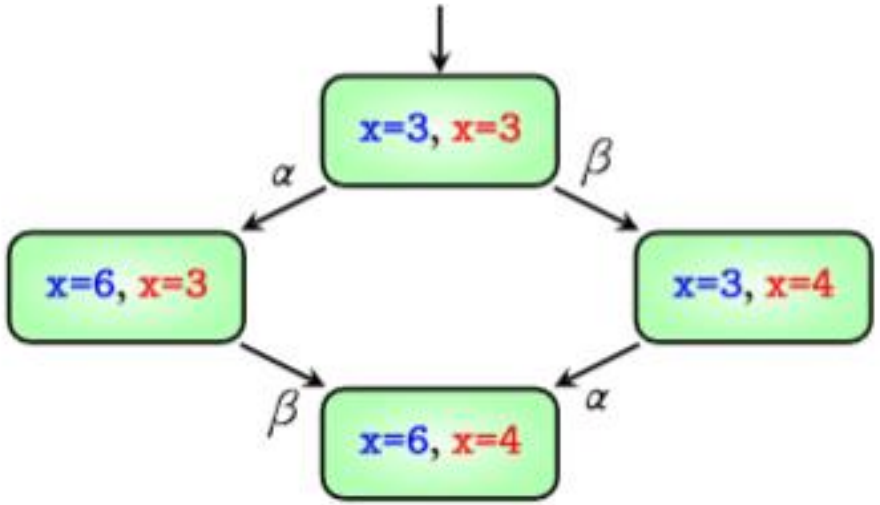
Variáveis compartilhadas



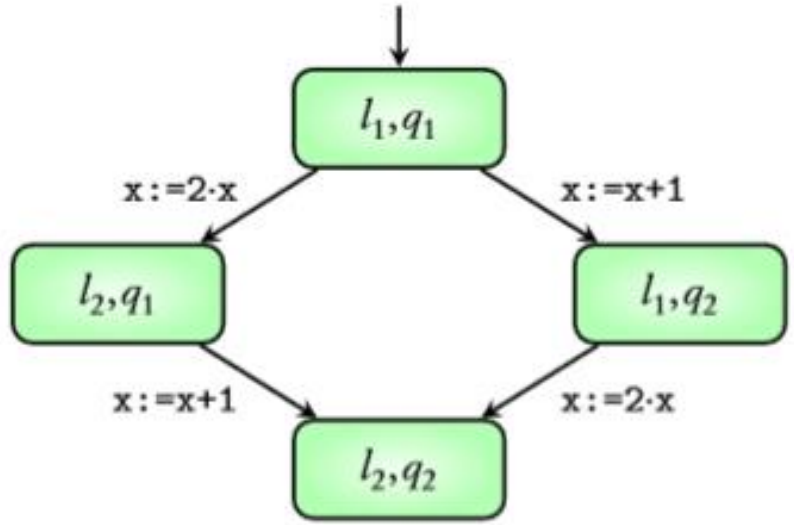
TS_1 TS_2



PG_1 PG_2

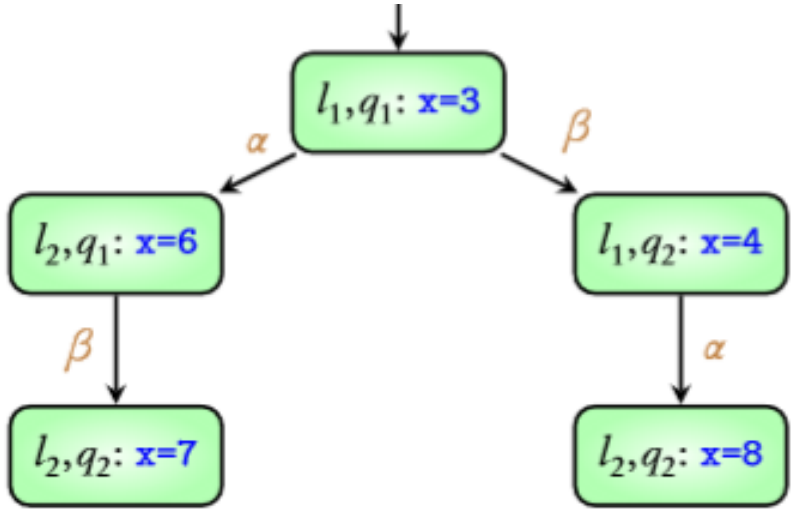


$TS_1 \;|||\; TS_2$

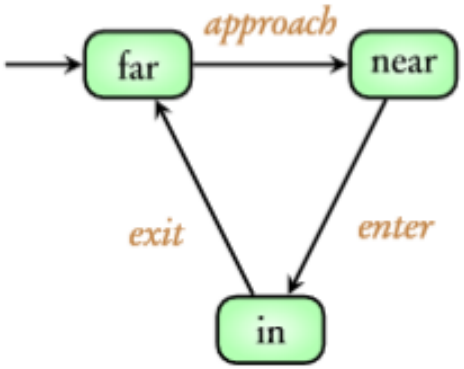


$PG_1 \;|||\; PG_2$

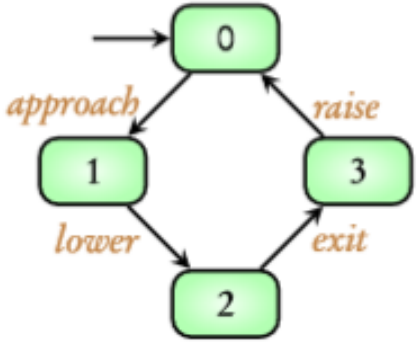
$TS(PG_1 \;|||\; PG_2)$:



