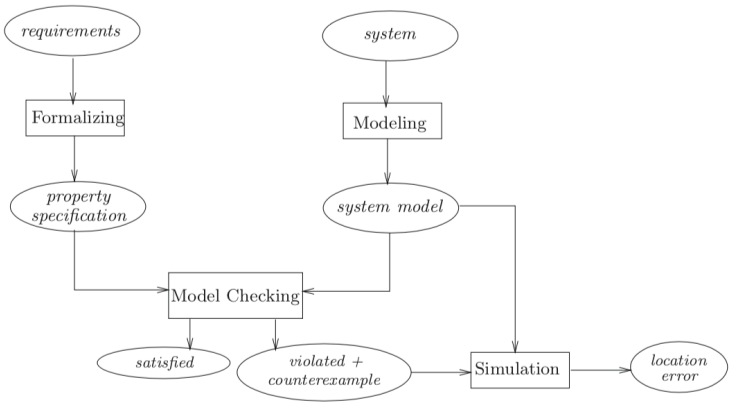
Aula 2

Model checking



Model checking

- Parte-se do modelo do sistema e não do sistema
- Explora-se exaustivamente os possíveis estados do modelo
- Os sistemas reactivos/concorrentes são descritos pelas suas propriedades temporais.
- Uma especificação é verificada se for satisfeita ao longo da execução do modelo.

Concorrência e atomicidade

$$\begin{aligned}
\textbf{proc}\ \mathsf{Inc} &= \textbf{while}\ \mathrm{true}\ \textbf{do if}\ x < 200\ \textbf{then}\ x := x + 1\ \textbf{fi od} \\
\textbf{proc}\ \mathsf{Dec} &= \textbf{while}\ \mathrm{true}\ \textbf{do if}\ x > 0\ \textbf{then}\ x := x - 1\ \textbf{fi od} \\
\textbf{proc}\ \mathsf{Reset} &= \textbf{while}\ \mathrm{true}\ \textbf{do if}\ x = 200\ \textbf{then}\ x := 0\ \textbf{fi od}
\end{aligned}$$

Os valores de x são sempre entre 0 e 200? Não, depende do escalonador:

- suponhamos $x = 200$
- o processo Dec testa x e passa o controlo ao processo Reset
- O processo Reset testa x e faz $x = 0$
- O controlo volta ao processo Dec e fica $x = -1$

Model checking

O *model checking* é técnica automática que dado um modelo com um número finito de estados e uma propriedade formal, sistematicamente verifica se essa propriedade é válida para (um estado do) o modelo.

Model checking

O *Model checking* é baseado em lógicas temporais.

Um modelo temporal $\mathcal{M}$ é constituído por um conjunto de estados e transições entre eles (sistema de transições).

Uma fórmula φ pode ser **V** nalguns estado e **F** noutros.

A noção de verdade não é absoluta, mas sim dinâmica.

Dado um estado s, pretende-se saber se:

$$\mathcal{M}, s \models \varphi$$

O verificador de modelos (*model checker*) é um algoritmo que decide este problema (responde sim ou não).

Fases

- Modelação: usando uma linguagem de descrição que permite a simulação do modelo
- Formalização da propriedade
- Execução do *model checker* para verificar se a propriedade é válida no modelo
- Análise: se a propriedade não se verificar um contra-exemplo é produzido e pode-se refinar o modelo (ou a propriedade)
- Pode acontecer que o *model checker* fique sem memória, no caso do número de estados ser demasiado grande (*state-space explosion*)

Propriedades a testar

***Reachability* (estados atingíveis)** propriedades que garantem que um dado estado pode ser atingido

***Safety* (segurança)** propriedades que têm de se verificar para todas as computações e em todos os estados

***Liveness* (vivacidade)** indicam que determinados estados do sistema têm de ser atingidos.

Persistência indicam que para todas as computações a partir de um dado estado uma certa propriedade tem de se verificar.

***Fairness* (razoabilidade)** indicam que uma dada propriedade tem de se verificar um número não finito de vezes. Exemplo: nenhum processo é esquecido sempre por um temporizador.