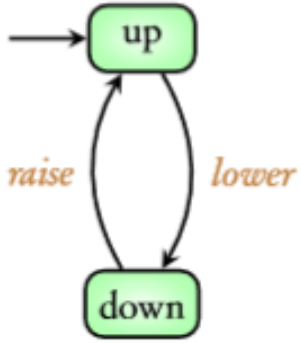
Ações comuns



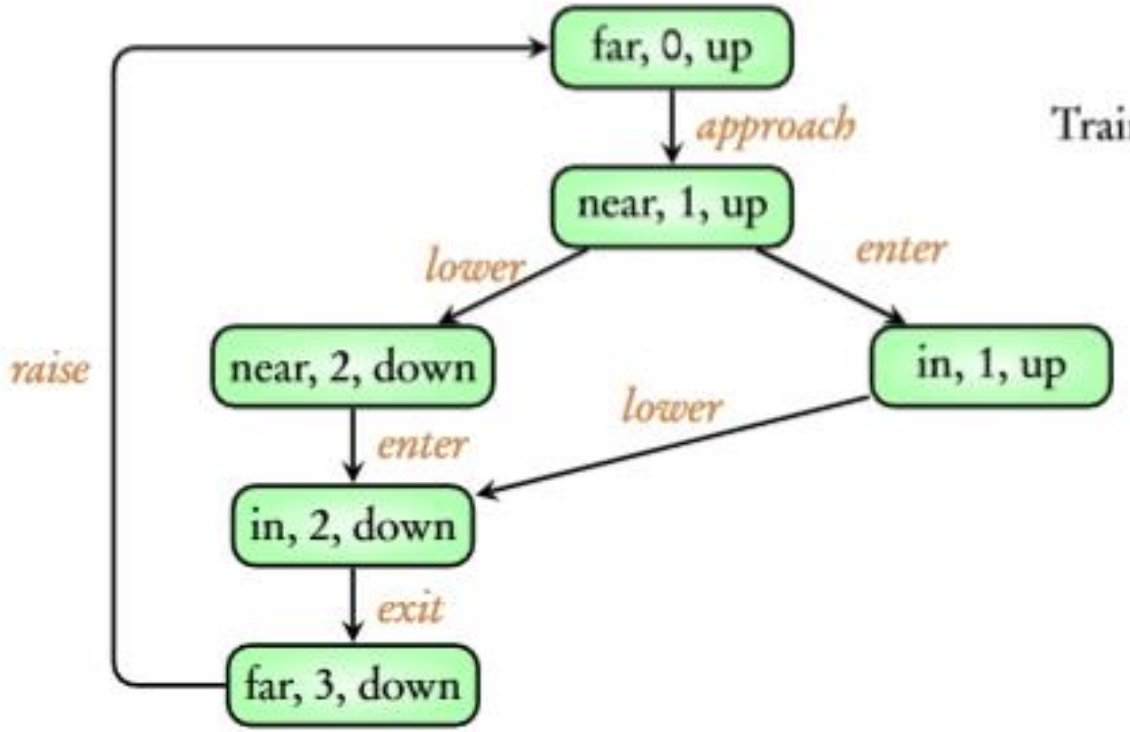
Train



Controller



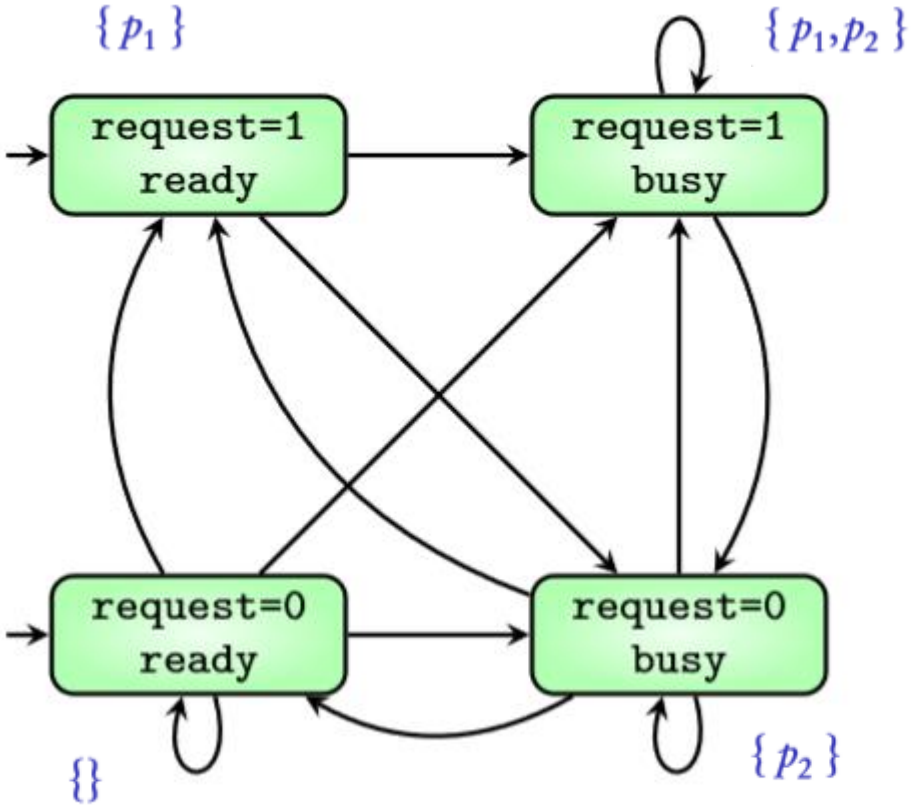
Gate



Train || Controller || Gate

A especificação como propriedade

- O que é uma propriedade
- Proposições atômicas
- Traces
- Relação de satisfação



- Uma proposição atômica é um sentença que pode ser avaliada em verdadeiro ou falso com base no TS que estamos analisando

Atomic propositions

p_1: (request=1) p_2: (status=busy)

- Dado um conjunto de proposições atômicas AP
- Temos o conjunto potencia de AP: 2^{AP}
- AP_{inf} é o conjunto de todas as palavras infinitas sobre o alfabeto 2^{AP}
- Uma execução gera um Trace, que é uma palavra infinita sobre o alfabeto 2^{AP}, ou seja, um elemento de AP_{inf}
- Traces(TS) é o conjunto de todas as execuções de TS, ou seja, um subconjunto de AP_{inf}
- Traces pode ser interpretado como o comportamento de um TS

Definindo uma propriedade

São descritas em cima das proposições atômicas

- p1 é sempre verdade
- p1 é verdade pelo menos uma vez e p2 é sempre verdade

Elas definem então um subconjunto de AP_{inf}

Relação de aceitação

- Como vimos que Traces(TS) e uma propriedade são ambos subconjuntos de AP_{inf}.
- Um TS satisfaz uma especificação se:

$$\text{Traces}(TS) \subseteq P$$

- As propriedades sobre AP são chamadas de propriedades temporais lineares

Tipos de propriedade

Invariants

- p1 é sempre verdade
- P1 e não p2 são sempre verdade

- São propriedades que ocorrem em todo o sistema (em todos os estados)

- Invariants são propriedades regulares

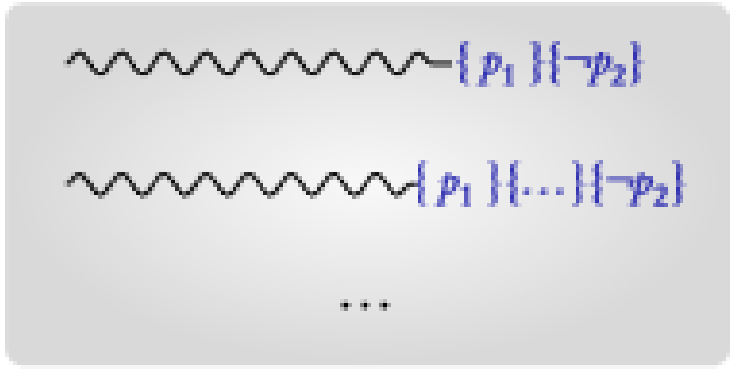
Algoritmo de Verificação de Invariants

- Dada uma invariant ϕ, TS satisfaz ϕ se para todo estado alcançável a partir dos estados iniciais, o estado satisfaz ϕ
- Busca em profundidade em TS

Safety Properties

- Determinam condições consideradas ruins que o sistema não deveria aceitar (Bad Prefixes)
- Sempre que p1 for verdade, o próximo passo terá p2 como verdade
- Uma propriedade é considerada uma Safety Property quando ela descreve Bad Prefixes e para toda palavra dessa propriedade, a palavra não contenha esse prefixo
- Uma Safety Property será regular caso seus Bad Prefixes são regulares

Bad Prefixes



- São determinados a partir das safety properties

- Sempre que p1 for verdade, o próximo passo terá p2 como verdade
- Um prefixo ruim seria p1 verdade em um passo e no seguinte p2 falso

- Sempre que p1 for verdade, o próximo do próximo passo terá p2 como verdade
- Um prefixo ruim seria p1 verdade e o próximo do próximo passo terá p2 falso

Algoritmo de verificação de safety properties

- Identificar Bad Prefixes da propriedade
- Caso os Bad Prefixes sejam regulares, construir autômato que reconhece-os
- Fazer a interseção entre o autômato dos Bad Prefixes e o autômato modelado a partir do TS
- Checar se o autômato resultante aceita apenas a linguagem vazia, se sim o modelo satisfaz as especificações, caso contrário não satisfaz

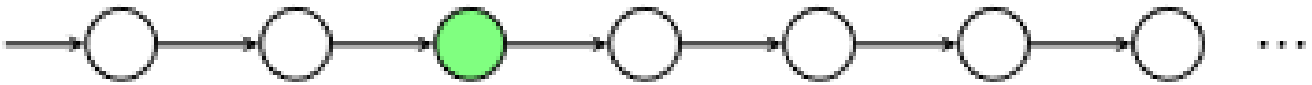
Liveness Properties

- Descreve que um evento deve ocorrer infinitamente sempre que possível (infinitely often)

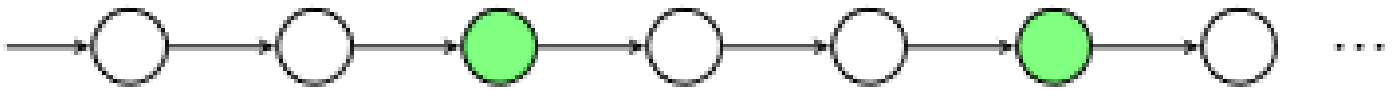
- Será detectada e implementada com o autômato de Büchi

p1 sempre verdade

p1 verdade ao menos uma vez



p1 verdade infinitamente (infinitely often)



Model Checking com autômatos finitos

Como já descrito, usamos autômatos finitos em dois casos

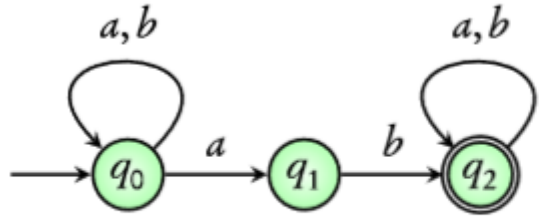
- Checar Invariants em um autômato construído a partir de um TS
- Checar Safety Properties regulares

Propriedades uteis dos autômatos Finitos

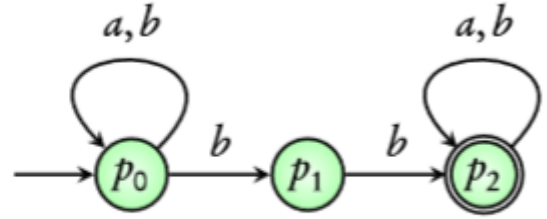
- Interseção
- Linguagem vazia

Interseção (Produto Síncrono)

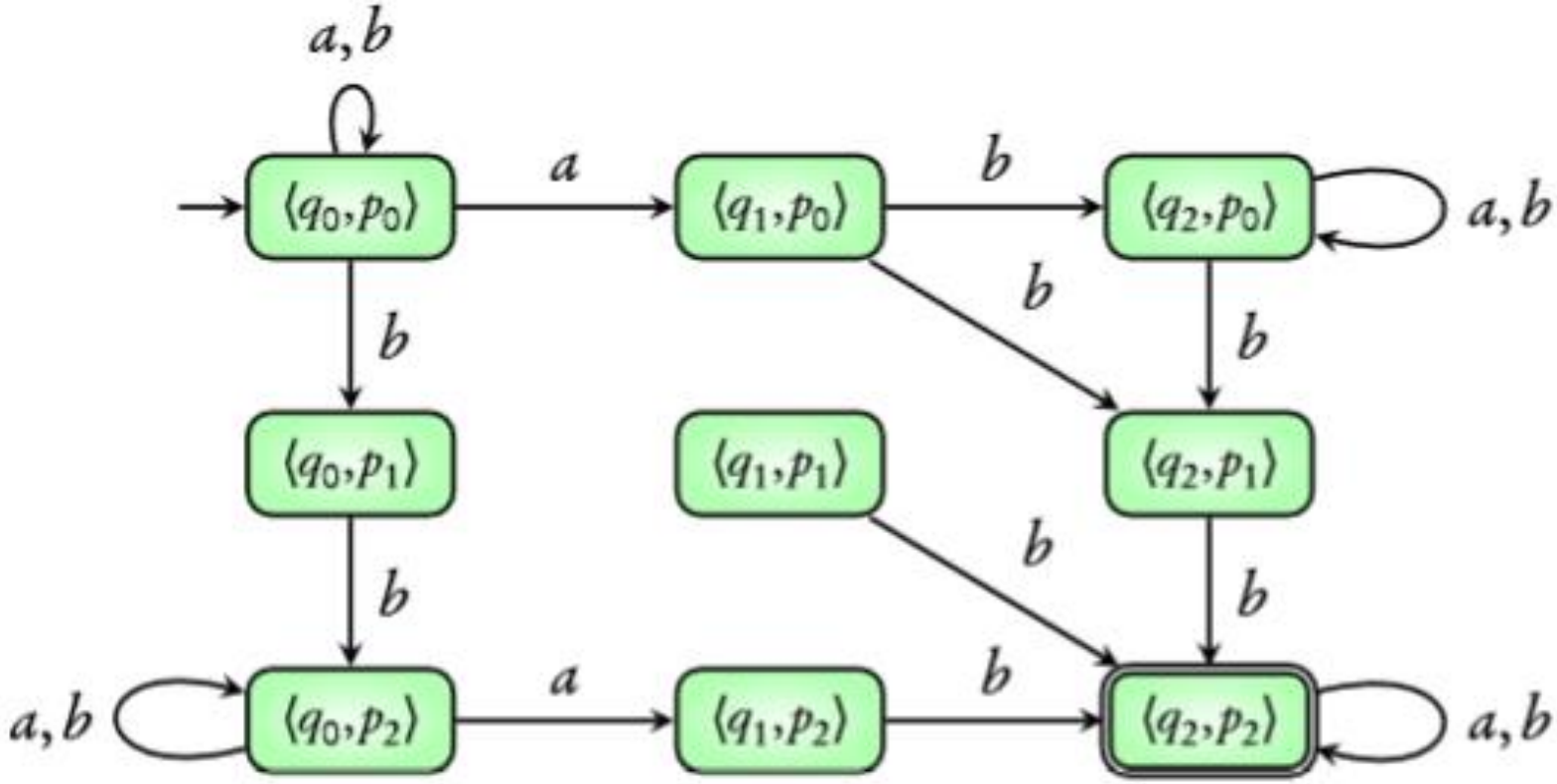
- Criar combinações de estados n a n
- Para cada tupla nesses novos estados, fazer transições apenas quando para um símbolo, todos os estados presentes nesse estado combinatório tenham transições com o mesmo símbolo
- O estado iniciais resultantes devem ter todos seus estados originais iniciais
- Os estados finais resultantes devem ter todos seus estados originais finais



$\Sigma^* ab \Sigma^*$



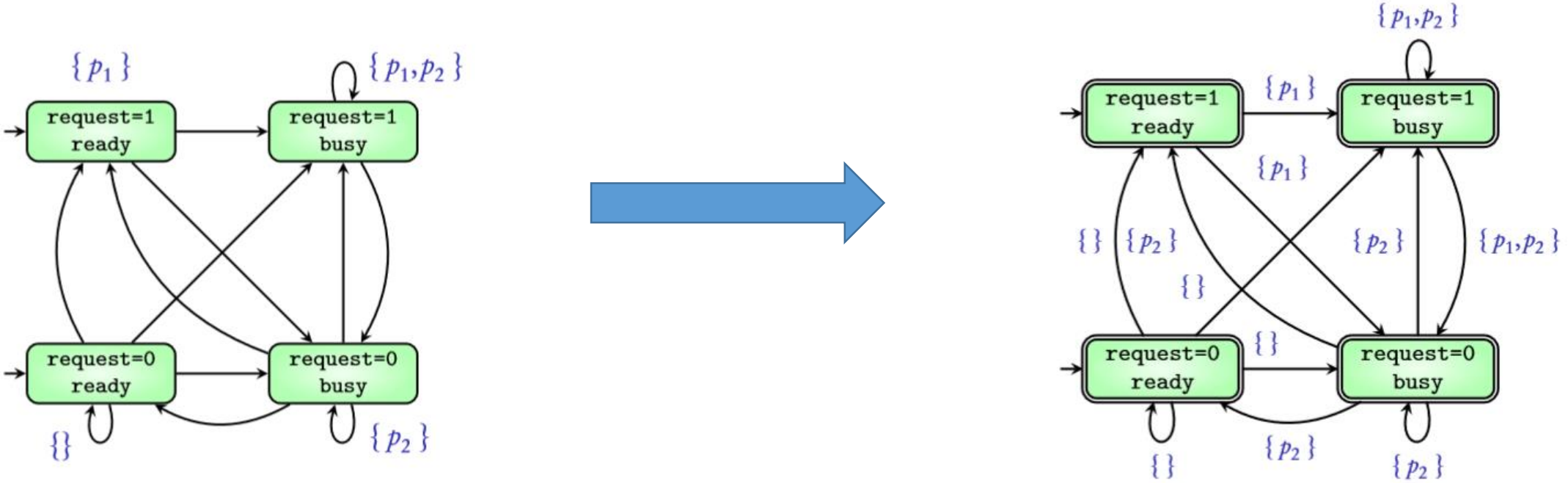
$\Sigma^* bb \Sigma^*$



$\Sigma^* ab \Sigma^* \cap \Sigma^* bb \Sigma^*$: words containing both *ab* and *bb*

Modelar um TS para um AFN

- Para cada transição de um estado do TS, colocar na transição a AP associada ao estado
- Marcar todos estados como finais



Autômato de Büchi

- Especificar propriedades em termos de um autômato de Büchi
- Criar algoritmo de determinação de satisfação da propriedade dado um modelo

Relação de Aceitação

- Para uma palavra infinita ser aceita por um autômato de Büchi, a execução (run) do autômato para essa palavra deve passar por um estado final infinitamente (infinitely often)

Propriedades ω - Regulares

Expressões regulares (usando como exemplo o alfabeto {a, b})

$$\epsilon \mid a \mid b \mid r_1\, r_2 \mid r_1 + r_2 \mid r^*$$

Onde r, r_1 *e* r_2 são expressões regulares

Expressões ω - Regulares

$$G = E_1 \cdot F_1^{\omega} + E_2 \cdot F_2^{\omega} + \cdots + E_n \cdot F_n^{\omega}$$

Onde E_n *e* F_n são expressões regulares e ε não pertence a F_n

Algoritmo de satisfação da especificação

Dada uma propriedade e um modelo

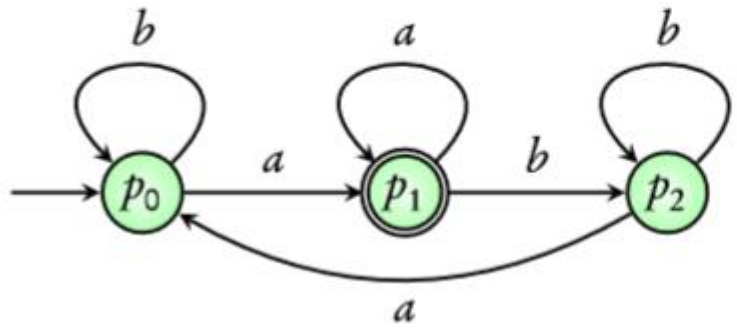
- Gerar autômato de Büchi, α, a partir da propriedade
- Gerar autômato que aceita a linguagem inversa a de α
- Gerar autômato de Büchi, β, a partir do TS
- Fazer a interseção entre α e β
- Determinar se a linguagem do autômato da interseção é vazia, se sim o modelo satisfaz as propriedades, caso contrário não satisfaz.