SWOT: Vantagens

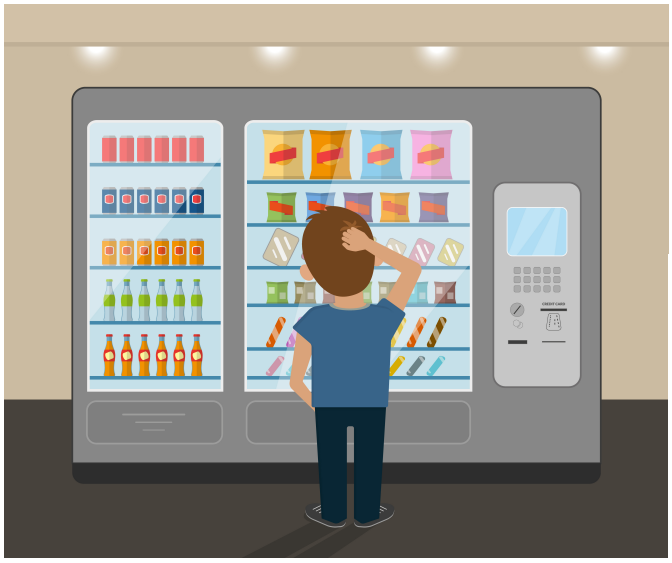
- É adequado a diversos tipos de aplicações: sistemas embebidos, desenho de hardware e no desenvolvimento de software.
- não necessita duma especificação completa do sistema
- fornece informação no caso da propriedade não ser válida
- pode ser facilmente integrado no ciclo de desenvolvimento de software
- é integro e matematicamente rigoroso

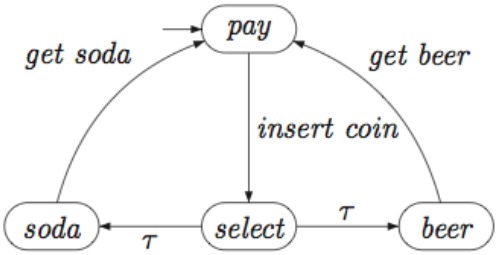
SWOT: Desvantagens

- é mais apropriado para aplicações de controlo e menos para grandes manipulações de dados
- verifica um modelo e não o sistema real
- só testa os requisitos expressos na propriedade e não a correção do sistema;
- sofre do problema da *explosão do espaço de estados*
- a modelação e especificação das propriedades requer um especialista em métodos formais

1 Sistemas de Transições

Máquina de venda





Sistemas de Transições (Modelo)

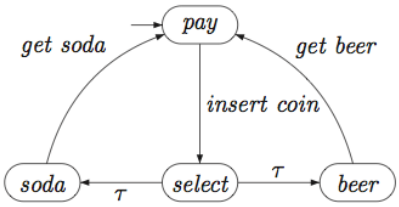
Um sistema de transições é um tuplo $T = (S, Act, \longrightarrow, I, AP, L)$ onde

- S é um conjunto de estados
- Act é um conjunto de acções
- $\longrightarrow \subseteq S \times Act \times S$ é a relação de transição
- $I \subseteq S$ é o conjunto de estados iniciais
- AP é um conjunto de variáveis proposicionais (proposições atómicas)
- $L : S \to 2^{AP}$ uma função de etiquetagem que associa a um estado um conjunto de variáveis proposicionais.

Sistemas de Transições

- se $(s, \alpha, s') \in \longrightarrow$, escreve-se $s \xrightarrow{\alpha} s'$
- o sistema T pode ser não determinístico
- dado um estado s, $L(s)$ indica quais as proposições que são verdade em s.
- podemos omitir as ações Act, caso só estejamos interessados na estrutura do sistema (p.e para saber os estados atíngíveis).
- podemos também não considerar estados iniciais

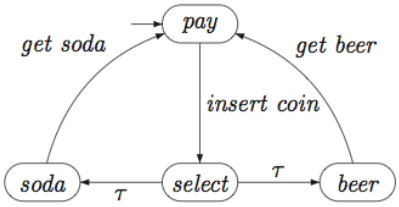
Máquina de venda- II



$$\begin{aligned} S &= \{pay, select, soda, beer\} \\ I &= \{pay\} \\ Act &= \{insert\_coin, get\_soda, get\_beer, \tau\} \\ \longrightarrow &= \{(pay, insert\_coin, select), (select, \tau, beer), \ldots\} \end{aligned}$$

τ descreve acções que não são relevantes no modelo (p.e internas ao sistema); o sistema é nãodeterminístico na escolha da bebida.