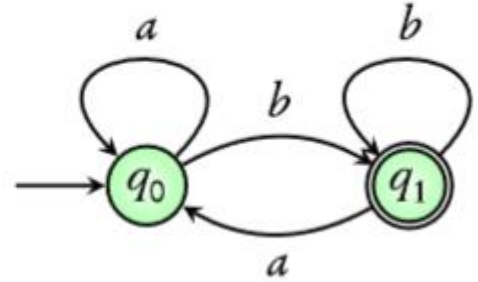
Propriedades Uteis do Autômato de Büchi

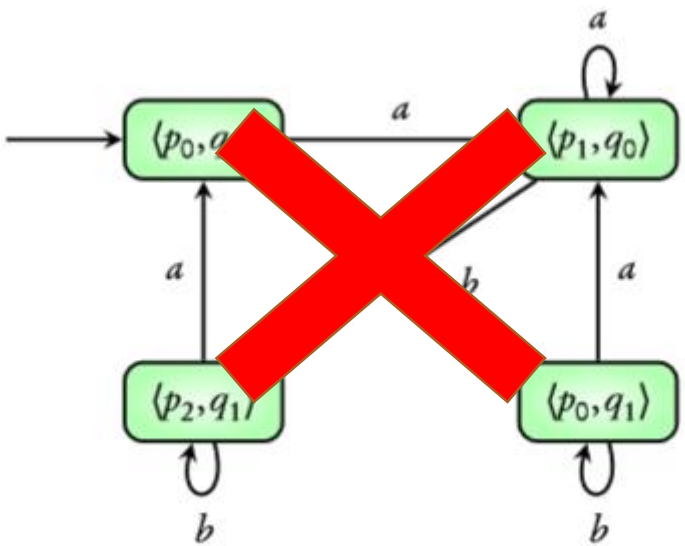
- Interseção (produto síncrono modificado)
- Linguagem vazia
- Complemento
- União
- (Generalização)

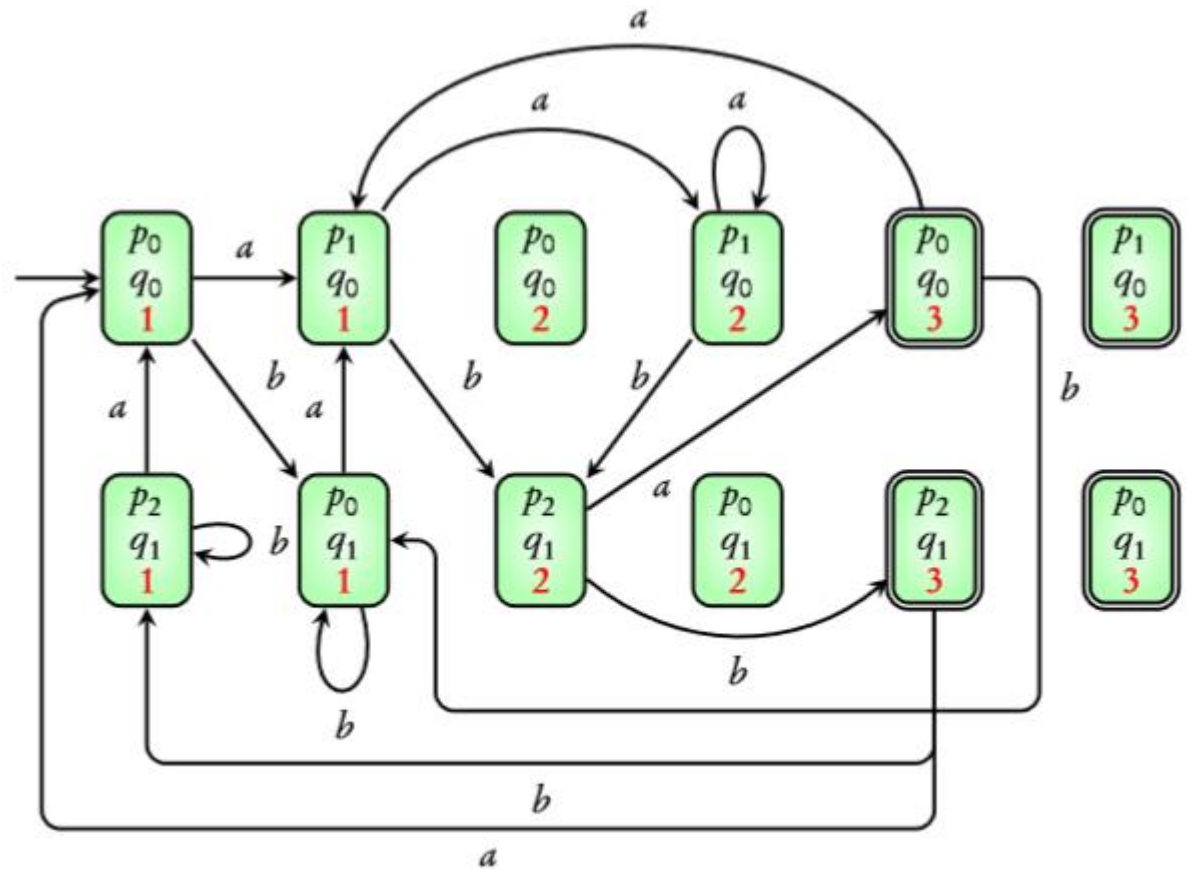
Interseção

Se utiliza da construção do produto síncrono dos autômatos finitos, porem usa índices para determinar se já houve passagem por algum estado final dos estados originais.


b
a
b
a
b
p0
p1
p2
a


a
b
b
q0
q1
a

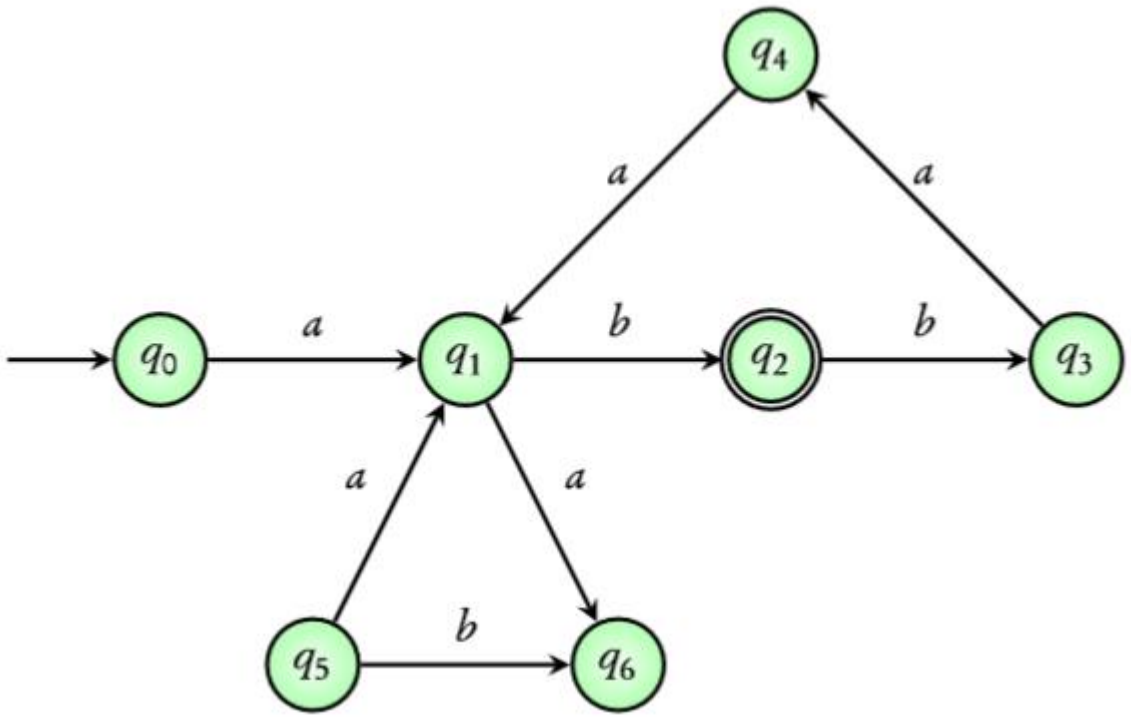

a
a
⟨p1,q0⟩
a
b
a
⟨p2,q1⟩
⟨p0,q1⟩
b
b

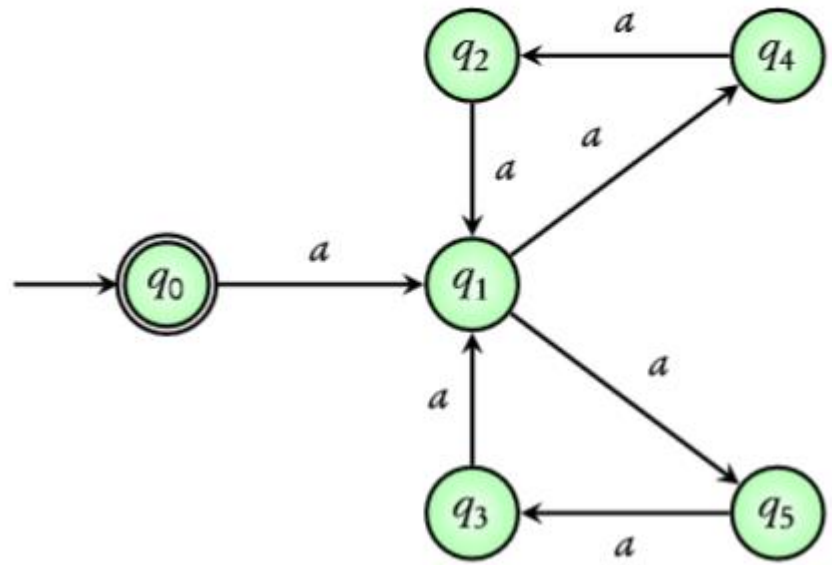

a
a
a
p0 q0 1
a
p1 q0 1
p0 q0 2
p1 q0 2
p0 q0 3
p1 q0 3
b
b
b
b
a
a
p2 q1 1
b
p0 q1 1
p2 q1 2
a
p0 q1 2
p2 q1 3
p0 q1 3
b
b
b
b
a

Linguagem Vazia

São usados duas buscas em profundidade para determinar se a linguagem aceita pelo autômato de Büchi é vazia

- Achar todos os estados alcançáveis a partir dos estados iniciais
- A partir de cada um desses estados, achar um caminho que te leve de volta ao mesmo estado de onde se partiu


q_4
a
a
q_0
a
q_1
b
q_2
b
q_3
a
a
q_5
b
q_6


q_2
a
q_4
a
a
q_0
a
q_1
a
a
q_3
a
q_5

Complemento

- Não se pode apenas inverter os estados finais com não finais como nos autômatos finitos.
- Pode-se afirmar que há como determinar o complemento de um autômato de Büchi