Máquina de venda- III



- Podemos supor que $AP = S$ e $L(s) = \{s\}$ para todo $s \in S$
- Ou por exemplo, $AP = \{paid, drink\}$ e $L(pay) = \emptyset$, $L(select) = \{paid\}$, $L(soda) = L(beer) = \{paid, drink\}$

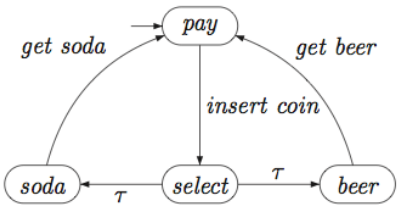
Sucessores e Predecessores

Seja $T = (S, Act, \longrightarrow, I, AP, L)$. Para $s \in S$ (ou $C \subseteq S$) e $\alpha \in Act$, *Post* é o

conjunto dos sucessores directos de α e Pre o dos predecessores:

$$
\begin{aligned}
Post(s,\alpha) &= \{s' \in S \mid s \xrightarrow{\alpha} s'\} \\
Pre(s,\alpha) &= \{s' \in S \mid s' \xrightarrow{\alpha} s\} \\
Post(C,\alpha) &= \bigcup_{s \in C} Post(s,\alpha) \\
Pre(C,\alpha) &= \bigcup_{s \in C} Pre(s,\alpha) \\
Post(s) &= \bigcup_{\alpha \in Act} Post(s,\alpha) \\
Pre(s) &= \bigcup_{\alpha \in Act} Pre(s,\alpha) \\
Post(C) &= \bigcup_{\alpha \in Act} Post(C,\alpha) \\
Pre(C) &= \bigcup_{\alpha \in Act} Pre(C,\alpha)
\end{aligned}
$$

Máquina de venda- III'



- Podemos supor que $AP = S$ e $L(s) = \{s\}$ para todo $s \in S$
- Ou por exemplo, $AP = \{paid, drink\}$ e $L(pay) = \emptyset$, $L(select) = \{paid\}$, $L(soda) = L(beer) = \{paid, drink\}$

Calcula $Post(select)$ e $Post(Pre(pay))$.

Estado Terminal

Um estado $s \in S$ é *terminal* se $Post(s) = \emptyset$.

- Para um modelo de um programa sequencial um estado terminal indica que o programa terminou.
- Para sistemas paralelos/reactivos, normalmente vamos supor que o modelo não tem estados terminais.

Determinismo

Um sistema de transições $T = (S, Act, \longrightarrow, I, AP, L)$ diz-se

1. *acção-determinístico* se $|I| \leq 1$ e $|Post(s, \alpha)| \leq 1$, para todo $s \in S$ e $\alpha \in Act$. *Isto é, cada estado tem no máximo uma saída por cada ação.*
2. *AP-determinístico* se $|I| \leq 1$ e $|Post(s) \cap \{s' \in S \mid L(s') = A\}| \leq 1$, para todo $s \in S$ e $A \in 2^{AP}$. *Isto é, dado um conjunto de variáveis A, cada estado tem no máximo uma transição para um estado etiquetado por A.*

Fragmento de Execução

Seja $T = (S, Act, \longrightarrow, I, AP, L)$.