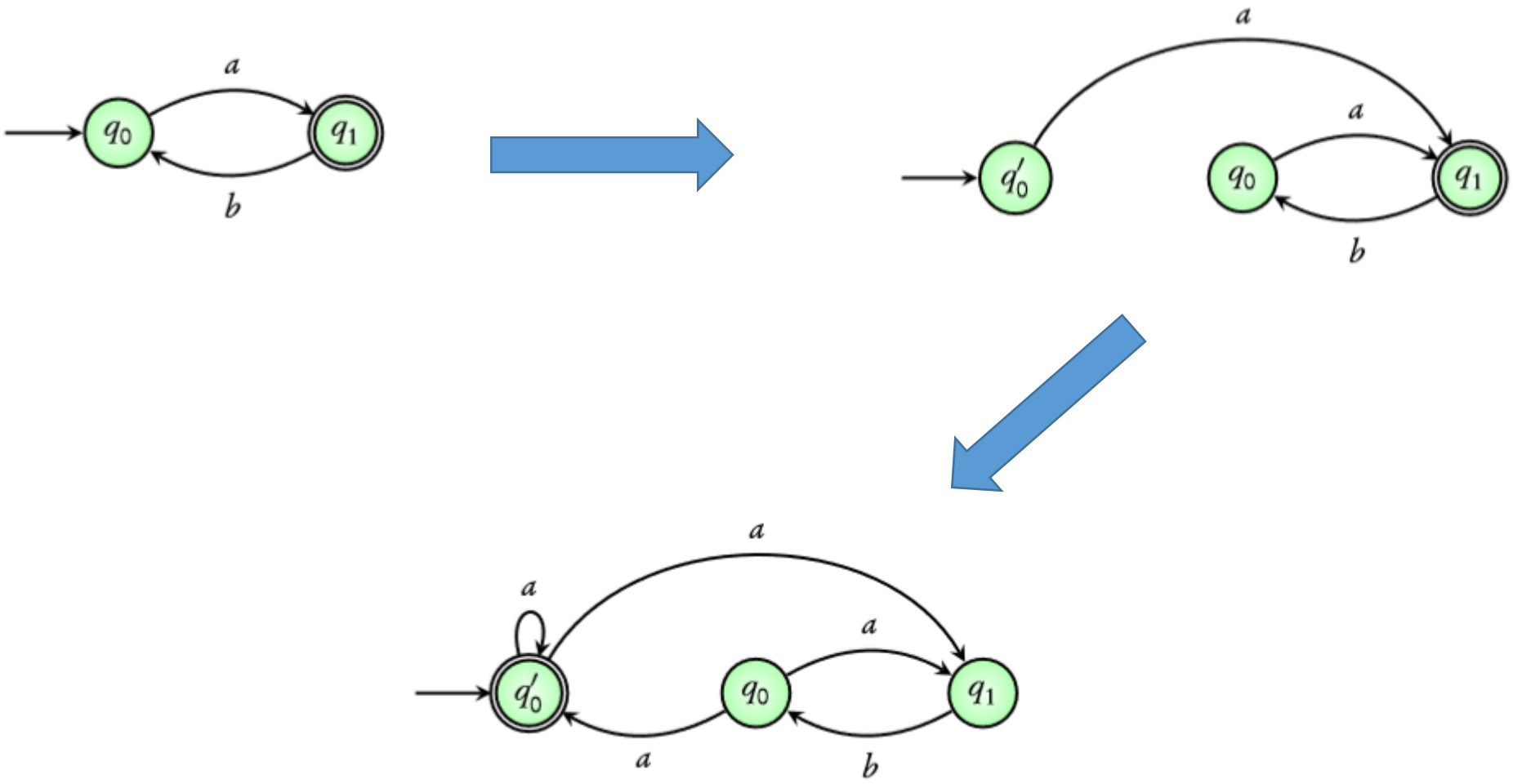
União

- Englobar os dois autômatos presentes na união como um só

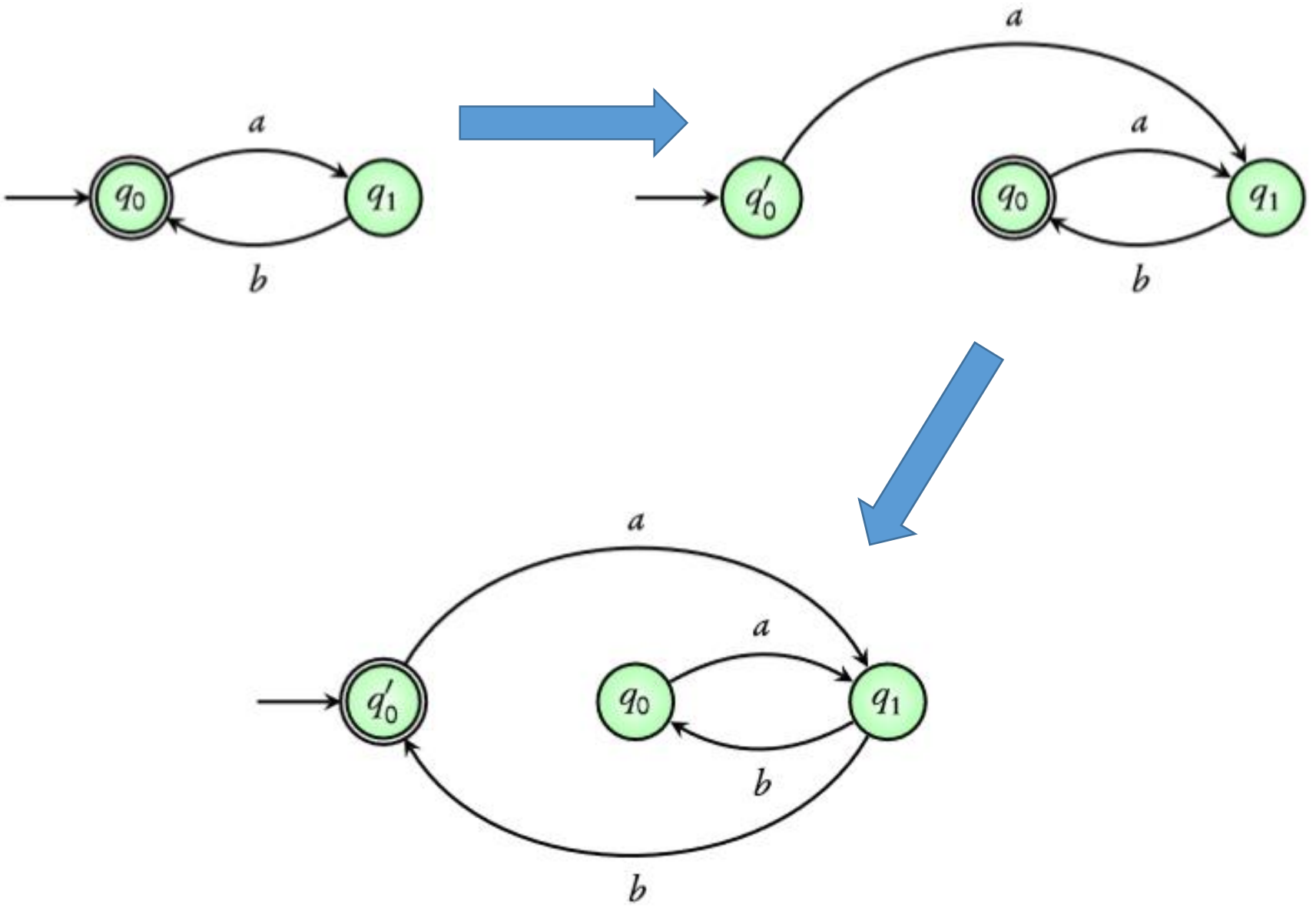
Algoritmo para Converter Expressão ω - Regular em ABN

1. Dada uma expressão regular U, achar autômato para U^{ω}
   - Cria-se um novo estado q_0'
   - Todas as transições com destino nos estados iniciais são copiadas tendo como origem q_0' (autômato padronizado)
   - Todas as transições quem chegam em um estado final são copiadas tendo como destino q_0'
   - Estados iniciais não são mais iniciais
   - Estados finais não são mais finais
   - q_0' se torna único estado final e inicial
2. Dados AFN U e ABN V, achar o autômato da concatenação
   - Transições para o estado final de U, são copiadas com destino aos estados iniciais de V.
   - Estados finais de U não são mais finais
   - Estados iniciais de V não são mais iniciais
3. Dados ABN U e V, achar o autômato para a união

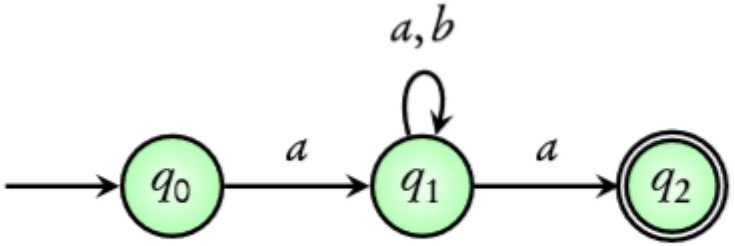
1.