- Um *fragmento finito de execução* é uma sequência alternada de estados e acções, ρ tal que
$$\rho = s_0\alpha_1 s_1\alpha_2 s_2 \ldots \alpha_n s_n,$$
tal que $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ para todo $0 \leq i \leq n$, $n \geq 0$. Ou seja,
$$\rho = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots \xrightarrow{\alpha_n} s_n$$
- O *fragmento de execução* é infinito se for uma sequência infinita
$$\rho = s_0\alpha_1 s_1\alpha_2 s_2 \ldots$$
ou seja
$$\rho = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots$$

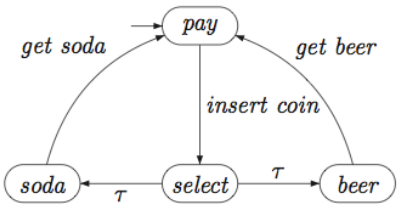
Execução

Um (fragmento de) execução é

- *maximal* se é finito e termina num estado terminal ou se é infinita;
- *inicial* se começa num estado inicial, i.e $s_0 \in I$

Uma *execução* é um fragmento inicial e maximal. *Isto é começa num estado inicial e ou termina num estado terminal ou é infinito.*

Máquina de venda- IV



$$
\begin{aligned}
\rho_1 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \cdots \\
\rho_2 &= select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \cdots \\
\rho_3 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda.
\end{aligned}
$$

ρ_1 é uma execução mas ρ_3 não porque não há estado terminal e ρ_2 também não porque não é inicial.

Estados atingíveis

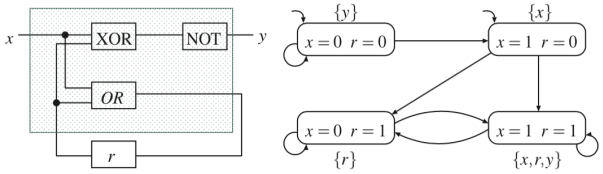
Seja $T = (S, Act, \longrightarrow, I, AP, L)$. Um estado $s \in S$ é *atingível* se existe uma um fragmento de execução inicial e finito que termina em s.

$$\rho = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots \xrightarrow{\alpha_n} s_n = s$$

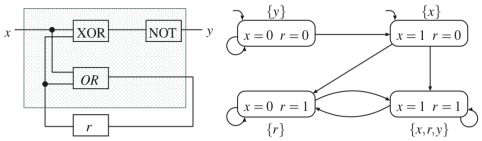
O conjunto $Reach(T)$ é conjunto de todos os estados atíngiveis.

Circuito Sequencial

Um circuito sequencial com um registo de um bit r, entrada de 1 bit x e saída de um bit y onde a função de controlo para y é $\lambda_y = \neg(x \oplus r)$ e a função para a actualização do valor de r é $\delta_r = x \vee r$.



Circuito Sequencial



- Supondo que no início $r = 0$,
- $S = Eval(x, r)$ é o conjunto dos possíveis valores de x e r
- $I = \{\langle x = 0, r = 0\rangle, \langle x = 1, r = 0\rangle\}$
- Neste caso as ações são irrelevantes

- As transições resultam de avaliar λ_y e δ_r. Por exemplo $\langle x = 0, r = 1\rangle \longrightarrow \langle x = 1, r = 1\rangle$ se o novo bit de entrada é 1.
- $AP = \{x, y, z\}$ e cada estado é etiquetado com as variáveis que são verdadeiras nesse estado. Por exemplo $L(\langle x = 0, r = 1\rangle) = \{r\}$ e $L(\langle x = 1, r = 1\rangle) = \{x, y, r\}$
- Podemos exprimir a propriedade:"o bit y é 1 um número infinito de vezes".

Qualquer circuito sequencial pode ser representado por um sistema de transições.

Programas e Transições condicionais

if $x \mod 2 = 1$ **then**
  $x := x + 1$
else
  $x := 2x$

Para modelar fragmentos de programa (com estruturas de dados) é necessário considerar *transições condicionais*. Para isso associa-se ao programa um digrafo (corresponde a uma semântica operacional) do qual se pode depois construir um sistema de transições.