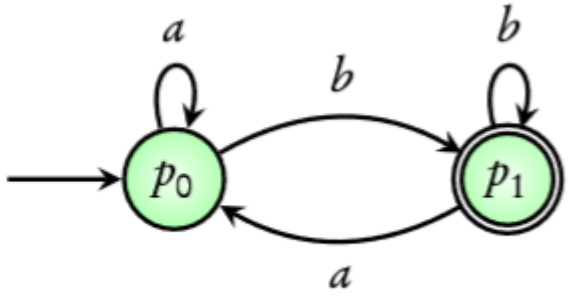


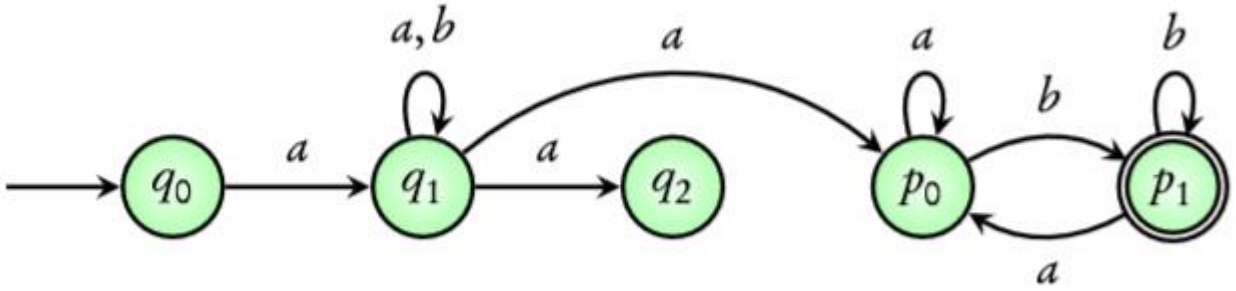
1.



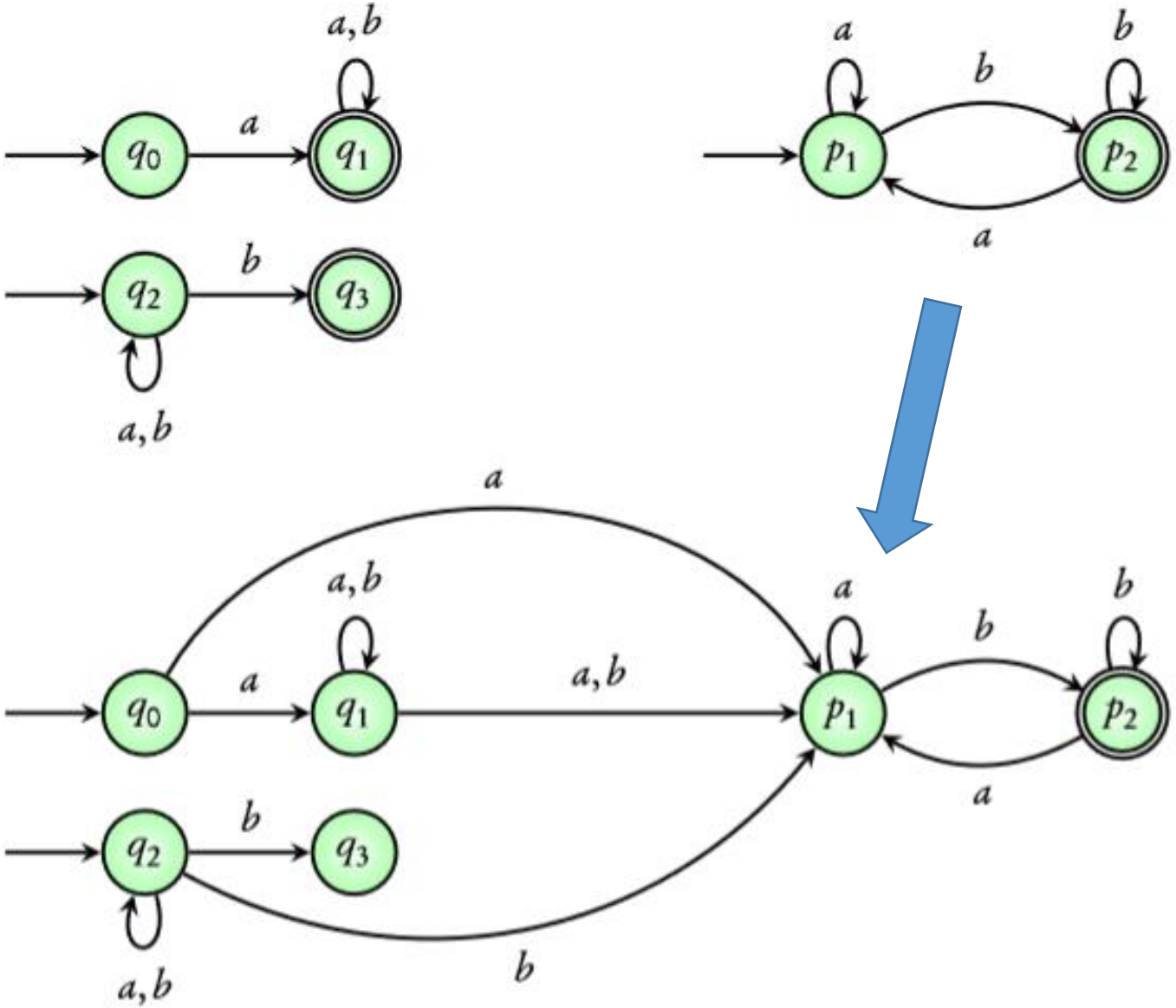
2.







2.



Lógica Temporal Linear

É o formalismo mais comum e conveniente para se escrever propriedades

- Fórmulas
- Operadores
- Descrevendo propriedades como LTL
- Converter uma formula LTL para um NBA

- Uma formula LTL esta descrevendo uma propriedade, que é um subconjunto de palavras infinitas sobre o alfabeto AP_{inf}, então as fórmulas LTL também descrevem subconjuntos de AP_{inf}

- Dado uma fórmula ϕ de LTL, podemos descrever todas as palavras que essa fórmula gera representado pelo conjunto Words(ϕ).

- De forma similar às verificações com os autômatos, devemos verificar se o subconjunto Traces(TS) está contido em Words(ϕ) para satisfação da propriedade pelo modelo.

Fórmulas LTL

Fórmulas básicas de LTL

$$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid X\,\phi \mid \phi_1\, U\, \phi_2$$

p_i pertence à AP

Fórmulas derivadas

$\mathbf{F}\, p_1$ $\quad true\ U\ p_1$

$\mathbf{G}\, p_1$ $\quad \neg(true\ U\ \neg p_1)$

Algoritmo para determinar se um modelo satisfaz uma fórmula em LTL

- Dada uma fórmula ϕ, gerar um NBA que reconheça $\neg \phi$
- Achar um NBA para o TS
- Gerar o autômato da interseção entre os autômatos
- Determinar se a linguagem do autômato interseção é vazia, se sim, o modelo satisfaz as especificações, caso contrario o modelo não satisfaz

Avaliar comportamento das fórmulas pela sua expansão

- Para construir o autômato, precisamos avaliar cada subfórmula para gerar estados a partir dos estados de cada subfórmula

- Ao construir uma expansão, respeitar as propriedades de cada fórmula

- Determinar a aceitação da palavra na expansão

$p_1 \ \mathrm{U} \ p_2$

	$\{p_1\}$	$\{p_1\}$	$\{p_1\}$	$\{p_1\}$	$\{p_2\}$	$\{p_1\}$	$\{p_1\}$	$\{p_1\}$	$\{p_1,p_2\}$ ⋯
p_1	1	1	1	1	0	1	1	1	1
p_2	0	0	0	0	1	0	0	0	1
$p_1 \ U \ p_2$	1	1	1	1	1	1	1	1	1

	$\{p_1\}$	$\{\}$	$\{p_1\}$	$\{\}$	$\{p_1\}$	$\{\}$	$\{p_1\}$	$\{\}$	$\{p_1\}$ ⋯
p_1	1	0	1	0	1	0	1	0	1
p_2	0	0	0	0	0	0	0	0	0
$p_1 \ U \ p_2$	0	0	0	0	0	0	0	0	0

$\mathrm{G}\,\mathrm{F}\,p_1$

$\mathrm{F}\,\phi = true\,\mathrm{U}\,\phi \qquad \mathrm{G}\,\phi = \neg\, true\,\mathrm{U}\,\neg\phi$

$\neg\ true\,\mathrm{U}\,\neg(true\,\mathrm{U}\,p_1)$

	$\{\}$	$\{\}$	$\{p_1\}$	$\{\}$	$\{\}$	$\{p_1\}$	$\{\}$	$\{\}$	$\{p_1\}$
p_1	0	0	1	0	0	1	0	0	1
$true$	1	1	1	1	1	1	1	1	1
$true\,\mathrm{U}\,p_1$	1	1	1	1	1	1	1	1	1
$\neg\, true\,\mathrm{U}\,p_1$	0	0	0	0	0	0	0	0	0
$true\,\mathrm{U}\,\neg\,(true\,\mathrm{U}\,p_1)$	0	0	0	0	0	0	0	0	0
$\neg\ true\,\mathrm{U}\,\neg\,(true\,\mathrm{U}\,p_1)$	1	1	1	1	1	1	1	1	1

Propriedades das fórmulas (compatibility)

	$\{\}$		$\{p_1\}$		$\{p_2\}$		$\{p_1,p_2\}$	
p_1	0		1		0		1	
p_2	0		0		1		1	

ϕ		0		1	
$\neg\phi$		1		0	

ϕ			0			1	
$X\,\phi$		0			1		

ϕ_1		1		0		1		0	
ϕ_2		1		1		0		0	
$\phi_1 \wedge \phi_2$		1		0		0		0	

ϕ_1	*		1		0		1		
ϕ_2	1		0		0		0		
ϕ_1 U ϕ_2	1		1	1		0		0	0

ϕ_1	1	1	1	1	1		
ϕ_2	0	0	0	0	0	• • •	
ϕ_1 U ϕ_2	1	1	1	1	1		

Não pode ocorrer para sempre que $\phi_1 U \phi_2 = 1, \phi_1 = 1$, mas $\phi_2 = 0$

Gerar NBA a partir de fórmula em LTL

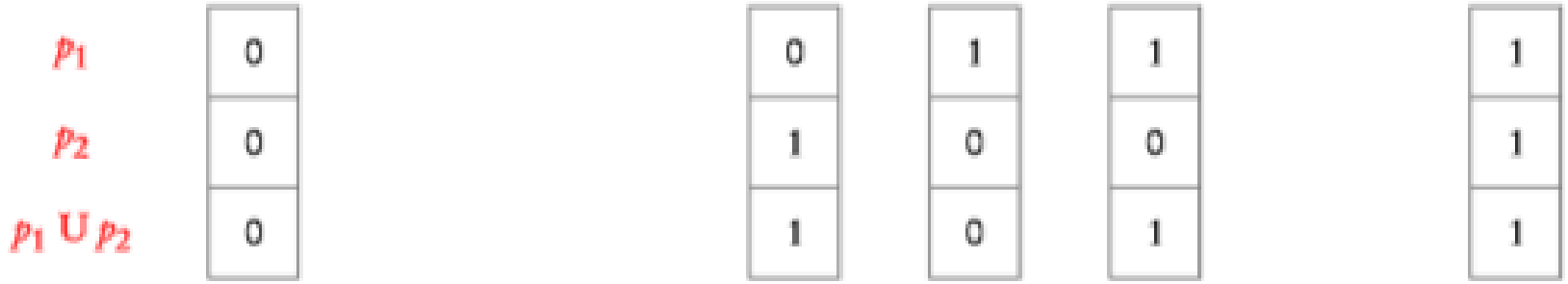
A ideia chave é construir um autômato aonde cada estado é uma coluna da expansão de aceitação da formula LTL

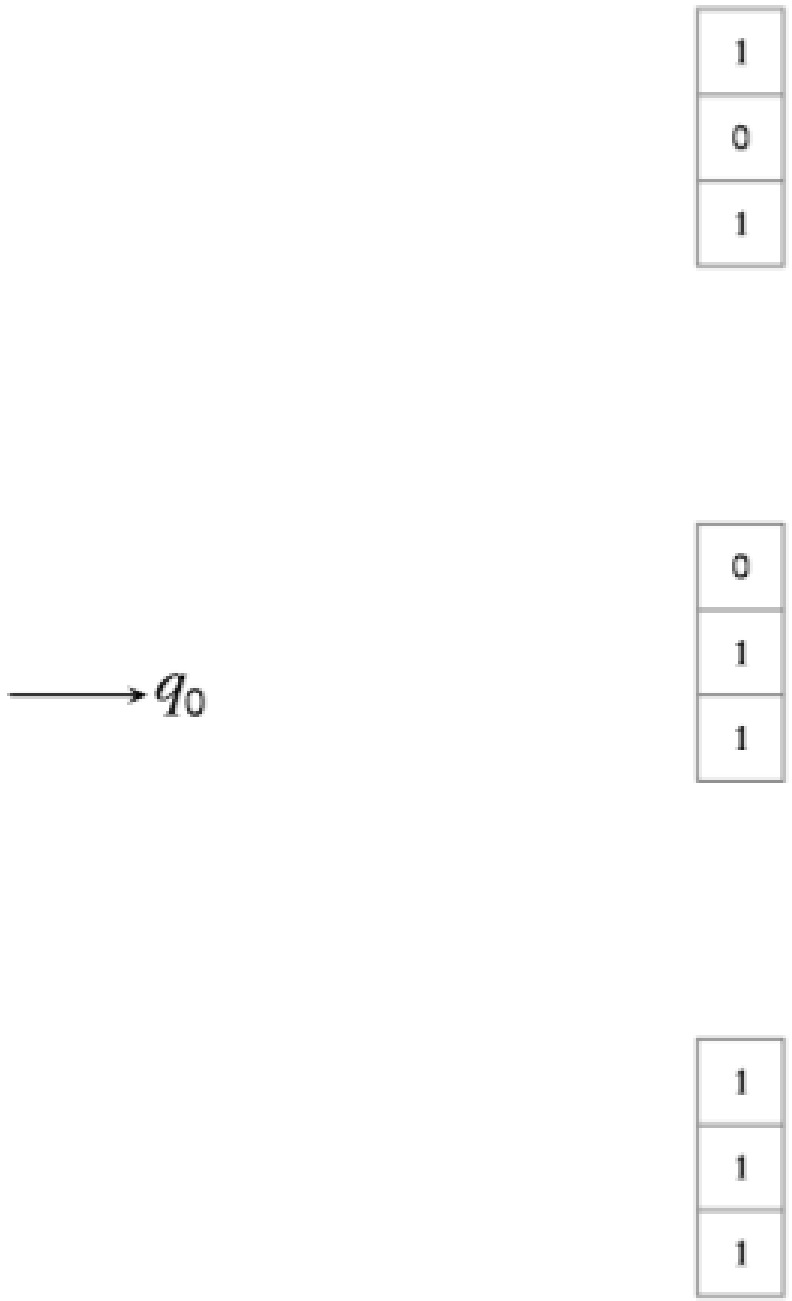
A expansão de aceitação deve ter sempre 1 na primeira coluna da fórmula final

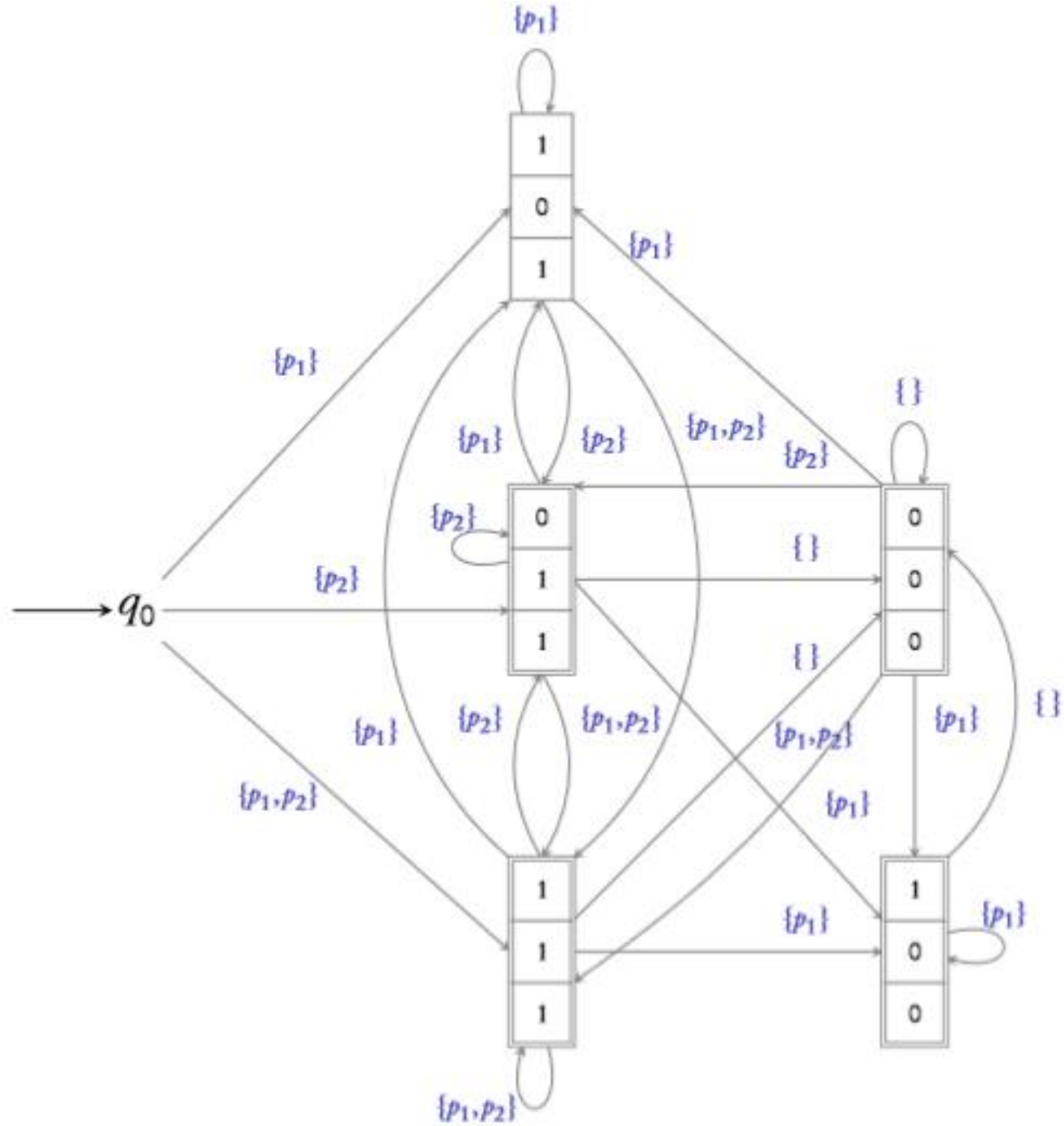
Algoritmo para construção

- Listar todas as subfórmulas da fórmula original
- Checar as propriedades das fórmulas LTL
  - AND, NOT, UNTIL,
  - Remova estados incompatíveis e adicione um estado inicial q_0
- Adicione transições que satisfaçam as propriedades das fórmulas LTL
  - PALAVRA, X, UNTIL
- Estados finais devem garantir a condição da eventualidade do UNTIL, caso as subfórmulas não contenham o UNTIL, todo estado é final
- O resultado será um NBA Generalizado