Grafo de programa

Seja Var um conjunto de variáveis de diversos tipos Var e cujos valores são dados por uma função de avaliação $\eta \in Eval(Var)$, designando-se por $dom(x)$ o domínio/tipo de x. Num grafo de programa

- os vértices são constituídos por uma componente de controlo, i.e. uma localização ℓ o ponto do programa que está a ser executado. Nesse ponto os valores da função de avaliação η das variáveis simula o estado da memória.
- os arcos estão etiquetados com condições sobre as variáveis e ações;
- Uma condição é $g \in Cond(Var)$ onde $Cond(Var)$ é um conjunto de condições Booleanas sobre Var onde as fórmulas atómicas são da forma

  $$\bar{x} \in \bar{D}$$

  onde $\bar{x} = (x_1, \ldots, x_n)$ e $\bar{D} \subseteq dom(x_1) \times \cdots \times dom(x_n)$.
- Sendo $(-3 < x - x' \leq 5) \wedge (x \leq 2x') \wedge y = green$ então $dom(x) = dom(x') = \mathbb{Z}$ e e.g. $dom(y) = \{red, green\}$. A condição $(-3 < x - x' \leq 5$ corresponde na notação anterior a

  $$(x, x') \in \{(n, m) \in \mathbb{N}^2 \mid -3 < n - m \leq 5\}$$

  . E $y = green$ a $y \in \{green\}$.

- o *efeito* duma ação (comando) é uma função

$$Effect : Act \times Eval(Var) \to Eval(Var)$$

.

- Se α é $x := x + 1$ e $\eta(x) = 3$ então $Effect(\alpha, \eta)(x) = 4$.

Grafo de programa

Um grafo de programa PG sobre um conjunto Var de variáveis tipificadas é um tuplo $(Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$

- Loc, conjunto de localizações
- Act, conjunto de ações (comandos)
- $Eval(Var)$ é o conjunto de avaliações das variáveis
- $Effect : Act \times Eval(Var) \to Eval(Var)$
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$ relação de transição condicional
- $Loc_0 \subseteq Loc$ é o conjunto das localizações iniciais
- $g_0 \in Cond(Var)$ é a condição inicial

Se $(\ell, g : \alpha, \ell') \in \hookrightarrow$ escreve-se $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$ e g é a *guarda*, que pode ser omitida se for uma tautologia. Se a guarda não for verdadeira a transição não é efectuada, e o sistema pára/bloqueia.

Máquina de venda - V

Suponhamos que a máquina de venda conta o número de bebidas e devolve o dinheiro se a máquina estiver vazia. Sejam

- $Var = \{nsoda, nbeer\}$ ambas com domínio $\{0, \ldots, max\}$.
- Então $\eta \in Eval(Var)$ é qualquer função que atribuí a $nsoda$ e $nbeer$ um valor nesse domínio.
- $Loc = \{start, select\}$
- $Loc_0 = \{start\}$
- $Act = \{bget, sget, coin, ret\_coin, refill\}$

•

$$\begin{aligned} Effect(coin, \eta) &= Effect(ret\_coin, \eta) = \eta \\ Effect(sget, \eta) &= \eta[nsoda = nsoda - 1] \\ Effect(bget, \eta) &= \eta[nbeer = nbeer - 1] \\ Effect(refill, \eta) &= \eta[nsoda = max, nbeer = max] \end{aligned}$$

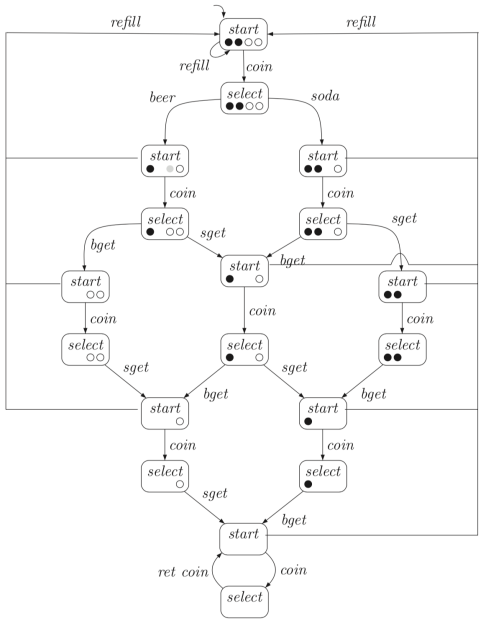
- g_0 é $(nsoda = max \wedge nbeer = max)$

Máquina de venda com contagem

- temos as seguintes transições $\hookrightarrow$:
- $start \overset{true:coin}{\hookrightarrow} select$
- $start \overset{true:refill}{\hookrightarrow} start$
- $select \overset{nsoda>0:sget}{\hookrightarrow} start$
- $select \overset{nbeer>0:bget}{\hookrightarrow} start$
- $select \overset{nsoda=0 \wedge nbeer=0:ret\_coin}{\hookrightarrow} start$

Um PG não é um sistema de transições mas pode ser expandido para um, sabendo-se o valor das variáveis (p.e. $max = 2$). Suponhamos • uma cerveja e ◦ a soda.