Construindo o autômato

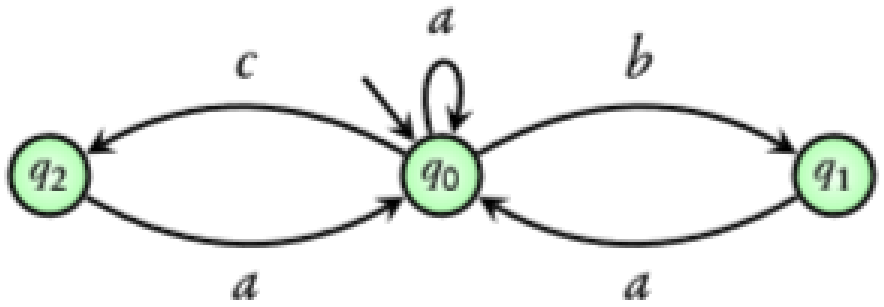



1
0
1
q_0
0
1
1
0
0
0
1
1
1
1
0
0


q_0
1
0
1
0
1
1
1
1
1
0
0
0
0
0
0
1
0
0
$\{p_1\}$
$\{p_2\}$
$\{p_1,p_2\}$
$\{\}$

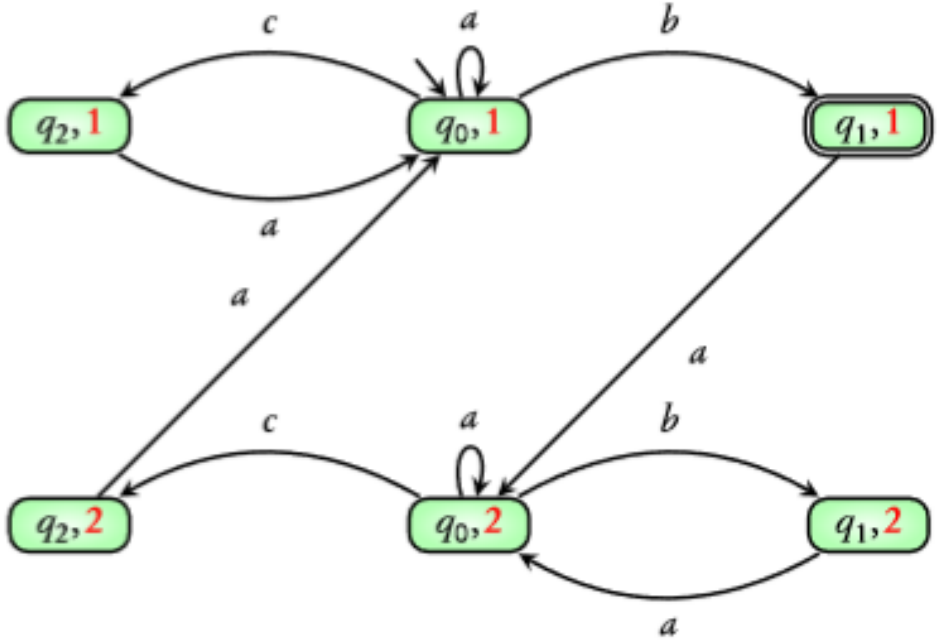
NBA Generalizado para NBA

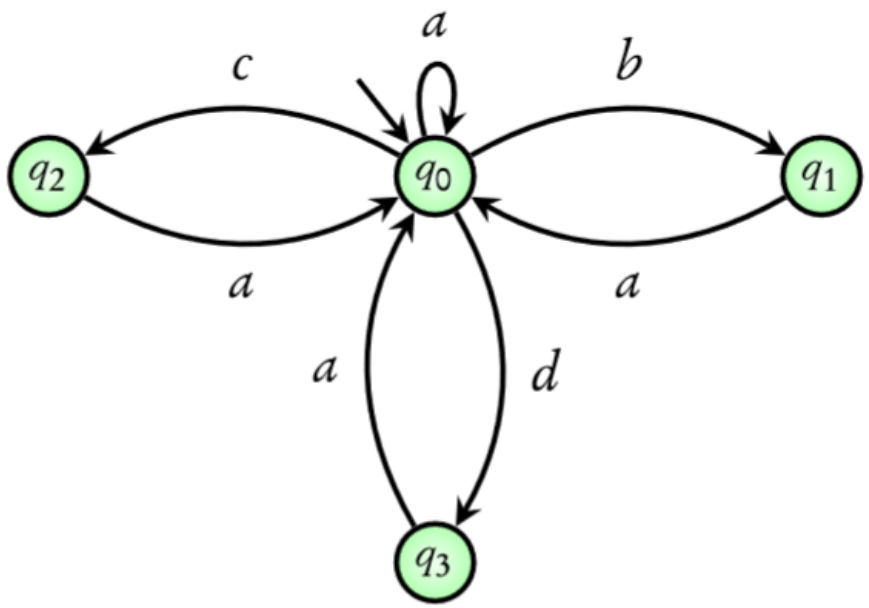
- Todo NBAG pode ser convertido para um NBA equivalente
- Deve-se alterar a condição de aceitação para o tipo:
  - $\{F_1, F_2, \ldots, F_n\}$
- O autômato aceita a palavra apenas se para cada F_i algum estado ocorrer infinitamente (infinitely often)



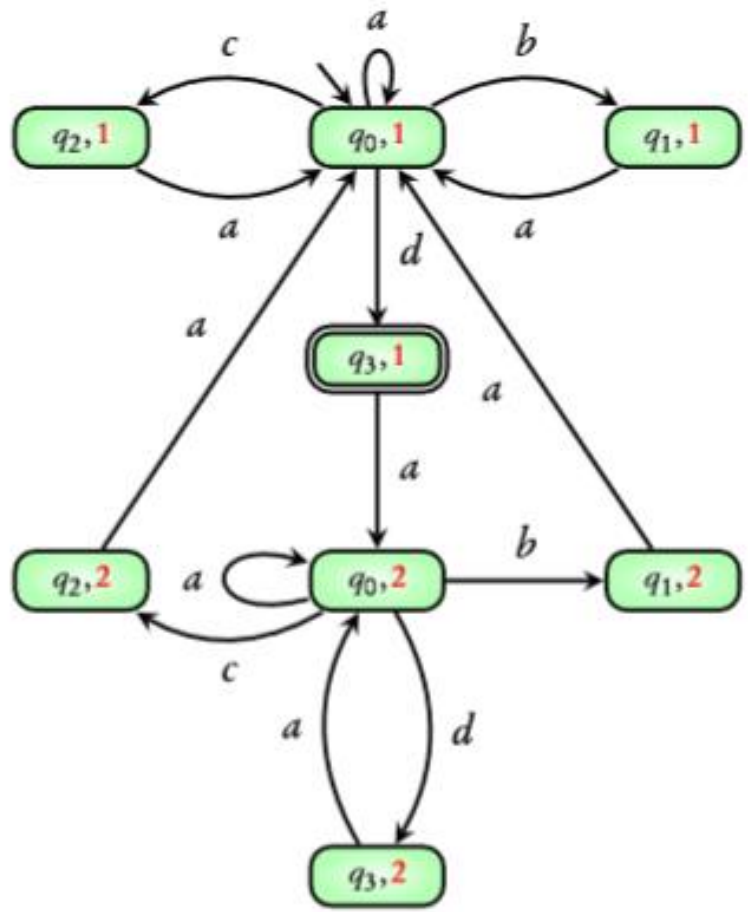
$$(a^*(b+c)a)^\omega$$

$$\{\{q_1\}, \{q_2\}\}$$




c
a
b
q_2
q_0
q_1
a
a
a
d
q_3

$\{\{q_3\}, \{q_1, q_2\}\}$


a
c
b
$q_2,1$
$q_0,1$
$q_1,1$
a
d
a
a
$q_3,1$
a
a
b
$q_2,2$
a
$q_0,2$
$q_1,2$
c
a
d
$q_3,2$

Referências

- Principles of Model Checking, Christel Baier and Joost-Pieter Katoen
- http://www.cmi.ac.in/~sri/Courses/MC2015/index.html