Máquina de venda com contagem



Sistema de transições de um grafo de programa

- os estados são constituídos por uma componente de controlo, i.e. uma localização ℓ e uma função de avaliação η das variáveis (simula o estado da memória e o ponto do programa que está a ser executado). I.e $\langle \ell, \eta \rangle$
- os estados iniciais são as localizações iniciais que satistazem a condição inicial
- AP tem as localizações (para se saber onde o programa está) e as condições Booleanas sobre as variáveis. Ex $(x \leq 5) \wedge (\ell \in \{1, 2\}$
- As transições são tais que se $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$ é uma transição condicional em PG e $\eta \models g$ então há uma transição por α de $\langle \ell, \eta \rangle$ para $\langle \ell', Effect(\alpha, \eta) \rangle$

Sistema de transições de um grafo de programa- Formalmente

Sendo $PG = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ o sistema de transições associado é $T(PG) = (S, Act, \longrightarrow, I, AP, L)$ onde

- $S = Loc \times Eval(Var)$
- $\longrightarrow \subseteq S \times Act \times S$ definido por

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \wedge \eta \models g}{\langle \ell, \eta \rangle \overset{\alpha}{\longrightarrow} \langle \ell', Effect(\alpha, \eta) \rangle}$$

onde $\eta \models g$ indica que g é verdade para a atribuição de valores às variáveis η.

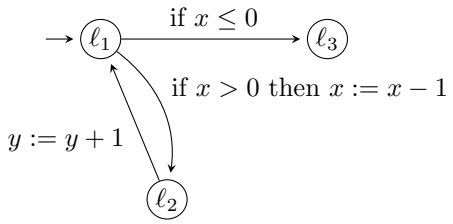
- $I = \{ \langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0 \}$
- $AP = Loc \cup Cond(Var)$
- $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{ g \in Cond(Var) \mid \eta \models g \}$

Exemplos

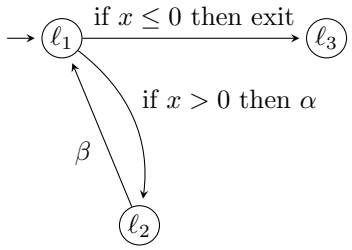
Determinar o grafo de programa do seguinte fragmento de programa, identificando as localizações (estados de controlo) com as respectivas instruções. Dar nomes às açcões.

while $x > 0$ **do**
  $x := x - 1$
  $y := y + 1$

O grafo de programa é:

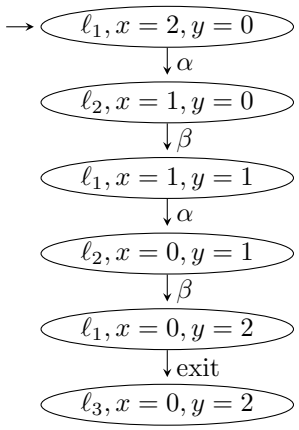


Seja $\alpha = x := x - 1$, $\beta = y := y + 1$



Exemplos

Determinar o respectivo sistema de transições supondo no início que $x = 2, y = 0$.



Exercícios

1. Ler os Cap. 1 e 2 de [AFPMdS11].
2. Ler os Cap. 1 e 2.1 de [BKL08].
3. Mostrar que para qualquer circuito sequencial se pode construir um sistema de transições.
4. Verificar o T(PG) da máquina de vendas usando as regras.
5. Resolver o Exercício 2.1 (a) de [BKL08].

Referências

[AFPMdS11] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. *Rigorous Software Development: An Introduction to Program Verification*. Springer, 2011.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.